Univerzitet u Nišu
Elektronski fakultet

Predrag S. Stanimirović
Gradimir V. Milovanović

PROGRAMSKI PAKET MATHEMATICA I PRIMENE

Edicija: **Monografije**

Predrag S. Stanimirović
Gradimir V. Milovanović

PROGRAMSKI PAKET MATHEMATICA I PRIMENE

Publikacije Elektronskog fakulteta u Nišu

Edicija: Monografije

RECENT DEVELOPMENTS IN ABSTRACT HARMONIC ANALYSIS WITH APPLICATIONS IN SIGNAL PROCESSING (Copublisher "Nauka", Beograd), Eds. R. Stanković, M. Stojić and M. Stanković, 1996

P.D. Rančić, PRILOZI SVETLOTEHNIČKIM KARAKTERIZACIJAMA. Sveska 4: Osvetljenje zatvorenih prostora. A ' raspodela svetlosnog fluksa i osvetljenosti, 1997

Z. Nikolić, PERFORMANSE SISTEMA SA PROŠIRENIM SPEKTROM, 1999

M.Č. Stefanović, DETEKCIJA SIGNALA U BELOM I OBOJENOM GAUSOVOM ŠUMU, 1999

D. Tasić, N. Rajaković, UTICAJ POTROŠNJE NA NAPONSKU NESTABILNOST ELEKTROENERGETSKOG SISTEMA, 2000

M.Č. Stefanović, PERFORMANSE DIGITALNIH TELEKOMUNIKACIONIH SISTEMA, 2000

D. Tasić, TERMIČKI ASPEKTI STRUJNE OPTERETLJIVOSTI PROVODNIKA NADZEMNIH ELEKTROENERGETSKIH VODOVA, 2002

S. Detelić, RAZVOJ SAVREMENOG DRUšTVA, 2002

P.S. Stanimirović, G.V. Milovanović, PROGRAMSKI PAKET MATHEMATICA I PRIMENE, 2002

Univerzitet u Nišu
Elektronski fakultet

Predrag S. Stanimirović
Gradimir V. Milovanović

PROGRAMSKI PAKET MATHEMATICA I PRIMENE

Edicija: **Monografije**

2002.

Dr Predrag S. Stanimirović, vanr. prof. Prirodno-matematičkog fakulteta u Nišu
Dr Gradimir V. Milovanović , red. prof. Elektronskog fakulteta u Nišu
PROGRAMSKI PAKET MATHEMATICA I PRIMENE

Izdavač: Elektronski fakultet u Nišu
P. fah 73, 18000 Niš
http://www.elfak.ni.ac.yu

Recenzenti: Dr Vera Kovačević -Vujčić
redovni profesor Fakulteta organizacionih nauka u Beogradu
Dr Ljubiša Kocić
redovni profesor Elektronskog fakulteta u Nišu

Glavni i odgovorni urednik: Prof. dr Mile Stojčev

Odlukom Naučno-nastavnog veća Elektronskog fakulteta u Nišu, br. 1/0-05-003/99-003 od 22.08.2002., rukopis je odobren za štampu kao

ISBN 86-80135-21-6

CIP - Каталогизација у публикацији
Народна библиотека Србије, Београд

517 396 41

STANIMIROVIĆ, Predrag S.
Programski paket MATHEMATICA i primene/ Predrag S. Stanimirović, Gradimir V. Milovanović. - Niš: Elektronski fakultet, 2002 (Leskovac: Đak). - XII, 242 str. : graf. prikazi; 24 cm. - (Edicija Monografije ; [Elektronski fakultet, Niš])

Na vrhu nasl. str.: Univerzitet u Nišu.

Tiraž 300. – Bibliografija uz svako poglavlje

Tiraž: 300 primeraka

Dr Predrag S. Stanimirović je rođen u Leskovcu 1959. godine. Diplomirao je, magistrirao i doktorirao iz oblasti matematičkih nauka na Filozofskom fakultetu u Nišu, studijska grupa za matematiku, 1983., 1990. i 1996. godine, respektivno. Sada je vanredni profesor Prirodno-matematičkog fakulteta u Nišu, za predmete: *Programski jezici, Matematičko programiranje* i *Operaciona istraživanja*. Oblasti njegovog naučnog interesovanja su matematičko programiranje, generalisani inverzi i programski jezici. Kao autor ili koautor objavio je preko 70 naučnih radova u domaćim i međunarodnim časopisima (videti http://pmf.pmf.ni.ac.yu). Član je YUPIMa, referent za Mathematical Reviews i recenzent u više domaćih i međunarodnih časopisa.

Dr Gradimir V. Milovanović je rođen 1948. godine u Zorunovcu kod Knjaževca. Diplomirao je 1971. godine na Elektronskom fakultetu u Nišu, a magistrirao 1974. i doktorirao 1976. iz oblasti matematičkih nauka. Od 1986. redovni je profesor na Elektronskom fakultetu u Nišu i šef Katedre za matematiku. Oblasti njegovog naučnog interesovanja su numerička analiza, teorija aproksimacija, ortogonalni sistemi i specijalni problemi matematičke analize. Objavio je desetak knjiga i monografija i preko 200 naučnih radova u međunarodnim časopisima (videti http://gauss.elfak.ni.ac.yu). Član je Naučnog društva Srbije i profesionalnih naučnih asocijacija (YUPIM, AMS, SIAM, GAMM), član redakcije većeg broja matematičkih časopisa u Srbiji i inostranstvu, rukovodilac više naučnih projekata. Referent je za Mathematical Reviews i Zentral-blatt für Mathematik, recenzent je za najpoznatije međunarodne matematičke časopise i predavač po pozivu na mnogim međunarodnim naučnim skupovima.

Edicija: **Monografije**

ISBN 86-80135-21-6

PREDGOVOR

Ovaj tekst ima za cilj da se opiše programski paket *MATHEMATICA* kao i njegova primena u većem broju matematičkih oblasti. Naravno, akcenat je pre svega na primeni simboličke obrade podataka. Koliko je autorima poznato, do sada nije objavljena knjiga slične sadržine na srpskom jeziku.

Saglasno osnovnoj nameri, knjiga je podeljena u dve glave. Prva glava sadrži opis programskog paketa *MATHEMATICA*, gde je učinjen pokušaj da se *MATHEMATICA* opiše kao programski jezik. U tom smislu, vršeno je upoređivanje standardnih funkcija iz *MATHEMATICA* sa odgovarajućim funkcijama i procedurama u proceduralnim programskim jezicima, kakvi su *PASCAL* i *C*. Čitaocu skrećemo pažnju da prva glava ne sadrži kompletan opis paketa *MATHEMATICA*. Takav opis bi sam po sebi zahtevao previše prostora. Osim toga, naš glavni cilj je bio da se *MATHEMATICA* primeni u rešavanju najvažnijih matematičkih problema. Stoga, izloženi su najvažniji pojmovi i oni su interpretirani u svetlu tradicionalnog stila programiranja. U tom smislu, može se reći da je prva glava uvodnog karaktera. U drugoj glavi je opisana simbolička implementacija većeg broja matematičkih problema. Svaka oblast primene predstavlja posebno poglavlje. U prvom poglavlju su opisani neki algoritmi sortiranja kao i njihova implementacija. Predmet drugog poglavlja jeste mogućnost višestruke upotrebe funkcionalnog argumenta u različitim programskim jezicima. U trećem poglavlju se izučava implementacija najvažnijih problema koji se rešavaju tehnikom pretraživanja sa vraćanjem. Četvrto poglavlje izučava numeričko i simboličko izračunavanje determinanti u paketu *MATHEMATICA*. U petom poglavlju se izučavaju neke primene linearnog programiranja. Predmet šestog poglavlja je simbolička implementacija nekih metoda višekriterijumske optimizacije. Sedmo poglavlje izučava primenu paketa *MATHEMATICA* u dokazivanju nekih stavova u Boole-ovoj algebri. Sedmo poglavlje izučava lokacijske i mrežne probleme, dok se primena simboličkog procesiranja podataka u generisanju regularnih izraza izučava u devetom poglavlju. Deseto poglavlje sadrži interesantnu primenu paketa *MATHEMATICA* u modeliranju Tjuringove mašine. Implementacija dvofaznog simpleks metoda i nekih njegovih modifikacija opisana je u poglavlju 11. Najzad, u dvanaestom poglavlju se izučava analogija unarnim par-funkcijama u paketu *MATHEMATICA*.

Simbolička implementacija metoda nelinearne optimizacije jeste prirodan nastavak sadržaja ove knjige. Međutim, zbog obima materijala, ovaj deo je predmet posebne knjige.

S obzirom na raznovrsnost matematičkihkih oblasti koje su obuhvaćene sadržajem ove knjige, odlučili smo da reference budu na kraju svakog poglavlja.

Pisanje ove knjige i odgovarajućih programa zahtevalo je mnogo napora i vremena. U tome smo imali pomoć velikog broja mlađih saradnika. Njima se ovom prilikom zahvaljujemo. Mr Milan Tasić, asistent Tehnološkog fakulteta u Leskovcu, napisao je i testirao veći broj programa i učestvovao u kompjuterskom procesiranju dela teksta. Dr Nebojša Stojković, asistent Ekonomskog fakulteta u Nišu, zaslužan je u velikoj meri za neke rezultate koji su sadržani u poglavlju 11. Za implementaciju simpleks metoda linearnog programiranja veliki doprinos dao je Ivan Stanković, asistent pripravnik na Prirodno-matematičkom fakultetu u Nišu.

Ova knjiga može biti korisna studentima redovnih i poslediplomskih studija na prirodno-matematičkim i tehničkim fakultetima. Knjiga sadrži neke od rezultata koje su autori publikovali u svojim prethodnim radovima.

Recenzenti dr Vera Vujčić-Kovačević, redovni profesor Fakulteta organizacionih nauka u Beogradu i dr Ljubiša Kocić, redovni profesor Elektronskog fakulteta u Nišu, pomogli su svojim savetima i sugestijama u poboljšanju kvaliteta teksta. Koristimo ovu priliku da im se zahvalimo za trud koji su uložili.

Niš, avgust 2002. *Autori*

S A D R Ž A J

I GLAVA

UVOD U PROGRAMSKI JEZIK *MATHEMATICA*®

1. O PAKETU MATHEMATICA

MATHEMATICA je programski paket za matematičke i druge primene. Do sada je korišćena u praktične svrhe, pomogla je rešavanju mnogih teorijskih problema. Takođe, *MATHEMATICA* je izučavana i od strane studenata. Primena mathematice se rasprostire u svim poljima nauke, u tehnologiji i biznisu. Ovaj programski paket ima velike mogućnosti. *MATHEMATICA* je posebno pogodna za sledeće primene:

- obrada numeričkih podataka,
- sposobnost simboličkog procesiranja,
- sistem za grafičko prikazivanje podataka i funkcija.

Za razliku od klasičnih (proceduralnih) programskih jezika kao što su *BASIC* ili *FORTRAN,* koji imaju oko 30 ugrađenih matematičkih operacija, *MATHEMATICA* ima hiljade različitih operacija.
MATHEMATICA je potpuno integrisano okruženje za tehnička izračunavanja. Često se kaže da nastanak paketa *MATHEMATICA* obeležava početak modernog tehničkog izračunavanja. Počev od 1960-tih nastali su individualni paketi za specifična numerička, algebarska, grafička izračunavanja i druge potrebe. Međutim, *MATHEMATICA* je ujedinila sve te asekte u jedan koherentni sistem na jedan jedinstveni način. Ključna ideja je bilo otkriće simboličkog programskog jezika koji može da manipuliše vrlo velikim opsegom objekata koji se pojavljuju u tehničkim izračunavanjima. *MATHEMATICA* igra ključnu ulogu u mnogim važnim otkrićima, i predstavlja bazu za hiljade naučnih radova. U inženjerstvu je postala standardni alat kako za istraživanja tako i za proizvodnju. U finansijama igra značajnu ulogu u razvitku iskustvenog finansijskog modelovanja i široko se koristi u mnogim problemima planiranja i analize. Takođe, njena uloga je važna u informatici i razvoju softvera.
Najveći deo korisnika paketa *MATHEMATICA* jesu profesionalci u tehničkim naukama. Međutim, ona se aktivno koristi i u obrazovanju. Više stotina kurseva, počev od visokoškolskih institucija pa do srednjih škola bazirano je na paketu *MATHEMATICA*. Takođe, zahvaljujući različitim verzijama za

studente, ona postaje važan alat kako za studente tehničkih nauka, tako i za ostale studente širom sveta. Korisnici ovog paketa se nalaze na svim kontinentima, uključuju sva godišta, sve vrste zanimanja, na primer, umetnike, kompozitore, lingviste i pravnike.

Na tehničkom nivou, *MATHEMATICA* se široko koristi kao najvažniji podvig softverskog inženjetrstva. Predstavlja jednu od najvećih aplikacija koja je ikada razvijena, i sadrži ogroman niz novih algoritama i važnih tehničkih inovacija. Među mnogim inovacijama je koncept interaktivnog dokumenta, poznat ka *notebook*.

MATHEMATICA je matematički softver (jezik i paket) za simboličko i numeričko rešavanje problema iz svih poznatih oblasti matematike, fizike i drugih oblasti nauke, tehnologije, finansija, medicine, istraživanja, obrazovanja, itd. Namenjena je kako korisnicima (đacima, studentima, inženjerima) za rešavanje već poznatih, proučenih problema, tako i istraživačima koji je mogu upotrebiti za najkomplikovanije proračune i analize.

MATHEMATICA je razvijena u softverskoj kompaniji Wolfram Research, vlasnika Stephena Wolframa. Prva verzija se pojavila 1988. godine, a 1989. verzija *MATHEMATICA* 2.0 za DOS operativni sistem. Verzija *MATHEMATICA* 2.2 vezana je za Windows 3.11 i zahteva najmanje 8Mb RAM memorije. *MATHEMATICA* 3.0 zahteva operativni sistem Windows, najmanje 486 procesor i 16Mb RAM memorije. Danas su aktuelne verzije 4.0 i 4.1, i razvijene su za mnoge operativne sisteme.

Glavni deo softvera *MATHEMATICA* je *Kernel* (jezgro) koje služi za obavljanje matematičkih operacija i radi nezavisno od računara na kome je instaliran. Korisnik postavlja svoje zahteve u radnom prostoru (*NoteBook*-beleška, sveska). Radni prostor je struktuirani interaktivni dokument koji se sastoji od niza ćelija (*cells*). Svaka ćelija je označena uglastom zagradom na desnoj strani dokumenta. Ćelija može da sadrži: tekst, formule ili grafiku.

Radni prostor uspostavlja dvosmernu vezu između korisnika i jezgra, tj. važi:

KORISNIK ⇔ *RADNI PROSTOR* ⇔ *JEZGRO*

Radni prostor prima podatke koje korisnik šalje. Kada se zatraži izračunavanje prosleđuje ih jezgru koje ih obrađuje. Zatim jezgro vraća rezultate koji se prikazuju u radnom prostoru. Za nove korisnikove zahteve, radni prostor stvara novu ćeliju.

Izračunavanje ćelije se zahteva istovremenim pritiskom tastera *Shift* i *Enter*. Tada se ćeliji dodeljuje oznaka oblika *In*Š*n*Ć*,* gde je *n* odgovarajući prirodan broj. Rezultat se upisuje u novu ćeliju koja je označeba sa *Out*Š*n*Ć*,* i te dve ćelije se grupišu.

Primer. otkucamo 1+1 i pritisnemo Š*Shift*Ć i Š*Enter*Ć. Dobijamo
InŠ1Ć:=1+1
OutŠ1Ć=2
Prekid računanja se postiže tasterima Š*Alt*ĆŠ,Ć ili Š*Alt*ĆŠ.Ć. Računanje se može nastaviti naredbom *Return*ŠĆ.

Kako *MATHEMATICA* pamti sve unose i rezultate od početka rada, zbog toga ponekad mora da se isprazni memorija. Najlakši način da se to postigne jesu naredbe *Quit* ili *Exit*, čime se prekida rad jezgra. To se može postići i pomoću glavnog menija, opcija *Kernel*. Pri prvom sledećem izračunavanju, jezgro se pokreće iz početka.
Radni prostor sofvera *MATHEMATICA* ima sledeći meni:

File, Edit, Cell, Format, Input, Kernel, Find, Window, Help.

Svaki od njih ima svoj podmeni.

(1) Meni ***File*** omogućava stvaranje novih dokumenata, otvaranje postojećih i pamćenje nove verzije, te štampanje dokumenata. Opcije na ovom meniju su: *New, Open, Close, Save, Save As, Save As Special, Open Special, Import, Send to, Send Selection, Palletes, Notebooks, Printing Settings, Print, Exit*.

(2) Meni ***Edit*** omogućava izmene sadržaja dokumenata. *Edit* ima sledeće opcije: *Undo, Cut, Copy, Paste, Clear, Copy As, Save Selection As, Select All, Insert Object, Motion, Expression Input, Preferences*.

(3) Meni ***Cell*** omogućava rad sa ćelijama dokumenta i njihovo organizovanje na različite načine. Njegove opcije su: *Convert To, Display As, Default Input Format Type, Default Output Format Type, Cell Properties, Cell Grouping, Divide Cells, Merge Cells*.

(4) Meni ***Format*** pruža mogućnost za izvođenje raznih operacija sa tekstom i graficima. Podmeniji su: *Style, Font, Size, Color, Show ToolBar, Magnification*.

(5) Meni ***Input*** omogućava različitu prezentaciju ulaznih i izlaznih podataka. Njegove opcije su: *Get Graphics Coordinates, 3D View Point Selector, Color Selector, Record Sound, Get File Path, Create Table*,

(6) Meni ***Kernel*** omogućava upravljanje jezgrom pomoću sledećih stavki: *Evaluation, Interrupt, Abort Evaluation, Start Kernel, Quit Kernel, Default Kernel, Kernel Configuration*.

(7) Meni ***Find*** služi za pretraživanje u dokumentu i izmmenu dokumenta: *Find, Enter Selection, Find Next, Find Previous, Find in Cell Tags, Replaces, Replace All, Make Index*.

(8) Meni ***Window*** omogućava da se podesi raspored prozora koji prikazuju otvorene dokumente. Njegove opcije su: *Stack Windows, Tile Windows Wide, Tile Windows Tall, Messages*.

(9) Meni ***Help*** pruža pomoć i informacije korisniku. Kompletna kniga *Mathematica Book* je ugrađena u help. Njegove opcije su: *Help, Registration, Find in Help, Why the Beep, About MATHEMATICA*.

1.1. INTERPRETATOR

MATHEMATICA je programsi jezik interpretatorskog tipa. U interaktivnom režimu rada interpretatora očekuje se unos od strane korsnika, obrađuje se uneti izraz, prikazuje se rezultat, i ponovo se se očekuje novi izraz za evaluaciju. Beskonačan ciklus u interpretatoru se odvija preko niza naredbi *In*Š*n*Ć*:*= i niza rezultata *Out*Š*n*Ć=. Iza promta := korisnik unosi naredbu a u liniji *Out*Š*n*Ć program prikazuje odgovarajući rezultat. Tasterima Š*Shift*ĆŠ*Enter*Ć se naznačuje da je završen unos u liniji :=, i tada može da počne obrada unetog izraza. Kao deo tog "glavnog ciklusa", *MATHEMATICA* održava i koristi različite globalne objekte. Upotreba objekata iz globalnog okruženja zamenjuje promenljive parametre iz proceduralnih programskih jezika.

MATHEMATICA pamti *Input* i *Output* linije. U velikima plikacijama, ovo pamćenje može da okupira mnogo kompjuterske memorije. Memorija koja je zauzeta za pamćenje vrednosti *In*ŠĆ i *Out*ŠĆ izraza se može osloboditi eksplicitno koristeći izraze *Unprotect*Š*In,Out*Ć, i *Clear*Š*In,Out*Ć. Može se pamtiti samo određeni broj *In*ŠĆ i *Out*ŠĆ izraza postavljanjem vrednosti globalne promenljive *$HistoryLength*.

1.1.1. Ulazne i izlazne naredbe

Ulazne naredbe obezbeđuju naknadni unos podataka u program. U *MATHEMATICA* se podaci mogu unositi interaktivno, direktno sa tastature. Funkcija *Input*ŠĆ učitava jedan izraz sa tastature i vraća njegovu vrednost kao rezultat. Izraz *Input*ŠĆ otvara prozor za unos podataka ili čitavog niza dodatnih naredbi u program. Ova naredba učitava izraze unete pomoću tastature i u daljem radu ih obrađuje kao sastavni deo programa.

Naredba *Read*ŠĆ i *OpenRead*ŠĆ obezbeđuju preuzimanje podataka iz datoteka ili neke ćelije. Rezultat računanja može se videti navođenjem imena rezultujuće promenljive ili neke izlazne naredbe. Izlazna naredba

*Print*ŠĆ vrši prenos podataka ili poruka na ekran. Tekst na izlazu se navodi između apostrofa.

InputŠ Ć	interaktivni unos jednog izraza
InputŠ"tekst"Ć	intraktivni unos, koristeći *tekst* kao odziv
InputStringŠ Ć	interaktivni unos niza karaktera
InputStringŠ"tekst" Ć	interaktivni unos niza karaktera, koristeći *tekst* kao odziv
PrintŠizr_1, izr_2,..Ć	ispisivanje vrednosti izraza na ekran.

Za vreme rada korisnik može da pristupi svim prethodno unetim izrazima, kao i dobijenim rezultatima.

InStringŠnĆ	tekstualni oblik *n*-te ulazne linije
%n ili OutŠnĆ	izraz *n*-te izlazne linije
%...% ili **OutŠ-nĆ**	izraz *n*-te izlazne linije, brojano od kraja.

Mogu se dodavati nove funkcije, potrebne u različitim specijalizovanim oblastima. Pre svega, mogu se koristiti funkcije koje su sadržane u "džepovima" (*packages*), učitavanjem njihovog sadržaja. Takođe, korisničke funkcije zapisane u nekoj datoteci mogu se uneti u program izrazom <<*ime_ fajla* gde izraz *ime_ fajla* predstavlja ime datoteke sa sadržajem željenih korisničkih funkcija. Na kraju, sve funkcije koje su definisane u aktuelnom okruženju mogu se evaluirati i koristiti.

1.1.2. Informacije preuzete iz jezgra

Različite informacije iz jezgra mogu se dobiti na sledeći način:

?ime	prikazuje informaciju za *ime*
??ime	prikazuje detaljniju informaciju za *ime*
?a*	prikazuje informaciju za sve objekte čija imena počinju slovom *a*
?Asa*	prikazuje informaciju za sve objekte čija imena počinju znacima *Asa*.

Informacije vezane za standardnu funkciju *Log* mogu se dobiti pomoću izraza *?Log*.

? Log

```
Log[z] gives the natural logarithm of z (logarithm
   to base e). Log[b, z] gives the logarithm to base b. More…
```

Mogu se zahtevati informacije za proizvoljni objekat, bilo da je on ugrađen, bilo učitan, ili uveden od strane korisnika. Dodatne informacije o objektu se mogu dobiti koristeći *??*.

```
?? Log
Log[z] gives the natural logarithm of z (logarithm
   to base e). Log[b, z] gives the logarithm to base b. More…
Attributes[Log] = {Listable, NumericFunction, Protected}
```

Informacije o svim objektima čija imena počinju sa *Lo* mogu se dobiti izrazom

```
? Lo*
System`
Locked  LogGamma      LogIntegral  Loopback
Log     LogicalExpand LongForm     LowerCaseQ
```

Izrazom ?*Asa* dobija se informacija o pojedinačnom objektu *Asa*. Koristeći "metakarakter" *, može se dobiti informacija o kolekciji objekata čija imena počinju sa *Asa*. Džoker * može da bude zamenjen bilo kojom sekvencom karaktera. Znak * se može postaviti na bilo kom mestu, ne samo na početku izraza o kojima se zahteva informacija. Na primere, *?*Expand* zahteva informaciju o svim objektima čija se imena završavaju sa *Expand*. Slično, izrazom *?x*0* dobija se informacija o svim objektima čija imena počinju sa *x*, završavaju sa *0*, i imaju proizvoljnu sekvencu karaktera između njih.

1.2. STILOVI PROGRAMIRANJA U PAKETU *MATHEMATICA*

Funkcionalni programski jezici se baziraju na primeni funkcija na aktuelne parametre. Funkcionalni programski jezici su pogodni za programiranje ekspertnih sistema sa bazama znanja. U ovakvom stilu programiranja osnovni pojam je funkcija. Funkcijski program je funkcija ili grupa funkcija, obično komponovan iz prostijih funkcija. Veza između funkcija je višestruka: jedna funkcija možze da poziva drugu ili pak rezultat jedne funkcije može da se koristi kao argument druge funkcije. *MATHEMATICA* nije čisto funkcionalni programski jezik. U njoj se mogu naći ideje logičkog i objektnog programiranja. Osim toga, ogromno bogatsvo upravljačkih struktura afirmiše programski paket *MATHEMATICA* kao proceduralni programski jezik. *MATHEMATICA* je proceduralna koliko i *C*, funkcionalna koliko i *LISP*. Osim toga, poseduje veoma razvijen aparat za simboličku manipulaciju podacima.

Zastupnici ideje funkcionalnog programiranja koriste sledeće argumente:
1. Programi su kraći.
2. Programi su pogodni za dokazivanje korektnosti.
3. Programi su dobro matematički zasnovani.
4. Aktivnosti koje se koriste su jednostavnije nego u drugim stilovima programiranja. Postoje dve vrste osnovnih aktivnosti:
   a) Definisanje funkcije: pridruživanje imenu funkcije vrednosti nekog izraza, koji može da sadrži i druge funkcije.
   b) Primena funkcije, tj. poziv funkcije sa zadatim argumentima.
5. Programi su pogodni za implementaciju na paralelnim računarima.

Funkcionalni stil programiranja spada u deklarativni stil programiranja, u kome je izračunavanje, tj. izvršavanje programa zasnovano na pojmu funkcije. Funkcije su ravnopravni objekti sa drugim tipovima podataka.
Za uspešno programiranje u funkcionalnom programskom jeziku potrebno je sledeće:
1. Bogat skup funkcija koje možemo odmah da koristimo, tzv. *ugrađene* (*primitivne* ili *standardne*) funkcije (*built-in functions*).
2. Mogućnost definisanja novih funkcija, koje se nazivaju *korisničke* funkcije (*user-defined functions*). Korisničke funkcije se mogu svrstati u biblioteke funkcija i pozivati po potrebi.

U funkcionalnom programskom jeziku, program se sastoji iz niza definicija funkcija i niza njihovih poziva.
Može se reći da *MATHEMATICA* podržava funkcionalni stil programiranja. Ona dopušta samo poziv parametara po adresi. Nedostatak poziva po adresi nadoknađuje se objektima koji se pamte u globalnom okruženju. *MATHEMATICA* poseduje mnoge funkcije od kojih su neke veoma moćne i rešavaju složene probleme. Međutim, postoje problemi koji se ne mogu rešiti samo pomoću ugrađenih funkcija. Tada je potrebno uraditi više operacija ili postupati po nekom algoritmu, tj. potrebno je programirati.

1.2.1. Korisničke funkcije

Elementarne funkcije se pozivaju izrazima oblika

*Funkcija*Š*x,y,z,...*Ć.

Pored toga što poseduje veliki broj ugrađenih funkcija *MATHEMATICA* omogućava korisniku da definiše i svoje funkcije. Funkcija više promenljivih se definiše izrazom

*funkcija*Š*x_,y_,z_,...*Ć*:= izraz.*

Pri definisanju funkcija treba voditi računa da se argumenti navode u uglastim zagradama, a umesto znaka jednakosti stoji znak :=. Takođe, svaki element u listi parametara se završava znakom '_' .

Primer. Izrazom
fŠx_Ć:=(ExpŠxĆ-1)/x
Definiše se funkcija

$$f(x)=\frac{e^x-1}{x}$$

Funkcija $g(x,y,z)= x^2+y^2+z^2$ definiše se izrazom
gŠx_,y_,z_Ć:=xČ2+yČ2+zČ2.

Funkcija *f* koja pamti vrednosti koje je ranije izračunavala poziva se izrazom oblika

*f*Š*x_,y_,…*Ć*:= f*Š*x,y,z,…*Ć*=izraz,*

Koristi se kada su nam više puta potrebne već izračunate vrednosti funkcija, na primer u rekurzivnim pozivima. Ovakvim definicijama se mogu ubrzati izračunavanja rekurzivnih funkcija.

Definicija funkcije *f* može se proveriti naredbom
?f
ili
DefinitionŠfĆ,
a brisanje se postiže naredbom
f=.
ili
ClearŠfĆ.

Primer.

InŠ1Ć:=fŠx_Ć:=xČ2	Definicija funkcije $f(x)$ = x^2
InŠ2Ć:=fŠa+1Ć	
OutŠ2Ć=$(1+a)^2$	Funkcija *f* kvadrira svoj argument
InŠ3Ć:=fŠ4Ć	
OutŠ3Ć=16	Argument funkcije može biti broj
InŠ4Ć:=fŠ3x+xČ2Ć	Argument funkcije može biti
OutŠ4Ć=$\mathbf{(3x+x^2)^2}$	komplikovaniji izraz
InŠ5Ć:=ExpandŠfŠ3x+xČ2ĆĆ	Funkcija *f* se može koristiti u
OutŠ5Ć=$9x^2+6x^3+x^4$	računanju kao argument neke druge funkcije
InŠ6Ć:=pŠx_+y_Ć:=pŠxĆ+pŠyĆ	Svojstvo funkcije *p*
InŠ7Ć:=pŠa+b+cĆ	Primena definisanog svojstva
OutŠ7Ć=pŠaĆ+pŠbĆ+pŠcĆ	

InŠ8Ć:=sŠšx__,a_,y__ć,a_Ć:=ša,x,x,y,yć
InŠ9Ć:=sŠš1,2,3,4,5,6ć,4Č
OutŠ9Ć=š4,1,2,3,1,2,3,5,6,5,6ć

U sledećem primeru pokazuje se prednost rekurzivnih funkcija koje "pamte" vrednosti koje su računale u odnosu na "obične" rekurzivne funkcije. Posmatramo funkciju *fib* za generisanje Fibonačijevih brojeva:

```
fib[1] := 1
fib[2] := 2
fib[n_] := fib[n - 1] + fib[n - 2]
```

Vreme potrebno za izračunavanje Fibonačijevih brojeva se drastično povećava sa povećanjem argumenta:

```
fib[10] // Timing
{0. Second, 89}
fib[20] // Timing
{0.220000000000255 Second, 10946}
fib[30] // Timing
{26.6900000000001 Second, 1346269}
```

S druge strane, pomoću sledeće definicije kojom se pamte izračunate vrednosti

```
fib1[1] := 1; fib1[2] := 2
fib1[n_] := fib1[n] = fib1[n - 1] + fib1[n - 2]
```

znatno se smanjuje vreme izračunavanja:

```
fib1[10] // Timing
{0. Second, 89}
fib1[20] // Timing
{0. Second, 10946}
fib1[30] // Timing
{0. Second, 1346269}
```

1.2.2. Procedure

Često se događa da se određeni nizovi naredbi ponavljaju. Te naredbe se mogu grupisati zajedno u procedure. Jedna procedura je niz naredbi razdvojenih znakom ;. Izrazi u proceduri izračunavaju se jedan za drugim s leva na desno. Konačni rezultat procedure je vrednost poslednjeg izraza.

izraz $_1$; izraz $_2$; ... ; izraz $_n$	Procedura od *n* izraza
(izraz $_1$; izraz $_2$; ... ; izraz $_n$)	Procedura se može staviti u zagrade

Primer. Analiziramo sledeći izraz:
InŠ1Ć:=t=TableŠi!,ši,5ćĆ;   x=MapŠLog,tĆ;   x//N

OutŠ1Ć=š0,0.693147, 1.79176, 3.17805, 4.78749ć

Funkcija *Map* u izrazu *x=Map*Š*Log,t*Ć primenjuje funkciju *Log* nad svakim elementom liste *t*. Kao rezultat ove procedure dobija se vrednost poslednjeg izraza *x*//*N*, što predstavlja numeričku vrednost izraza *x*.

```
z = a; Do[Print[z *= z + i], {i, 3}]
a (1 + a)
a (1 + a) (2 + a (1 + a))
a (1 + a) (2 + a (1 + a)) (3 + a (1 + a) (2 + a (1 + a)))
```

Funkcija se može definisati pomoću sekvence izraza jednostavnim navođenjem niza naredbi razdvojenim znakom ; iza operatora odložene dodele *:=*. U jednoj takvoj funkciji se može izvršiti više nezavisnih operacija. Tako definisana funkcija se poziva na isti način kao i sve ostale funkcije. Vrednost poslednjeg izraza se vraća kao rezultat funkcije. Kada se funkcija definiše kao procedura, niz naredbi koje joj pripadaju se mora navesti u malim zagradama. *MATHEMATICA* daje mogućnost pisanja programa na mnogo različitih načina.

Teorijske (čiste) funkcije dozvoljavaju da se definišu funkcije koje mogu da se primene kao argumenti, bez davanja eksplicitnog imena funkciji. Postoji nekoliko zapisa teorijskih funkcija:

FunctionŠx,bodyĆ	teorijska funkcija u kojoj se x zamenjuje zadatim argumentom
FunctionŠšx$_1$,...,x$_n$ć,bodyĆ	teorijska funkcija u kojoj se koristi nekoliko argumenata
body&	teorijska funkcija u kojoj su argumenti specificirani sa # ili #1,#2,#3,...

Za rešavanje različitih problema možemo koristiti kompozicije postojećih funkcija.

Primer.

InŠ1Ć:=PositionŠš1,2,3,4,5ć/2,_IntegerĆ

OutŠ1Ć=šš2ć,š4ćć

InŠ2Ć:=MapIndexedšPower,ša,b,c,dćĆ

OutŠ2Ć=ššać,šb^{2}ć,šc^{3}ć,šd^{4}ćć

InŠ3Ć:=FixedPointListŠIfŠEvenQŠ#ĆĆ,#2,#&,10Č5Ć

OutŠ3Ć=š100000,50000,25000,125000,6250,3125,3125ć

InŠ4Ć:=ReplaceListŠšh,b,c,d,eć,->šk__,y__ć

OutŠ4Ć=šššhć,šb,c,d,ećć,ššh,bć,šc,d,ećć,ššh,b,cć,šd,ećć,ššh,b,c,dć,šeććć

1.2.3. Elementi objektno-orijentisanog programiranja

MATHEMATICA ima i neke elemente **objektno-orijentisanog programiranja**, jer omogućava korisniku da definiše svoje objekte. Pri definisanju osobina objekata prvo se navodi ime objekta koje je odvojeno znacima /: od definicije osobine objekta.

Primer. U ovom primeru su simbolu *h* dodeljene tri osobine:

```
h /: h[x_] + h[y_] := hplus[x, y];
h /: P[h[x_], x_] := hp[x];
h /: f_[h[x_]] := fh[f, x];
```

Ove osobine se mogu koristiti u programu:

```
h[2] + h[3]
hplus[2, 3]
```

Definicijama oblika *f*Š*args*Ć=*rhs* ili *f*Š*args*Ć:= *rhs* vrši se vezivanje tih definicija sa objektom *f*. Informacije o takvim definicijama se dobijaju posle izraza ?*f*. Za sve izraze kojima se dodeljuju osobine simbolu *f*, a koji imaju *f* kao *glavu* koristi se naziv *donje vrednosti* (*downvalues*) za *f*. *MATHEMATICA* takođe podržava definiciju *gornjih vrednosti* (*upvalues*), kojim se pridružuju definicije simbolima koji se ne pojavljuju direktno kao glave u tim definicijama. Gornje vrednosti se uvode koristeći znakove Č:. Posmatrajmo na primer definiciju oblika *Exp*Š*g*Š*x*_ĆĆ:= *rhs*. Ova definicija je vezana za simbol *Exp*, i može se posmatrati kao donja vrednost za *Exp*. Međutim, ovakav pristup nije najbolji, zbog toga što je *Exp* standardna funkcija, i to može da nepotrebno uspori izračunavanje izraza. bolje je da se izrazom oblika *ExpŠgŠx_ĆĆ*Č:=*rhs* definicija asocira za *g*, i da predstavlja gornju vrednost objekta *g*.

fŠargsĆ /:rhs	definicija donje vrednosti za *f*
fŠgŠargsĆ,...ĆČ:=rhs ili **fŠg,...ĆČ:=rhs**	definicija gornje vrednosti za *g*

Donja vrednost za *f* se može definisati na sledeći način:

fŠgŠx_ĆĆ := fgŠxĆ

Ova definicija se može videti kada se traži informacija za *f:*

?f

```
Global`f
f[g[x_]] := fg[x]
```

Gornja vrednost za objekat *g* se može definisati na sledeći način:

***Exp*Š*g*Š*x*_ĆĆ *Č:= expg*Š*x*Ć**

Ova definicija je asocirana za *g*:

?g

```
Global`g
e^g[x_] ^:= expg[x]
```

i nije asocirana za *Exp:*

??Exp

```
Exp[z] is the exponential function.
Attributes[Exp] = {Listable, NumericFunction, Protected, ReadProtected}
```

Definicija gornje vrednosti za *g* se koristi pri evaluaciji sledećeg izraza:

ExpŠgŠ5ĆĆ

```
expg[5]
```

Definicija za *f*Š*g*Š*x*ĆĆ bi trebalo da predstavlja gornju vrednost za *g* u slučajevima kada je funkcija *f* poznatija (češće u upotrebi) u odnosu na funkciju *f*. Na primer, u slučaju izraza *Exp*Š*g*Š*x*ĆĆ, funkcija *Exp* je standardna funkcija, dok *g* verovatno predstavlja korisničku funkciju.

Još jedna gornja vrednost za *g* je definisan izrazom

g/: gŠx_Ć + gŠy_Ć := gplusŠx, yĆ

Sve definicije vezane za objekat *g* se prikazuju na sledeći način:

?g

```
Global`g
e^g[x_] ^:= expg[x]

g[x_] + g[y_] ^:= gplus[x, y]
```

Definicija za zbir *g*-ova se koristi kad god je moguće:

gŠ5Ć + gŠ7Ć

```
gplus[5, 7]
```

Kako je unutrašnji oblik izraza *g*Š*x*_Ć+*g*Š*y*_Ć jednak *Plus*Š*g*Š*x*_Ć,*g*Š*y*_ĆĆ, prethodna definicija se može tretirati kao donja vrednost za *Plus*. U tom slučaju, kad god se definišu izrazi oblika *g*Š*x*_Ć+*g*Š*y*_Ć kao donja vrednost za *Plus*, ove definicije se koriste kad god se pojavi poziv funkcije *Plus*. Ovim se usporava evaluacija ovakvih izraza. Međutim, ako se definicija za *g*Š*x*_Ć+*g*Š*y*_Ć posmatra kao gornja vrednost za *g*, tada se pokušava primena te definicije samo kada se *g* pojavi unutar funkcije *Plus*. S obzirom da se *g* pojavljuje mnogo manje u odnosu na funkciju *Plus*, ova definicija ima mnogo više opravdanja.

Gornja vrednost se mogu upotrebiti za konstrukciju "baze" osobina partikularnog objekta. Svaka definicija koja se učini kao gornja vrednost se može asocirati sa objektom koji se posmatra, radije nego sa osobinom koja se specificira.

Sledeća definicija određuje gornju vrednost za objekat *square* kojim se daje njegova površina:

areaŠsquareĆ Č= 1

```
1
```

Sledeći izraz dodaje definiciju za *perimeter:*

perimeterŠsquareĆ Č= 4

```
4
```

Obadve definicije su sada asocirane sa objektom *square:*

?square

```
Global`square
area[square] ^= 1

perimeter[square] ^= 4
```

Izrazom oblika *f*Š*args*Ć može se definisati gornja vrednost za *g,* bilo direktno, bilo za neki objekat sa glavom *g* koji se pojavljuje u *args.* Ne mogu se asocirati definicije vezane za neki objekat *g* ako se on pojavljuje u suviše niskom nivou na levoj strani izraza koji se koristi za takvu definiciju.

U sledećem izrazu *g* predstavlja glavu jednog argumenta, i sa njim se može asocirati definicija gornje vrednosti.

g/: hŠwŠx_Ć, gŠy_ĆĆ := hwgŠx, yĆ

U izrazu koji sledi, *g* se pojavljuje isuviše duboko na levoj strani izraza koji se sa njim povezuje, te definicija gornje vrednosti koja je asocirana sa *g* ne uspeva:

g/: hŠwŠgŠx_ĆĆ, y_Ć := hwŠx, yĆ

```
TagSetDelayed::tagpos :
  Tag g in h[w[g[x_]], y_] is too deep for an assigned rule to be found.
$Failed
```

Moguće pozicije simbola u definicijama date su u sledećoj tabeli:

fŠ...Ć:=rhs	donja vrednost za *f*
f/: fŠgŠ...Ć...ĆŠ...Ć:=rhs	donja vrednost za *f*
g/: fŠ...,g,...Ć:=rhs	gornja vrednost za *g* koji prestavlja argument
g/: fŠ...,gŠ...Ć,...Ć:=rhs	gornja vrednost za *g* koji prestavlja glavu

U opštem slučaju, možete poželeti da definišete klasu apstraktnih matematičkih objekata sa imenom *quat*. Ovi objekti se mogu predstaviti izrazom oblika *quat*Š*data*Ć, i mogu imati posebne osobine u odnosu na aritmetičke operacije (na primer sabiranje i množenje). Ove osobine se mogu postaviti definicijama gornjih vrednosti za *quat* u odnosu na *Plus* i *Times*.

Definicija gornje vrednosti objekta *quat* u odnosu na *Plus*:

quatŠx_Ć + quatŠy_Ć Č:= quatŠx + yĆ

Ova definicija se koristi u simplifikaciji sledećeg izraza:

quatŠaĆ + quatŠbĆ + quatŠcĆ

```
quat[a + b + c]
```

Kada se definiše gornja vrednost za *quat* u odnosu na neku operaciju, na primer *Plus*, u stvari se proširuje domen operacije *Plus* na objekte *quat*. U definiciji "sabiranja" *quat* objekata, može se koristiti specijalna operacija sabiranja, na primer *quatPlus*, za koju se definiše odgovarajuća donja vrednost. Ovakav pristup je pogodniji nego da se koristi standardna operarcija sabiranja *Plus*, tako što se njeno standardno dejstvo predefiniše za objekte tipa *quat*.

Gornja vrednost u stvari predstavlja mogućnost da se implementira jedan aspekt objektno-orijentisanog programiranja. Simbol sličan *quat* predstavlja objekat određenog tipa. Različite gornje vrednosti za *quat* specificiraju "metode" koje definišu kako se *quat* objekti ponašaju pod određenim operacijama

2. OSNOVNI ELEMENTI JEZIKA

2.1. REZERVISANE REČI

Reči jednog jezika čije je značenje utvrđeno i ne može se promeniti ni u jednom programu napisanom na tom jeziku nazivaju se *rezervisane reči*. U *MATHEMATICA* su sve rezervisane reči *zabranjene*, tj. ne mogu se koristiti kao identifikatori u programima. Na primer, rezultat izvršenja naredbe dodele *DO*=2 jednak je 2, ali rezervisana reč *DO* nije dobila vrednost, tj. nije mogla da se upotrebi kao identifikator.

```
Do = 2
Set::wrsym : Symbol Do is Protected.
2
```

U *MATHEMATICA* postoji veliki broj rezervisanih reči. Najveći deo službenih reči je rezervisan imenima ogromnog broja standardnih funkcija, a veliki broj služi za označavanje karakterističnih konstanti.

2.2. KONSTANTE

Veličine čija se vrednost ne može menjati u toku izvršavanja programa naziva se *konstanta*. Konstantama se mogu dodeljivati simbolička imena, koja se mogu koristiti umesto njih. Ovakve simboličke konstante sreću se i u programskim jezicima *PASCAL* (naredba *const*) i *C* (naredba *#define*). Simbolička konstanta u *MATHEMATICA* je deo globalnog okruženja.

Postoje sledeći tipovi brojeva:

Integer	celi brojevi (*integers*) proizvoljne dužine
Rational	racionalni brojevi oblika *integer/integer* u najmanjoj formi
Real	približni realni brojevi sa proizvoljnom specificiranom tačnošću
Complex	kompleksan broj oblika x+*y I*, gde su *x* i *y* realni brojevi

Ono što karakteriše paket *MATHEMATICA* jeste mogućnost rada sa velikim brojevima.

Racionalni brojevi se zapisuju u obliku *brojilac/menilac*.

Realni brojevi se mogu zapisati u fiksnom ili eksponencijalnom zapisu:
- a) fiksni oblik: ±*celi.decimalni*;
- b) eksponencijalni oblik: ±*mantisa*bazaČeksponent*.

Kompleksni (*Complex*) brojevi se zapisuju u obliku *x+yI*, gde je *I* imaginarna jedinica a *x* i *y* su proizvoljni realni brojevi.

Primer.
InŠ1Ć:= **3/7**
OutŠ1Ć= $\frac{3}{7}$
InŠ2Ć:= **21/7**
OutŠ2Ć= 3
InŠ3Ć:= **1+2I**
OutŠ3Ć= 1+2 i

Takođe, može se koristiti brojni sitem sa proizvoljnom osnovom. Zapis dekadnog broja *n* u bazi *b* dobija se funkcijom:

*BaseForm*Š*n*,*b*Ć.

Primer. Vrednost izraza *BaseForm*Š37,2Ć jeste binarni broj 100101.
Brojna vrednost u proizvoljnoj bazi se može zadati izrazom

baseČČdigits,

u kome *base* predstavlja osnovu brojnog sistema, dok izraz *digits* sadrži cifre u tom brojnom sistemu.
Približna vrednost broja se dobija primenom funkcije

*NŠbroj,broj_cifara*Ć ili *broj//N.*

Primer.

2ČČ100101

```
37
```

BaseFormŠ37, 2Ć

100101_2

(* Brojna vrednost prevedena iz osnove 16 u osnovu 10 *)

16ČČffffaa00

```
4294945280
```

16ČČfffaa2 + 16ČČff - 1

```
16776096
```

Poznate simboliče konstante imaju svoja posebna, rezervisana imena:

Pi	$\pi \simeq 3.14159$
E	e≃2.71828
Degree	$\pi/180$: faktor konverzije stepena u radijane
I	i=$\sqrt{-1}$
Infinity	∞

Pi Č 2 //N

```
9.869604401089358
```

SinŠPi/2Ć

```
1
```

SinŠ20 DegreeĆ //N

```
0.3420201433256687
```

LogŠE Č 5Ć

```
5
```

LogŠ2, 256Ć

```
8
```

Strukturni izrazi su dominantni u *MATHEMATICA*. Međutim mogu se koristiti i stringovi sa velikim brojem standardnih funkcija i za manipulaciju stringovima. Proizvoljni string se zadaje u obliku *"text"*. Kada se unosi string od strane korisnika, uvek mora da bude ograđen navodnicima. Međutim, kada *MATHEMATICA* prikazuje string, navodnici se ne prikazuju.

"Ovo je string"

```
Ovo je string
```

Navodnici se mogu videti ako se zahteva ulazni oblik (*InputFform*) stringa.

InputForm[%]

"Ovo je string"

2.3. PROMENLJIVE

Promenljive su veličine koje menjaju svoju vrednost u programu. Svaka promenljiva ima svoje simboličko ime. Promenljive se nazivaju *simboli,* i predstavljaju osnovne imenovane objekte u jeziku *MATHEMATICA*. Ime koje se koristi kao simbol mora da bude sekvenca slova i cifara, koja ne počinje cifrom. Velika i mala slova se razlikuju. Vrednost simbola se definiše naredbom

promenljiva=izraz

a briše naredbom

promenljiva=.

Specijalne dodele su analogne odgovarajućim dodelama u jeziku *C*:

i++, i--	povećanje (smanjenje) vrednosti promenljive *i* za 1
++i, --i	povećanje (smanjenje) vrednosti promenljive *i* za 1
i+=di, i- =di	povećanje (smanjenje) vrednosti promenljive *i* za *di*
x*=c, x/=c	množenje (deljenje) promenljive *x* sa *c*

2.4. KOMENTARI I DOKUMENTOVANJE PROGRAMA

Ponekad je zbog boljeg razumevanja programa potrebno koristiti razne komentare. Komentari se omeđuju znacima (* i *) i ubacuju bilo gde u tekst programa. Time se čitljivost programa povećava.

(* tekst *)	komentar koji može da se nađe bilo gde u programu

```
If[a > b, (*onda*) p, (*inace*) q]
If[a > b, p, q]
```

U principu bi trebalo dokumentovati sve funkcije koje se jednom definišu da bi kasnije bile korišćene više puta. Svakoj funkciji se može dodeliti kratak tekst u kojem se opisuje šta funkcija radi i kakve parametre očekuje. Ovakav opis se funkciji *f* dodeljuje kao vrednost simbola *f::usage*, a dobija se pomoću naredbe *?f*.

f::usage="tekst"	dodeljivanja opisa funkciji *f*
?f	informacije o funkciji *f*
??f	detaljnije informacije o funkciji *f*

```
f[x_] := x^2
f::usage = "f[x] kvadrira x"
f[x] kvadrira x
??f
f[x] kvadrira x
f[x_] := x2
```

3. TIPOVI PODATAKA

MATHEMATICA spada u programske jezike sa slabim tipovima podataka. Sve promenljive koje nisu lokalne pripadaju globalnom okruženju. Ni globalne ni lokalne promenl;jive nemaju eksplicitnu definiciju tipa. Osim toga, ista promenljiva se može koristiti u veoma različitim kontekstima i može da uzima vrednosti različitih tipova:

```
a = 2
2
a = {z, b, c}
{z, b, c}
a = 2.34
2.34
```

Kao i u *LISP*u, postoje dva osnovna tipa podataka: atomi i liste. Osnovni atomični tipovi podataka su: simboli, celi brojevi, realni brojevi, racionalni brojevi, kompleksni brojevi i stringovi. Lista je uređeni skup objekata, i omogućuju da se različiti objekti grupišu i da se nad njima izvršavaju operacije. Liste su najmoćniji i najfleksibilniji tip podataka u *MATHEMATICA*. Koristeći liste mogu se izvoditi operacije sa matricama i vektorima jednostavnim pozivom odgovarajuće funkcije, a takođe su ugrađene i funkcije za izračunavanje determinante, inverzne matrice, karakterističnih korena i vektora, itd. Mnoge kombinatorne funkcije čija primena u programskim jezicima zahteva izvestan trud i znanje, dobijaju se pozivom ugrađene funkcije u *MATHEMATICA*. Lista se može definisati eksplicitnim navođenjem svih elemenata ili definisanjem pravila kojima se određuju:

š1,3,5,7ć	lista brojeva
šx,xČ2,xČ3,aabbać	lista je sastavljena od simboličnih izraza

Atomični tipovi podataka su opisani sledećom tabelom:

Symbol	simbol čije se ime izdvaja koristeći *SymbolName*

String	string čiji se karakteri izdvajaju funkcijom *Characters*
Integer	ceo broj čije se cifre izdvajaju koristeći *IntegerDigits*
Real	realni broj čije se cifre izdvajaju koristeći *RealDigits*.
Rational	racionalni broj čiji se delovi izdvajaju funkcijama *Numerator* i *Denominator*
Complex	kompleksni broj čiji se delovi izdvajaju pomoću *Re* i *Im*

Predikatske funkcije kojima se ispituje tip podataka navedene su u sledećoj tabeli:

IntegerQŠexprĆ	ispitati da li je expr ceo broj
EvenQŠexprĆ	test za paran broj
OddQŠexprĆ	test za neparan broj
PrimeQŠexprĆ	test za prost broj
NumberQŠexprĆ	test za broj bilo koje vrste
NumericQŠexprĆ	test za numeričku vrednost.
PolynomialQŠexpr,šx_1,x_2,...ć Ć	test za polinom od x_1,x_2,...
VectorQŠexprĆ	test za listu koja predstavlja vektor
MatrixQŠexprĆ	test za listu koja predstavlja matricu

Svi strukturni tipovi podataka imaju jednoobraznu strukturu , koja ima oblik

*Head*Š*Arg1, ... ,Argn*Ć.

Eksplicitna deklaracija tipova se može opciono primeniti jedino na formalne parametre funkcija.

```
Q[x_] := x^2 + 3 x
Q[5]
40
Q[1.2]
5.039999999999999
R[x_Integer] := x^3 - 2 x
R[5]
115
R[1.3]
R[1.3]
```

3.1. FUNKCIJE ZA RAD SA NUMERIČKIM VREDNOSTIMA

3.1.1. Ispitivanje tipa brojeva

Neke transformacije podataka zahtevaju da ti podaci budu određenih tipova. U takvim slučajevim se greške mogu izbeći prethodnim ispitivanjem tipova podataka pre primene takvih transformacija.

NumberQŠxĆ	testira da li je broj bilo kog tipa
IntegerQŠxĆ	testira da li je *x* ceo broj (*integer*)

Vrednost izraza *NumberQ*Š*exp*Ć je *True* u slučajevima kada je glava (*Head*) izraza *exp* jednaka *Complex, Integer, Rational* ili *Real*.

Primer.

```
NumberQ[3.76]
True
NumberQ[p]
False
Head[p]
Symbol
```

Vrednost izraza *IntegerQ*Š*exp*Ć je *True* u slučajevima kada je glava izraza *exp* jednaka *Integer*. Postoje i funkcije kojima se testiraju određene osobine celih brojeva:

EvenQŠxĆ,	**t**estira da li je *x* neparan broj
OddQŠxĆ	**t**estira da li je *x* paran broj

Vrednost izraza *EvenQ*Š*exp*Ć (*OddQ*Š*exp*Ć) je *True* ukoliko je *exp* neparan (paran) ceo broj.

PrimeQŠxĆ,	*True* ako je vrednost za *x* prost broj, inače *False*
PositiveŠxĆ	*True* ako je vrednost za *x* pozitivan broj
NegativeŠxĆ	*True* ako je vrednost za *x* negativan broj

3.1.2. Konverzija između različitih formi brojeva

U sledećoj tabeli su date osnovne funkcije za konverziju numeričkih vrednosti iz jednog oblika u drugi.

NŠx,nĆ	konvertuje *x* u približan realan broj sa najviše *n* cifara preciznosti
RationalizeŠxĆ	racionalan broj koji predstavlja aproksimaciju za *x*
RationalizeŠx,dx Ć	racionalan broj koji predstavlja aproksimaciju za *x* sa tolerancijom *dx*

```
InŠ4Ć:= N[3/7, 30]
OutŠ4Ć= 0.428571428571428571428571428571
InŠ5Ć:= N[%, 20]
OutŠ5Ć= 0.42857142857142857143
InŠ6Ć:= Rationalize[N[Pi], 10^-5]
```

OutŠ6Ć= $\frac{355}{113}$

```
InŠ7Ć:= Rationalize[N[Pi], 0]
```

OutŠ7Ć= $\frac{245850922}{78256779}$

Takođe, postoje i druge funkcije za konverziju numeričkih podataka:

RoundŠxĆ	ceo broj Š*x*Ć najbliži *x*
FloorŠxĆ	najveći ceo broj ne veći od *x*
CeilingŠxĆ	najmanji broj ne manji od *x*
SignŠxĆ	1 za *x*>0, -1 za *x*<0, i 0 za *x*=0
AbsŠxĆ	apsolutna vrednost od *x*

U sledećoj tabeli je ilustrovano dejstvo funkcija za konverziju različitih tipova brojeva.

x	**RoundŠxĆ**	**FloorŠxĆ**	**CeilingŠxĆ**
2.4	2	2	3
2.5	2	2	3
2.6	3	2	3
-2.6	-3	-3	-2

Cifre celog ili realnog broja mogu da se izdvoje u listu na više načina:

IntegerDigitsŠnĆ	lista decimalnih cifara u celom broju *n*
IntegerDigitsŠn,bĆ	cifre broja *n* u bazi *b*
RealDigitsŠxĆ	lista svih cifara realnog broja *x*
RealDigitsŠx,bĆ	cifre realnog broja *x* u bazi *b*

```
IntegerDigits[1234135634, 16]
{4, 9, 8, 15, 6, 10, 5, 2}
RealDigits[123.1356342398476]
{{1, 2, 3, 1, 3, 5, 6, 3, 4, 2, 3, 9, 8, 4, 7, 6}, 3}
```

3.1.3. Funkcije za određivanje numeričke preciznosti

MATHEMATICA omogućava rad sa proizvoljnim brojem decimala. Određena preciznost u izračunavanjima se može zahtevati sledećim funkcijama:

PrecisionŠxĆ	ukupan broj značajnih cifara u broju *x*
AccuracyŠxĆ	broj značajnih cifara iza decimalne tačke u *x*
NŠexpĆ ili **expr//**	izračunava *expr* numerički, koristeći mašinsku tačnost

Funkcija *Precision* daje meru relativne greške, dok se funkcijom *Accuracy* meri apsolutna greška. Kada se koristi izraz *NŠexpr,n*Ć, *MATHEMATICA* izračunava vrednost izraza *expr* koristeći brojeve sa *n* cifara preciznosti. Međutim, to ne znači uvek da je tačnost *n* cifara: tačnost je približna ili jednaka broju *n*.

```
N[Pi^25, 30]
```

$2.68377941431776454900992812440 \times 10^{12}$

```
Precision[%]
30
```

3.1.4. Numerička izračunavanja

U *MATHEMATICA* se mogu koristiti uobičajene aritmetičke operacije, koristeći prioritet prema standardnim matematičkim konvencijama. Prioritet se može promeniti upotrebom zagrada. *MATHEMATICA* može da radi sa velikim brojevima.

```
2.3+5.63
7.93
2.4/8.9^2+(3-5)
-1.969700795354122
2^100
1267650600228229401496703205376
2^500
327339060789614187001318969682759915221664204604306478948329136809613379640467455488327009\
 2325904157150886684127560071009217256545885393053328527589376
452/62
```

$\frac{226}{31}$

```
452./62
7.290322580645161
```

Osnovne funkcije za rad sa kompleksnim brojevima date su u sledećoj tabeli:

x+Iy	kompleksni broj *x+Iy*
ReŠzĆ, ImŠzĆ	realni i imaginarni deo od *z*
AbsŠzĆ	apsolutna vrednost.
ConjugateŠzĆ	konjugivano-kompleksan broj
ArgŠzĆ	argument ϕ takav da je z=đzđe$^{i\phi}$

```
Sqrt[-4]
2 i
(4+3I)/(2-I)
1+2 i
Arg[(4+3I)/(2-I)]
ArcTan[2]
```

3.1.5. Pseudoslučajni brojevi

Postoji veći broj standardnih funkcija kojima se mogu generisati pseudoslučajni brojevi različitih tipova i u različitim opsezima.

RandomŠ Ć	pseudoslučajni broj između 0 i 1
RandomŠReal, x_{max}Ć	pseudoslučajni realni broj između 0 i x_{max}
RandomŠReal, šx_{min}, x_{max}ćĆ	pseudoslučajni realni broj između x_{min} i x_{max}
RandomŠReal, šx_{min}, x_{max}ć,PrecisionĆ	pseudoslučajni broj između x_{min} i x_{max} sa tačnošću *Precision*
RandomŠComplexĆ	pseudoslučajni komleksan broj u jediničnom krugu
RandomŠComplex, šz_{min}, z_{max}ćĆ	pseudoslučajni kompleksni broj u pravougaoniku koji je definisan pomoću z_{min} i z_{max}
RandomŠInteger, ši_{min}, i_{max}ćĆ	pseudoslučajni ceo broj između i_{min} i i_{max}, inkluzivno

Vrednost sledećeg izraza je pseudoslučajni broj između 0 i 1:

```
Random[]
0.8511132434464095
```

Pseudoslučajni veo broj iz intervala Š-500,100Ć generiše se na sledeći način:

```
Random[Integer, {-500, 100}]
-433
```

Pseudoslučajni realni broj iz intervala Š1.1,22Ć sa 50 decimala preciznosti:

```
Random[Real, {1.1, 22}, 50]
13.397676187040943496217274471227481457828872883412
```

Pseudoslučajni kompleksni broj u pravougaoniku sa uglovima *Đ10-10I* i *10+10I*.

```
Random[Complex, {-10 - 10 ⅈ, 10 + 10 ⅈ}]
1.5448419530971371 - 7.886986380193752 ⅈ
```

SeedRandomŠnĆ	resetovanje generatora pseudoslučajnih brojeva, koristeći *n* kao generator
SeedRandomŠĆ	resetovanje generatora pseudoslučajnih brojeva, koristeći vreme kao generator

Posle naredbe *SeedRandom* dobija se ista sekvenca pseudoslučajnih brojeva:

```
Table[Random[], {4}]
{0.7865985779841621, 0.8483394783354097, 0.6128273120449554, 0.39803759651651716}
SeedRandom[5]
Table[Random[], {4}]
{0.7865985779841621, 0.8483394783354097, 0.6128273120449554, 0.39803759651651716}
```

3.1.6. Celobrojne funkcije iz teorije brojeva

Postoji veliki broj standardnih funkcija za određivanje maksimuma i miminuma, ostatka pri deljenju, celobrojnog dela količnika, ispitivanja da li je broj prost, i mnoge druge. Mogu se koristiti različiti brojni sistemi.

MaxŠx_1,x_2,...Ć **MaxŠšx_1,x_2,...ćĆ**	maksimum za x_1, x_2,...
MinŠx_1,x_2,...Ć **MinŠšx_1,x_2,...ćĆ**	minimum za x_1, x_2,...
ModŠk,nĆ	ostatak pri deljenju *k* sa *n*
QuotientŠm,nĆ	celobrojni deo od *m*/*n*
GCDŠn_1, n_2,...Ć	najveći zajednički delilac (NZD) za n_1, n_2,...
LCMŠn_1, n_2,...Ć	najveći zajednički sadržalac (NZS) za n_1, n_2,...
IntegerDigitsŠn,bĆ	cifre broja *n* u bazi *b*
FaktorIntegerŠnĆ	lista prostih faktoraza *n* i hjihovih eksponenata
DividorsŠnĆ	lista celih brojeva koji dele *n*
PrimeŠkĆ	*k*-ti prost broj
PrimePiŠxĆ	broj *Pi*(*x*) - kardinalni broj prostih brojeva manjih od *x*

PrimeQŠnĆ	*True* ako je *n* prost, inače *False*

```
Mod[17, 3]
2
Quotient[17, 3]
5
FactorInteger[24]
{{2, 3}, {3, 1}}
FactorInteger[11111111111111111111111111111111111111111]
{{83, 1}, {1231, 1}, {538987, 1}, {201763709900322803748657942361, 1}}
PrimePi[10^9]
50847534
```

3.1.7. Kombinatorne funkcije

Osnovne kombinatorne funkcije navedene su u sledećoj tabeli.

n!	Faktorijel
n!!	Dupli faktorijel
BinomialŠn,mĆ	$\binom{n}{m}$
MultinomialŠ$\mathbf{n_1}$,$\mathbf{n_2}$,...Ć	$\frac{(n_1+n_2+...)!}{n_1!\,n_2!...}$

```
Binomial[n, 2]
```

$\frac{1}{2}(-1+n)\,n$

```
Multinomial[6, 5]
462
```

Da li ste pokušali da izražunate 1000! u *PASCAL*u ili *C*? Ko je pokušao, shvatiće jednostavnost sledećeg izraza:

```
1000!
```

```
4023872600770937735437024339230039857193748642107146325437999104299385123986290205920442084869694\
048004799886101971960586316668729948085589013238296699445909974245040870737599188236277271887325\
197795059509952761208749754624970436014182780946464962910563938874378864873371191810458257836478\
499770124766328898359557354325131853239584630755574091142624174743493475534286465766116677973966\
688202912073791438537195882498081268678383745597317461360853795345242215865932019280908782973084\
313928444032812315586110369768013573042161687476096758713483120254785893207671691324484262361314\
125087802080002616831510273418279777047846358681701643650241536913982812648102130927612448963599\
287051149649754199093422215668325720808213331861168115536158365469840467089756029009505376164758\
477284218896796462449451607653534081989013854424879849599533191017233555566021394503997362807501\
378376153071277619268490343526252000158885351473316117021039681759215109077880193931781141945452\
572238655414610628921879602238389714760885062768629671466746975629112340824392081601537808898939\
645182632436716167621791689097799119037540312746222899880051954444142820121873617459926429565817\
466283029555702990243241531816172104658320367869061172601587835207515162842255402651704833042261\
439742869330616908979684825901254583271682264580665267699586526822728070757813918581788896522081\
643483448259932660433676601769996128318607883861502794659551311565520360939881806121385586003014\
356945272242063446317974605946825731037900840244324384656572450144028218852524709351906209290231\
364932734975655139587205596542287497740114133469627154228458623773875382304838656889764619273838\
149001407673104466402598994902222217659043399018860185665264850617997023561938970178600408118897\
299183110211712298459016419210688843871218556461249607987229085192968193723886426148396573822911\
231250241866493531439701374285319266498753372189406942814341185201580141233448280150513996942901\
534830776445690990731524332782882698646027898643211390835062170950025973898635542771967428222487\
575867657523442202075736305694988250879689281627538488633969099598262809561214509948717012445164\
612603790293091208890869420285106401821543994571568059418727489980942547421735824010636774045957\
417851608292301353580818400969963725242305608559037006242712434169090041536901059339838357779394\
109700277534720000000000000000000000000000000000000000000000000000000000000000000000000000000000\
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000\
00000000000000000000000000000000000000000000000000000000000000000000000
```

Može se izračunati i 10000! To ovde nije učinjeno zbog uštede prostora.

3.1.8. Transcedentne funkcije

Pored transcendentnih funkcija poznatih u drugim programskim jezicima prepoznaćete i neke nove u sledećoj tabeli:

ExpŠzĆ	e^{z}.
LogŠzĆ	*log*Š*z*Ć.
LogŠb,zĆ	$log_{b}z$.
SinŠzĆ, CosŠzĆ, TanŠzĆ, CscŠzĆ, SecŠzĆ, CotŠzĆ	trigonometrijske funkcije sa uglovima u radijanima
ArcSinŠzĆ, ArcCosŠzĆ, ArcTanŠzĆ, ArcCscŠzĆ,	inverzne trigonometrijske funkcije (rezultat u radijanima)

ArcSecŠzĆ, ArcCotŠzĆ	
SinhŠzĆ, CoshŠzĆ, TanhŠzĆ, SechŠzĆ,CothŠzĆ,CschŠzĆ	hiperboličke funkcije
ArcSinhŠzĆ, ArcCoshŠzĆ, ArcTanhŠzĆ, ArcSechŠzĆ, ArcCothŠzĆ, ArcCschŠzĆ	inverzne hiperboličke funkcije

Sve funkcije iz tabele se mogu koristiti sa proizvoljnom tačnošću.

```
Log[2, 1024]
10
N[Log[2], 40]
0.6931471805599453094172321214581765680755
N[Log[2 + 8 I]]
2.1097538525880535 + 1.3258176636680326 ⅈ
Sin[Pi / 2]
1
N[Sin[30 Degree]]
0.5
N[Pi, 50]
3.1415926535897932384626433832795028841971693993751
```

3.2. FUNKCIJE ZA RAD SA STRINGOVIMA

Funkcije navedene u sledećoj tabeli imaju analoge u proceduralnim programskim jezicima.

"s_1" <> " s_2" <> … **StringJoinŠ"s_1", " s_2", … Ć** **StringJoinŠ"s_1", " s_2", … Ć**	string dobijen konkatenacijom stringova "s_i"
StringLengthŠ"*string*"Ć	broj karaktera u stringu "*string*"
StringReverseŠ"*string*"Ć	obrnut redosled karaktera u "*string*"
StringTakeŠ"string", nĆ	string koji sadrži prvih *n* karaktera u "*string*"
StringTakeŠ"*string*", -*n*Ć	string koji sadrži poslednjih *n* karaktera u "*string*"
StringTakeŠ"*string*", š*n*ćĆ	*n*-ti karakter u "*string*"
StringTakeŠ"*string*", š*m*, *n*ćĆ	karakteri od pozicija *m* do *n* u "*string*"
StringDropŠ"*string*", *n*Ć	izostavlja prvih *n* karaktera iz "*string*"
StringDropŠ"*string*", -*n*Ć	izostavlja poslednjih *n* karaktera iz "*string*"
StringDropŠ"*string*", š*n*ćĆ	izostavlja *n*-ti karakter iz "*string*"

StringDropŠ"*string*", š*m*, *n*ćĆ	izostavlja karaktere od pozicije *m* do pozicije *n* iz "*string*"
StringInsertŠ"*string*", "*snew*", *n*Ć	string "*snew*" se insertuje u poziciji *n* stringa "*string*"
StringInsertŠ"*string*", "*snew*", -*n*Ć	string *snew* se insertuje u poziciji *n* s kraja stringa "*string*"
StringInsertŠ"*string*", "*snew*", šn_1,n_2,…ćĆ	insertuje kopiju "*snew*" u svakoj od pozicija n_i
StringPositionŠ"*string*", "*sub*"Ć	lista koja sadrži početnu i startnu poziciju u kojima se "*sub*" pojavljuje u "*string*"
StringPositionŠ"*string*", "*sub*", *k*Ć	ukjlučuje samo prvih *k* pozicija stringa "*sub*"
StringPositionŠ"*string*", š"sub_1", "sub_2",… ćĆ	pozicije stringova "sub_i"

Primer.

```
"You can join one string "<>"with another."
You can join one string with another.
StringJoin["You can join one string ", "with another."]
You can join one string with another.
StringPosition["Pojavljivanja znaka t u recenici.", "v"]
{{5, 5}, {9, 9}}
StringInsert["Ovo je string.", "novi ", 8]
Ovo je novi string.
StringReplacePart["qrstuvwxyz", "AA", {3, 6}]
qrAAwxyz
```

3.3. FUNKCIJE ZA RAD SA LISTAMA

Svaka kolekcija izraza naziva se lista. Elementi liste se pišu između zagrada š i ć. Sve funkcije za rad sa listama mogu se svrstati u dve grupe: funkcije koje izdvajaju delove liste kao i funkcije koje se koriste za izgradnju novih listi iz postojećih.

3.3.1. Izdvajanje delova liste

Kao i u *LISP*u, svaka lista poseduje glavu i rep (ostatak).

FirstŠlistaĆ	prvi element liste
RestŠlistĆ	ostatak liste

Takođe, postoje standardne funkcije za pristup određenim elementima liste:

LastŠlistaĆ	poslednji element liste
PartŠlista,nĆ ili **listaŠŠnĆĆ**	*n*-ti element liste
PartŠlista,-nĆ ili **listaŠŠ-nĆĆ**	*n*-ti element liste, gledano od kraja
PartŠlista, šn_1,n_2,...ćĆ ili **listaŠŠn_1,n_2,...ĆĆ**	lista elemenata koji se nalaze u pozicijama n_1, n_2, ... u polaznoj listi

```
t = {10, 20, 30, 40, 50}
{10, 20, 30, 40, 50}
First[t]
10
Last[t]
50
Part[t, {3, 4}]
{30, 40}
```

Elementima matrice se moče pristupiti na više načina:

mŠŠi,jĆĆ	(i,j)-ti element matrice *m*
mŠŠši_1,... ,i_rć,šj_1,...,j_sćĆĆ	$r\times s$ submatrica matrice *m* sa elementima koji se nalaze u preseku vrsta $i_1,...,i_r$ i kolona $j_1,...,j_s$

Postoje i funkcije za izdvajanje ili uklanjanje određenih delova u listi:

TakeŠlista,nĆ	izdvajanje prvih *n* elemenata u listi
TakeŠlista,-nĆ	izdvajanje poslednjih *n* elemenata iz liste
TakeŠlist,šm,nćĆ	izdvajanje elemenata *m* do *n* (uključujući *m* i *n*)
DropŠlist,nĆ	lista bez svojih prvih *n* elemenata
DropŠlist,-nĆ	lista bez svojih poslednjih *n* elemenata
DropŠlist,šm,nćĆ	lista sa izbačenim elementima od pozicije *m* do *n*

3.3.2. Konstrukcija novih listi iz postojećih

Listama se mogu dodavati elementi na proizvoljnom mestu od početka ili od kraja liste. Takođe, listama se mogu uklanjati određeni elementi.

PrependŠlista,elemĆ	dodavanje *elem* na početak liste *lista*
AppendŠlista,elemĆ	dodavanje *elem* na kraj liste *lista*

InsertŠlista,elem,iĆ	umetanje elem na *i*-to mesto liste
InsertŠlista,elem,-iĆ	umetanje elem na *i*-to mesto od kraja liste
InsertŠlista,elem,ši,j,...ć Ć	umetanje na š*i*,*j*,...ć-to mesto u listi
DeleteŠlista,iĆ	uklanjanje elementa sa i-tog mesta u listi
DeleteŠlista,-iĆ	uklanjanje elementa sa *i*-tog mesta gledano od kraja liste
DeleteŠlista,ši,j,...ćĆ	uklanjanje elementa sa š*i*,*j*,...ć-te pozicije u listi

```
l = {a, b, c, d}
{{x, x^2, x^3, aba}, b, c, d}
Prepend[l, x]
{x, {x, x^2, x^3, aba}, b, c, d}
a
{x, x^2, x^3, aba}
Take[l, 3]
{{x, x^2, x^3, aba}, b, c}
Take[l, -3]
{b, c, d}
Take[l, {2, 5}]
Take::take : Cannot take positions 2 through 5 in {{x, x^2, x^3, aba}, b, c, d}.
Take[{{x, x^2, x^3, aba}, b, c, d}, {2, 5}]
Take[l, {2, 4}]
{b, c, d}
Take[l, {0, 4}]
{}
Take[l, {1, 4}]
{{x, x^2, x^3, aba}, b, c, d}
Rest[t]
{20, 30, 40, 50}
Drop[l, 3]
{d}
Drop[l, {3, 3}]
{{x, x^2, x^3, aba}, b, d}
t = {{a, b, c}, {d, e, f}}
{{{x, x^2, x^3, aba}, b, c}, {d, e, f}}
a =.
t = {{a, b, c}, {d, e, f}}
{{a, b, c}, {d, e, f}}
t[[1]]
```

```
{a, b, c}
t[[1, 2]]
b
t[1, 2]
{{a, b, c}, {d, e, f}}[1, 2]
t[[{2, 2, 1}]]
{{d, e, f}, {d, e, f}, {a, b, c}}
```

Element *elem*, koji predstavlja argument funkcija u prethodnoj tabeli, može biti nova lista:

```
Append[{a, b, c}, {a, c, d}]
{a, b, c, {a, c, d}}
```

ReplacePartŠlista, elem, iĆ	Zameniti element u poziciji *i* sa *elem*
ReplacePartŠlista, elem, -iĆ	Zameniti *i*-ti element sa kraja liste sa *elem*
ReplacePartŠlista, elem, ši, j, ...ćĆ	Zameniti *lista*ŠŠ*i, j, ...*ĆĆ sa *elem*
ReplacePartŠlista, elem, ši$_1$, j$_1$,...ć, ši$_2$,j$_2$,...ć, ...Ć	Zameniti sve delove *lista*ŠŠi_k,,j_k,,...ĆĆ vrednošću *elem*.

```
ReplacePart[{a, b, c, d}, x, 3]
{a, b, x, d}
ReplacePart[{a, b, c, d}, x, {{1}, {4}}]
{x, b, c, x}
ReplacePart[{{1, 0, 0}, {0, 1, 0}, {0, 0, 1}}, X, {2, 2}]
{{1, 0, 0}, {0, X, 0}, {0, 0, 1}}
```

SortŠlistaĆ	Sortiranje elemenata liste
ReverseŠlistaĆ	Invertovanje liste
RotateLeftŠlista, nĆ	Rotiranje ulevo za *n* mesta
RotateRightŠlista, nĆ	Rotiranje udesno za *n* mesta
RotateLeftŠlistaĆ	Rotacija za jedno mesto ulevo
RotateRightŠlistaĆ	Rotacija za jedno mesto udesno

Liste se mogu posmatrati i kao skupovi:

UnionŠlistaĆ	Sortiranje elemenata liste uklanjajući duplikate

3.3.3. Kombinatorne operacije

Funkcijom *Permutrations*Š*l*Ć generiše se lista koja sadrži sve permutacije elemenata koji su sadržani u *l*.

PermutationsŠša, b, ccĆ	Sva moguća uređenja liste
OrderedQŠlistaĆ	*True* ako su elementi liste uređeni

```
Permutations[{a, b, c}]
{{a, b, c}, {a, c, b}, {b, a, c}, {b, c, a}, {c, a, b}, {c, b, a}}
```

3.3.4. Matematičke operacije sa listama

Aritmetičke funkcije mogu imati za svoj argument listu. U tom slučaju se one primenjuju na svaki element te liste.

ApplyŠPlus, listaĆ ili **Plus ŽŽ lista**	Sabrati sve elemente liste
ApplyŠTimes, listaĆ ili **Times ŽŽ lista**	Množenje svih elemenata u listi

```
Apply[Plus, {a, b, c, d}]
a + b + c + d
Apply[Plus, {{1, 2, 3}, {3, 2, 1}}]
{4, 4, 4}
```

4. UPRAVLJAČKE STRUKTURE

Prilikom rešavanja nekih problema, potrebno je ponavljanje niza naredbi, ili prelazak na određeno mesto u programu u zavisnosti od izvesnih parametara. Drugim rečima, potrebna je kontrola nad tokom izvršavanja programa. Osnovni mehanizmi kontrole toka izvršavanja programa, koje imaju skoro svi programski jezici, jesu petlje, grananja i skokovi. Pored navedenih, *MATHEMATICA* ima još neke upravljačke strukture koje olakšavaju pisanje raznovrsnih, jednostavnih i efikasnih programa.

4.1. SEKVENCA NAREDBI I BLOK

Kao što je rečeno, u sekvenci naredbi (proceduri) se naredbe međusobno odvajaju znakom ž;' i izvršavaju jedna za drugom u redosledu u kome su napisane. Rezultat takve sekvence naredbi je rezultat izvršenja poslednje naredbe u sekvenci. Ukoliko iza poslednje naredbe stoji znak ; rezultat sekvence naredbi je *NULL*, i ne prikazuje se.

$expr_1$, $expr_2$,...	izvršiti naznačene operacije, i vratiti rezultat poslednje
$expr_1$, $expr_2$,...;	izvršiti naznačene operacije, ali bez rezultata

x = 4; y = 6; z = y + 6

```
12
```

Znak ; na kraju ulazne linije signalizira da se rezultat ne prikazuje:

x = 67 - 5 ;

Međutim, i dalje se može koristiti % da se dobije rezultat koji nije prikazan:

%

```
62
```

Razlika u odnosu na proceduralne programske jezike se sastoji u tome da sekvenca naredbi ne mora da se piše između zagrada kakve su na primer *begin* i end u *PASCAL*u ili zagrade š i ć u jeziku *C*. To je posledica činjenice da se svi izrazi u *MATHEMATICA* tretiraju na jedinstven način, u obliku *Head*Šarg_1,arg_2,...Ć. Takvu strukturu imaju i naredbe za grananje u programu, naredbe za definisanje ciklusa kao i deklaracija potprograma.

4.2. RELACIONI I LOGIČKI OPERATORI

Relacioni izrazi se koriste za upoređivanje vrednosti dva objekta. Kao rezultat primene ovih operatota, dobija se logička vrednost *True* ako je upoređivanje tačno, odnosno *False*, ako je upoređivanje netačno. Relacijski operatori su identični kao u jeziku *C*:

x==y ili **EqualŠx,yĆ**	jednakost
x=y=z	jednakost više izraza
x!=y	različitost
x!=y!=z	nejednakost više izraza
x>y	veće
x>y>z	strogo opadajući izrazi
x<y	manje
x<y<z	strogo rastući izrazi
x>=y	veće ili jednako
x<=y	manje ili jednako

x = 4

```
4
```

x == 6

```
False
```

```
x == 4
True
```

Ovakva testiranja uključuju samo brojeve, i uvek daju rezultat tipa *True* ili *False*. Međutim, mogu se testirati i simbolički izrazi:

```
var == 2
var == 2
```

U ovom slučaju, rezultat nije definisan sve dok simbol *var* ne dobije određenu vrednost.

Postoje i komplikovaniji relacioni izrazi.

TrueQŠexprĆ	*True* ako je *expr*=*True* inače *False*
lhs===rhs ili **SameQŠlhs, rhsĆ**	*True* ako su *lhs* i *rhs* identični, inače *False*
lhs=!=rhs ili **UnsameQŠlhs, rhsĆ**	*True* ako *lhs* i *rhs* nisu identični
MatchQŠexpr, formĆ	*True* ako se šablon *form* slaže sa *expr*

Relacioni operator === se koristi za ispitivanje strukture izraza, za razliku od relacionog operatora == koji se koristi za ispitivanje matematičke jednakosti.

```
x == y
x == y
```

(*Izraz ostaje kao simbolička jednačina*)

```
TrueQ[x == y]
False
```

(* TrueQ podrazumeva da je svaki izraz False, osim ako se manifestuje kao True*)

```
x == y
x == y
TrueQ[x == y]
False
x === y
False
```

(*Za razliku od ==, funkcija === uvek vraća True ili False*)

```
Head[a + b + c]
Plus
Head[a + b + c] === Times
False
```

(*Funkcija === ispituje strukturu izraza *)

```
Head[a + b + c] == Times
Plus == Times
```

(*Funkcija == daje manje koristan rezultat*)

Često se koriste različite kombinacije kondicionala pomoću standardnih logičkih operatora. Oni su poznati iz jezika *C*:

$expr_1$ && $expr_2$ &&...	evaluacija izraza $expr_i$, dok jedan od njih ne bude *False*, što je rezultat; ako su svi izrazi $expr_i$ jednaki *True*, i rezultat je *True*
$expr_1$ đđ $expr_2$ đđ ...	evaluacija izraza $expr_i$, sve dok jedan od njih ne bude *True*, i ta vrednost je rezultat. Inače, rezultat je *False*
!p	negacija izraza *p*
XorŠp,q,...Ć	ekskluzivna disjunkcija

```
t=.
t[x_] := (x ≠ 0 && 1/x < 3)
t[2]
True
```

4.3. USLOVNI IZRAZI

MATHEMATICA obezbeđuje različite načine za definisanje uslovnih izraza, koji specificiraju parcijalne izraze koji se evaluiraju ako su određeni uslovi ispunjeni. U *MATHEMATICA* se uslovni izrazi *If-Then* i *If-Then-Else* predstavljaju pozive funkcija.

IfŠtest, thengranaĆ	Izvršavaju se izrazi u *thengrana* ako test ima vrednost *True*
IfŠtest,thengrana, elsegranaĆ	izvršava se *thengrana* ako test ima vrednost *True*, inače se izvršava *elsegrana*

```
x = 7
7
If[x < 10, b = 0, b = 20]
0
b
0
If[7, a = 0, a = 20]
If[7, a = 0, a = 20]
a
```

```
a

If[7 > 8, x, y]
y
y
```

Takođe, jednu vrstu uslovnih izraza čine izrazi sledećeg tipa, koji omogućavaju da se definicija koristi samo ako je određen uslov ispunjen:

lhs:=rhs/; test	koristi definiciju samo ako *test* evaluira u *True*

Pojedinačne definicije funkcija koje u svojoj desnoj strani sadrže naredbe uslovnog prelaska mogu se zameniti sa nekoliko definicija, od kojih je svaka kontolisana odgovarajućim uslovom /*;test*.

Na primer, u sledećem izrazu je definisana funkcija **fŠxĆ=Š** $\begin{cases} 1, & x>0 \\ -1, & x\leq 0 \end{cases}$

fŠx_Ć:=IfŠx>0, 1, -1Ć

Ekvivalentna sa funkcijom f je sledeća funkcija g:
gŠx_Ć:=1/; x>0 (* Pozitivni deo funkcije f *)
gŠx_Ć:=-1/; x<=0 (*Negativni deo funkcije f *)
?g
Global'g
gŠx_Ć:=1/; x>0
gŠx_Ć:=-1/; x<=0

Važan detalj simboličkog izračunavanja u *MATHEMATICA* je da uslovni izrazi mogu da produkuju rezultate koji su različiti i od *True* i od *False*. U slučaju da test u *If* izrazima produkuje rezultat koji nije ni *True*, ni *False*, tada ovi izrazi ostaju ne evaluirani.

```
If[x == y, a, b]
If[x == y, a, b]
```

Može se dodati četvrti argument naredbi *If*, koji se koristi ako *test* produkuje rezultat koji nije ni *True* ni *False*.

*If*Š*test, thengrana, elsegrana, default*Ć

Ovakav izraz je jedna forma naredbe If koja uključuje izraz default, koji se koristi ako test nije ni *True* ni *False*.

```
If[x == y, a, b, c]
c
```

Funkcija *If* dozvoljava izbor između dve alternative. Međutim, često je potrebno da se testira veći broj uslova. To se može učiniti pomoću većeg broja umetnutih *If* funkcija. Mnogo je efikasnije da se to uradi funkcijama *Which* i *Switch*.

WhichŠtest$_1$, value$_1$, test$_2$, value$_2$, ...Ć	evaluira se redom *test*$_1$,*test*$_2$,... pri čemu je rezultat vrednost asocirana sa prvim *test*$_i$ koji je *True*
SwitchŠexpr,form$_1$, value$_1$, form$_2$, value$_2$, ...Ć	upoređuje se *expr* sa svakim od izraza *form*$_1$,..., a rezultat je vrednost izraza *value*$_i$, koji je asociran sa prvim izrazom *form*$_i$ koji se slaže sa *expr*
SwitchŠexpr,form$_1$,value$_1$, form$_2$, value$_2$,..., defĆ	koristi se *def* kao vrednost koja se vraća ako se nijedan od izraza *form*$_i$ ne slaže sa *expr*

U poslednjoj tabeli svaki od izraza *value*$_1$,*value*$_2$,... predstavlja jedan izraz ili sekvencu izraza koji su razdvojeni znakom *;*.
Na primer, izrazom:

```
h[x_] := Which[x < 0, x^2, x > 5, x^3, True, 0]
```

definisana je funkcija *h* sa tri grane. Treća grana je uvek ispunjena ako nisu prethodne dve.

```
h[-5]
25
```
(*Koristi se prvi slučaj u Which*)

```
h[2]
0
```
(*Koristi se treći slučaj*)

```
residue[x_] := Switch[Mod[x, 3], 0, a, 1, b, 2, c]
```
(*Rezultat zavisi od ostatka pri deljenju argumenta sa 3*)

```
residue[7]
b
```
(*ModŠ7,3Ć=1, pa se koristi drugi slučaj*)

```
Switch[17, 0, a, 1, b, _, g]
g
```
(*17 se ne slaže sa 1, ali se slaže sa _*)

```
res[x_] := Switch[Mod[x, 3], 0, Print["Ostatak je 0 "]; a, 1, Print["Ostatak je 1 "]; b,
  2, Print["Ostatak je 2 "]; c]
res[17]
Ostatak je 2
c
```

4.4. CIKLUSI

U *MATHEMATICA* postoje upravljačke strukture kojima se definiše višestruko ponavljanje određene sekvence naredbi. Kao i u ostalim savremenim programskim jezicima postoje naredbe za definisanje petlji sa unapred zadatim brojem izvršenja tela petlje. To su tzv. brojačke petlje, kod kojih postoji brojač kojim se upravlja izvršenjem petlje. Takođe, postoje *pre-test* i *post-test* petlje kojima se definišu iterativni programski ciklusi.

4.4.1. Do ciklusi

Do ciklusi se koriste za definisanje brojačkih programskih ciklusa. Različite varijante ovih ciklusa prikazane su u sledećoj tabeli.

DoŠexpr, ši, imaxćĆ	evaluirati *expr* za espektivno, pri čemu *i* uzima vrednosti iz skupa š1*, ..., imaxć*
DoŠexpr,ši, imin, imax, dićĆ	evaluirati expr sa vrednostima *i* od *imin* do *imax* sa korakom *di*
DoŠexpr, šnćĆ	evaluacija izraza *expr n* puta

```
Do[Print[i^2], {i, 4}]
1
4
9
16
```

(* Evaluira *PrintŠiČ2Ć* sa vrednostima za i od 1 do 4 *)

```
t = x; Do[t = 1/(1 + k t), {k, 2, 6, 2}]; t
```

$$\frac{1}{1+\frac{6}{1+\frac{4}{1+2x}}}$$

Može se postaviti više ugnježdenih ciklusa pomoću sekvence iteracija.

```
Do[Print[{i, j}], {i, 4}, {j, i - 1}]
{2, 1}
{3, 1}
{3, 2}
{4, 1}
{4, 2}
{4, 3}
```

(*Dvostruki ciklus za i=1,4,   j=1, i-1*)

Ponekad je potrebno da se određena operacija ponovi određeni broj puta, bez promene vrednosti neke iterativne promenljive.

```
t = x; Do[t = 1/(1 + t), {3}]; t
```

$$\frac{1}{1+\frac{1}{1+\frac{1}{1+x}}}$$

```
t = 67; Do[Print[t]; t = Floor[t / 12], {3}]
67
5
0
```

Suma prvih 1000 prirodnih brojeva se može izračunati na sledeći način:

```
sum = 0;
Do[sum += i, {i, 1000}];
sum
500500
```

4.4.2. While i For ciklusi

Pored *Do* petlje, *MATHEMATICA* ima petlje *For* i *While*, koje podsećaju na istoimene petlje u programskim jezicima *PASCAL* i *C*. Za razliku od *Do* petlje koja se uvek izvršava zadati broj puta, broj izvršavanja petlji *For* i *While* je određen nekim izlaznim kriterijumom. Tačnije, te petlje se izvršavaju samo dok određeni uslov ima vrednost *True*.

WhileŠtest,bodyĆ	izvršava *body* sve dok je *test=True*

While petlja je pogodna kada se ne zna koliko je puta potrebno ponoviti određene operacije.

```
n = 17; While[n ≠ 0, Print[n]; n = Floor[n / 2]]
17
8
4
2
1
t = {MATHEMATICA, Pi, 15}; While[t ≠ {}, Print[First[t]]; t = Rest[t]]
MATHEMATICA
π
15
```

U nekim slučajevima je potrebna veća kontrola nad izvršavanjem programa. U tim slučajevima se koristi fleksibilnija, ali i komplikovanija *For* petlja.

ForŠstart,test, incr, bodyĆ	izvršava *start*, zatim respektivno evaluira *body* i *incr* sve dok *test* ne postane *False*

U slučaju izraza *For*Š*start, test, incr*Ć izvršava se ciklus sa praznim telom. Sve do evaluacije izraza *Return*Š*expr*Ć ili *Throw*Š*expr*Ć, finalna vrednost *For* ciklusa je *Null*.

Primer.

```
f[x]
x^2
For[tot = 0; i = 0, i < 3, i++, tot += f[i]]
tot
5
```

Napomenimo da su uloge znakova , i ; obrnute u odnosu na programski jezik *C*.

```
For[i = 1, i < 4, i++, Print[i]]
1
2
3
For[i = 1; t = x, i^2 < 10, i++, t = t^2 + i; Print[t]]
1 + x^2
2 + (1 + x^2)^2
3 + (2 + (1 + x^2)^2)^2
```

U *While* i *For* ciklusima se uslovni izraz *test* kojim se prekida ciklus, evaluira pre evaluacije tela ciklusa. Telo ciklusa se izvršava sve dok je vrednost izraza *test* jednaka *True*.

4.4.3. Kontrola petlji

Ponekad je potrebno preskočiti neki korak u petlji, ili čak izaći iz petlje pre nego što se ona završi. *MATHEMATICA* ima dve funkcije koje služe takve intervencije. Naredbom *Break*ŠĆ u telu ciklusa, koji je označen sa *body*, završava se izvršenje *For* ciklusa. Naredba *Continue*ŠĆ završava evaluaciju tela *body*, i nastavlja ciklus evaluacijom izraza *incr*.

BreakŠ Ć	izlazak iz petlje
ContinueŠ Ć	prelazak na sledeći korak u petlji

```
Do[Print[i]; If[i == 3, Break[]], {i, 10}]
1
2
3
For[i = 1, i < 5, i++, If[i == 3, Continue[]]; Print[i]]
```

```
1
2
4
```

4.5. POSEBNE VRSTE CIKLUSA

Pored opisanih tipova ciklusa koje poseduje većina savremenih programskih jezika, *MATHEMATICA* poseduje specifične funkcije za realizaciju brojačkih i iterativnih programskih ciklusa.

NestŠf, expr, nĆ	primenjuje *n* puta funkciju *f* na *expr*
FixedPointŠf, exprĆ	počinje sa *expr* i primenjuje funkciju *f* sve dok se rezultat ne menja više
FixedPointŠf, expr, SameTest⟶comp Ć	zaustavlja se kada je funkcija *comp* primenjena na dva susedna rezultata daje *True*

Ciklusi definisani funkcijom *Nest* su brojački, dok se funkcijom *FixedPoint* definišu iterativni programski ciklusi.

```
Nest[f, x, 3]
f[f[f[x]]]
Nest[Function[t, 1/(1+t)], x, 3]
```

$$\frac{1}{1+\frac{1}{1+\frac{1}{1+x}}}$$

```
FixedPoint[Function[z, Print[z]; Floor[z/2]], 67]
67
33
16
8
4
2
1
0
0
```

4.6. BEZUSLOVNI SKOK

Pomoću funkcija *Label* i *Goto* se može izvršiti bezuslovni skok na određeno mesto u proceduri. Kada se funkcija *Label* navede u proceduri, funkcijom *Goto* se izvršavanje programa nastavlja počevši od elementa označenog sa *Label*.

LabelŠimeĆ	oznaka za funkciju *Goto*
GotoŠimeĆ	prelazi na funkciju *Label*Š*ime*Ć u trenutno aktivnoj proceduri

Funkcija *Goto* se može koristiti samo kada se u istoj proceduri nalazi i odgovarajuća funkcija *Label*. U opštem slučaju treba izbegavati upotrebu *Goto*, jer ona narušava strukturu programa, pa su praćenje izvršavanja i kasnije modifikacije time otežani.

```
(t = 5; Label[start]; Print[t]; t -= 2;    If[t > 0, Goto[start]])
5
3
1
```

4.7. IZLAZAK IZ FUNKCIJE SA VRAĆANJEM VREDNOSTI

Funkcija *Return*Š*izraz*Ć vraća izraz kao rezultat tekuće funkcije, i zatim izlazi iz nje. Upotrebom te funkcije se pojednostavljuje pisanje korisničkih funkcija, jer rezultat funkcije ne mora uvek biti poslednji izraz.

ReturnŠizraz Ć	vraća vrednost izraza *izraz* kao rezultat tekuće funkcije

```
f[k_] :=
  Module[{z},
     z = 4 - k;
     If[z < 0, Return[minus]];
     Table[a, {z}]
  ]
General::spell1 :
 Possible spelling error: new symbol name "minus" is similar to existing symbol "Minus".
f[1]
{a, a, a}
f[7]
minus
```

Funkcija *Return* se može koristiti i za izlazak iz petlje. Ako *MATHEMATICA* naiđe na izraz oblika *Return*Š*izraz*Ć u toku izvršavanja petlje, ta petlja će se prekinuti, a njen rezultat će biti izraz.

```
Do[Print[i];
   If[i == 2, Return[izlaz]], {i, 6}
]
1
2
```

4.8. NE-LOKALNI POVRATAK I OBRADA GREŠKE

Nekada se javlja potreba da se u jednom trenutku izađe iz više petlji ili funkcija ođednom, te da se određeni izraz vrati kao konačni rezultat neke korisničke funkcije. Izlaz iz većeg broja ugnježdenih ciklusa ili više funkcija naziva se ne-lokalni povratak. *MATHEMATICA* podržava takvu vrstu povratka pomoću funkcija *Throw* i *Catch*.

CatchŠprocĆ	prima vrednost funkcije *Throw* iz procedure *proc*, ako takva postoji
ThrowŠizraz Ć	šalje izraz nadređenoj funkciji *Catch*

```
h[x_] := If[x > 10, Throw[greska], x!]
Catch[Sqrt[1 + h[12]]]
greska
Catch[4 + h[4]]
28
Clear[h]
h[x_] := If[x > 10, greska, x!]
Catch[Sqrt[1 + h[12]]]
√1 + greska
```

4.9. PRAĆENJE IZRAČUNAVANJA

Standardni tok u kojem *MATHEMATICA* radi je da izraz koji predstavlja ulaz izračunava i izdaje krajnji rezlutat, ali nekada je potrebno pratiti i sve međukorake koji se dešavaju između. To nam na različite načine omogućavaju sledeće funkcije:

TraceŠexprĆ	generiše listu svih izračunavanja.
TraceŠexpr, formĆ	uključuje samo izračunavanja koja odgovaraju obliku *form*

```
Trace[1 + 1]
```

```
{1+1, 2}
Trace[3+2]
{3+2, 5}
Trace[2^3+4]
{{2^3, 8}, 8+4, 12}
Trace[2^3+4^2+1]
{{2^3, 8}, {4^2, 16}, 8+16+1, 25}
fac[n_] := fac[n-1]; fac[1] = 1
1
Trace[fac[3]]
{fac[3], fac[3-1], {3-1, 2}, fac[2], fac[2-1], {2-1, 1}, fac[1], 1}
Trace[fac[3], fac[n_]]
{fac[3], fac[3-1], fac[2], fac[2-1], fac[1]}
Trace[fac[10], fac[n_ /; n > 5]]
{fac[10], fac[10-1], fac[9], fac[9-1], fac[8], fac[8-1], fac[7], fac[7-1], fac[6]}
fib[n_] := fib[n-1] + fib[n-2]
fib[0] = fib[1] = 1
1
```

(* Izveštaj o argumentima funkcije *fib* koji su korišćeni u evaluaciji *fib*Š5Ć *)

```
Trace[fib[5], fib[n_] → n]
{5, {4, {3, {2, {1}, {0}}, {1}}, {2, {1}, {0}}}, {3, {2, {1}, {0}}, {1}}}
```

(* Prikazati sve pojave *fib* u evaluaciji *fib*Š3Ć *)

```
Trace[fib[3], fib[_]]
{fib[3], {fib[2], {fib[1]}, {fib[0]}}, {fib[1]}}
```

TraceŠexpr, fŠ___ĆĆ	prikazuje sve pozive funkcije *f*
TraceŠexpr, i=_Ć	prikazuje dodeljivanja vrednosti simbolu *i*
TraceŠexpr, _=_Ć	prikazuje sve vrednosti
TraceŠexpr, MessageŠ___ĆĆ	prikazuje sve generisane poruke
TraceŠexpr, fĆ	prikazuje sva izračunavanja koja koriste pravila transformacije vezane za simbol *f*
TraceŠexpr, f/gĆ	prikazuje izračunavanja koja koriste ili *f* ili *g*

```
Trace[fac[6], fac[_]]
{fac[6], fac[6-1], fac[5], fac[5-1], fac[4],
 fac[4-1], fac[3], fac[3-1], fac[2], fac[2-1], fac[1]}
fp[n_] := fp[n-1] /; n > 1
?fp
```

Globalžfp

```
fp[n_] := fp[n - 1] /; n > 1
fp[6]
fp[1]
Trace[fp[6]]
{fp[6], {{6 > 1, True}, RuleCondition[$ConditionHold[$ConditionHold[fp[6 - 1]]], True],
  $ConditionHold[$ConditionHold[fp[6 - 1]]]}, fp[6 - 1], {6 - 1, 5}, fp[5],
 {{5 > 1, True}, RuleCondition[$ConditionHold[$ConditionHold[fp[5 - 1]]], True],
  $ConditionHold[$ConditionHold[fp[5 - 1]]]}, fp[5 - 1], {5 - 1, 4}, fp[4],
 {{4 > 1, True}, RuleCondition[$ConditionHold[$ConditionHold[fp[4 - 1]]], True],
  $ConditionHold[$ConditionHold[fp[4 - 1]]]}, fp[4 - 1], {4 - 1, 3}, fp[3],
 {{3 > 1, True}, RuleCondition[$ConditionHold[$ConditionHold[fp[3 - 1]]], True],
  $ConditionHold[$ConditionHold[fp[3 - 1]]]}, fp[3 - 1], {3 - 1, 2}, fp[2],
 {{2 > 1, True}, RuleCondition[$ConditionHold[$ConditionHold[fp[2 - 1]]], True],
  $ConditionHold[$ConditionHold[fp[2 - 1]]]}, fp[2 - 1], {2 - 1, 1}, fp[1], {{1 > 1, False},
  RuleCondition[$ConditionHold[$ConditionHold[fp[1 - 1]]], False], Fail}, fp[1]}
Trace[fac[10], fac[n_ /; n > 5]]
{fac[10], fac[10 - 1], fac[9], fac[9 - 1], fac[8], fac[8 - 1], fac[7], fac[7 - 1], fac[6]}
Trace[fp[6], fp[_]]
{fp[6], fp[6 - 1], fp[5], fp[5 - 1], fp[4], fp[4 - 1], fp[3], fp[3 - 1], fp[2], fp[2 - 1], fp[1]}
log[x_* y_] := log[x] + log[y]
Trace[log[ab* cd]]
{log[ab cd], log[ab] + log[cd]}
Trace[log[ab* cd], log]
{log[ab cd], log[ab] + log[cd]}
```

Mogu se koristiti komplikovaniji načini za praćenje evaluacije:

TraceŠexpr, lhs->rhsĆ	nalazi sve izraze oblika *lhs* koji se javljaju tokom izračunavanja *expr* i zamenjuje ih sa *rhs*
TraceŠexpr, form, TraceDepth->nĆ	prati izračunavanje *expr*, ignorišući korake koji vode do ugnježdenih listi nivoa većeg od *n*

```
fib[n_] := fib[n - 1] + fib[n - 2]
fib[0] = fib[1] = 1
1
Trace[fib[5], fib[n_] -> n]
{5, {4, {3, {2, {1}, {0}}, {1}}, {2, {1}, {0}}}, {3, {2, {1}, {0}}, {1}}}
Trace[fib[3], fib[_], TraceDepth -> 2]
```

```
{fib[3], {fib[2]}, {fib[1]}}
```

Opcije koje se mogu koristiti u funkciji *Trace* opisane su u sledećoj tabeli:

TraceŠexpr,form, optsĆ	prati izračunavanje izraza *expr* koristeći određene opcije

Opcije se mogu predstaviti sledećom tabelom

TraceForward->True	uključuje poslednji izraz u nizu izračunavanja koji sadrži *form*
TraceForward->All	uključuje sve izraze u nizu izračunavanja koji su iza *form*
TraceBackward->True	uključuje prvi izraz u nizu izračunavanja koji uključuje *form*
TraceBackward->All	uključuje sve izraze u nizu izračunavanja koji prethode *form*
TraceAbove->True	uključuje prvi i poslednji izraz u svim nizovima izvršavanja koji sadrže *form*
TraceAbove->All	uključuje sve izraze u nizu izračunavanja koji sadrže *form*

```
Trace[fac[4], fac[_], TraceForward -> True]
{fac[4], fac[4 - 1], fac[3], fac[3 - 1], fac[2], fac[2 - 1], fac[1], 1}
Trace[fac[4], fac[_], TraceForward -> All]
{fac[4], fac[4 - 1], fac[3], fac[3 - 1], fac[2], fac[2 - 1], fac[1], 1}
Trace[fac[10], TraceAbove -> True]
{fac[10], fac[10 - 1], {10 - 1, 9}, fac[9], fac[9 - 1], {9 - 1, 8}, fac[8],
 fac[8 - 1], {8 - 1, 7}, fac[7], fac[7 - 1], {7 - 1, 6}, fac[6], fac[6 - 1],
 {6 - 1, 5}, fac[5], fac[5 - 1], {5 - 1, 4}, fac[4], fac[4 - 1], {4 - 1, 3},
 fac[3], fac[3 - 1], {3 - 1, 2}, fac[2], fac[2 - 1], {2 - 1, 1}, fac[1], 1}
Trace[fib[5], fib[2], TraceAbove -> True]
{fib[5], {fib[4], {fib[3], {fib[2], 2}, 3}, {fib[2], 2}, 5}, {fib[3], {fib[2], 2}, 3}, 8}
```

4.10. KORIŠĆENJE STEKA IZRAČUNAVANJA

Stek je struktura koja se vrlo često koristi. U *MATHEMATICA* za korišćenje steka koristimo funkciju *Stack*. Ukoliko prekinemo program *MATHEMATICA* usred izračunavanja, možemo koristiti stek da bi pregledali šta je dotad urađeno.

StackŠĆ	daje listu oznaka povezanih sa izračunavanjima koji su u

	toku
StackŠ_Ć	daje listu svih izraza čije je izračunavanje u toku
StackŠform Ć	uključuje samo izraze koji odgovaraju *form*

```
f[g[Print[Stack[_]]]]
{f[g[Print[Stack[_]]]], g[Print[Stack[_]]], Print[Stack[_]]}
f[g[Null]]
f[g[Print[Stack[]]]]
{f, g, Print}
f[g[Null]]
```

StackInhibitŠexprĆ	generiše izraz *expr* ne menjajući stek
StackBeginŠexprĆ	generiše izraz *expr* sa novim stekom
StackCompleteŠexpr Ć	generiše *expr* sa među-izračunavanjima u nizu izračunavanja uključenih u stek

```
f[g[StackInhibit[Print[Stack[]]]]]
{f, g}
f[g[Null]]
f[StackBegin[g[h[StackInhibit[Print[Stack[]]]]]]]
{g, h}
f[g[h[Null]]]
```

4.11. KONTROLA BESKONAČNIH IZRAČUNAVANJA

Generalni princip koji *MATHEMATICA* sledi u izračunavanju izraza je primenjivanje pravila transformacija sve dok se rezultat više ne menja. Izraz oblika x=x+1 se potencijalno može izračunavati beskonačno dugo. Praktično, *MATHEMATICA* prestaje sa rekurzivnim pozivima posle određenog broja koraka određenih vrednošću globalne promenljive *$RecursionLimit*. Na taj način je obezbeđena kontrola nad rekurzinim pozivanjem.

```
x = x + 1
$RecursionLimit::reclim : Recursion depth of 256 exceeded.
255 + Hold[1 + x]
ReleaseHold[%]
$RecursionLimit::reclim : Recursion depth of 256 exceeded.
510 + Hold[1 + x]
```

$RecursionLimit	maksimalna dubina steka izračunavanja
$IterationLimit	maksimalna dužina lanca (niza) izračunavanja

```
{a, b} = {b, a}
$IterationLimit::itlim : Iteration limit of 4096 exceeded.
$IterationLimit::itlim : Iteration limit of 4096 exceeded.
{Hold[b], Hold[a]}
$RecursionLimit = 20
```
(*Sledi cirkularna definicija čije se izračunavanje zaustavlja sa *$IterationLimit* *)
```
20
t = {t}
$RecursionLimit::reclim : Recursion depth of 20 exceeded.
{{{{{{{{{{{{{{{{{{{Hold[{t}]}}}}}}}}}}}}}}}}}}}
```

4.12. PREKIDI I NASILNI IZLASCI

Nasilni prekidi programa i ciklusa se često koriste u programiranju.

InterruptsŠ Ć	prekid izračunavanja (sa povratkom)
AbortŠ Ć	prekid izračunavanja (bez povrataka)
CheckAbort Šexpr, failexprĆ	izračunava izraz *expr* i vraca *failexpr* u slučaju *Abort*-prekida
AbortProtectŠexprĆ	izračunava *expr* maskirajući prekid sve dok se izračunavanje ne završi

Možemo koristiti funkciju *Abort* za nasilni prekid u programu, ali u skoro svim slučajevima treba koristiti funkcije *Return* i *Throw* koje omogućavaju bolju kontrolu.
U sledećim izrazima funkcija *Abort* prekida izračunavanja, tako da se izvršavaju samo prva dva poziva funkcije *Print*.
```
Clear[a]
Print[a]; Print[c]; Abort[]; Print[b]
a
c
$Aborted
```

Funkcija *CheckAbort* prihvata prekid i štampa vrednost koja je do tada izračunata.
```
CheckAbort[Print[a]; Abort[]; Print[b], aborted]
a
aborted
```
U sledećem izrazu nije bilo *Abort*-prekida:
```
CheckAbort[Print[a]; Print[b], aborted]
```

```
a
b
```

Rezultat funkcije *Abort* sadržan je u pozivu funkcije *CheckAbort,* pa se drugi *Print* izvršava.

```
Clear[b]
?aborted
```

Globalžaborted

```
CheckAbort[Print[a]; Abort[], aborted]; Print[b]
a
b
```

Abort se ukida sve dok se *AbortProtect* ne završi.

```
AbortProtect[Abort[];Print[a]];Print[b]
a
$Aborted
```

U sledećem izrazu funkcija *CheckAbort* prihvata (uviđa) prekid, ali se kao rezultat ne vraća njen drugi argument.

```
AbortProtect[Abort[]; CheckAbort[Print[a], x]]; Print[b]
```

5. STRUKTURNI TIPOVI PODATAKA

Strukturni tipovi podataka, kakvi su nizovi, slogovi, skupovi i datoteke, poznat je u višim programskim jezicima. U jeziku *MATHEMATICA* dominantan strukturni tip podataka su liste i izrazi. Liste obuhvataju nizove, skupove i slogove kao strukturne tipove podataka. Takođe, svi strukturni tipovi podataka imaju jedinstvenu strukturu.

5.1. INDEKSIRANI OBJEKTI

U mnogim vrstama izračunavanja koriste se *indeksirani objekti* (*arrays*) koji sadrže sekvence izraza, od kojih je svaki specificiran određenim indeksom. Jedan način da se definišu sekvence u *MATHEMATICA* jesu liste. Na primer, može se definisati lista oblika *a*= š*x*,*y*,*z*,…*ć*. Elementima takve liste se može pristupati koristeći izraze *a*ŠŠ*i*ĆĆ.Takođe, elementi ove liste se mogu modifikovati izrazima oblika *a*ŠŠ*i*ĆĆ=*value*.Ovakav pristup ima nedostatak što zahteva da se zadaju svi elementi kada se prvi put kreira lista. Često je mnogo praktičnije da se koriste indeksirani objekti u kojima se mogu zadati samo oni elementi koji su potrebni u određenom trenutku. Definicije indeksiranih objekata su oblika *a*Š*i*Ć.

Definicija vrednosti *a*Š1Ć:

aŠ1Ć = 9

9

i vrednosti *a*Š2Ć*:*

aŠ2Ć = 7

7

Sve do sada definisane vrednosti koje su povezane sa *a* mogu se pregledati izrazom

?a

Global`a

a[1] = 9

a[2] = 7

Može se definisati vrednost za *a*Š5Ć iako nisu definisane vrednosti *a*Š3Ć i *a*Š4Ć.

aŠ5Ć = 0

0

Lista vrednosti indeksiranih objekata *a*Š*i*Ć se može definisati na sledeći način:

TableŠ*a*ŠiĆ, š*i*, 5ćĆ

{9, 7, a[3], a[4], 0}

Izraz oblika *a*Š*i*Ć se može posmatrati kao "indeksirana" promenljiva. Manipulacija indeksiranim objektima opisana je u sledećoj tabeli

aŠiĆ=value	definisati ili predefinisati vrednost
aŠiĆ	pristup vrednosti
aŠiĆ=.	uklanjanje vrednosti
ClearŠaĆ	brisanje svih definicija vezanih za *a*
TableŠaŠiĆĆ, ši,1,nćĆ **ArrayŠa,nĆ**	konstrukcija indeksiranih objekata

U izrazu oblika *a*Š*i*Ć nije obavezno da "indeks" *i* bude broj. U stvari, indeks može da bude proizvoljni izraz. Upotreba simbola za indekse omogućava da se definišu jednostavne baze podataka.

Definicija objekta *area* sa "indeksom" *square* sa vrednošću 1:

areaŠsquareĆ = 1

1

Sledećim izrazom se dodaje novi rezultat za "bazu" *area:*

areaŠtriangleĆ = 1/2

$\frac{1}{2}$

Veličine koje odgovaraju bazi *area* mogu se pregledati na sledei način:

?area

```
Global`area
area[square] = 1
```

area[triangle] = $\frac{1}{2}$

Ovakve definicije se mogu koristiti u proizvoljnim izrazima.

4 areaŠsquareĆ + areaŠpentagonĆ

```
4 + area[pentagon]
```

5.2. VEKTORI I MATRICE

Vektori se predstavljaju listama čiji su elementi atomi, dok se matrice predstavljaju listama čiji su elementi liste.

ša,b,cć	vektor š*a,b,c*ć
šša,bć,šc,dćć	matrica čija je prva vrsta vektor š*a,b*ća druga vektor š*c,d*ć

Definicija matrice *m*:

m = šša, bć, šc, dćć

```
{{a, b}, {c, d}}
```

Prvi element matrice *m*:

mŠŠ1ĆĆ

```
{a, b}
```

Element matrice *m* u prvoj vrsti i drugoj koloni:

mŠŠ1,2ĆĆ

```
b
```

Vektor *v* ima dve komponente:

v = šx,yć

```
{x, y}
```

Objekti *p* i *q* se tretiraju kao skalari:

p v + q

```
{q + p x, q + p y}
```

Vektori se sabiraju po koordinatama:

v + šxp, ypć + šxpp, yppć

```
{x + xp + xpp, y + yp + ypp}
```

Simbol tačka označava skalarni proizvod dva vektora:

šx, yć . šxp, ypć

```
x xp + y yp
```

Mogu se množiti matrice i vektori odgovarajućih dimenzija.

m . v

```
{a x + b y, c x + d y}
```

Mogu se množiti i dve matrice.

m . m

$\{\{a^2+b\,c,\ a\,b+b\,d\},\ \{a\,c+c\,d,\ b\,c+d^2\}\}$

Pomoću operacija nad vektorima mogu se dobiti sklalarne veličine.

v . m . v

$x\,(a\,x+c\,y)+y\,(b\,x+d\,y)$

Kako se liste koriste za reprezentaciju vektora i matrica, ne može se uočiti razlika između vektora koji predstavljaju vrste i vektora koji prestavljaju kolone matrica.

5.3. DATOTEKE

MATHEMATICA koristi tekstualne datoteke. One se mogu obrađivati pomoću editora teksta. Tako pripremljene lako se prenose u druge aplikacije. Postojeće datoteke se mogu otvoriti pomoću menija *File>Open*, a zapisati poću *File>Save As*, a tekstualne pomoću *File>Save AsSpecial>Text*.
U aktivni notebook, mogu se dozvati naredbom *<<ime datoteke*, čime se pozivaju na izvršenje. Isto dejstvo ima izraz *Get*Š*"datoteka"*Ć.
Upisivanje nekog izraza u datoteku uz brisanje prethodnog sadržaja datoteke, postiže se naredbom
izraz >> datoteka
ili
*Put*Š*"datoteka"*Ć.
Izraz se dodaje na kraj datoteke uz čuvanje prethodnog sadržaja pomoću naredbe
izraz >>> datoteka
ili
*PutAppend*Š*"datoteka"*Ć.

Direktni pristup podacima u datoteci ostvaruje se pomoću kanala (*channel*). *MATHEMATICA* podržava kanale za čitanje i pisanje podataka.

(1) Otvaranje kanala i uspostavljanje veze sa određenom datotekom:

k=OpenReadŠ"dat"Ć	otvaranje kanala *k* za čitanje datoteke *dat*
k=OpenWriteŠ"dat"Ć	otvaranje kanala *k* za upisivanje u datoteku *dat* pri čemu se briše prethodni sadržaj
k=OpenAppendŠ"dat" Ć	otvaranje kanala *k* za upisivanje u datoteku *dat* pri čemu se novi podaci dodaju prethodnom sadržaju datoteke

(2) čitanje ili upisivanje podataka predstavljeno je sledećom tabelom:

ReadŠkĆ	čitanje jednog podatka sa kanala *k*

ReadŠk,tipĆ	čitanje podataka datog tipa sa kanala *k*
WriteŠk,izraz1,izraz2,... Ć	upisivanje niza izraza u kanal *k* posle čega se prelazi u novi red

(3) Zatvaranje kanala postičze se naredbom *Close*Š*k*Ć.

Kada datoteka sadrži veći broj podataka razdvojenih prazninama ili zarezima, ovom naredbom učitavaju se svi podaci iz datoteke i prevode u listu.

ReadListŠ"dat"Ć	učitavanje liste podataka iz datoteke
ReadListŠ"dat",tipĆ	učitavanje liste podataka određenog tipa iz datoteke
ReadListŠ"dat",RecordList-> TrueĆ	učitavanje liste podataka iz datoteke pri čemu se svaki red smešta u posebnu podlistu.

Primer.
Neka je "ulaz" datoteka u kojoj se nalaze brojevi 1, 2, 3, 4. Posle izraza
a=ReadListŠ"ulaz",šReal,RealćĆ;
PrintŠaĆ;
DetŠaĆ
Prikazuje se matrica šš1.,2.ć,š3.,4.ćć i vrednost njene determinante -2.

5.4. IZRAZI

MATHEMATICA manipuliše sa objektima različitih tipova: matematičkim formulama, listama, graficima. Iako oni izgledaju različito, *MATHEMATICA* ih predstavlja na univerzalan način: svi su oni izrazi. Prototip izraza u je oblika *f*Š*x,y,...* Ć. Ime funkcije je *f,* a argumenti su *x,y,* ... Izrazi se ne moraju uvek pisati u obliku *f*Š*x,y,...* Ć. Na primer, *x+y* je takođe izraz. *MATHEMATICA* konvertuje taj izraz u standardni oblik *Plus*Š*x+y*Ć. Isti princip važi za sve ostale operatore. U stvari, svaka naredba u *MATHEMATICA* se tretira kao izraz.
Objekat *f* u izrazu *f*Š*x,y, ...*Ć naziva se *glava* (*head*) tog izraza. On se može izdvojiti koristeći izraz *Head*Š*f*Š*x,y,* ĆĆ. Uopšte, glava izraza *expr* može se izdvojiti pomoću *Head*Š*expr*Ć.U toku pisanja programa često puta je potrebno da se testira glava nekog izraza da bi se odredila vrsta izraza.

Primer. U sledećoj tabeli su prikazane reprezentacije nekih izraza.

x+y+z	**PlusŠx,y,zĆ**

x y z	**TimesŠx,y,zĆ**
xČn	**PowerŠx,nĆ**
ša,b,cć	**ListŠa,b,cĆ**
a->b	**RuleŠa,bĆ**
a=b	**SetŠa,bĆ.**

Unutrašnja forma bilo kog izraza može se dobiti pomoću izraza *FullForm*Š*expr*Ć.

```
Head[f[x, y, z]]
f
Head[a + b + c]
Plus
Head[{a, b, c}]
List
Head[123344]
Integer
Head[123344.229]
Real
FullForm[x + y + z]
Plus[x, y, z]
FullForm[x + y + y]
Plus[x, Times[2, y]]
FullForm[x + y + y + z^2]
Plus[x, Times[2, y], Power[z, 2]]
FullForm[x + y + y + (z + q)^2]
Plus[x, Times[2, y], Power[Plus[q, z], 2]]
```

Izrazi se mogu iskoristiti za kreiranje korisničkih struktura. Na primer, tačke u trodimenzinalnom prostoru se mogu predstaviti izrazom *point*Š*x,y,z*Ć. "Funkcija" *point* ne izvršava operacije, već omogućava da se tri koordinate objedine, a da se rezultujući objekat označi sa *point*.

5.4.1. Posebni načini za unošenje izraza

Generalno, postoji četiri načina za pisanje izraza u *MATHEMATICA*:

fŠx,yĆ	standardna forma za *f*Š*x,y*Ć
fŽx	prefiksna forma za *f*Š*x*Ć
x//f	postfiksna forma za *f*Š*x*Ć
x~f~y	infiksna forma za *f*Š*x,y*Ć

```
2~List~a~List~b
{{2, a}, b}
x+y//f
f[x+y]
xy//List
{xy}
x, y//List
Syntax::tsntxi : "x, y//List" is incomplete; more input is needed.
x, y//List
{a, b, c}~Join~{d, e}
{a, b, c, d, e}
{a, b, c}[[2]]
b
```

Napomenimo da operator // ima vrlo mali prioritet. Ako se unese // *f* na kraju izraza koji sadrži aritmetički ili logički operator, tada se *f* primenjuje na ceo izraz. Prefiksna forma Ž ima mnogo veći prioritet, tako da je *f*Ž*x*+*y* ekvivalentno sa *f*Š*x*Ć+*y*, a ne sa *f*Š*x*+*y*Ć. Umesto *f*Š*x*+*y*Ć možemo pisati *f*Ž(*x*+*y*).

5.4.2. Delovi izraza

Delovima izraza može da se pristupa kao i delovima liste. Pri tome se izrazi posmatraju u svojoj unutrašnjoj formi, koja se može dobiti pomoću funkcije *FullForm*.

PartŠexpr,nĆ ili **exprŠŠnĆĆ**	*n*-ti element izraza *expr*
PartŠexpr,šn$_1$,n$_2$,...ćĆ ili **exprŠŠn$_1$,n$_2$,...ĆĆ**	element izraza *expr* u poziciji š*n*$_1$, *n*$_2$, ...ć

```
(x+y+z)[[2]]
y
(x+y+z)[[0]]
Plus
f[g[a], g[b]][[1]]
g[a]
FullForm[f[g[a], g[b]]]
f[g[a], g[b]]
g[a][[1]]
a
f[g[a], g[b]][[1, 1]]
```

a

(1+ x^2)[[2, 1]]

x

FullForm[x / y]

Times[x, Power[y, -1]]

(x / y)[[2]]

$\frac{1}{y}$

Neke funkcije za zamenu delova izraza date su u sledećoj tabeli:

ReplacePartŠexpr,new,nĆ	izraz dobijen zamenom *n*-tog dela izraza *expr* sa *new*
ReplacePartŠexpr,new,ši,j,... ćĆ	zamenjuje deo izraza *expr* u poziciji š*i*,*j*, ...ć
ReplacePartŠexpr,new,ši$_1$,j$_1$,...ć,ši$_2$,j$_2$,...ćĆ	zamenjuje delove sa nekoliko pozicija sa *new*

ReplacePart[a+ b + c + d, x^2, 3]

$a+b+d+x^2$

t = 1+ (3 + x) ^2 / y

$1+\frac{(3+x)^2}{y}$

FullForm[t]

Plus[1, Times[Power[Plus[3, x], 2], Power[y, -1]]]

t[[2, 1, 1]] = x

x

t

$1+\frac{x^2}{y}$

ReplacePart[t, z^2, {2, 1, 1}]

$1+\frac{z^4}{y}$

{a, b, c, d}[[2, 4]]

Part::partd : Part specification {a, b, c, d}⟦2, 4⟧ is longer than depth of object.

{a, b, c, d}⟦2, 4⟧

{a, b, c, d, e, f}[[{2, 4}]]

{b, d}

(a+ b + c + d + e + f)[[{2, 4}]]

b + d

(a+ b + c + d + e + f)[[{2, 4, 1}]]

```
a + b + d
```

5.4.3. Izrazi kao liste

Većina funkcija za rad sa listama može se primenjivati na proizvoljnim izrazima.

```
t = 1 + x + x^2 + y^2
1 + x + x^2 + y^2
Take[t, 2]
1 + x
Length[t]
4
FreeQ[t, x]
False
Variables[t]
{x, y}
f[a, b, c, d]
f[a, b, c, d]
Append[%, e]
f[a, b, c, d, e]
Reverse[Append[f[a, b, c, d], e]]
f[e, d, c, b, a]
```

5.4.4. Izrazi kao stabla

Svaki izraz se može predstaviti kao stablo. Struktura tog stabla se može prikazati koristeći funkciju *FullForm*.

```
FullForm[x^3 + (1 + x)^2]
Plus[Power[x, 3], Power[Plus[1, x], 2]]
TreeForm[x^3 + (1 + x)^2]
Plus[ |          , |                      ]
      Power[x, 3]   Power[ |         , 2]
                           Plus[1, x]
```

Prvi čvor (top čvor) je *Plus*. Odatle polaze dve "grane", xČ3 i (1+x)Č2, koje se ponovo prikazuju kao stablo.

```
{{a, b, c, d^2}, {x^3, y^4}}
{{a, b, c, d^2}, {x^3, y^4}}
TreeForm[%]
```

```
List[ |                          , |                            ]
     List[a, b, c, |            ]  List[ |           , |         ]
                  Power[d, 2]          Power[x, 3]   Power[y, 4]
```

5.4.5. Nivoi u izrazima

Funkcija *Part* dozvoljava pristup određenim delovima izraza. Međutim, kada su izrazi komplikovaniji, pogodno je da se odredi kolekcija delova izraza prema nekim kriterijumima. Nivoi (*levels*) dozvoljavaju da se specificiraju delovi u izrazima. Mnoge funkcije dozvoljavaju da se specificiraju delovi izraza u kojima će se primenjivati.

PositionŠexpr,form,nĆ	daje pozicije u kojima se form nalazi u nivoima od 1 do *n* izraza *expr*
PositionŠexpr,form,šnć Ć	daje poziciju samo u nivou *n*

O nivoima se može razmišljati u terminima stabla. Tačnije, nivo nekog dela izraza jeste njegovo rastojanje u stablu u odnosu na glavu izraza (top izraz), čiji je nivo 0. Nivo izraza se može specificirati na sledeći način:

n	nivoi 1 do *n*
Infinity	svi nivoi
šnć	samo nivo *n*
šn$_1$,n$_2$ć	nivoi od n_1 do n_2
Heads → True	uključivanje glave izraza
Heads → False	isključivanje glave izraza
LevelŠexpr,lev Ć	lista delova izraza *expr* u nivoima specificiranim sa *lev*
DepthŠexprĆ	ukupan broj nivoa u *expr*

```
(t = {x, {x, y}, y}) // TreeForm
List[x, |          , y]
     List[x, y]
Position[t, x, 1]
{{1}}
Position[t, x, 2]
{{1}, {2, 1}}
Position[t, x, {2}]
{{2, 1}}
```

```
(n = f[g[a], a, a, h[a], f]) // TreeForm
f[ |    , a, a,  |    , f]
  g[a]          h[a]
Position[n, a, {2, Infinity}]
{{1, 1}, {4, 1}}
Position[n, f, Heads -> False]
{{5}}
Position[n, f, Heads -> True]
{{0}, {5}}
Level[n, {2}]
{a, a}
Level[n, {1}]
{g[a], a, a, h[a], f}
Level[n, {-2}]
{g[a], h[a]}
Depth[g[a]]
2
Level[u, {-2}]
{}
```

6. POTPROGRAMI

Potprogrami se definišu kao programske celine koje se zatim po potrebi pozivaju, bilo u okviru programa u kome su definisani, bilo u drugim programima. Potprogrami u *MATHEMATICA* poseduju osnovne osobine kao i u većini drugih programskih jezika.

1. Svaki potprogram ima jednu ulaznu tačku.
2. Programska jedinica koja poziva potprogram prekida svoju aktivnost sve dok se ne završi pozvani potprogram. To znači da se u jednom trenutku izvršava samo jedan potprogram.
3. Po završetku potprograma upravljanje tokom izvršenja se prenosi na pozivajuću programsku jedinicu, na mestu iza poziva potprograma.

Za svaki potprogram su karakteristična sledeća četiri elementa:

- ime potprograma,
- lista imena argumenata
- telo potprograma,
- okruženje u kome je potprogram definisan.

6.1. LOKALNE PROMENLJIVE

U programiranju je često potrebno koristiti promenljive koje se koriste samo za vreme izvršavanja potprograma ili nekog izraza. Takve promenljive ne treba da postoje izvan takvih struktura, pa se zato nazivaju još i lokalne promenljive. Stukture koje omogućavaju korišćenje raznih vrsta lokalnih promenljivih u *MATHEMATICA* su *Module, With* i *Block*.

6.1.1. Moduli i lokalne promenljive

Lokalne promenljive u *MATHEMATICA* se mogu koristiti u modulima. U modulu se zadaje lista lokalnih simbola, koji postoje samo u modulu, i ne utiču na simbole istog imena van tog modula. Modul je posebno označeni deo programa koji se definiše naredbom

*Module*Šš*promenljiva*1*, promenljiva*2*,...ć, program* Ć

ili

*Module*Šš*promenljiva=vrednost,...ć, program* Ć

ModuleŠšx, y, ...ć, proceduraĆ,	modul sa lokalnim promenljivim *x, y, ...*
ModuleŠšx=x_0, y=y_0,...ć, proceduraĆ	postavljanje početnih vrednosti za *x, y, ...*

Pri definisanju početnih vrednosti za lokalne promenljive *x, y, ...* Izrazi x_0, y_0, ... se računaju pre izvršenja procedure.

Svakoj lokalnoj promenljivoj u modulu se dodeljuje jedinstveno ime oblika *ime*$*broj*, gde se *broj,* pri svakom nardnom dodeljivanju povećava za 1. Lokalnoj promenljivoj u modulu se u toku rada nikada neće dva puta dodeliti ista vrednost.

Primer.

```
nula := n Pi
nula
n π
nula - nula
0
nula := Module[{n}, n Pi]
nula
n$30 π
nula - nula
n$31 π - n$32 π
nula := Module[{n}, n*Pi]; Print[nula]; Print[nula - nula]
```

```
n$9 π
n$10 π - n$11 π
```

Primer. Pomoću modula možemo definisati funkcije.

```
s[n_] := Module[{h = 2*Pi/n},
      Table[{i*h, N[Sin[i*h]]}, {i, 0, n}]
   ]
```

Promenljiva *h* se definiše odmah na početku, tako da *s*Š4Ć daje vrednost:

```
s[4]
```

$\left\{\{0, 0.\}, \left\{\frac{\pi}{2}, 1.\right\}, \{\pi, 0.\}, \left\{\frac{3\pi}{2}, -1.\right\}, \{2\pi, 0.\}\right\}$

Promenljiva *h* je lokalna i ne vidi se izvan modula. Izraz *?h* potvrđuje da ne postoje definicije vezane za ovaj simbol.

Izlazak iz funkcije sa vraćanjem vrednosti se zadaje naredbom

*Return*Š*izraz*Ć.

Ova naredba vraća vrednost izraza kao rezultat tekuće funkcije i zatim se kontrola programa prenosi u pozivajuću programsku jedinicu.

```
f[k_] :=
  Module[{m},
          m = 4 - k;
          If[m < 0, Return[m je negativno]];
          Table[a, {i, 1, m}]
  ]
f[1]
{a, a, a}
f[7]
-3 je negativno
```

Posmatrajmo funkciju

```
g[m_, n_] :=
    Module[{k},
        k = m + n;
        If[k <= 0, Return[brojevi nisu pozitivni]];
        Table[1, {i, 1, k}]
    ]
```

Za *g*Š2,3Ć dobijamo rezultat š1,1,1,1,1ć, ali za *g*Š-2,-3Ć poruku da brojevi nisu pozitivni:

```
g[2, 3]
{1, 1, 1, 1, 1}
g[-2, -3]
```

```
brojevi nisu pozitivni
```

Naredba *Return* se može korisiti i za izlazak iz ciklusa.

```
Do[ Print[i]; If[i == 5, Return[izlaz]],
   {i, 1, 6}
  ]
1
2
3
4
5
izlaz
```

6.1.2. Blokovi i lokalne promenljive

U modulima se imena promenljivih se tretiraju kao lokalna. Međutim, pnekad je potrebno da promenljiva bude globalna, ali da dobije lokalnu vrednost. U programu *MATHEMATICA* gloabalne i lokalne promenljive postoje unutar strukture *Block*.

BlockŠšx, y, ...ć, proceduraĆć	blok sa lokalnim rednostima simbola *x y, ...*
BlockŠšx=x$_0$, y=y$_0$, ...ć, proceduraĆ	postavljanje početnih vrednosti za *x,y, ...*

```
Sin[n Pi]
Sin[n π]
Block[{n = 1/2}, %]
1
```

(*U bloku je *n* dobila konkretnu vrednost i izračunava se SinŠ*Pi*/2Ć*)
OutŠ8Ć=1

```
n
n
```

(*Van bloka, *n* i dalje nema vrednost *)

Primer. Posmatrajmo sledeću sekvencu izraza:

```
f[n_] := Sin[n*Pi];
Block[{n = 1/2}, Print[f[n]] ];
Print[n] ;
1
n
Null^3
```

Unutar bloka *n* dobija konkretnu vrednost (n=1/2), i izračunava se $\sin(n\pi/2)=\sin(\pi/2)=1$. Van bloka, *n* i dalje ne poseduje vrednost.

Primer.

```
f[x_] := Sin[t*x];
Print[f[a]];

Block[{t = 3}, Print[f[a]]];
Block[{t}, Print[f[a]]]
Sin[a t]
Sin[3 a]
Sin[a t]
Null^4
```

6.1.3. Razlika između modula i blokova

Promenljiva *x* u modulu *Module*Šš*xć, procedura*Ć predstavlja jedinstven simbol koji dobija različita imena svaki put kada se modul koristi, i nezavisan je od globalnog simbola *x* ako on postoji. Nasuprot tome, promenljiva *x* u bloku *Block*Šš*xć, procedura*Ć je i dalje globalni simbol. Blok samo čini njegovu vrednost lokalnom. Originalna vrednost simbola *x* se vraća nakon izlaska iz bloka.

```
t = 42
42
Module[{t}, Print[t]]
t$33
Block[{t}, Print[t]]
t
t
42
```

U modulu lokalna promenljiva *t* dobija jedinstveno ime *t$*33, a u bloku njeno ime ostaje globalno dok vrednost postaje lokalna. Međutim, čim se iz bloka kao rezultat vrati izraz koji sadrži promenljivu *t*, ona ponovo dobija svoju globalnu vrednost 42 i rezultujući izraz se ponovo izračunava. Da bi se prikazala vrednost promenljive *t* akva je bila u bloku, koristi se funkcija *Print*Š*t*Ć.

```
f[x_] := Sin[42 x] - 2
f[a]
-2 + Sin[42 a]
```

```
Block[{t = 3}, f[a]]
-2 + Sin[42 a]
Block[{a = 3}, f[a]]
-2 + Sin[126]
Block[{a}, Print[f[a]]]
-2 + Sin[42 a]
```

6.1.4. Lokalne promenljive u *With*

U modulima se definišu lokalne promenljive, kojima se u okviru modula mogu proizvoljno dodeljivati razne vrednosti. Međutim, u praksi su često potrebne lokalne konstante kojima se samo jednom dodeljuje vrednost. Struktura *With* služi za definisanje konstanti.

WithŠšx=x_0, y=y_0, ...ć, proceduraĆ	lokalne konstante *x*,*y*,... u proceduri *procedura*.

```
W[x_] := With[{t = x - 1}, 2 + t^2]
W[a]
2 + (-1 + a)^2
t
42
```

Naredba *With*Ššx=x_0,, ... *ć, procedura*Ć zamenjuje sve pojave simbola *x*, ... u *procedura* izrazima x_0, ... a zatim je izvršava. Prednost upotrebe strukture *With* umesto modula je čitljivost programa. Za razliku od modula, već iz liste zaglavlja strukture *With* jasno se vidi koji su simboli konstante i koju vrednost imaju u proceduri.
Konstrukcije *With* mogu biti ugnježdene jedna u drugoj. U slučaju da se više struktura *With* koje definišu iste konstante, nalaze jedna unutar druge, u proceduri važe vrednosti konstanti iz unutrašnje strukture. Dakle, lokalne promenljive iz unutrašnje strukture imaju prioritet nad onima iz spoljašnje (preklapaju spoljašnje).

Primer.

```
With[{t = 5}, With[{t = 7}, t^2]]
49
Module[{t = 8}, With[{t = 9}, t^2]]
81
With[{t = a}, With[{n = b}, t + n]]
a + b
```

Primer. Činjenica da vrednosti konstanti iz unutrašnje strukture preklapaju vrednosti iz spoljašnje strukture ilustruje se sledećim primerima. Vrednost izraza

WithŠšt=0*ć, WithŠšt*=1*ć,tĆ* Ć

jednaka je 1. U izrazu

WithŠšt=0*ć,* (*WithŠšt*=1*ć,PrintŠt*+1*ĆĆ; PrintŠtĆ*) Ć

prvo se štampa 2, a zatim 0:

```
With[{t = 0}, (With[{t = 1}, Print[t + 1]]; Print[t])]
2
0
```

Primer. Posmatrajmo sledeće izraze:

```
t = 1; f[x_] := With[{t = x + 1}, 2 + t^2]
f[a]
```

$2+(1+a)^2$

Globalna vrednost promenljive *t* jednaka je 1. U funkciji *fŠx_Ć*, *x* je lokalna promenljiva. Vrednost funkcije *fŠaĆ* jednaka je 2+(1+*a*)*Č*2, ali je globalna vrednost za *t* ostala 1.

```
t
1
```

Primer. Početne vrednosti u strukturi *With* se izračunavaju pre nego što se počne izvršavati sekvenca naredbi u njenom telu. Zbog toga dobijamo sledeće:

```
t = 5; With[{t = t + 1}, 3*t]
18
```

Primer 5. Konstanta iz spoljašnje strukture se vidi u unutrašnjoj strukturi, iako nije tamo definisana. Tako je vrednost izraza

WithŠšt=ać, WithŠšu=bć,t+uĆ Ć

jednaka *a*+*b*.

6.2 SEKVENCE IZRAZA

Svako izračunavanje u *MATHEMATICA* se izvršava preko sekvence koraka. Svaki korak se može napisti u posebnom redu. Međutim, veći broj koraka se može napisati i u istom redu. To se može postići razdvajanjem pojedinačnih izraza znakom ;. Izrazi u takvoj sekvenci se izvršavaju jedan za drugim s leva na desno. Konačni rezultat procedure je vrednost poslednjeg izraza. Ponekad se sekvenca izraza razdvojenih znakom ; naziva i procedura.

izraz$_1$; ...; izraz$_n$	sekvenca od *n* izraza.

Ako iz procedure ne treba da se vrati vrednost poslednjeg izraza, tada se na kraj procedure stavlja znak ;. *MATHEMATICA* zapravo i tada vraća vrednost izraza, ali je u tom slučaju poslednji izraz prazna naredba.

6.3. FUNKCIJE KAO SEKVENCE IZRAZA

Funkcija se definiše kao procedura jednostavnim navođenjem niza naredbi razdvojenim znakom ; iza operatora odložene dodele :=. Tako definisana funkcija se poziva na isti način kao i sve ostale funkcije, sa tom razlikom da se u jednoj funkciji može izvršiti i više nezavisnih operacija. Vrednost poslednjeg izraza se vraća kao rezultat funkcije. Kada se funkcija definiše kao procedura, niz naredbi koje joj pripadaju se moraju navesti u malim zagradama.

fŠargĆ:=(izr$_1$; ...; izr$_n$)	funkcija *f* definisana kao procedura

```
S[n_] =
  (step = 2 Pi / n;
   Table[{i step, n [Sin[i step]]},
         {i, 0, n}
   ]   )
Table::iterb : Iterator {i, 0, n} does not have appropriate bounds.
Table[{i step, n[Sin[i step]]}, {i, 0, n}]
S[4]
```

$$\left\{\{0, 4[0]\}, \left\{\frac{2\pi}{n}, 4\left[\text{Sin}\left[\frac{2\pi}{n}\right]\right]\right\}, \left\{\frac{4\pi}{n}, 4\left[\text{Sin}\left[\frac{4\pi}{n}\right]\right]\right\}, \left\{\frac{6\pi}{n}, 4\left[\text{Sin}\left[\frac{6\pi}{n}\right]\right]\right\}, \left\{\frac{8\pi}{n}, 4\left[\text{Sin}\left[\frac{8\pi}{n}\right]\right]\right\}\right\}$$

```
?step
```

Globalžstep

$$\text{step} = \frac{2\pi}{n}$$

```
Remove[step]
```

Ako bi se definicija funkcije *SŠn_Ć* iz prethodnog primera navela bez malih zagrada, *MATHEMATICA* bi takav izraz protumačila kao proceduru u kojoj je prva naredba definisanje funkcije *SŠn_Ć:=step=2Pi/n,* dok se ostale naredbe izvršavaju nezavisno od nje.

Osnovni problem pri definisanju funkcija kao procedura jesu bočni efekti tj. definisanje pomoćnih promenljivih koje ostaju nakon poziva funkcije. Taj problem se eliminiše upotrebom lokalnih promenljivih.

```
S[n_] := Module[{step = 2Pi/n}, Table[{i step, N[sin[i step]]}, {i, 0, n}]]
```

Promenljivoj *step* se vrednost dodeljuje odmah na početku.

```
S[4]
```

$\left\{\{0, \sin[0.]\}, \left\{\frac{\pi}{2}, \sin[1.5707963267948966]\right\}, \{\pi, \sin[3.141592653589793]\}, \left\{\frac{3\pi}{2}, \sin[4.71238898038469]\right\}, \{2\pi, \sin[6.283185307179586]\}\right\}$

(* Ovog puta je promenljiva step lokalna pa se ne vidi izvan modula*)

```
?step
```

Globalžstep

$step = \frac{2\pi}{n}$

7. SIMBOLIČKA IZRAČUNAVANJA

Ono što u najvećoj meri karakteriše programski paket *MATHEMATICA* jesu njene velike mogućnosti u simboličkoj obradi podataka. Algebarskim formulama se može manipulisati kao sa brojevima.

Numerička izračunavanja su prisutna u sledećem primeru

3 + 62 - 1

64

Sledeći izraz sadrži simbolička izračunavanja

3x - x + 2

2 + 2 x

U toku ovih simboličkih izračunavanja izvršena je simplifikacija izraza. Mnoga poznata algebarska pravila se koriste za simplifikaciju algebarskih izraza:

SqrtŠ1 + xĆČ4

$(1 + x)^2$

Za sledeći izraz ne može se primeniti ni jedna od poznatih transformacija, pa je rezultat jednak ulaznom izrazu:

LogŠ1 + CosŠxĆĆ

Log[1 + Cos[x]]

Kada se vrši simplifikacija izraza $x+x$ u $2x$, promenljiva x se koristi u čisto formalnom modalitetu. Tada x predstavlja simbol koji stoji na mestu proizvoljnog izraza. Simbol x se može zameniti određenom vrednošću. Takva vrednost može da bude brojna vrednost, ali može da bude i drugi izraz.

7.1. OPERACIJE NA POLINOMIMA

Postoji više načina da se izraze neki algebarski izrazi. Na primer, izraz $(1+x)^2$ se može napisati i u obliku $1+2x+x^2$. Sledeće funkcije se mogu koristiti za konvertovanje između različitih oblika algebarskih izraza:

ExpandŠpolyĆ	ekspanzija proizvoda i stepena u polinomu *poly*
FactorŠpolyĆ	faktorizacija izraza *poly*
FactorTermsŠpolyĆ	izdvajanje zajedničkih faktora u *poly*
CollectŠpoly,xĆ	srediti polinom u obliku sume stepena "dominantne promenljive" *x*
CollectŠpoly,šx,y,...ć Ć	srediti polinom u obliku sume stepena od *x,y,...*
PowerExpandŠexpr Ć	ekspanzija $(ab)^c$ i $(a^b)^c$ u *expr*

```
(2 + 4 x^2) ^2 (x - 1) ^3
```
$(-1+x)^3\,(2+4x^2)^2$

```
t = Expand[%]
```
$-4+12x-28x^2+52x^3-64x^4+64x^5-48x^6+16x^7$

```
Factor[t]
```
$4\,(-1+x)^3\,(1+2x^2)^2$

```
Expand[(1 + 2 x + y) ^3]
```
$1+6x+12x^2+8x^3+3y+12xy+12x^2y+3y^2+6xy^2+y^3$

```
Collect[%, x]
```
$1+8x^3+3y+3y^2+y^3+x^2\,(12+12y)+x\,(6+12y+6y^2)$

```
Collect[Expand[(1 + x + 2 y + 3 z) ^3], {x, y}]
```
$1+x^3+8y^3+9z+27z^2+27z^3+x^2\,(3+6y+9z)+$
$y^2\,(12+36z)+y\,(6+36z+54z^2)+x\,(3+12y^2+18z+27z^2+y\,(12+36z))$

```
Expand[(x + 1) ^2 (y + 1) ^2, x]
```
$(1+y)^2+2x\,(1+y)^2+x^2\,(1+y)^2$

```
(xy) ^n
```
xy^n

```
PowerExpand[%]
```
xy^n

Faktorizacija izraza x^{10}-1 se može izvršiti izrazom

FactorŠ xČ10 - 1 Ć

$(-1+x)\,(1+x)\,(1-x+x^2-x^3+x^4)\,(1+x+x^2+x^3+x^4)$

U ovom slučaju funkcija *Expand* daje "jednostavniji" zapis:

ExpandŠ % Ć

```
-1 + x^10
```

7.2. ISPITIVANJE STRUKTURE POLINOMA

Postoji veći broj funkcija za simboličku manipulaciju polinomima.

PolynomialQŠexpr,xĆ	testira da li je *expr* polinom od *x*
PolynomialQŠexpr,šx_1,x_2,...ćĆ	testira da li je *expr* polinom od x_i
VariablesŠpolyĆ	lista promenljivih u *poly*
LengthŠpolyĆ	ukupan broj terama u *poly*
ExponentŠpoly,xĆ	maksimalni eksponent sa kojim se *x* pojavljuje u *poly*
CoefficientŠpoly,exprĆ	koeficijent uz *expr* u *poly*
CoefficientŠpoly,expr,nĆ	koeficijent uz *exprČn* u *poly*
CoefficientŠpoly,expr,0Ć	koeficijent u *poly* koji je nezavisan od *expr*
CoefficientListŠpoly,šx_1,x_2,...ć Ć	generiše listu koeficijenata uz x_i u *poly*

```
t = Expand[(1 + x)^3 (1 - y - x)^2]
1 + x - 2 x^2 - 2 x^3 + x^4 + x^5 - 2 y - 4 x y + 4 x^3 y + 2 x^4 y + y^2 + 3 x y^2 + 3 x^2 y^2 + x^3 y^2
PolynomialQ[t, x]
True
PolynomialQ[t, z]
True
PolynomialQ[x + Sin[x], x]
False
Variables[t]
{x, y}
Length[t]
14
Exponent[t, x]
5
Coefficient[t, x^2]
-2 + 3 y^2
CoefficientList[a + 3 x^2 + 4 x^4, x]
{a, 0, 3, 0, 4}
CoefficientList[t, {x, y}]
{{1, -2, 1}, {1, -4, 3}, {-2, 0, 3}, {-2, 4, 1}, {1, 2, 0}, {1, 0, 0}}
```

x^a + x^b + y^c

$x^a + x^b + y^c$

Exponent[%, x]

Max[0, a, b]

u = (-4x + x^2) / (-x + x^2) + (-4 + 3x + x^2) / (-1 + x^2)

$$\frac{-4x + x^2}{-x + x^2} + \frac{-4 + 3x + x^2}{-1 + x^2}$$

7.3. STRUKTURNE OPERACIJE SA RACIONALNIM IZRAZIMA

ExpandNumeratorŠexprĆ	ekspanzija brojioca
ExpandDenominatorŠexprĆ	ekspanzija imenioca
ExpandŠexprĆ	ekspanzija brojioca, pri čemu se svaki term deli imeniocem
ExpandAllŠexprĆ	ekspanzija i brojioca i imenioca
TogetherŠexprĆ	svođenje na NZS u izrazu *expr*
ApartŠexprĆ	ispisuje izraz u obliku sume terama sa prostim imeniocima
CancelŠexprĆ	skratiti zajedničke faktore između brojioca i imenioca
FactorŠexprĆ	izvršiti kompletnu faktorizaciju
ApartŠexpr,varĆ	koristi se u izrazu za nekoliko varijabli za parcijalnu dekompoziciju razlomaka u odnosu na različite varijable

Together[u]

$$\frac{2\,(-4 + x^2)}{(-1 + x)\,(1 + x)}$$

Factor[%]

$$\frac{2\,(-2 + x)\,(2 + x)}{(-1 + x)\,(1 + x)}$$

Apart[u]

$$2 - \frac{3}{-1 + x} + \frac{3}{1 + x}$$

Cancel[u]

$$\frac{-4 + x}{-1 + x} + \frac{4 + x}{1 + x}$$

v = (x^2 + y^2) / (x + xy)

$$\frac{x^2+y^2}{x+xy}$$

```
Apart[v, x]
```

$$x-xy+\frac{xy^2+y^2}{x+xy}$$

```
Apart[v, y]
```

$$\frac{x^2}{x+xy}+\frac{y^2}{x+xy}$$

7.4. ALGEBARSKE OPERACIJE SA POLINOMIMA

PolynomialQuotientŠpoly$_1$,poly,xĆ	količnik deljenja polinoma $poly_1$ po x polinomom $poly_2$, pri čemu se odbacuje ostatak
PolynomialRemainderŠpoly$_1$,poly$_2$ Ć	ostatak deljenja polinoma $poly_1$ po x polinomom $poly_2$
PolynomialGCDŠpoly$_1$,poly$_2$Ć	NZD dva polinoma
PolynomialLCMŠpoly$_1$,poly$_2$Ć	NZS dva polinoma
PolynomialModŠpoly,mĆ	redukcija poly po modulu *m*

```
PolynomialRemainder[x^2, x + 1, x]
1
PolynomialQuotient[x^2, x + 1, x]
-1 + x
%%
1
Simplify[(x + 1) % + %%]
2 x
{PolynomialRemainder[x + y, x - y, x], PolynomialRemainder[x + y, x - y, y]}
{2 y, 2 x}
```

7.4. SIMPLIFIKACIJA ALGEBARSKIH IZRAZA

U mnogim situacijama je potrebna "najprostija" forma nekog izraza. Iako je teško odrediti koja je forma najprostija, *MATHEMATICA* poseduje funkcije za nalaženje najprostijeg oblika nekog izraza. Ove funkcije upoređuju različite oblike izraza i nalaze najjednostavniji oblik, koji sadrži najmanji broj delova.

SimplifyŠexprĆ	najjednostavniji oblik izraza *expr* pomoću algebarskih transformacija

FullSimplifyŠexpr Ć	najjednostavniji oblik izraza *expr*, dobijen primenom svih poznatih transformacija

SimplifyŠxČ2 + 2x + 1Ć

$(1+x)^2$

Funkcija *Simplify* se može koristiti za "prečišćavanje" komplikovanih izraza koji su dobijeni različitim izračunavanjima.

```
Integrate[1/(x^4-1), x]
```

$-\frac{\text{ArcTan}[x]}{2}+\frac{1}{4}\text{Log}[-1+x]-\frac{1}{4}\text{Log}[1+x]$

```
D[%, x]
```

$\frac{1}{4(-1+x)}-\frac{1}{4(1+x)}-\frac{1}{2(1+x^2)}$

```
Simplify[%]
```

$\frac{1}{-1+x^4}$

7.5. MANIPULACIJA JEDNAČINAMA

U *MATHEMATICA* jednačine se tretiraju kao logičke naredbe. Na primer, jednačina *xČ*2+3*x*==2 tretira se kao logička naredba kojom se izraz *xČ*2+3*x* upoređuje sa 2. Ako se dodeli eksplicitna vrednost za *x*, npr. *x*=4, *MATHEMATICA* može da eksplicitno determiniše da je vrednost logičke naredbe *xČ*2+3*x*==2 jednaka *False*. Ako eksplicitna vrednost za x nije dodeljena, *MATHEMATICA* neće računati vrednost izraza *xČ*2+3*x*==2, već ostavlja jednačinu u simboličkoj formi.
Jednačine se mogu kombinovati kao i logičke naredbe, a takođe se mogu izračunavati i njihova rešenja.

Jednačine se mogu rešavati funkcijom *Solve*. Na primer, izrazom

*Solve*Š*izraz*1 == *izraz*2, *x*Ć

izračunava se vrednost *x* koja zadovoljava jednačinu *izraz*1==*izraz*2. Eksplicitne formule se dobijaju za polinomne jednačine stepena manjeg od 5, za neke polinomne jednačine specijalnog oblika, kao i za trigonometrijske jednačine. Ako jednačina ima više rešenja, ne moraju se dobiti sva.

```
x=.
b=.
Solve[a x^2 + b x + c == 0, x]
```

$\left\{\left\{x\to\frac{-b-\sqrt{b^2-4ac}}{2a}\right\},\left\{x\to\frac{-b+\sqrt{b^2-4ac}}{2a}\right\}\right\}$

```
Solve[x^6 - 64 == 0, x]
```

```
{{x→-2}, {x→2}, {x→-2 (-1)^(1/3)}, {x→2 (-1)^(1/3)}, {x→-2 (-1)^(2/3)}, {x→2 (-1)^(2/3)}}
N[Solve[x^6 - 64 == 0, x]]
{{x→-2.}, {x→2.}, {x→-1. - 1.7320508075688774 ⅈ}, {x→1. + 1.7320508075688774 ⅈ},
 {x→1. - 1.7320508075688776 ⅈ}, {x→-1. + 1.7320508075688776 ⅈ}}
Solve[x^6 == 1, x]
{{x→-1}, {x→1}, {x→-(-1)^(1/3)}, {x→(-1)^(1/3)}, {x→-(-1)^(2/3)}, {x→(-1)^(2/3)}}
N[%]
{{x→-1.}, {x→1.}, {x→-0.5 - 0.8660254037844387 ⅈ}, {x→0.5 + 0.8660254037844387 ⅈ},
 {x→0.5 - 0.8660254037844388 ⅈ}, {x→-0.5 + 0.8660254037844388 ⅈ}}
```

Ako se zahtevaju numerička rešenja jednačine, može se koristiti funkcija *NSolve*.

```
NSolve[x^3 + 7.8 x + 1 == 0, x]
{{x→-0.12793666149664779}, {x→0.06396833074832389 - 2.795044872987942 ⅈ},
 {x→0.06396833074832389 + 2.795044872987942 ⅈ}}
NSolve[Sqrt[1 - x] + Sqrt[1 + x] == a, x]
```

$$\left\{\left\{x \to -0.5\, a \sqrt{4.-1.\, a^2}\right\}, \left\{x \to 0.5\, a \sqrt{4.-1.\, a^2}\right\}\right\}$$

Funkcijom

*Solve*Šš*izraz*1==*izraz*2, *izraz*3==*izraz*4,...ć,š*x,y,*ćĆ

mogu se rešavati sistemi jednačina. Tako se dobija jedno rešenje specificiranog sistema po promenljivim *x,y,...,* iako može postojati više rešenja. Ako sistem nema rešenja, kao izlaz se dobija prazna lista. Ukoliko rešenja postoje samo za specijalne vrednosti parametara, one se određuju pomoću izraza

*Reduce*Šš*izraz*1==*izraz*2, *izraz*3==*izraz*4,...ć, š*x,y,...*ćĆć

Takođe, ako sistem ima više rešenja, mogu se dobiti funkcijom *Reduce*.

Funkcijom

*Eliminate*Šš*izraz*1==*izraz*2, *izraz*3==*izraz*4,...ć,š*x,y,...*ćĆ

može se sistem jednačina pojednostaviti koristeći eliminaciju navedenih promenljivih.

```
Solve[Sqrt[1 - x] + Sqrt[1 + x] == a, x]
```

$$\left\{\left\{x \to -\frac{1}{2} a \sqrt{4-a^2}\right\}, \left\{x \to \frac{1}{2} a \sqrt{4-a^2}\right\}\right\}$$

```
Solve[{x^2 == 4, x == a}, x]
{}
Reduce[{x^2 == 4, x == a}, x]
a == -2 && x == -2 || a == 2 && x == 2
```

U poslednjem rezultatu rešenje postoji za specijalni slučaj a=2 ili a=-2.

```
Eliminate[{3 x + 6 y == 2, bx + 2 y == 3}, y]
```

$7+3x==3\,bx$

Solve[%]

Solve::svars : Equations may not give solutions for all "solve" variables.

$\{\{bx \to \frac{7}{3} + x\}\}$

Reduce[%%]

$\frac{1}{3}(7+3x) == bx$

Solve[{ax + by == 1, x - y == 2}, {x, y}]

Solve::svars : Equations may not give solutions for all "solve" variables.

$\{\{x \to 2 + y\}\}$

Solve[{x^2 + y^2 == 1, x + y == a}, {x, y}]

$\{\{x \to \frac{1}{2}(a - \sqrt{2-a^2}), y \to \frac{1}{2}(a + \sqrt{2-a^2})\}, \{x \to \frac{1}{2}(a + \sqrt{2-a^2}), y \to \frac{1}{2}(a - \sqrt{2-a^2})\}\}$

Solve[x == x, x]

{{}}

Reduce[x == x, x]

True

Solve[x + y == 1 && x - y == 2, {x, y}]

$\{\{x \to \frac{3}{2}, y \to -\frac{1}{2}\}\}$

Solve[x + y == 1 || x - y == 2, {x, y}]

Solve::svars : Equations may not give solutions for all "solve" variables.

$\{\{x \to 1 - y\}, \{x \to 2 + y\}\}$

Solve[x^3 == x && x ≠ 0, x]

$\{\{x \to -1\}, \{x \to 1\}\}$

Solve[x^3 == x && x ≠ 1 || x^2 == 2, x]

$\{\{x \to -1\}, \{x \to 0\}, \{x \to -\sqrt{2}\}, \{x \to \sqrt{2}\}\}$

7.6. PRAVILA TRANSFORMACIJE

Transformacije izraza u *MATHEMATICA* nisu bazirane na njihovom algebarskom značenju, već na njihovoj strukturi.

1 + x^2 + x^4 /. x^2 → a

$1 + a + x^4$

1 + x + xČ2 /. x -> 2 - y

$3 + (2-y)^2 - y$

MATHEMATICA tretira pravilo transformacije slično ostalim simboličkim izrazima:

x -> 3 + y

$x \to 3 + y$

Takvo pravilo transformacije se može primeniti na izraz *xČ2 Đ* 9:

xČ2 - 9 /. %

$-9+(3+y)^2$

7.7. DIFERENCIRANJE

Postoje standardne funkcije za numeričko i simboličko diferenciranje.

DŠf,xĆ	parcijalni izvod $\frac{\partial f}{\partial x}$
DŠf,x$_1$,x$_2$,...Ć	višestruki izvod $\frac{\partial}{\partial x_1}\frac{\partial}{\partial x_2}\ldots f$
DŠf,šx,nćĆ	$\frac{\partial^n f}{\partial x}$

```
D[x^n, x]
```
$n\,x^{-1+n}$

```
D[ArcTan[x], x]
```
$\frac{1}{1+x^2}$

```
D[f[x], x]
```
f′[x]

```
D[2 x f[x^2], x]
```
$2\,f[x^2]+4\,x^2\,f'[x^2]$

```
D[x^2 + y^2, x]
```
$2\,x$

DŠgŠxČ2, yČ2Ć, xĆ

$2\,x\,g^{(1,0)}[x^2,\,y^2]$

f'ŠxĆ	prvi izvod funkcije *f* jedne promenljive
f$^{(n)}$ŠxĆ	*n*-ti izvod funkcije jedne promenljive
f$^{(n1,n2,....)}$ ŠxĆ	izvod funkcije nekoliko promenljivih, n_i puta u odnosu na *i*-tu promenljivu
f''ŠxĆ	drugi izvod
f'''ŠxĆ	treći izvod

7.8. DIFERENCIJALNE JEDNAČINE

Funkcija *DSolve* može da rešava diferencijalne jednačine i sisteme diferencijalnih jednačina u čisto funkcionalnom obliku.

DSolveŠeqns, yŠxĆ,xĆ	rešava diferencijalnu jednačinu za *y*Š*x*Ć, smatrajući *x* za nezavisnu

	promenljivu
DSolveŠšeqn1,eqn$_2$,...ć,šy$_1$,y$_2$,...ć,xĆ	rešava sistem diferencijalnih jednačina

```
DSolve[y'[x] == a y[x], y[x], x]
```

$$\{\{y[x] \to e^{a x} C[1]\}\}$$

```
DSolve[{y'[x] == a y[x], y[0] == 1}, y[x], x]
```

$$\{\{y[x] \to e^{a x}\}\}$$

MATHEMATICA može da rešava linearne i nelinearne obične diferencijalne jednačine, kao i sisteme diferencijalnih jednačina.

```
DSolve[{x'[t] == y[t], y'[t] == x[t]}, {x[t], y[t]}, t]
```

$$\left\{\left\{x[t] \to \frac{1}{2} e^{-t} \left(1 + e^{2t}\right) C[1] + \frac{1}{2} e^{-t} \left(-1 + e^{2t}\right) C[2],\right.\right.$$
$$\left.\left. y[t] \to \frac{1}{2} e^{-t} \left(-1 + e^{2t}\right) C[1] + \frac{1}{2} e^{-t} \left(1 + e^{2t}\right) C[2]\right\}\right\}$$

```
DSolve[y''[x] == a y'[x] + y[x], y, x]
```

$$\left\{\left\{y \to \text{Function}\left[\{x\}, e^{\frac{1}{2}\left(a-\sqrt{4+a^2}\right)x} C[1] + e^{\frac{1}{2}\left(a+\sqrt{4+a^2}\right)x} C[2]\right]\right\}\right\}$$

```
DSolve[y''[x] == a y'[x] + y[x], y, x, DSolveConstants -> K]
```

$$\left\{\left\{y \to \text{Function}\left[\{x\}, e^{\frac{1}{2}\left(a-\sqrt{4+a^2}\right)x} K[1] + e^{\frac{1}{2}\left(a+\sqrt{4+a^2}\right)x} K[2]\right]\right\}\right\}$$

```
DSolve[{y''[x] == a y'[x] + y[x], y[0] == 1, y'[0] == 0}, y, x]
```

$$\left\{\left\{y \to \text{Function}\left[\{x\}, \frac{1}{2\sqrt{4+a^2}}\right.\right.\right.$$
$$\left.\left.\left.\left(a e^{\frac{1}{2}\left(a-\sqrt{4+a^2}\right)x} + \sqrt{4+a^2}\, e^{\frac{1}{2}\left(a-\sqrt{4+a^2}\right)x} - a e^{\frac{1}{2}\left(a+\sqrt{4+a^2}\right)x} + \sqrt{4+a^2}\, e^{\frac{1}{2}\left(a+\sqrt{4+a^2}\right)x}\right)\right]\right\}\right\}$$

Poslednje rešenje se može proveriti:.

```
{y''[x] == a y'[x] + y[x], y[0] == 1, y'[0] == 0} /. % // Simplify
{{True, True, True}}
```

7.9. RAZVOJ FUNKCIJE U RED

Do sada su razmatrane samo tačne operacije. Međutim u mnogim slučajevima, tačan rezultat nije potreban, već je potrebno da se nađe neka približna formula. Funkcije navedene u sledećoj tabeli generišu približne formule za funkcije.

SeriesŠexpr, šx,x$_0$,nćĆ	razvoj izraza *expr* u stepeni red po x u okolini x_0 sa najviše n članova
NormalŠredĆ	deo razvoja u red bez ostatka

Series[(1+x)^n, {x, x0, 3}]

$$(1+x0)^n+n\,(1+x0)^{-1+n}\,(x-x0)+\frac{1}{2}\,(-1+n)\,n\,(1+x0)^{-2+n}\,(x-x0)^2+\frac{1}{6}\,(-2+n)\,(-1+n)\,n\,(1+x0)^{-3+n}\,(x-x0)^3+O[x-x0]^4$$

Series[(1+x)^n, {x, 0, 3}]

$$1+n\,x+\frac{1}{2}\,(-1+n)\,n\,x^2+\frac{1}{6}\,(-2+n)\,(-1+n)\,n\,x^3+O[x]^4$$

Series[Exp[-a t] (1+Sin[2 t]), {t, 0, 4}]

$$1+(2-a)\,t+\left(-2\,a+\frac{a^2}{2}\right)t^2+\left(-\frac{4}{3}+a^2-\frac{a^3}{6}\right)t^3+\left(\frac{4\,a}{3}-\frac{a^3}{3}+\frac{a^4}{24}\right)t^4+O[t]^5$$

Series[Exp[x], {x, 0, 5}]

$$1+x+\frac{x^2}{2}+\frac{x^3}{6}+\frac{x^4}{24}+\frac{x^5}{120}+O[x]^6$$

Normal[%]

$$1+x+\frac{x^2}{2}+\frac{x^3}{6}+\frac{x^4}{24}+\frac{x^5}{120}$$

Expand[%]

$$1+x+\frac{x^2}{2}+\frac{x^3}{6}+\frac{x^4}{24}+\frac{x^5}{120}$$

Series[Exp[x], {x, 2, 4}]

$$e^2+e^2\,(x-2)+\frac{1}{2}\,e^2\,(x-2)^2+\frac{1}{6}\,e^2\,(x-2)^3+\frac{1}{24}\,e^2\,(x-2)^4+O[x-2]^5$$

7.10. GRANIČNE VREDNOSTI

Mogu se izračunavati granične vrednosti u simboličkoj formi.

LimitŠizraz, x->x$_0$Ć	granična vrednost izraza *izraz* kada x->x_0
LimitŠizraz, x->x$_0$,Direction ->1Ć	granična vrednost izraza *izraz* kada x->x_0 odozdo
LimitŠizraz, x->x$_0$,Direction->-1Ć	granična vrednost izraza *izraz* kada x->x_0 odozgo

Limit[Sin[x] / x, x→ 0]

1

Limit[Sin[x] / x^2, x→ 0]

∞

Limit[x Log[x], x→ 0]

0

```
Limit[Exp[a Sin[x]], x → 0]
1
Limit[Sin[1/x], x → 0]
Interval[{-1, 1}]
```

*Interval*Ššx_{min},x_{max}ćĆ predstavlja neodređenu vrednost koja leži u intervalu Šx_{min},x_{max}Ć.

```
Limit[1/x, x → 0, Direction → 1]
-∞
Limit[1/x, x → 0, Direction → -1]
∞
```

7.11. INTEGRACIJA

Neodređeni integral se izračunava funkcjom *Integrate*.

IntegrateŠf,xĆ	neodređeni integral $\int f(x)dx$
IntegrateŠf,šx,xmin,xmaxć Ć	neodređeni integral $\int_{x\,min}^{x\,max} f(x)dx$
IntegrateŠf,šx,xmin,xmaxć, šy,ymin,ymaxćĆ	neodređeni integral $\int_{x\,min}^{x\,max} fdx \int_{x\,min\,y\,min}^{x\,max\,y\,max} fdy$

Određeni integral se izračunava funkcjom *NIntegrate*.

NIntegrateŠf,xĆ	numerička aproksimacija integrala $\int f(x)dx$

8. LINEARNA ALGEBRA

8.1. LISTE KAO VEKTORI I MATRICE

Koncept listi u *MATHEMATICA* je generalizacija matematičkih i računarskih principa, poznatih u proceduralnim programskim jezicima. Zahvaljujući takvom pristupu mogu se izvoditi operacije sabiranja i množenja sa matricama i vektorima jednostavnim navođenjem operacije, zatim složenije operacije kao što su određivanje inverzne matrice, izračunavanje determinante, karakterističnih korena i slično. Mnoge kombinatorne funkcije čija primena u programskim jezicima zahteva izvestan trud i znanje se dobijaju pozivom odgovarajuće ugrađene funkcije. Liste omogućavaju da se objekti grupisu i da se nad njima vrše operacije.

Gotovo sve standardne funkcije, primenjene na listu, deluju na svaki element liste posebno. Ako korisnik definiše novu funkciju, ta funkcija tretira listu kao jedan objekat, pa se mora posebno naglasiti ako se funkcija primenjuje na svaki element liste pojedinačno. To se može postići funkcijom *Map*:

MapŠf,ša,b,...ćĆ - Funkcija *f* se primenjuje na svaki element liste i dobija se lista š*f*Š*a*Ć,*f*Š*b*Ć,...ć

InŠ3Ć:=1+ša,b,ccČ2

OutŠ3Ć=š1+a^2,1+b^2,1+c^2ć

InŠ4Ć:=TableŠiČj,ši,4ć,šj,ićĆ

OutŠ4Ć=šš1ć,š2,4ć,š3,9,27ć,š4,16,64,256ććć

TableŠf,ši,i_{max}ć,šj,j_{max}ćĆ - Lista vrednosti funkcije *f* kad promenljiva i uzima vrednosti od 1 do i_{max} a promenljiva j od 1 do $j_{max.}$

InŠ5Ć:=FlattenŠ%Ć - Ova funkcija generiše listu sastavljenu od pojedinačnih elemenata argumenta.

OutŠ5Ć=š1,2,4,3,9,27,4,16,64,256ć

InŠ6Ć:=PartitionŠ%,2Ć - liste i elemenata podlisti

OutŠ6Ć=šš1,1ć,š4,3ć,š9,27ć,š4,16ć,š64,256ćć **-** (oslobađa se podlisti)

PartitionŠl,nĆ - Generiše novu listu čiji su elementi podliste formirane od po *n* elemenata liste l uzetih redom od početka ka kraju

InŠ7Ć:=a=šx,xČ2,xČ3, aabbać

OutŠ7Ć=šx,x^2,x^3,aabbać

Lista a se može diferencirati. Kao rezultat nastaje lista čiji su elementi diferencijali odgovarajućih elemenata polazne liste.

InŠ8Ć:=DŠa,xĆ

OutŠ8Ć=š1,2x,3x^2,0ć

InŠ9Ć:=a/.x->5 (* Pravilo zamene *)

OutŠ9Ć=š5,25,125,aabbaćć

{1, 2, 3, 4}

{1, 2, 3, 4}

a = {x, x^2, x^3, aba}

{x, x^2, x^3, aba}

{6, 7, 8} - {3.5, 4, 2.5}

{2.5, 3, 5.5}

Exp[%]

{12.182493960703473, e^3, 244.69193226422038}

Exp[%] // N

{195339.42408031868, $5.284913114854919\times10^{8}$, $1.8550514042316475\times10^{106}$}

Exp[%%] // N

```
{12.182493960703473, 20.085536923187664, 244.69193226422038}
```

8.2. OSNOVNE OPERACIJE SA VEKTORIMA I MATRICAMA

Vektor je jednodimezionalna lista brojeva. Skalarni proizvod vektora *a* i *b* se močze izračunati pomoću izraza *a.b*, a vektorski izrazom *Cross*Š*a,b*Ć. Matrice su dvodimenzionalne liste brojeva.

VectorQŠexprĆ	*True* ako je *expr* vektor, inače *False*
MatrixQŠexprĆ	*True* ako je *expr* matrica, inače *False*
DimensionsŠexprĆ	dimenzije vektora ili matrice

```
VectorQ[{a, b, c}]
True
VectorQ[x + y]
False
Dimensions[{1, 2, 3}, {3, 2, 1}]
Dimensions::innf :
 Non-negative integer or Infinity expected at position 2 in Dimensions[{1, 2, 3}, {3, 2, 1}].
Dimensions[{1, 2, 3}, {3, 2, 1}]
Dimensions[{{1, 2, 3}, {3, 2, 1}}]
{2, 3}
```

Veliki broj ugrađenih funkcija se može primeniti posebno na svaki element liste.

```
Log[{a, b, c}]
{Log[a], Log[b], Log[c]}
Log[{1, 2, 3}]
{0, Log[2], Log[3]}
N[Log[{1, 2, 3}]]
{0., 0.6931471805599453, 1.0986122886681098}
{a, b} + {1, 2}
{1 + a, 2 + b}
1 + {a, b}
{1 + a, 1 + b}
{a, b} + c
{a + c, b + c}
k {a, b}
{a k, b k}
```

```
{a, b} + p
{a + p, b + p}
%/. p→ {c, d}
{{a + c, a + d}, {b + c, b + d}}
```

c v , c m	množenje svih elemenata vektora *v* ili matrice *m* skalarom *c*
v.v, v.m, m.m	množenje vektora ili matrica
MatrixPowerŠm, nĆ	*n*-ti stepen matrice *m*
DetŠmĆ	determinanta matrice *m*
TransposeŠmĆ	transponovana matrica matrice *m*
InverseŠmĆ	inverzna matrica matrice *m*
MinorsŠm, kĆ	minori $k\times k$ matrice *m*
TrŠmĆ	trag matrice *m*

```
Det[{{a, b}, {c, d}}]
-b c + a d
m = Array[a, {3, 3}]
{{a[1, 1], a[1, 2], a[1, 3]}, {a[2, 1], a[2, 2], a[2, 3]}, {a[3, 1], a[3, 2], a[3, 3]}}
Det[m]
-a[1, 3] a[2, 2] a[3, 1] + a[1, 2] a[2, 3] a[3, 1] + a[1, 3] a[2, 1] a[3, 2] -
 a[1, 1] a[2, 3] a[3, 2] - a[1, 2] a[2, 1] a[3, 3] + a[1, 1] a[2, 2] a[3, 3]
Minors[m, 2]
{{-a[1, 2] a[2, 1] + a[1, 1] a[2, 2],
  -a[1, 3] a[2, 1] + a[1, 1] a[2, 3], -a[1, 3] a[2, 2] + a[1, 2] a[2, 3]},
 {-a[1, 2] a[3, 1] + a[1, 1] a[3, 2], -a[1, 3] a[3, 1] + a[1, 1] a[3, 3],
  -a[1, 3] a[3, 2] + a[1, 2] a[3, 3]}, {-a[2, 2] a[3, 1] + a[2, 1] a[3, 2],
  -a[2, 3] a[3, 1] + a[2, 1] a[3, 3], -a[2, 3] a[3, 2] + a[2, 2] a[3, 3]}}
Sum[m[[i, i]], {i, 2}]
a[1, 1] + a[2, 2]
```

8.3. ČLANSTVO U LISTI

Članstvo elementa u listi se ispituje na više različitih načina.

PositionŠlista, exprĆ	lista pozicija u kojima se izraz *expr* nalazi u listi *lista*
CountŠlista, exprĆ	broj pojavljivanja *expr* u *lista*
MemberQŠlista,	testira da li je *expr* element liste *lista*

exprĆ	
FreeQŠlista, exprĆ	*True* ako se *expr* ne nalazi u listi *lista*

```
Position[{a, b, c, d}, a]
{{1}}
Position[{a, b, c, a, d}, a]
{{1}, {4}}
Count[{a, b, c, a, d}, a]
2
MemberQ[{a, b, c, a, d}, a]
True
m = IdentityMatrix[3]
{{1, 0, 0}, {0, 1, 0}, {0, 0, 1}}
FreeQ[m, 0]
False
Position[m, 0]
{{1, 2}, {1, 3}, {2, 1}, {2, 3}, {3, 1}, {3, 2}}
```

8.4. REŠAVANJE LINEARNIH SISTEMA

Sistem linearnih jednačina se zadaje eksplicitno u formi *ls*==ds, ili se posmatra se u matričnoj formi *m.x=b*, gde je *m* matrica, a *x* je vektor promenljivih.

SolveŠls==ds, xĆ	rešavanje jednačine *ls==ds* po *x*
LinearSolveŠm, bĆ	određivanje vektora x kojim se rešava matrična jednačina *m.x=b*
NullSpaceŠmĆ	lista različitih vektora čije linearne kombinacije zadovoljavaju matričnu jednačinu *m.x=0*
RowReduceŠmĆ	uprošćena forma matrice *m,* dobijena linearnim transformacijama vrsti

```
m = {{1, 5}, {2, 14}}
{{1, 5}, {2, 14}}
m.{x, y} == {a, b}
{x + 5 y, 2 x + 14 y} == {a, b}
Solve[%, {x, y}]
```

$$\left\{\left\{x \to \frac{1}{4}(14a - 5b),\ y \to \frac{1}{4}(-2a + b)\right\}\right\}$$

```
Det[m]
```

```
4
LinearSolve[m, {a, b}]
```

$$\left\{\frac{1}{4}(14a-5b),\ \frac{1}{4}(-2a+b)\right\}$$

```
NullSpace[m]
{}
```

8.5. SOPSTVENI VEKTORI I SOPSTVENE VREDNOSTI

MATHEMATICA poseduje sledeće standardne funkcije za rešavanje problema sopstvenih vrednosti matrica:

EigenvaluesŠmĆ	lista sopstvenih vrednosti matrice *m*
EigenvectorsŠmĆ	lista sopstvenih vektora matrice *m*
EigensystemŠmĆ	lista elemenata oblika ššsopstvene_vrednostić, šsopstveni_vektor*ićć*
EigenvaluesŠNŠm, kĆĆ	numeričke vrednosti sopstvenih vrednosti sa *k* decimala preciznosti

```
m = {{2.3, 3.4}, {4.5, 5.6}}
{{2.3, 3.4}, {4.5, 5.6}}
Eigenvalues[m]
{8.195291509425472, -0.2952915094254718}
Eigenvectors[m]
{{-0.4995979013302471, -0.8662574311291146}, {-0.7948887021812828, 0.6067552646203869}}
Eigensystem[m]
{{8.195291509425472, -0.2952915094254718},
 {{-0.4995979013302471, -0.8662574311291146}, {-0.7948887021812828, 0.6067552646203869}}}
```

8.6. KONSTRUKCIJA TABELA VREDNOSTI

Tabele vrednosti mogu biti generisane kao vektori, matrice ili indeksirani objekti.

TableŠf, šimaxćĆ	lista vrednosti š*f*Š1Ć, ..., *f*Š*imax*Ć*ć*
TableŠf, ši, imaxćĆ	lista vrednosti funkcije *f* za *i* od 1 do *imax*
TableŠf, ši, imin, imaxćĆ	lista vrednosti funkcije *f* za *i* od *imin* do *imax*
TableŠf, ši, imin, imax, dićĆ	kao prethodno, sa korakom *di*
TableŠf, ši, imin, imaxć, šj,jmin,jmaxć,...Ć	generiše višedimenzionalnu tabelu

TableFormŠlistaĆ	prikazuje listu u tabelarnoj formi
TableŠ0, šmć, šnćĆ	nula matrica dimenzije $m\times n$
TableŠIfŠi>=j, 1, 0Ć, ši, mć, šj, nćĆ	donja trougaona matrica
TableŠIfŠi<=j, 1, 0Ć, ši, mć, šj, nćĆ	gornja trougaona matrica
ArrayŠa, nĆ	vektor dužine *n* sa elementima š*a*Š1Ć,...,*a*Š*n*Ćć
ArrayŠa, šm, nćĆ	$m\times n$ matrica čiji je (*i*,*j*) element jednak *a*Š*i*, *j*Ć
RangeŠnĆ	kreira listu š1, 2 ,..., *n*ć
RangeŠn_1, n_2Ć	kreira listu šn_1, n_1+1, ..., n_2ć
RangeŠn_1, n_2, dĆ	kreira listu šn_1, n_1+*d*, ..., n_2ć
ColumnFormŠlistaĆ	prikazuje elemente liste u koloni
IdentityMatrixŠnĆ	jedinična $n\times n$ matrica
DiagonalMatrixŠlistĆ	generiše dijagonalnu matricu sa elementima iz liste *list* na glavnoj dijagonali

```
m = Table[i - j, {i, 2}, {j, 2}]
{{0, -1}, {1, 0}}
m[[1]]
{0, -1}
m[[1, 2]]
-1
TableForm[m]
0     -1
1     0
Table[x^i + 2 i, {i, 5}]
{2 + x, 4 + x^2, 6 + x^3, 8 + x^4, 10 + x^5}
Table[If[i ≤ j, 1, 0], {i, 2}, {j, 2}]
{{1, 1}, {0, 1}}
Array[p, {3, 2}]
{{p[1, 1], p[1, 2]}, {p[2, 1], p[2, 2]}, {p[3, 1], p[3, 2]}}
DiagonalMatrix[{a, b, c}]
{{a, 0, 0}, {0, b, 0}, {0, 0, c}}
```

9. FUNKCIONALNE OPERACIJE

9.1. IMENA FUNKCIJA KAO IZRAZI

U izrazima oblika *f*Š*x,y,...*Ć "ime funkcije" *f* je izraz, i mogu se nad njim izvršavati različite operacije. Ovakva sposobnost da se imena funkcija tretiraju kao i ostali tipovi izraza je značajna posledica mogućnosti *MATHEMATICA* u simboličkoj obradi podataka. Postoje i funkcije koje predstavljaju funkcionalne operacije, i koje mogu da manipulišu ne samo sa običnim tipovima podataka, već i sa funkcijama.

f[x] + f[1 - x] /. f → g

(* Mogu se zameniti imena funkcija koristeći pravila transformacija *)

```
g[1 - x] + g[x]
```

p1 = p2; p1[x, y] (* Dodeljivanje imena funkciji *)

```
p2[x, y]
```

pf[f_, x_] := f[x] + f[1 - x]

(*Definicija funkcije koja koristi ime funkcije kao argument*)ć

pf[Log, q]

(*Poziv koristi *Log* kao stvarni parametar.*)

```
Log[1 - q] + Log[q]
```

Standardna funkcija *InverseFunction* koristi drugu funkciju kao argument, a rezultat je inverzna funkcija argumenta.

InverseFunction[ArcSin]

```
Sin
```

%x

```
Sin x
```

%[x] (*Funkcija kao rezultat se moze primenjivati *)

```
(Sin x)[x]
```

InverseFunction[f][x]

$f^{(-1)}[x]$

9.2. REPETITIVNO KORIŠĆENJE FUNKCIJA

Funkcija kao argument može biti primenjena više puta u jednom pozivu neke druge funkcije.

NestŠf,x,nĆ	primeniti funkciju *f* uzastopno *n* puta na *x*; rezultat je *f*Š*f*Š...*f*Š*x*Ć...ĆĆ
NestListŠf,x,nĆ	generisati listu š*x*,*f*Š*x*Ć,*f*Š*f*Š*x*ĆĆ, ,*f*Š*f*Š... *f*Š*x*Ć...ĆĆć.
FixedPointŠf,xĆ	primenjivati funkciju *f* sve dok se rezultat menja; rezultat je poslednja primena funkcije
FixedPointListŠf,x Ć	generisati listu š*x*,*f*Š*x*Ć,*f*Š*f*Š*x*ĆĆ,...ć; zaustaviti se kada se rezultat više ne menja
FixedPointŠf,x, SameTest->compĆ	zaustaviti se kada funkcija comp primenjena na dve sukscesivne transformacije daje *True*

```
Nest[f, x, 4]
f[f[f[f[x]]]]
NestList[f, x, 4]
{x, f[x], f[f[x]], f[f[f[x]]], f[f[f[f[x]]]]}
recip[x_] := 1/(1+x)
Nest[recip, x, 3]          (* Iterativno korišćenje funkcije Recip pomoću Nest *)
```

$$\frac{1}{1+\frac{1}{1+\frac{1}{1+x}}}$$

Funkcije *Nest* i *NestList* su pogodne za iterativnu primenu funkcija.

```
newtonŠx_Ć:=NŠ1/2 (x+3/x)Ć
General::spell1 :
 Possible spelling error: new symbol name "newton" is similar to existing symbol "Newton".
NestList[newton, 1.0, 5]
{1., 2., 1.75, 1.7321428571428572, 1.7320508100147274, 1.7320508075688774}
NestList[newton, 1.0]
NestList::argrx : NestList called with 2 arguments; 3 arguments are expected.
NestList[newton, 1.]
FixedPoint[newton, 1.0]
1.7320508075688772
FixedPoint[newton, 5.0]
1.7320508075688774
FixedPointList[newton, 1.0]
{1., 2., 1.75, 1.7321428571428572,
 1.7320508100147274, 1.7320508075688774, 1.7320508075688772}
```

Funkcionalne operacije *Nest* i *NestList* koriste funkciju od jednog argumenta, i primenjuju je repetitivno. U svakom koraku se koristi rezultat iz prethodnog koraka.

Korisno je da se ova mogućnost generalizuje na dvoargumentne funkcije.

FoldListŠf,x,ša,b,ć Ć	generiše listu š*x*,*f*Š*x*,*a*Ć,*f*Š*f*Š*x*,*a*Ć,*b*Ć,...ć
FoldŠf,x,ša,b,ćĆ	poslednji element liste generisan sa *FoldList*Š*f*,*x*,š*a*,*b*,ćĆ

```
FoldList[f, x, {a, b, c}]
{x, f[x, a], f[f[x, a], b], f[f[f[x, a], b], c]}
Fold[f, x, {a, b, c}]
f[f[f[x, a], b], c]
FoldList[Plus, 0, {a, b, c}]
{0, a, a + b, a + b + c}
FoldList[Plus, x, {a, b, c}]
{x, a + x, a + b + x, a + b + c + x}
```

Primer. Koristeći funkcije definisane na sledeći način
nextdigitŠa_,b_Ć:=10a+b
brojŠlistacifara_Ć:=FoldŠnextdigit,0,listacifaraĆ
dobija se
brojŠš1,3,5,2ćĆ
1352

9.3. PRIMENA FUNKCIJA NA LISTE I OSTALE IZRAZE

U izrazima oblika *f*Šš*a*,*b*,*c*,ćĆ funkcija f koristi listu kao svoj argument. Često puta je potrebno da se funkcija primeni direktno na elemente liste, a ne na celu listu. To se u *MATHEMATICA* može pisati koristeći *Apply*.

ApplyŠf,ša,b,...ćĆ	rezultat je *f*Š*a*,*b*,...Ć
ApplyŠf,exprĆ ili f ŽŽ expr	primeniti *f* na top nivo izraza *expr*
ApplyŠf,expr,levĆ	primeniti *f* na specificirane nivoe u izrazu *expr*

```
Apply[f, {a, b, c}]
f[a, b, c]
Apply[Plus, {a, b, c}]
a + b + c
sval[l_] := Apply[Plus, l] / Length[l]
sval[{1, 2, 3}]
2
Apply[List, a + b + c]
{a, b, c}
```

```
m = {{a, b, c}, {b, c, d}}
{{a, b, c}, {b, c, d}}
Apply[f, m]
f[{a, b, c}, {b, c, d}]
Apply[f, m, {1}]
{f[a, b, c], f[b, c, d]}
Apply[f, m, {0, 1}]
f[f[a, b, c], f[b, c, d]]
Apply[f, m, {1, 2}]
{f[a, b, c], f[b, c, d]}
```

9.5. PRIMENA FUNKCIJA NA DELOVE IZRAZA

Često puta je potrebno da se funkcija primeni posebno na svaki element liste. Rezultati te primene mogu biti grupisani na različite načine. Najčešće se rezultati primene grupišu u listu. To se može učiniti funkcijom *Map*:

MapŠf, ša,b,...ćĆ =šfŠaĆ, fŠbĆ, ...ć

```
Map[f, {a, b, c}]
{f[a], f[b], f[c]}
```

Funkcija *Map* koristi se često u kombinaciji sa funkcijom *Head*:

MapŠf ,headŠa,b, ...ĆĆ=headŠfŠaĆ ,fŠbĆ, ...Ć

```
Map[f, Head[a, b, c]]
Head::argx : Head called with 3 arguments; 1 argument is expected.
Head::argx : Head called with 3 arguments; 1 argument is expected.
Head[f[a], f[b], f[c]]
take2[l_] := Take[l, 2]
Map[take2, {{1, 3, 4}, {3, 4, 5}, {4, 4, 5, 6}}]
{{1, 3}, {3, 4}, {4, 4}}
Map[f, a + b + c]
f[a] + f[b] + f[c]
Map[f, Plus[a, b, c]]
f[a] + f[b] + f[c]
Map[Sqrt, q[x^2, x^3]]
```

$q\left[\sqrt{x^2}, \sqrt{x^3}\right]$

```
m = {{a, b, c}, {b, c, d}}
{{a, b, c}, {b, c, d}}
Map[f, m]
{f[{a, b, c}], f[{b, c, d}]}
```

U primeni funkcije *Map* može se koristiti specifikacija nivoa u kojima se zahteva primena funkcionalnog argumenta. Na taj način se dobijaju različiti oblici pozivanja funkcije *Map* ili neke njene modifikacije.

MapŠf,exprĆ ili f/Ž expr	primeniti *f* na delove u prvom nivou izraza *expr*
MapAllŠf,exprĆ ili f//Žexpr	primeniti *f* na sve delove izraza *expr*
MapŠf, expr,levĆ	primeniti *f* na delove izraza *expr* koji su specificirani sa *lev*

```
MapAll[f, m]
f[{f[{f[a], f[b], f[c]}], f[{f[b], f[c], f[d]}]}]
Map[f, m, {1, 2}]
{f[{f[a], f[b], f[c]}], f[{f[b], f[c], f[d]}]}
Map[f, m, {1, 2}, {2, 3}]
Map::nonopt : Options expected (instead of {2, 3}) beyond position 3 in
   Map[f, {{a, b, c}, {b, c, d}}, {1, 2}, {2, 3}]. An option must be a rule or a list of rules.
Map[f, {{a, b, c}, {b, c, d}}, {1, 2}, {2, 3}]
Map[f, m]
{f[{a, b, c}], f[{b, c, d}]}
Map[f, m, Heads→True]
f[List][f[{a, b, c}], f[{b, c, d}]]
```

Funkcija *MapAt* se definiše na sledeći način:

MapAtŠf, expr, šši$_1$,*j*$_1$,..*ć*,š*i*$_2$,*j*$_2$,..*ć*,..*ć*Ć = š*f*Š*expr*ŠŠ*i*$_1$,j$_1$,...ĆĆĆ,
*f*Š*expr*ŠŠ*i*$_2$,*j*$_2$,...ĆĆĆ, ...*ć*

```
MapAt[f, m, {{1, 2}, {2, 3}}]
{{a, f[b], c}, {b, c, f[d]}}
MapAt[f, m, {{1, 2}, {{2}, {3}}}]
MapAt::psl :
 Position specification {{2}, {3}} in MapAt[f, {{a, b, c}, {b, c, d}}, {{1, 2}, {{2}, {3}}}]
   is not an integer or a list of integers.
MapAt[f, {{a, b, c}, {b, c, d}}, {{1, 2}, {{2}, {3}}}]
MapAt[f, {{a, b, c}, {b, c, d}}, {{1, 2}, {{2}, {3}}}]
MapAt::psl :
 Position specification {{2}, {3}} in MapAt[f, {{a, b, c}, {b, c, d}}, {{1, 2}, {{2}, {3}}}]
   is not an integer or a list of integers.
MapAt[f, {{a, b, c}, {b, c, d}}, {{1, 2}, {{2}, {3}}}]
MapAt[f, {a, b, c, d}, {{2}, {3}}]
{a, f[b], f[c], d}
t = 1+ (3 + x)^2 / x
```

$1+\frac{(3+x)^2}{x}$

FullForm[**t**]

Plus[1, Times[Power[x, -1], Power[Plus[3, x], 2]]]

MapAt[**f**, **t**, {{**2**, **1**, **1**}, {**2**, **2**}}]

$1+\frac{f[(3+x)^2]}{f[x]}$

MapIndexedŠf,exprĆ	primeniti *f* na elemente izraza, pri čemu je drugi element funkcije *f* jednak poziciji svakog od tih elemenata
MapIndexedŠf,expr,lev Ć	primeniti *f* na specificirane nivoe, pri čemu su sukcesivni argumenti funkcije *f* dati listama indeksa za svaki deo

MapIndexed[**f**, {**a**, **b**, **c**}]

{f[a, {1}], f[b, {2}], f[c, {3}]}

MapIndexed[**f**, {{**a**, **b**}, {**c**, **d**}}, **2**]

{f[{f[a, {1, 1}], f[b, {1, 2}]}, {1}], f[{f[c, {2, 1}], f[d, {2, 2}]}, {2}]}

Funkcija *Map* dozvoljava da se funkcija od jednog argumenta primeni na delove izraza. Međutim, može se primeniti više argumenta funkcija više puta na odgovarajuće delove nekoliko različitih izraza.

MapThreadŠf, šexpr$_1$,expr$_2$,...ćĆ	primeniti *f* na odgovarajuće elemente u svakom izrazu *expr*$_1$, ...
MapThreadŠf, šexpr$_1$,expr$_2$,...ć, levelĆ	primeniti *f* na delove izraza *expr*$_1$,... u specificiranim nivoima

MapThread[**f**, {{**a**, **b**, **c**}, {**ap**, **bp**, **cp**}}]

{f[a, ap], f[b, bp], f[c, cp]}

MapThread[**f**, {{**a**, **b**, **c**}, {**ap**, **bp**, **cp**}, {**app**, **bpp**, **cpp**}}]

{f[a, ap, app], f[b, bp, bpp], f[c, cp, cpp]}

MapThread[**f**, {**a** + **b** + **c**, **d** + **e** + **f**}]

```
MapThread::mptd : Object a + b + c at position {2, 1} in
   MapThread[f, {a + b + c, d + e + f}] has only 0 of required 1 dimensions.
```

MapThread[f, {a + b + c, d + e + f}]

ScanŠf,exprĆ	primeniti *f* na svaki element iz *expr*, bez generisanja novog rezultata
ScanŠf,expr, levĆ	primeniti *f* na delove izraza expr koji su specificirani sa *lev*

```
Scan[Print, {a, b, c}]
a
b
c
Scan[Print, a + x^2, Infinity]
a
x
2
x^2
```

9.6. ČISTE FUNKCIJE

Kada se koriste funkcionalne operacije kakve su *Nest* i *Map*, uvek se mora specificirati funkcija koja se primenjuje. U gore navedenim primerima za specifikaciju funkcije ja korišćeno "ime" funkcije. Čiste funkcije dozvoljavaju da se definišu funkcije koje mogu da se primene kao argumenti, bez davanja eksplicitnog imena funkciji. Postoji nekoliko zapisa čistih funkcija:

FunctionŠx,bodyĆ	čista funkcija u kojoj se *x* zamenjuje zadatim argumentom
FunctionŠšx_1,...,x_nć,body Ć	čista funkcija u kojoj se koristi nekoliko argumenata
body&	čista funkcija u kojoj su argumenti specificirani sa # ili # 1, # 2, …

```
h[x_] := f[x] + g[x]      (* Definicija funkcije h *)
Map[h, {a, b, c}]      (* Koristi se ime definisane funkcije u Map *)
{f[a] + g[a], f[b] + g[b], f[c] + g[c]}
Map[f[#] + g[#] &, {a, b, c}]
(* Isti se rezultat može dobiti pomocu cistih funkcija*)
{f[a] + g[a], f[b] + g[b], f[c] + g[c]}
Function[x, x^2]
(* Čista funkcija koja postavlja operaciju kvadriranja*)
Function[x, x^2]
%[n]
n^2
Map[Function[x, x^2], a + b + c]
a^2 + b^2 + c^2
Nest[Function[g, 1 / (1 + g)], x, 3]
```

$$\frac{1}{1+\frac{1}{1+\frac{1}{1+x}}}$$

9.7. IZGRADNJA LISTI IZ FUNKCIJA

Rezultati uzastopne primene funkcija mogu biti zapamćeni kao indeksirani objekti ili kao specifične liste.

ArrayŠf,nĆ	generiše listu š*f*Š1Ć,...,*f*Š*n*Ćć
ArrayŠf,šn_1,n_2,...ćĆ	generiše $n_1\times n_2\times$... ugneždenu listu, čiji je element ŠŠi_1,i_2, ,...ĆĆ jednak *f*ŠŠi_1,i_2,...,ĆĆ, i_1=*1*,...,n_1;..., i_2=*1*,...,n_2
NestListŠf,x,nĆ	generiše listu oblika š*x*,*f*Š*x*Ć,*f*Š*f*Š*x*ĆĆ,...ć dužine *n*
FoldlistŠf,x,ša,b,...ćĆ	generiše listu š*x*,*f*Š*x*,*a*Ć,*f*Š*f*Š*x*,*a*Ć,*b*Ć,...ć
ComposeŠšf_1,f_2,...ć,xĆ	generiše listu š*x*,f_1Š*x*Ć,f_2Šf_1Š*x*ĆĆ,...ć

```
Array[p, 5]
{p[1], p[2], p[3], p[4], p[5]}
Table[p[i], {i, 5}]
{p[1], p[2], p[3], p[4], p[5]}
m =.
Array[m, {2, 3}]
{{m[1, 1], m[1, 2], m[1, 3]}, {m[2, 1], m[2, 2], m[2, 3]}}
```

9.8. FUNKCIJE KAO OPERATORI

Funkcija *f*Š*x*Ć se može posmatrati kao primena operatora *f* na izraz *x*. Takođe,izraz oblika *f*Š*g*ŠxĆĆ se može posmatrati kao primena kompozicije operatora *f* i *g* na izraz *x*.

CompositionŠf,g,...Ć	kompozicija funkcija *f*,*g*,...
InverseFunctionŠfĆ	inverzna funkcija za *f*
Identity	identična funkcija.

```
Composition[f, g, h]
Composition[f, g, h]
InverseFunction[%]
Composition[h^(-1), g^(-1), f^(-1)]
Function[x, x^2][a + b]
```

```
(a + b)^2
```

9.9. STRUKTURNE OPERACIJE

MATHEMATICA sadrži neke mocne funkcije za izmenu strukture izraza.

SortŠexprĆ	sortirati elemente liste ili drugog izraza u standardni poredak
SortŠexpr,predĆ	sortirati izraz *expr* koristeći funkciju *pred*
OrderedQŠexprĆ	rezultat je *True* ako su elementi u *expr* u standardnom poretku, inače *False*
OrderŠ$\mathbf{expr_1}$,$\mathbf{expr_2}$Ć	rezultat je 1 ako je $expr_1$ pre $expr_2$ u standardnom poretku, inače -1

Funkcija *SortŠexpr*Ć se koristi ne samo za sortiranje listi, već i proizvoljnih izraza, sa proizvoljnom glavom.

```
Sort[f[c, a, b]]
f[a, b, c]
f[List[c, a, b]]
f[{c, a, b}]
Sort[{5.1, 8.2}, (#2 < #1) &]
{8.2, 5.1}
```

Funkcija *Distribute* dozvoljava da se u izrazima primene svojstva distributivnosti i linearnosti.

DistributeŠfŠa+b+...,c+d+...,... ĆĆ	Distribucija *f* u odnosu na +, a rezultat je *f*Š*a,,c,...*Ć+ *f*Š*b,d,...*Ć+ ···
DistributeŠfŠargsĆ,gĆ	Distribucija *f* u odnosu na argumente koji imaju glavu *g*
DistributeŠexpr,g,fĆ	Distribucija *f* u odnosu na izraze sa glavom *g*
DistributeŠexpr,g,f,gp,fpĆ	Distribucija *f* u odnosu na g, zamenjujući ih sa *fp* i *gp* respektivno

```
Distribute[f[a + b]]
f[a] + f[b]
Distribute[f[a + b, c + d]]
f[a, c] + f[a, d] + f[b, c] + f[b, d]
Distribute[f[{a, b}, {c, d}], List]
```

```
{f[a, c], f[a, d], f[b, c], f[b, d]}
Distribute[{a, b}, {c, d}]
{c, d}[{a, b}]
Distribute[f[{a, b}, {c, d}]]
f[{a, b}, {c, d}]
Distribute[f[{a, b}, {c, d}], List, f]
{f[a, c], f[a, d], f[b, c], f[b, d]}
Distribute[f[{a, b}, {c, d}], List, f, gp, fp]
gp[fp[a, c], fp[a, d], fp[b, c], fp[b, d]]
```

ThreadŠfŠša$_1$,a$_2$,...ć,šb$_1$,b$_2$,...ć,... ĆĆ	rezultat je lista š*f*Š*a*$_1$,*b*$_1$, ...Ć,*f*Š*a*$_2$,*b*$_2$,, ...Ćć
ThreadŠfŠargsĆ,gĆ	primenjuje *Thread* na objektima u *args* koji imaju glavu *g*

```
Thread[f[{a1, a2}, {b1, b2}]]
{f[a1, b1], f[a2, b2]}
Thread[f[{a1, a2}, {b1, b2}, c, d]]
{f[a1, b1, c, d], f[a2, b2, c, d]}
Log[x == y]
Log[x == y]
Thread[%, Equal]
Log[x] == Log[y]
```

9.10. ŠABLONI

Šabloni (*patterns*) se koriste za reprezentaciju klasa izraza. Primer šablona je izraz *f*Š_Ć*,* što označava klasu izraza oblika *f*Š*bilocega*Ć. Na taj način, mnoge operacije u *MATHEMATICA* mogu da se izvrše ne samo na pojedinačnim izrazima, već na čitavim klasama izraza.

fŠn_Ć	*f* sa proizvoljnim argumentom, sa imenom *n*
fŠn_,m_Ć	*f* sa dva argumenta, koji su imenovani *n* i *m*
xČn_	*x* sa eksponentom proizvoljnog tipa, koji je imenovan sa *n*
x_Č n_	proizvoljan izraz i proizvoljan eksponent
a_+b_	suma dva proizvoljna izraza
ša_,b_ć	lista dva proizvoljna izraza
fŠn_,n_Ć	*f* sa dva identična argumenta

```
f[{a, b}] + f[c] /. f[{x_, y_}] → p[x + y]
f[c] + p[a + b]
```

```
g[list_] := Part[list, 1]^Part[list, 2]
g[{x, y}]
x^y
{1, x, x^2, x^3} /. x^n_ -> r[n]
{1, x, r[2], r[3]}
{1, x, x^2, x^3} /. x^n -> r[n]
{1, x, x^2, x^3}
{a / b, 1 / b^2} /. b^n_→ d[n]
{a d[-1], d[-2]}
```

9.11. NALAŽENJE IZRAZA KOJI VRŠE SLAGANJE ŠABLONA

Slaganje šablona je pojam koji odgovara čuvenoj unifikaciji u programskom jeziku PROLOG.

CasesŠlist,formĆ	elementi iz *list* koji se slažu sa *form*
CountŠlist,formĆ	broj elemenata u *list* koji se slažu sa form
PositionŠlist,form;šlćĆ	pozicije elemenata u listi *list* koji se slažu sa *form*
SelectŠlist,testĆ	elementi iz *list* za koje test daje test

```
Cases[{3, 4, x, x^2, x^3}, x^_]
{x^2, x^3}
Count[{3, 4, x, x^2, x^3}, x^_]
2
```

CasesŠexpr,lhs->rhsĆ	elementi u *expr* koji se slažu sa *lhs*; rezultat je lista rezultata uz primenu pravila transformacije
CasesŠexpr,lhs->rhs,levĆ	testira delove iz *expr* u nivoima specificiranim sa *lev*
CountŠexpr,form,levĆ	broj delova koji se slažu sa *form* u nivoima specificiranim sa *lev*
PositionŠexpr,form,levĆ	daje pozicije delova koji se slažu sa form u nivoima specificiranim sa *lev*

```
Cases[{3, 4, x, x^2, x^3}, x^n_→ n]
{2, 3}
Cases[{3, 4, x, x^2, x^3}, _Integer, Infinity]
{3, 4, 2, 3}
```

```
Cases[{3, 4, x, x^2, x^3}, _Integer -> Infinity]
{∞, ∞}
```

DeleteCasesŠexpr,formĆ	briše elemente iz *expr* koji se slažu sa *form*
DeleteCasesŠexpr,form,levĆ	briše delove iz *expr* koji se slažu sa form a nalaze se u nivoima specificiranim sa *lev*

```
DeleteCases[{3, 4, x, x^2, x^3}, x^n_]
{3, 4, x}
(* Brisanje svih celih brojeva na bilo kom nivou *)
DeleteCases[{3, 4, x, x^2, x^3}, _Integer, Infinity]
{x, x, x}
DeleteCases[{3, 4, x, x + 2, x + 3}, _Integer, Infinity]
{x, x, x}
```

9.12. IMENOVANJE DELOVA ŠABLONA

Imena za delove šablona često se koriste u pravilima transformacije. Objekat oblika *x_* stoji umesto proizvoljnog izraza, pri čemu se izraz imenuje sa *x*. Ovo ime se može koristiti na desnoj strani pravila transformacije. Izraz *f*Š*x_*,*x_*Ć označava izraz sa glavom *f* u kome su argumenti isti. Izraz *f*Š_,_Ć označava izraz sa glavom *f* u kome su argumenti proizvoljni.

_	proizvoljni izraz
x_	proizvoljni izraz, imenovan *x*
x:pattern	izraz imenovan *x*, koji se slaže sa obrascem *pattern*.

```
{f[a, a], f[a, b]} /. f[x_, x_] → p[x]
{p[a], f[a, b]}
f[a^b] /. f[x : _^_] → p[x, n]
p[a^b, n]
f[a^b] /. f[x : _^_] → p[x]
p[a^b]
f[a^b] /. f[x : _^n_] → p[x, n]
p[a^b, b]
{a, 4, 5, b} /. x_Integer → p[x]
{a, p[4], p[5], b}
```

9.13. SPECIFICIRANJE TIPOVA IZRAZA U šABLONIMA

Specifikacija tipova izraza prikazana je u sledećoj tabeli:

x_ h	izraz čija je glava *h*
x_ Integer	ceo broj
x_ Real	realni broj
x_ Complex	kompleksni broj
x_ List	lista
x_ Symbol	simbol

```
gama[4] + gama[x]
6 + gama[x]
gama[4.]
gama[4.]
d[x_^n_Integer] := n x^(n - 1)
d[x^4] + d[(a + b)^3] + d[x^(1/2)]
```

$3\,(a+b)^2 + 4\,x^3 + d[\sqrt{x}]$

9.14. POSTOJANJE OGRANIČENJA NA ŠABLONE

MATHEMATICA obezbeuje generalni mehanizam za specificiranje uslova u šablonima. Taj mehanizam je oblika /;condition, i stavlja se na kraj šablona. Njime se označava da se slaganje šablona primenjuje samo kada je vrednost izraza condition jednaka *True*.

pattern/; condition	šablon *pattern* koji se slaže samo kada je uslov *condition* ispunjen
lhs->rhs/; condition	pravilo koje se primenjuje samo kada je uslov *condition* ispunjen
lhs:=rhs/;condition	definicija koja se primenjuje samo kada je uslov *condition* ispunjen

```
fac[n_ /; n > 0] := n!
fac[6] + fac[-4]
720 + fac[-4]
Cases[{3, -4, 5, -2}, x_ /; x < 0]
{-4, -2}
fac[n_] := n! /; n > 0
fac[6] + fac[-4]
720 + fac[-4]
v[x_, 1 - x_] := p[x]
```

```
v[a^2, 1-a^2]
p[a^2]
v[4, -3]
v[4, -3]
w[x_, y_] := p[x] /; y == 1-x
w[4, -3]
p[4]
```

Korisničke funkcije se mogu dodati u globalno okruženje. Na primer, može se dodati funkcija koja kvadrira svoj argument:

fŠx_ Ć:=xČ2	definicija funkcije $f(x)=x^2$

Definicija koja je vezana za simbol *f* može se prikazati pomoću izraza
?f
Sve definicije za f mogu se obrisati pomoću *Clear*Š*f*Ć.

```
f[x_] := x^2
f[a+1]
(1+a)^2
f[4]
16
f[3x+x^2]
(3x+x^2)^2
Expand[f[x+1+y]]
1+2x+x^2+2y+2xy+y^2
?f
Globalžf
f[x_] := x^2
```

Imena koja se koriste za funkcije u *MATHEMATICA* su samo simboli. Zbog toga bi trebalo izbegavati korišćenje imena koja počinju velikim slovima, da bi se izbegle eventualne konfuzije sa ugrađenim funkcijama u *MATHEMATICA*.

```
h[x_, y_] := (x-y)^2/y
```

(* Korisničke funkcije mogu imati proizvoljan broj argumenata *)

```
2+h[x, 3.5]
```

(* Funkcija h se koristi kao i svaka ugrađena funkcija *)

```
2+0.2857142857142857 (-3.5+x)^2
```

Funkcije koje se definišu u *MATHEMATICA* jesu u suštini procedure koje izvršavaju zadate komande. U korisničkim funkcijama se može zadati više koraka, odvojenih simbolima ; (semicolon). Rezultat koji se dobija iz cele funkcije jeste vrednost njenog poslednjeg izraza.

```
exprod[n_] := Expand[Product[x + i, {i, 1, n}]]
exprod[5]
120 + 274 x + 225 x^2 + 85 x^3 + 15 x^4 + x^5
cex[n_, i_] := (t = exprod[n]; Coefficient[t, x^i])
cex[5, 3]
85
```

9.15. PRAVILA TRANSFORMACIJE ZA FUNKCIJE

Ranije je pokazano kako se mogu koristiti pravila transformacije oblika *x⟶v* za zamenu simbola vrednostima. Međutim, pojam pravila transformacije u *MATHEMATICA* je mnogo opštiji. Pravila transormacije se mogu definisati ne samo za simbole, već za proizvoljne izraze.

```
Clear[f]
1 + f[x] + f[y] /. x → 3
(* Zameniti x sa 3 *)
1 + f[3] + f[y]
1 + f[x] + f[y] /. f[x] → p
 (* Pravilo transformacije se može  koristiti za fŠxĆ. To nema uticaj na fŠyĆ *)
1 + p + f[y]
1 + f[x] + f[y] /. f[t_] → t^2
(* fŠt_Ć je šablon koji se  zamenjuje *)
1 + x^2 + y^2
f[a, b] + f[c, d] /. f[x_, y_] → f[x] + f[y]
f[a] + f[b] + f[c] + f[d]
1 + x^2 + x^4 /. x^p_ → f[p]
(* Pravilo transformacije za xČp_ *)
1 + f[2] + f[4]
```

U izrazu oblika *fŠxĆ,* ime funkcije *f* je jedan izraz, i može se tretirati kao i svaki drugi izraz.

```
f[x] + f[1 - x] /. f → g
g[1 - x] + g[x]
(* Imena funkcija se mogu zameniti  koristeci pravila transformacija *)
p1 = p2; p1[x, y]
p2[x, y]
pf[f_, x_] := f[x] + f[1 - x]
 (* Definise se funkcija koja ima funkciju ju kao argument *)
pf[Log, g]
```

```
Log[1 - g] + Log[g]
```
(* Log je ime funkcije koja se koristi kao stvarni parametar*)

10. NEKOLIKO JEDNOSTAVNIJIH PRIMERA

10.1. PROGRAM ZA GENERISANJE MAGIČNOG KVADRATA

Generisanje magičnog kvadrata za neparan broj *n* može se izvesti na sledeći način:

```
MagicniKvadratŠn_Ć:=
 BlockŠšm=RoundŠ(n-1)/2Ć,i,j,k,lć,
  IfŠ(ModŠn+1,2Ć==0)&&(n<=19)&&(n>0),
    PrintŠ
      MatrixFormŠ
        TableŠk=j-i+m;
          l=j+j-i;
          IfŠk>=n,k=k-n,IfŠk<0,k=k+nĆĆ;
          IfŠl>n,l=l-n,IfŠl<=0,l=l+nĆĆ;
          k*n+1 , ši,nć,šj,nć
        Ć Ć
      Ć,
      PrintŠ"Uneli ste pogresan broj"Ć
 Ć Ć
```

MagicniKvadratŠ5Ć

```
11  16  21  1   6
 6  11  16  21  1
 1   6  11  16  21
21   1   6  11      16
16  21   1  6   11
```

10.2. HORNEROVA ŠEMA

Program za izračunavanje vrednosti polinoma proizvoljnog stepena *n* za različite vrednosti argumenta *x* prema *Hornerovoj* šemi.

```
HornerŠn_,a_,x_Ć:=
 BlockŠšpć,
   p=aŠŠ1ĆĆ;
   ForŠi=2,i<=n,i++,p=p*x+aŠŠiĆĆ Ć;
    ReturnŠšx,pćĆ;
 Ć;
```

InŠ1Ć:=HornerŠ3,š6,5,4ć,2.Ć

OutŠ1Ć=š2.,38.ć

10.3. LAGRANGEOV INTERPOLACIONI POLINOM

Vrednost Lagrangeovog interpolacionog polinoma $P(x)=\sum_{k=0}^{n} y_k L_k(x)$ za proizvoljnu vrednost argumenta x može se izračunati na sledeći način:

```
lagrangeŠx_,y_,xarg_Ć:=
 BlockŠšp,r.lć,
  p=0;l=LengthŠxĆ;
  ForŠj=1,j<=l,j++,r=1;
   ForŠk=1,k<=l,k++,
    IfŠj!=k,r=r*(xarg-xŠŠkĆĆ)/(xŠŠjĆĆ-xŠŠkĆĆ)Ć
   Ć;
   p=p+yŠŠjĆĆ*r
  Ć;
  PrintŠ"Lista cvorova",x,",vred. u cvor. ",yĆ;
  PrintŠ"u tacki ",xarg," vred. je ",pĆ;
 Ć
```

U programu je promenljiva x označena kao *xarg*. Naravno, čvorovi x_k moraju biti različiti da bi se izbeglo deljenje nulom.

10.4. GAUSS-SEIDELOV METOD

Neka je dat sistem linearnih jednačina pomoću matrične jednačine $Ax=b$. Treba proveriti da li je matrica A dijagonalno dominantna i u slučaju da jeste primeniti Gauss-Seidelov metod za dobijanje približnog rešenja. Za početnu aproksimaciju uzeti vektor $XP=0$. Rešenje naći sa tačnošću $EPS=10^{-6}$ sa maksimalnim brojem iteracija $MAX=99$.

```
gsŠa_,b_Ć:=
BlockŠšn,eps=0.000001,maxi=99,x,xp,k=0,l=0,r=1,i,j,u,s,elć,
 n=FirstŠDimensionsŠaĆĆ;
 x=TableŠ0,ši,1,nćĆ;  xp=TableŠ0,ši,1,nćĆ;
 DoŠu=0.;
   DoŠel=aŠŠi,jĆĆ; IfŠi !=j,u=u+AbsŠelĆĆ,    šj,1,nćĆ,
   IfŠu>AbsŠaŠŠi,iĆĆĆ,l=1Ć, ši,1,nć
 Ć;
 IfŠl==1,PrintŠ"Matrica A nije dijagonalno dominantna "Ć,
 WhileŠk<maxi  r>=eps,
   k++;    r=0;
   DoŠs=bŠŠiĆĆ;
     DoŠIfŠj!=i, s=s-aŠŠi,jĆĆ*xpŠŠjĆĆĆ,  šj,1,nćĆ;
     xŠŠiĆĆ=s/aŠŠi,iĆĆ;  r=r+AbsŠxŠŠiĆĆ-xpŠŠiĆĆĆ;
     xpŠŠiĆĆ=xŠŠiĆĆ,    ši,1,nć
   Ć;
```

```
 Ć;
 ReturnŠxĆ
Ć
```

10.5. DEKARTOV PROIZVOD

Sledećom funkcijom se može generiše Dekartov proizvod dve liste.

```
DekartovProizvod Ša_,b_Ć :=
 ModuleŠ(l=(), pom,pom2,i,j,n,m),
   n=LengthŠaĆ;  m=LengthŠbĆ:
   ForŠi=1,i<=n,i++,
     ForŠj=1,j<=m,j++,
      pom=(PartŠa,iĆ,PartŠb,jĆ); pom2=(PartŠb,jĆ,PartŠa,iĆ);
      l=AppendŠl,pomĆ; l=AppendŠl,pom2Ć;
     Ć
   Ć;
 ReturnŠlĆ;
Ć
```

10.6. AKERMANOVA FUNKCIJA

Rekurzivna definicija Akermanove funkcije dobro je poznata.

```
AkerŠn_,x_,y_Ć:=AkerŠn-1,AkerŠn,x,y-1Ć,xĆ;
AkerŠn_,x_,y_Ć:=x+1/; n==0;
AkerŠn_,x_,y_Ć:=x/; (n==1)&&(y==0);
AkerŠn_,x_,y_Ć:=0/; (n==2)&&(y==0);
AkerŠn_,x_,y_Ć:=1/; (n==3)&&(y==0);
AkerŠn_,x_,y_Ć:=2/; (n>3)&&(y==0)
```

10.7. CIFRE KAO REČI

Sledeća funkcija generiše listu koja sadrži reči koje odgovaraju ciframa datog broja..

```
cifraŠn1_Ć:=
 ModuleŠšn=n1,c,cif,rezultat=šćć,
   WhileŠn>0,
      c=ModŠn,10Ć;
      cif=SwitchŠc,0,"nula",1,"jedan",2,"dva",3,"tri",
              4,"cetiri",5,"pet",6,"sest",7,"sedam",
              8,"osam",9,"devet"Ć;
      rezultat=PrependŠrezultat,cifĆ; n=QuotientŠn,10Ć
   Ć;
   ReturnŠrezultatĆ
 Ć
```

10.8. VREDNOSTI HERMITEOVOG POLINOMA

Sledećim programom se generiše matrica čiji su elementi vrednost Hermiteovih polinoma za dato *x*.

```
hŠ0ĆŠx_Ć:=1-3xČ2+2xČ3;
hŠ1ĆŠx_Ć:=x-2xČ2+xČ3;
hŠ2ĆŠx_Ć:= -xČ2+xČ3;
hŠ3ĆŠx_Ć:=3xČ2-2xČ3
PraviMatricuŠn_,x_Ć:=ModuleŠšić,
           MatrixFormŠTableŠNestListŠhŠiĆ,x,nĆ,ši,0,3ćĆĆĆ
```

10.9. ERATOSTENOVO SITO

Napraviti program za izračunavanje prostih brojeva do zadatog broja *n* koristeći princip *Eratostenovog sita*.

```
pozicijaŠa_,b_Ć:=
  ModuleŠšl=šć,ić,
         ForŠi=1,i<=LenghtŠaĆ,i++,
           IfŠModŠaŠŠiĆĆ,bĆ==0, l=AppendŠl,aŠŠiĆĆĆ
         Ć Ć;
         ReturnŠlĆ
  Ć;
eratostenŠn_Ć:=
 ModuleŠšprom=n,b=šć,a,l=šć,i,jć,
  a=DeleteŠRangeŠpromĆ,1Ć;
  WhileŠa!=šć,
    l=pozicijeŠa,FirstŠaĆĆ; b=AppendŠb,FirstŠaĆĆ;
    a=ComplementŠa,lĆ
  Ć;
  ReturnŠbĆ
 Ć
```

11. GRAFIKA

11.1. DVODIMENZIONALNA GRAFIKA

Osnovna grafička funkcija u *MATHEMATICA* je *Plot*. Ona iscrtava proizvoljnu funkciju jedne promenljive u zadatom intervalu. Pored toga moguće je i crtanje više različitih funkcija na istom grafiku.

PlotŠf, šx, xmin, xmaxćĆ	crtanje funkcije *f* u zavisnosti od parametra *x* u intervalu Š*xmin,xmax*Ć

PlotŠšf$_1$, f$_2$, ...ć, šx, xmin, xmaxćĆ	crtanje više funkcija zajedno

Postoje dva načina na koje *MATHEMATICA* može da nacrta grafik funkcije *f*(*x*). Prvi način je da se funkcija prvo izvrši, kako bi se eventualno dobio jednostavniji izraz, koji se zatim izra□unava nad nizom konkretnih vrednosti za *x*. Drugi način je da se prvo odredi niz vrednosti *x*, a zatim da se funkcija izvrši za svaku od tih vrednosti. Ako se pozove funkcija *Plot* izrazom *Plot*Š*f*,š*x*,*xmin*,*xmax*ćĆ, *MATHEMATICA* postupa na drugi način, tj. prvo bira neki niz vrednosti *x* iz datog intervala, a zatim izračunava funkciju za svaku od njih.
Međutim, u nekim slučajevima je pogodnije da se crtanje obavi na pravi način pozivom funkcije *Plot*Š*Evaluate*Š*f*Ć, š*x, xmin, xmax*ćĆ. Tipičan slučaj je da funkcija *f* pomoću funkcije *Table* generiše listu funkcija koje treba nacrtati. Tada je mnogo brže da se lista funkcija generiše jednom, na početku, umesto da se funkcija *Table* izvršava za svaku novu vrednost *x*.

PlotŠf, šx, xmin, xmaxćĆ	prvo se bira niz vrednosti *x*, zatim se funkcija *f* izračunava za svaku od njih
PlotŠEvaluateŠfĆ, šx, xmin, xmaxćĆ	prvo se izvršava funkcija *f*, a zatim se odre□uje niz vrednosti *x*
PlotŠEvaluateŠTableŠf, ...ĆĆ, šx, xmin, xmaxćĆ	generisanje listi funkcija koje se crtaju zajedno na jednom grafiku.

Rezultat grafičkih funkcija je objekat -*Grafics*-, koji sadrži sve podatke o nacrtanom grafiku. Objekat -g*rafics*- se može ponovo prikazati na ekranu funkcijom *Show*. Takoe, on se može dodeliti promenljivoj, elementu niza, ili staviti u listu-drugim rečima, sa njima se može postupati isto kao i sa svim drugim objektima u *MATHEMATICA*.

ShowŠ%Ć	prikazivanje grafika iz prethodne naredbe
g=PlotŠf, ...Ć	dodeljivanje grafika simbolu *g*
ShowŠgĆ	prikazivanje grafika dodeljenog simbolu *g*
ShowŠg_1, g_2, ...Ć	prikazivanje više grafika, zajedno.

MATHEMATICA sadrži i funkcije za crtanje određenih grafičkih primitiva.

PointŠšx, yćĆ	tačka na poziciji (*x*,*y*)
LineŠššx1, y$_1$ć, šx$_2$, y$_2$ć, ...ćĆ	linija kroz tačke šx_1, y_1ć,šx_2, y_2ć, …

RectangleŠšxmin, yminć, šxmax, ymaxćĆ	crtanje pravougaonika
PolygonŠššx$_1$, y$_1$ć, šx$_2$, y$_2$ć, ...ćĆ	crtanje poligona
CircleŠšx, yć, rĆ	krug sa centrom u (x,y) i poluprečnikom r

11.1.1. Opcije pri radu sa dvodimenzionalnom grafikom

MATHEMATICA pri crtanju grafika mora da izabere između mnogih mogućnosti: u kojim tačkama će računati funkciju, kako će nacrtati ose, i slično. U većini slučajeva *MATHEMATICA* vrši pravilan izbor i daje zadovoljavajući crtež. Ponekad je, međutim, neophodno da korisnik sam podesi neke opcije, kako bi grafik dobio željeni izgled.
U grafičkim funkcijama se kao poslednji argument može navesti niz pravila oblika opcija to vrednost, kojima se podešavaju razne opcije.

PlotŠf, šx, x$_{min}$, x$_{max}$ć, opcija->vrednostĆ	crtanje grafika funkcije f, dok opcija uzima određene vrednosti
ShowŠ%,opcija$_1$->vrednost$_1$, opcija$_2$-> vrednost$_2$, ...Ć	prikazivanje grafika iz prethodne naredbe, sa podešavanjem raznih opcija

Jednom dobijeni grafik se može ponovo prikazati na ekranu funkcijom Show, pri čemu se takoe može zadati niz operacija. Na taj način se uz malo eksperimentisanja može dobiti lepo uređen grafik. Sledeće opcije koje navodimo su opcije za dvodimenzionalne grafike, koje deluju u funkcijama *Plot*, *ListPlot* i *ParametricPlot*.

AspectRatio	**1/GoldenRatio**	odnos visine i širine grafika
Axes	**True**	iscrtavanje koordinantnih osa
AxesLabel	**None**	oznake za koordinantne ose
AxesOrigin	**Automatic**	tačka u kojoj se seku koordinantne ose
Frame	**False**	iscrtavanje okvira oko grafika
FrameLabel	**None**	oznake za okvir
FrameTics	**Automatic**	koordinate koje treba obeležiti na okviru
GridLines	**None**	iscrtavanje pomoćnih linija
PlotJoined	**False**	spajanje tačaka na grafiku
PlotLabel	**None**	naslov grafika
PlotPoints	**25**	najmanji broj tačaka u kojima se računa vrednost funkcije
PlotRange	**Automatic**	oblast grafika koji se prikazuje

Ticks	**Automatic**	koordinate koje treba obeležiti na koordinantnim osama

UnderbarAspectRatio	određuje odnos ozmeu širine i visine grafika

U graficima parametarski zadanih krivih se vrednost obično postavlja na *Automatic*, da bi krive imale svoj "prirodni" izgled. Tada se pri iscrtavanju grafika odnos širine i visine postavlja tako da jedinična duž ima istu dužinu i po *x* i po *y* osi.

1/GoldenRatio	standardna vrednost za funkciju *Plot*
Automatic	odnos širine i visine se određuje iz apsolutnih *x* i *y* koordinata
izraz	vrednot izraza se uzima za odnos visine i širine
UnderbarAxes	Određuje da li se na grafiku crtaju koordinantne ose
True	crtaju se obe koordinantne ose
False	ose se ne crtaju
šTrue, Falseć	crta se *x* osa, ali ne i *y* osa
AxesLabel	Određuje oznake koje se ispisuju na koordinantnim osama; kao oznaka se može navesti izraz ili tekst pod navodnicima
None	nema oznaka
Ylabel	oznaka za *y* osu
šxlabel, ylabelć	*xlabel* je oznaka za *x* osu, a *ylabel* za *y* osu
AxesOrigin	određuje koordinate tačke u kojoj se seku koordinatne ose

Koordinatne ose se postavljaju u koordinatni početak kad god je to moguće. Ukoliko to nije moguće, postavljaju se što bliže koordinatnom početku.
Opcija *Automatic* omogućava da se tačka preseka koordinatnih osa bira automatski.

šx_0,y_0ć	eksplictno zadata tačka preseka.
underbarFrame	određuje da li se oko grafika crta okvir.
True	okvir se crta
False	okvir se ne crta
FrameLabel	služi za označavanje stranica okvira
AxesLabel	za obeležavanje koordinatnih osa
None	bez oznake
šx_{dole},y_{levo}ć	oznake za donju i levu stranicu
šx_{dole},y_{levo},x_{gore},y_{desno}ć	oznake za sve četiri stranice

underbarFrameTicks	određuje tačke koje se obeležavaju na stranicama okvira

Način obeležavanja tačaka se može zadati za ceo okvir ođednom, ili za svaku stranicu posebno. U drugom slučaju se zadaje lista od četri elementa, od kojih svaki određuje način obeležavanja za po jednu stranicu okvira.

None	bez obeleženih tačaka
Automatic	automatski izbor obeleženih tačaka
šx_1,x_2,...ć	eksplictno zadate tačke
šx_{dole}, x_{levo}ć	pazni načini obeležavanja donje i leve stranice
šx_{dole},y_{levo},x_{gore},y_{desno}ć	razni načini obeležavanja za sve stranice
GridLines	kontroliše crtanje pomoćnih linija paralelnih koordinatnim osama

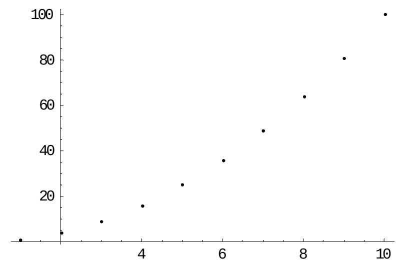
Način iscrtavanja pomoćnih linija se odreuje slično kao kod opcije FrameLabel. Ukoliko se navede lista od dva elementa prvi kontroliše vertikalne pomoćne linije, a drugi horizontalne.

None	bez pomoćnih linija
Automatic	automatsko postavljanje pomoćnih linija
šx_1,x_2,...ć	eksplicitno zadate pozicije pomoćnih linija
švlines,hlinesć	*vlines* kontroliše vertikalne, a *hlines* horizontalne linije

PlotJoined ima efekat u funkciji *ListPlot* i određuje da li se susedne tačke na grafiku spajaju linijom.

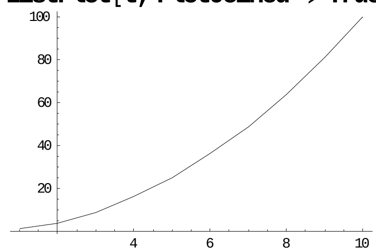
True	tačke se spajaju
False	tačke se ne spajaju
PlotLabel	predstavlja naslov grafika. Kao vrednost se može navesti bilo koji izraz ili tekst između navodnika
izraz ili "tekst"	naslov grafika

```
t = Table[i^2, {i, 10}]
ListPlot[t]
{1, 4, 9, 16, 25, 36, 49, 64, 81, 100}
```



- Graphics -

```
ListPlot[t, PlotJoined -> True]
```



PlotPoints	određuje najmanji broj tačaka u kojima se računa vrednost funkcije

Ako je *PlotPoints=N,* funkcija se računa u najmanje *N* tačaka. Kako se vrednost funkcije uvek računa u konačnom broju tačaka, moguće je da neke funkcije neće biti tačno nactane. Povećanjem vrednosti *PlotPoints* se može povećati preciznost grafika, ali tada programu *MATHEMATICA* treba više vremena za računanje. Opcija PlotPoints nema efekta u funkciji *Show.*

PlotRange	definiše oblast grafika koji se prikazuje

MATHEMATICA automatski određuje oblast tako da grafik ne bude definisan sa nepotrebno mnogo tačaka, ali zato ponekad delovi grafika ne stanu na crtež. Ukoliko se oblast previše smanji u odnosu na originalnu veličinu, može se dogoditi da grafik postane neprecizan. Tada treba ponovo nacrtati željenu funkciju u novom, manjem intervalu.

All	prikazivanje svih tačaka grafika
Automatic	automatski izbor oblasti
šy_{min}, y_{max}ć	eksplicitno zadate granice po *y* osi
ššx_{min}, x_{max}ć, šy_{min}, y_{max}ćć	eksplicitno zadata oblast u *xy* ravni.
Ticks	služi za obeležavanje tačaka na koordinantnim osama

Moguće je zadati način obeležavanja za obe ose ođednom, ili za svaku posebno.

None	nema oznaka
Automatic	automatsko obeležavanje
ššx_1, x_2, ...ć, šy_1, y_2, ...ćć	eksplicitno zadate tačke na *x* i *y* osi
šxticks, yticksć	*xticks* odreuje oznake na *x* osi, a *yticks* oznake na *y* osi

11.1.2. Stilovi i boje

Način i sortiranje pojedinih krivih na grafiku zadaje se opcijom *PlotStyle*, koja sadrži listu stilova za svoju vrednost. Pojedinačne krive na grafiku se mogu iscrtavati različitim stilovima da bi se bolje razlikovale. Pritom se pod stilom podrazumeva lista grafičkih kontrolnih funkcija koje određuju izgled krive ili niza tačaka.

šgkf_1, gkf_2, ...ć	lista grafičkih kontrolnih funkcija čini stil
PloStyle->stil	stil koji se primenjuje na sve krive na grafiku
PlotStyle->ššstil$_1$ć, šstil$_2$ć, ...ćć	stilovi koji se primenjuju na pojedinačne krive na grafiku

Grafičke kontrolne funkcije imaju efekat samo ako se navedu kao vrednosti opcija funkcije *PlotStyle*. Ovde su navedene samo neke od mnogobrojnih grafičkih kontrolnih funkcija.

GrayLavelŠiĆ	nijansa sivog izmeu 0 (crno) i 1 (belo)
RGBColorŠr, g, bĆ	boja određena crvenom, zelenom i plavom komponentom, svaka između 0 i 1
HueŠhĆ	boja *h* iz spektra
HueŠh, s, bĆ	boja *h*, zasićenje *s* i osvetljenje *b*
PointSize	prečnik tačke zadat relativno u odnosu na širinu celog grafika
AbsolutePointSizeŠdĆ	prečnik tačke u apsolutnim jedinicama
ThicknessŠrĆ	debljina linije u odnosu na širinu grafika
AbsoluteThicknessŠdĆ	debljina linije u apsolutnim jedinicama
DashingŠšr_1, r_2, ...ćĆ	dužine segmenta za crtanje neprekidnih linija zadate u odnosu na širinu grafika
AbsoluteDashingŠšd_1, d_2, ...ćĆ	dužine segmenata u apsolutnim jedinicama

Debljina i način iscrtavanja neprekidnih linija, kao i prečnik tačaka se mogu zadati relativno u odnosu na širinu celog grafika ili u apsolutnim jedinicama. Apsolutna jedinica iznosi 1/72 inča, što približno odgovara visini jedne tačke na štampaču.
Uobičajeni stilovi su navedeni u sledećoj tabeli:

GrayLavelŠ0.5Ć	siva boja
RGBColorŠ1, 0, 0Ć	crvena boja
ThicknessŠ.05Ć	debela linija
DashingŠš.05, .05ćĆ	obična isprekidana linija
DashingŠš.01, .05, .05, .05ćĆ	isprekidana linija sa tačkama

Pored opcije *PlotStyle* koja određuje stilove za pojedine krive, postoje i opcije koje utiču na boje celog grafika.

Background->boja	boja za pozadinu grafika
DefaultColor->boja	boja kojom se iscrtava grafik
ColorOutput->GrayLavel	generisanje crno-bele slike na monitoru u boji

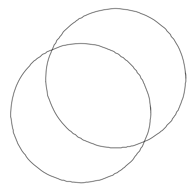
11.1.3. Prikazivanje i kombinovanje grafika

Svaki nacrtani grafik se pamti i kasnije se može prikazivati. Kad grafik ponovoi prikazuje mogu se promeniti neke od korišćenih opcija. Za prikazivanje grafika koristi se funkcija *Show*.

ShowŠplotĆ	prikazivanje grafika
ShowŠplot->valueĆ	prikazivanje grafika sa promenjenim opcijama
ShowŠplot_1, plot_2, ...Ć	kombinovanje nekoliko grafika
ShowŠGraphicsArrayŠšplot$_1$, plot$_2$,...ć, ...ćĆĆ	crtanje liste grafika
InpurFormŠplotĆ	prikazivanje informacija sačuvanih o grafiku
ShowŠGraphicsArrayŠšplot$_1$, pplot$_2$, ...ćĆĆ	prikazivanje više grafika jedan uz drugi
ShowŠGraphicsArrayŠššplot$_1$ć, šplot$_2$ć, ...ćĆĆ	prikazivanje grafika u koloni
ShowŠGraphicsArrayŠššplot$_{11}$, plot$_{12}$, ...ć, ...ćĆĆ	prikazivanje grafika u pravougaonom okviru

ShowŠGraphicsArrayŠplots, GraphicsSpacing->šh, vćĆĆ	postavljanje specificiranih horizontalnih i vertikalnih linija između grafika

```
Show[Graphics[{Circle[{0, 0}, 2],
                    Circle[{1, 1}, 2]}], AspectRatio -> Automatic]
```



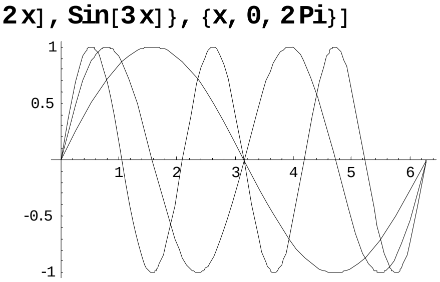
Podrazumevane vrednosti opcija: Grafičke funkcije poput *Plot* imaju mnogo opcija, koje su na pov cetku rada sa *MATHEMATICA* postavljene na neke podrazumevane vrednosti. Funkcijom *SetOptions* se podrazumevane vrednosti mogu promeniti, tako da se svi kasniji grafici iscrtavaju sa novim podrazumevanim vrednostima. Za razliku od lokalnih opcija koje se zadaju u funkciji plot, opcije podešene funkcijom *SetOptions* važe od trenutka zadavanja, pa sve dok se ne izmene ili dok se ne završi rad u *MATHEMATICA*.

SetOptionsŠf, opcija$_{1\text{->}}$vrednost$_1$, opcija$_{2\text{->}}$ vrednost$_2$, ...Ć	podešavanje podrazumevanih vrednosti opcija za funkciju *f*

Funkcija *SetOptions* se često upotrebljava da bi se smanjila debljina linije kojom se iscrtavaju grafici.

SetOptionsŠPlot, PlotStyle->ThicknessŠ0ĆĆ	postavljanje standardne debljine linije u grafiku na minimum.

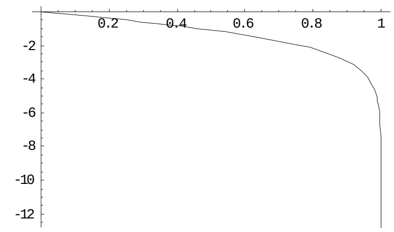
```
Plot[{Sin[x], Sin[2 x], Sin[3 x]}, {x, 0, 2 Pi}]
```



- Graphics -

```
g2[x_] := Log[(1 - x) / (1 + x)]
Definition[g2]
a = Plot[g2[x], {x, 0, 1}]
```

$$g2[x\_] := \mathrm{Log}\left[\frac{1-x}{1+x}\right]$$



- Graphics -

11.2. TRODIMENZIONALNA GRAFIKA

Osnovna funkcija za crtanje trodimenzionalne grafike je *Plot3D*, koja je analogna funkciji *Plot* u dve dimenzije. Glavna razlika između ove dve funkcije je u tome v sto se kod funkcije *Plot3D* umesto jedne navode dve promenljive, *x,* i *y,* koje određuju oblast u *xy* ravni nad kojom se iscrtava željena funkcija *f*.

Plot3DŠf, šx, xmin, xmaxć, šy, ymin, ymaxćĆ	cranje trodimenzionalnog grafika funkcije *f* u zavisnosti od promenljivih *x* i *y*
Plot3DŠšf, sć, šx, xmin, xmaxć, šy, ymin, ymaxćĆ	crtanje trodimenzionalnog grafika funkcije *f* čija je boja određena sa *s*
SunfaceGraphicsŠheight, colorĆ	grafički objekat koji predstavlja površinu sa određenim oblikom (visinama) i bojama

11.2.1. Trodimenzionalne grafičke primitive

U tabeli je navedeno nekoliko osnovnih trodimenzionalnih figura.

PointŠšx, y, zćĆ	tačka sa koordinatama (*x, y, z*)
LineŠššx_1, y_1, z_1ć, šx_2, y_2, z_2ć, ...ćĆ	linija kroz tačke šx_1, y_1, z_1ć, šx_2, y_2, z_2ć, ...
PolygonŠššx_1, y_1, z_1ć, šx_2, y_2, z_2ć, ...ćĆ	poligon sa zadatom listom temena
CuboidŠšxmin, ymin,zminć, šxmax, ymax, zmaxćĆ	paralelopiped

TextŠexpr, šx, y, zćĆ	tekst na poziciji š*x, y, z*ć
CuboidŠšx, y, zćĆ	jedinična kocka sa naspramnim temenima koje imaju koordinate š*x, y, z*ć i š*x*+1, *y*+1, *z*+1ć.

11.2.2. Opcije pri crtanju grafika

U trodimenzionalnom grafiku mogu se podešavati različiti parametri.

AmbientLight	**GrayLevelŠ0 Ć**	osvetljenje trodimenzionalnog grafika
Axes	**True**	iscrtavanje koordinantnih osa
AxesLabel	**None**	oznake za koordinantne ose
Boxed	**True**	iscrtavanje grafika unutar kvadra
BoxRatios	**š1, 1, .4ć**	odnos dimenzija stranica kvadra
FaceGrids	**None**	iscrtavanje pomoćnih linija na stranama kvadra
Lighting	**True**	za senčenje grafika (koristi simulirano osvetljenje)
LightSources		postavljanje izvora svetlosti
Mesh	**True**	iscrtavanje mreže na grafiku
PlotLabel	**None**	naslov grafika
PlotPoints	**15**	broj tačaka u oba pracvca u kojima se računa vrednost funkcije
PlotRange	**Automatic**	oblast grafika koja se prikazuje
Shading	**True**	senčanje grafika
ViewPoint	**š1.3, -2.4, 2ć**	koordinate tečke iz koje se posmatra grafik.

AmbientLight određuje kako se osvetljava prostor u kome se iscrtava grafik. Podrazumevana vrednost je *GrayLevel*Š*0*Ć, a kao vrednost opcije navodi se neka od grafičkih kontrolnih funkcija za boju.

GreyLevelleftŠ0,5Ć	siva
RGBColorleftŠ1,1,0Ć	žuta

Parametar *Axes* određuje da li i koje se koordinatne ose crtaju. Kao i u dvodimenzionalnom slučaju, i ovde je moguće uticati na iscrtavanje pojedinačnih osa.

True	ose se crtaju

False	ose se ne crtaju
šTrue,True,Falseć	crtaju se *x* i *y* ose, a osa *z* ne
underbarAxsesLabel	određuje oznake koje se ispisuju na koordinatnim osama
None	nema oznaka
zlabel	oznaka za *z* osu
šxlabel,ylabel,zlabelć	oznake za sve tri ose
underbarBoxed	kontroliše iscrtavanje kvadra koji uokviruje grafik
True	kvadar se crta
False	kvadar se ne crta
underbarBoxRatios	određuje odnos dužina stranica kvadra koji uokviruje grafik, a samim tim i odnos dimenzija grafika
š1,1,0.4ć	standardni odnos dužina stranica kvadra
Automatic	odnos se određuje iz koordinata sa grafika
šx_r,y_r,z_rć	eksplicitno zadat odnos dužina stranica kvadra
underbarFaceGrids	kontroliše iscrtavanje mreže pomoćnih linija na stranicama kvadra koji uokvirava grafik

Ukoliko se kao vrednost navede *All* mreža se iscrtava na svim stranicama. Pojedinačne stranice se označavaju listom brojeva šdir_x, dir_y, dir_zć, u kojoj dva broja moraju biti 0, a treći +1 ili -1. Pozicije pomoćnih linija se mogu eksplicitno zadati slično kao u opciji *GridLines* kao kod dvodimenzionalnih grafika.

None	nema pomoćnih linija
All	pomoćne linije se iscrtavaju na svim stranicama
šstr_1, str_2, ...ć	eksplicitno zadavanje pomoćnih linija za stranicu str_i
šš0, 0, -1ćć	pomoćne linije se iscrtavaju na desnoj stranici
underbarLighting	određuje da li se za senčenje grafika koristi simulirano osvetljenje, tj. izvori svetlosti zadati opcijama *AmbijentLight* i *LightSources*

U slučaju da se simulirano osvetljenje isključi, površi se senče u zavisnosti od visine.

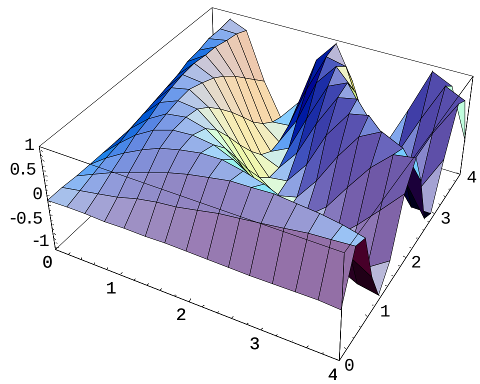
True	senčenje u zavisnosti od osvetljenja
False	senčenje u zavisnosti od visine

LightSources služi za postavljanje izvora svetlosti koji osvetljava površ. Kao vrednost se navodi lista parova špos_i,$boja_i$ć, gde je pos_i uređena trojka koja određuje poziciju izvora svetlosti, a *boja*$_i$ grafička kontrolna funkcija za izbor odgovarajuće boje.

šššx$_0$,y$_0$,z$_0$ćć,bojać	izvor svetlosti sa koordinatama šx_0, y_0, z_0ć
šššx$_0$,y$_0$,z$_0$ć,RGBColorŠ1,1,0Ćć,... ć	izvor svetlosti žute boje
Mesh	određuje da li se na grafiku iscrtava mreža koja pokriva površ
True	mreža se iscrtava
False	mreža se ne iscrtava
PlotLabel	određuje naslov grafika.
None	nema naslova (podrazumevana vrednost)
izraz ili "izraz"	naslov grafika
underbarPlotPoints	određuje broj tačaka u pravcu *x* i *y* ose u kojima se funkcija računa pri crtanju grafika. što je taj broj veći, površ će biti preciznije nacrtana, ali će crtanje trajati duže
n	funkcija se računa $n \times n$ tačaka
šn$_1$,n$_2$ć	funkcija se računa u $n_1 \times n_2$ tačaka. Podrazumevana vrednost je 15

PlotRange	definiše oblast grafika koji se prikazuje
All	prikazivanje celog grafika
Automatic	automatski izbor oblasti
šz$_{min}$, z$_{max}$ć	ograničenje po *z* osi
ššx$_{min}$,x$_{max}$ć,šy$_{min}$, y$_{max}$ć, šz$_{min}$, z$_{max}$ćć	eksplicitno zadata oblast
Shading	određuje da li se grafik senči ili ne (Za vrednost *True* grafik se senči)
False	grafik se crta bez senčanja
ViewPoint	određuje koordinatne tačke iz koje se posmatra grafik, relativno u odnosu na kvadar koji ga uokviruje. Pri tome se koristi koordinatni sistem u kome centar kvadra ima koordinate š0,0,0ć, a dužina najveće stranice je 1
šx$_0$,y$_0$,z$_0$ć	koordinate tačke iz koje se grafik posmatra

```
Plot3D[Sin[x y], {x, 0, 4}, {y, 0, 4}];
```



11.3. CRTANJE LISTI BROJEVA

Lista brojeva šy_1,... ,y_nć se može grafički prikazati kao niz tačaka sa koordinatama (i, y_i), i=1,...,n. Funkcija *ListPlot* crta grafik na osnovu liste brojeva. Umesto brojeva, u listi se takođe može navesti i niz uređenih parova oblika šx_i, y_ić, koji određuju obe koordinate tačaka.

ListPlotŠšy$_1$, y$_2$, ...ćĆ	crtanje tačaka sa koordinatama (1, y_1), (2, y_2),...
ListPlotŠšx$_1$, y$_1$ć, šx$_2$, y$_2$ć, ...Ć	crtanje tačaka sa koordinatama (x_1, y_1), (x_2, y_2),...
ListPlotŠlista, PlotJoined->TrueĆ	crtanje grafika na kome su susedne tačke spojene linijama

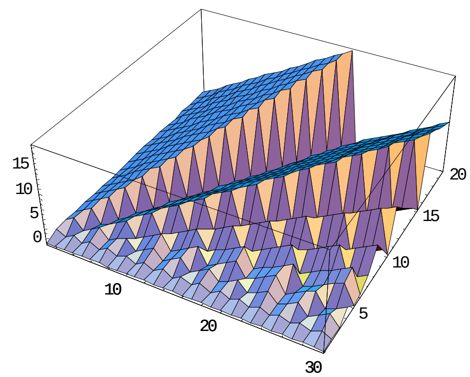
Većina opcija za dvodimenzionalne grafike, koje se koriste u funkciji *Plot*, deluje i u funkciji *ListPlot*.

11.3.1. Trodimenzionalna lista brojeva

Na osnovu realne matrice može se nacrtati trodimenzionalni grafik. Pojedinačni elementi matrice se tada posmatraju kao visine tačaka (z koordinate), a indeks elemenata kao x i y koordinate.

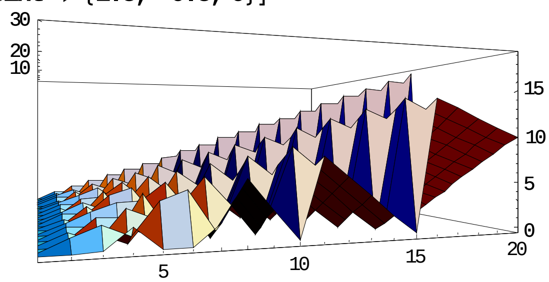
ListPlot3DŠššz$_{11}$, z$_{12}$, ...ć, šz$_{21}$, z$_{22}$, ...ć,...ćĆ	crtanje trodimenzionalnog grafika na osnovu matrice vrednosti za z

```
t3 = Table[Mod[x, y], {y, 20}, {x, 30}];
ListPlot3D[t3]
```



- SurfaceGraphics -

```
Show[%, ViewPoint -> {1.5, -0.5, 0}]
```



- SurfaceGraphics -

11.4. PARAMETARSKI ZADATE KRIVE I POVRŠI

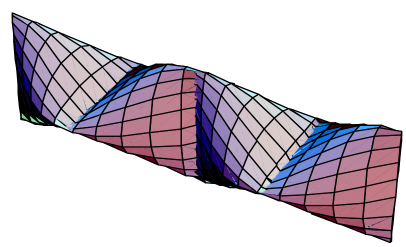
Funkcijom *Plot* se crtaju krive kod kojih je *y* koordinata svake tačke zadata kao funkcija *x* koordinata. *MATHEMATICA* takođe može da crta parametarski zadate krive, kod kojih su *x* i *y* koordinate svake tačke zadate kao funkcije jednog parametra.

ParametricPlotŠšf_x, f_yć, št, t_{min}, t_{max}ćĆ	crtanje parametarski zadate krive
ParametricPlotŠšf_x, f_yć, šg_x, g_yć,...ć, št,t_{min}, t_{max}ćĆ	crtanje više parametarski zadatih krivih zajedno
ParametricPlotŠšf_x,f_yć,šz,z_{min},z_{max}ć, AspectRatio->AutomaticĆ	podešavanje razmere da bi se očuvao pravilan oblik krive

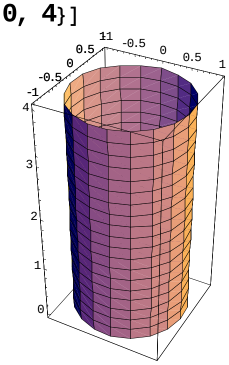
U tri dimenzije se mogu crtati parametarski zadate krive i površi. Krive su određene sa tri funkcije parametra *t*, koje određuju koordinate tačaka krive. Površi se zadaju preko tri funkcije parametara *u* i *t*. Pri zadavanju površi treba obratiti pažnju na pravilnu parametrizaciju. Granice parametra *u* i *t* treba postaviti tako da se svi delovi površi pokriju samo jednom, jer u suprotnom iscrtavnje može da potraje mnogo duže.

ParametricPlot3DŠšf$_x$, f$_y$, f$_z$ć, št, t$_{min}$,t$_{max}$ćĆ	crtanje parametarski zadate krive u tri dimenzije
ParametricPlot3DŠšf$_x$, f$_y$, f$_z$ć, št, t$_{min}$,t$_{max}$ć, šu, u$_{min}$, u$_{max}$ćĆ	crtanje parametarski zadate površi
ParamericPlot3DŠššf$_x$, f$_y$, f$_z$ć, šg$_x$, g$_y$, g$_z$ć, ...ć, ...Ć	crtanje više krivih ili površi zajedno.

```
- Graphics -
    g = ParametricPlot3D[{x, Cos[t] Sin[x], Sin[t] Cos[x]}, {x, -Pi, Pi},
        {t, 0, 2Pi}, Axes -> False, Boxed -> False]
```



```
- Graphics3D -
ParametricPlot3D[{Sin[t], Cos[t], u},
        {t, 0, 2Pi}, {u, 0, 4}]
```



11.5. NEKI SPECIJALNI GRAFICI

Postoje takođe i neki specijalni oblici grafičkog prikazivanja:

LogPlotŠf, šx, x$_{min}$, x$_{max}$ćĆ	generiše logaritamski-linearni grafik

LogLogPlotŠf, šx, x_{min}, x_{max}ćĆ	generiše logaritamski-logaritamski grafik
LogListPlotŠlistĆ	generiše logaritamski-linearni grafik iz liste podataka
LogLogListPlotŠlistĆ	generiše logaritamski-logaritamski grafik iz liste podataka
PolarPlotŠr, št, t_{min}, t_{max}ćĆ	generiše grafik pomoću polarnih koordinata radiusa *r* kao funkciju od ugla *t*
ErrorListPlotŠššx$_1$, y$_1$, dy$_1$ć, ...ćĆ	generiše grafik na osnovu podatka sa greškama
TextListPlotŠššx$_1$, y$_1$, "s$_1$"ć, ...ćĆ	tabela podataka gde je svaka tačka data sa tekst-stringom s_i
BarChartŠlistĆ	tabela podataka u obliku grafikona (pravougaonog dijagrama)
PieChartŠlistĆ	lista podataka u obliku pitastog dijagrama
PlotVectorFieldŠšf$_x$, f$_y$, f$_z$ć, šx, x_{min}, x_{max}ć, šy, y_{min}, y_{max}ćĆ	vektorsko polje koje odgovara vektorskoj funkciji *y*
ListPlotVectorFieldŠlistĆ	vektorsko polje koje odgovara dvo-dimenzionalnom nizu vektora u listi
SphericalPlotŠr,links,štheta,min,max, šphi, min, maxćĆ	generiše trodimenzionalni sferni graf (površ)

Literatura

[1] P. Abbot, Tricks of the trade, *The Mathematica Journal* , 3 (1993), 18—22.

[2] N. Blachman, *Mathematica: A Practical Approach,* Englewood Cliffs, New Jersey: Prentice-Hall, 1992.

[3] T. Gray and J. Glynn, *Exploring Mathematics in Mathematica*, Redwood City, California: Adisson-Wesley 1991.

[4] N. Krejić i đ. Herceg, Matematika i *MATHEMATICA*, Računari u Univerzitetskoj Praksi, Novi Sad, 1993.

[5] R. Maeder, *Programming in Mathematica, Third Edition*, Redwood City, California: Adisson-Wesley 1996.

[6] G.V. Milovanović, *Numerička analiza I,* Naučna knjiga , Beograd, 1991.

[7] G.V. Milovanović, *Numerička analiza II,* Naučna knjiga , Beograd, 1991.

[8] G.V. Milovanović, *Numerička analiza III,* Naučna knjiga , Beograd, 1991.

[9] C. Smith and N. Blachman, *The Mathematica Graphics Guidebook*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1995.

[10] S. Wolfram, *Mathematica: a System for Doing Mathematics by Computer*, Addison-Wesley Publishing Co, Redwood City, California, 1991.

[11] S. Wolfram, *Mathematica Book, Version 3.0,* Addison-Wesley Publishing Co, Redwood City, California, 1997.

II GLAVA

PRIMENE PROGRAMSKOG PAKETA *MATHEMATICA*®

1. SORTIRANJE

Sortiranje predstavja jedan od osnovnih problema programiranja, pa je izučavanju i pronalaženju novih metoda sortiranja pridodata značajna pažnja. Pod sortiranjem se podrazumeva uređivanje nekih vrsta podataka u opadajući odnosno rastući poredak. Najčešće se sortiranje primenjuje za nizove čiji podaci mogu da budu određenog tipa (celobrojnog, iz nekog konačnog skupa podataka itd.), dok su drugi veoma opšti, odnosno ne postoje nikakva ograničenja.

1.1. SORTIRANJE IZBOROM UZASTOPNIH MINIMUMA

Pod sortiranjem izborom uzastopnih minimuma podrazumeva se postupak koji se zasnva na sledećim pravilima:

1. na prvom mestu u sortiranom nizu mora biti najmanji element niza, odnosno obavja se razmena elemenata niza sa pozicije *a*Š1Ć sa svim onim elementima iz skupa *a*Š2Ć,...,*a*Š*n*Ć, koji su manji od *a*Š1Ć;
2. na drugom mestu se postavja najmanji od preostalih elemenata niza, tj. od svih elemenata osim prvog. To znači da se na mestu *a*Š2Ć postavja najmanja vrednost između vrednosti *a*Š2Ć, *a*Š3Ć,...,*a*Š*n*Ć;
3. na trećem mestu se postavlja najmanji element od dela niza koji ostaje posle isključivanja dva predhodno određena elementa
4. ovaj postupak nastavjamo tako što sledeči put biramo najmanji od preostalih elemenata.

Ovaj način sortiranja poznat je u literaturi na engleskom pod nazivom *Selection sort*. Na osnovu iznetog razmatranja ovog načina sortiranja možemo napisati funkciju na sledeći način:

```
Selectionsort Šl_Ć:=
 ModuleŠšl1=l,s,n=LengthŠlĆć,
   ForŠi=1,i<n,i++,
     ForŠj=i+1,j<=n,j++,
       IfŠl1ŠŠiĆĆ>l1ŠŠjĆĆ,
         s=l1ŠŠiĆĆ; l1ŠŠiĆĆ=l1ŠŠjĆĆ; l1ŠŠjĆĆ=s
     Ć Ć
```

```
 Ć;
  ReturnŠl1Ć;
 Ć
```

Iz same funkcije se vidi da se najmanja vrednost dela niza od *k*-te pozicije do poslednje *n*-te pozicije dodeljuje *k*-tom elementu.

Primer. Neka je neki niz od osam elemenata zada na sledeći način 43, 57, 10, 40, 98, 13, 2, 72. Ako posmatramo svaki od koraka u spoljašnjoj petlji onda ćemo dobiti sledeće vrednosti za elemente niza:

2	57	43	40	97	13	10	72
2	10	57	43	97	40	13	72
2	10	13	57	97	43	49	72
2	10	13	40	97	57	43	72
2	10	13	40	43	97	57	72
2	10	13	40	43	57	72	97
2	10	13	40	43	57	72	97

Predhodno definisana funkcija je koncipirana tako da svaka razmena zahteva 3 dodeljivanja. Efikasnije od takve razmene je da se izračuna redni broj (indeks) najmanjeg elementa dela niza i tek nakon toga razmeni vrednost *n*-tog elementa niza i najmanjeg elementa niza. Pa tako dobijemo sledeću funkciju:

```
SelectionsortŠl1_Ć:=
 ModuleŠšl1=l,i,j,minind,n=LengthŠlĆć,
   ForŠi=1,i<n,i++,minind:=i;
     ForŠj=i+1,j<n,j++,
       IfŠl1ŠŠjĆĆ<l1ŠŠminindĆĆ,minind:=jĆ;
       IfŠi!=minind,
         p:=l1ŠŠiĆĆ;l1ŠŠiĆĆ:=l1ŠŠminindĆĆ;
         l1ŠŠminindĆĆ:=p
   Ć Ć Ć
   ReturnŠl1Ć
 Ć
```

1.2. SORTIRANJE UMETANJEM ELEMENATA NIZA NA ODGOVARAJUĆA MESTA

Sortiranje umetanjem elemenata niza na odgovarajuća mesta (engl. *insertion sort*) obavlja se dodavanjem jednog po jednog elementa niza u sortirani niz koji se sastoji od već obrađenih (dodatih) elemenata niza. Ako pretpostavimo da smo od prvih k elemenata napravili sortirani niz onda dodajemo (k+1)-element niza tako da posle toga opet imamo sortirani niz sa jednim elementom više. Na početku taj element se dodaje na kraj sortiranog niza. Zatim se poredi sa predhodnimnim elementom (ako takav postoji) i ako je manji od njega zamene im se mesta. Postupak ovakvog poređenja i zamene mesta se izvršava dotle dok postoji predhodnik koji je veći od datog elementa.

Primer. Niz od 8 elemenata 44, 55, 12, 42, 94, 18, 6, 67 kroz proces uređenja ima sledeće vrednosti:

44	55	12	42	94	18	6	67
44	55	12	42	94	18	6	67
12	44	55	42	94	18	6	67
12	42	44	55	94	18	6	67
12	42	44	55	94	18	6	67
12	18	42	44	55	94	6	67
6	12	18	42	4	55	94	67
6	12	18	42	44	55	67	94

Za ovako izloženi postupak imamo sledeču funkciju:

```
InsertionsortŠl_Ć:=
ModuleŠšl1=l,d=LengthŠl1Ć,i,j,kć,
  ForŠi=2,i<n,i++,
    ForŠj=1,(j>0)&&(l1ŠŠjĆĆ<l1ŠŠj-1ĆĆ),j--,
      k=l1ŠŠjĆĆ; l1ŠŠjĆĆ=l1ŠŠj-1ĆĆ; l1ŠŠj-1ĆĆ=k
  Ć Ć;
  ReturnŠl1Ć
Ć
```

Može se primetiti da je kriterijum za prekid izvršenja unutrašnje petlje dat u obliku konjukcije dva poređenja. Drugo poređenje se ne može izbeći, ali bi se prvo moglo izbeći ako ispred prvog elementa sortiranog dela dodamo još jedan element i izjednačimo taj element sa elementom koji upravo umećemo u sortirani deo. Na taj način čemo prvo poređenje izbeći jer će u delu niza sigurno postojati element od koga upravo obrađivani element niza nije manji.

```
InsertionsortŠl_Ć:=
  ModuleŠšl1=l,i,j,k,n=LenghtŠlĆć,
    l1=AppendŠl1,l1ŠŠ1ĆĆĆ;
    ForŠi=3,i<=n+1,i++,
      ForŠl1ŠŠ1ĆĆ=l1ŠŠj=iĆĆ,l1ŠŠjĆĆ<l1ŠŠj-1ĆĆ,j--,
        k=l1ŠŠjĆĆ; l1ŠŠjĆĆ=l1ŠŠj-1ĆĆ; l1ŠŠj-1ĆĆ=k;
      Ć
    Ć;
    ReturnŠl1=DeleteŠl1,1ĆĆ;
  Ć
```

1.3. SORTIRANJE POREđENJEM PAROVA UZASTOPNIH ELEMENATA

U postupku sortiranja poređenjem parova uzastopnih elemenata (*Bublle sort*) porede se svaka dva elementa niza, počev od njegovog prvog elementa. Ako je u nekom paru uzastopnih elemenata predhodnik veći od sledbenika, ta dva elementa razmenjuju mesta u nizu. Ovakvim poređenjem svih parova susednih elemenata, najveći od njih će "isplivati" na kraj niza. Zbog toga se i zove metod mehurova ili bublle sort.

Na primer, za niz od 5 elemenata dovoljna su četiri prolaza da bi se ovaj niz sortirao mehanizmom *bublle sort*. U prvom prolazu se porede svih pet elemenata, i najveci dolazi na petu poziciju. Kako se sada največi broj nalazi na kraj niza, u sledečem prolazu treba porediti samo prva četiri elementa. Ovaj postupak traje do poređenja prva dva elementa, posle čega je niz sortiran.
Za niz sa elementima 2,7,5,8,1 proces sortiranja bi se odvijao na sledeći način:

nesortiran niz: 2 7 5 8 1
I prolaz: 2 7<->5 8 1
2 5 7 8 1
2 5 7 1 8
II prolaz: 2 5 7<->1 8
2 5 1 7 8
III prolaz: 2 5<->1 7 8
2 1 5 7 8
IV prolaz: 2<->1 5 7 8

Odgovarajuća funkcija izgleda ovako:

```
BubllesortŠl_Ć:=
 ModuleŠšl1=l,n=LenghtŠlĆ,i,j,kć,
   ForŠi=n,i>1,i--,
    ForŠj=1,j<i,j++,
      IfŠl1ŠŠjĆĆ>l1ŠŠj+1ĆĆ,
         k=l1ŠŠjĆĆ;l1ŠŠjĆĆ=l1ŠŠj+1ĆĆ; l1ŠŠj+1ĆĆ=k
   Ć Ć  Ć;
   ReturnŠl1Ć
  Ć
```

Poređenja se mogu vršiti počev od poslednjeg para. U tom slučaju u prvom prolazu određujemo najmanji element niza (prvi), u sledećem najmanji od preostalih (drugi) itd. Može se desiti da u jednom izvršavanju spoljašnje petlje ne obavimo ni jednu zamenu vrednosti para elemenata. To će značiti da je niz vreć sortiran. Dalje sortiranje je suvišno, odnosno sortiranje treba prekinuti. Stoga se može uvesti indikator koji će nam govoriti o tome da li smo u poslednjem izvršavanju petlje imali zamene.

```
BubllesortŠl_Ć:=
 ModuleŠšl1=l,n=LengthŠlĆ,i,j,k,bć,
   ForŠb=True;i=n,b&&(i>1),i++,
     ForŠj=1;b=False,j<i,j++,
       IfŠl1ŠŠjĆĆ>l1ŠŠj+1ĆĆ,
         k=l1ŠŠjĆĆ; l1ŠŠjĆĆ=l1ŠŠj+1ĆĆ;l1ŠŠj+1ĆĆ=k,
         b=True
   Ć Ć Ć;
   ReturnŠl1Ć
Ć
```

1.5. SORTIRANJE DELJENJEM NIZA

U algoritmu sortiranja deljenjem niza (*quick sort*) postupak sortiranja se sastoji iz dva dela. U prvom delu se niz deli na dva dela. Jedan deo čine elementi koji su manji ili jednaki nekoj unapred zadatoj vrednosti (obično je to neki element niza, prvi, srednji ili poslednji). Drugi deo čine svi oni elementi veći od od zadate vrednosti. Za delitelj niza je odabran prvi njegov element.

Deljenje zadatog niza se obavlja na taj način što uzimamo jedan po jedan element niza i dodajemo odgovarajučem delu niza. Tako u proizvoljnom trenutku možemo imati tri dela niza:

- elemente koje još nismo obradili (od $(i+1)$-og do poslednjeg);
- elemente koji su obrađeni i čine grupu manjih ili jednakih elemenata niza (od prvog do j-tog) i
- elemente koji su obrađeni i čine grupu večih elemenata niza (od $(j+1)$-og do i-tog člana niza).

7	4	9	10	1	2	16	8	3	14
7	4	9	10	1	2	16	8	3	14
7	4	9	10	1	2	16	8	3	14
7	4	9	10	1	2	16	8	3	14
7	4	1	10	9	2	16	8	3	14
7	4	1	2	9	10	16	8	3	14
7	4	1	2	9	10	16	8	3	14
7	4	1	2	9	10	16	8	3	14
7	4	1	2	3	10	16	8	9	14
7	4	1	2	3	10	16	8	9	14
3	4	1	2	7	10	16	8	9	14

Slika 1

Ako sledeći element pripada grupi manjih ili jednakih, prvi element iz grupe većih ((j+1)-vi u nizu) zamenjuje mesto sa analiziranim ((i+1)-im). Tako je ((i+1)-vi) dodat grupi manjih ili jednakih, i ona je uvećana za jedan element (povećava se vrednost promenljive j).

Ako sledeći element pripada grupi većih, tada se samo grupa većih uvećava za jedan element, dok grupa manjih ostaje ista (ne menja se vrednost promenljive j).

Nakon obrade svih elemeneta niz, se sastoji iz samo dva dela: grupa manjih (ili jednakih) i grupa većih (grupa elemenata koji nisu obrađeni je postala prazna). Potrebno je nultom i i-om elementu zameniti mesta. Tako će svi elementi levo od j-og biti manji ili jednaki od njega a svi desno veći od njega.

Sada se može napisati funkcija *Deli* koja deli niz na dva dela:

```
DeliŠl_Ć:=
 ModuleŠšl1=šć,l2=šć,el=FirstŠlĆ,ić,
  ForŠi=1,i<=LonghtŠlĆ,i++,
    IfŠlŠŠiĆĆ<el,
      l1=AppendŠl1,lŠŠiĆĆ, l2=AppendŠl2,lŠŠiĆĆ
  Ć Ć;
  ReturnŠšl1,l2ćĆ
 Ć
```

U funkciji *deli* delitelj *el* je prvi element niza.

Sada se za ovako defiisanu funkvciju *Deli* može napraviti funkciju *Quicksort* oblika:

```
QuisksortŠl1_Ć:=
   ModuleŠšl=l1,m=šć,v=šć,el=FirstŠlĆ,pom=šćć,
     IfŠLenghtŠlĆ<=1,ReturnŠlĆ,
```

```
        el=lŠŠ1ĆĆ;  l=DropŠl,1Ć;  pom=DeliŠl,elĆ;
        m=pomŠŠ1ĆĆ;       v=pomŠŠ1ĆĆ;
        m=QuicksortŠmĆ;    v=QuicksortŠvĆ;
        l=AppendŠm,šelć,vĆ;
        ReturnŠlĆ
      Ć
   Ć
```

1.6. SORTIRANJE KORIŠĆENJEM STEKA

Naziv sortiranja *korišćenjem steka* (engl. *Steck sort*) dolazi otud što se u postupku sortiranja koriste dva steka. Stek je implementiran pomoću jednodimenzionalnog niza i pokazivača na vrh steka (indeks elementa u kome se nalazi poslednji smešteni element u stek). Sortiranje se obavlja tako što se elementi niza, jedan po jedan, dodaju u jedan od stekova tako da važi:

(i) U prvi stek elementi su poređani od dna ka vrhu steka u neopadajući poredak.

(ii) U drugi stek je obrnuto, od vrha ka dnu steka elementi su poređani u neopadajući poredak.

(iii) U drugom steku su svi elementi veći od elemenata u prvom steku (dovoljno je da je element na vrhu drugog steka veći ili jednak elementu na vrhu prvog seka).

Dodavanje elemenata se postiže tako što se iz jednog u drugi stek prepisuju elementi, tako da se dobije:

(i) Na vrh prvog steka (ako nije prazan) element veći ili jednak elementu koji se dodaje.

(ii) Na vrhu drugog steka (ako nije prazan) elementi veći ili jednaki elementu niza koji se dodaje.

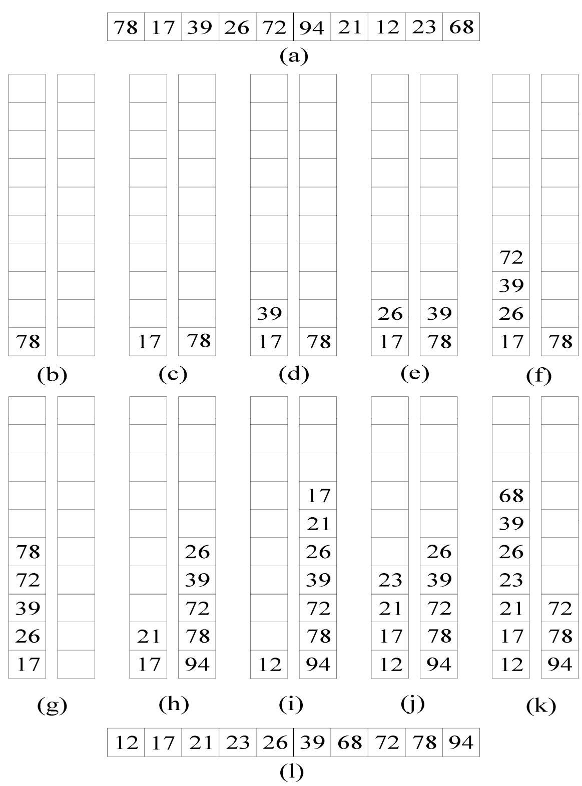
Tada se element može dodati na vrh bilo kog od dva steka.

Odgovarajuću funkciju možemo napisati na sledeći način:

```
StackSortŠl_Ć:=
  ModuleŠšpoml=l,n=LengthŠlĆ,stekm=šć,stekv=šć,i,j,
        vrhm=0, vrhv=0ć,
    ForŠi=1,i<=n,i++,
      WhileŠ(vrhm!=0)&&(stekmŠŠvrhmĆĆ>pomlŠŠiĆĆ),
       vrhv ++ ;
       stekv=AppendŠstekv,stekmŠŠvrhm--ĆĆ Ć;
stekm=DeleteŠstekm,vrhm+1Ć
      Ć ;
      WhileŠ(vrhv!=0)&&(stekvŠŠvrhvĆĆ<pomlŠŠiĆĆ),
       vrhm ++ ;
       stekm=AppendŠstekm,stekvŠŠvrhv--ĆĆĆ ;
```

```
    stekv=DeleteŠstekv,vrhv+1Ć
   Ć ;
   vrhm ++ ;    stekm=AppendŠstekm,pomlŠŠiĆĆ Ć ;
 Ć;
 poml=šć;
 ForŠi=vrhm, i>0, i--,
   poml=PrependŠpoml, stekmŠŠiĆĆ Ć
 Ć;
 PrintŠ"stekm = ",stekmĆ; PrintŠ"stekv = ", stekvĆ;
 ForŠi=vrhv; j=vrhm+1, i>0, i--, j++;
 poml=AppendŠpoml, stekvŠŠiĆĆĆĆ;
 ReturnŠpomlĆ
Ć
```

Na slici 2 je prikazan postupak sortiranja niza brojeva odnosno postupak dodavanja elemenata u stekove i situacija po dodavanju svih elemenata.



Slika 2

1.7. SORTIRANJE KORIŠĆENJEM DRVETA

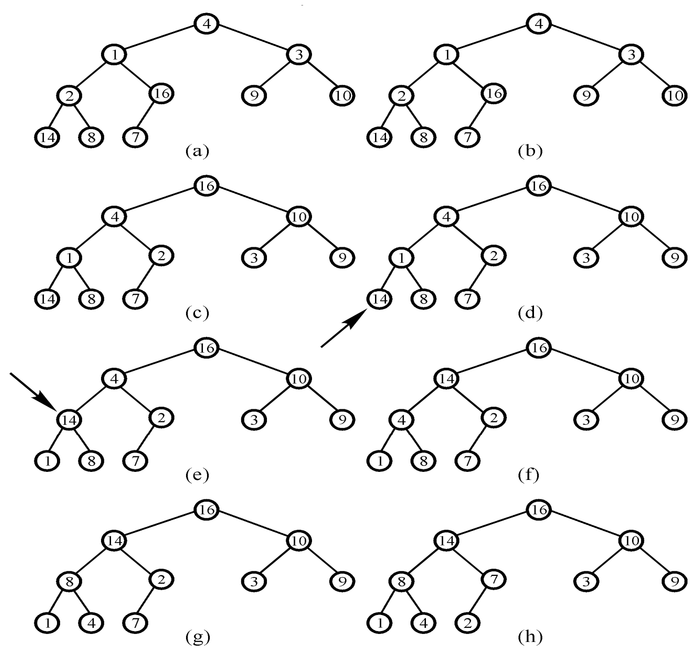
Sortiranje korišćenjem drveta (engl. *Heap sort*) je postupak sortiranja putem binarnog drveta. U toku sortiranja se formira binarno drvo za koje važi:

(i) Svi čvorovi osim čvorova na preposlednjem nivou imaju naslednika.

(ii) određen broj krajnje levih čvorova na predposlednjem nivou ima oba naslednika, zatim eventualno jedan čvor može imati samo levog naslednika (postoji najviše jedan čvor i to isključivo sa levim naslednikom) i preostali čvorovi nemaju naslednika.

(iii) Svim čvorovima su pridružene vrednosti iz niza koji sortiramo i pri tome je vrednost pridružena proizvoljnom čvoru veća (ili jednaka) od vrednosti pridruženih njegovim naslednicima.

Na slici 3 prikazano je jedno binarno drvo sa navedenim svojstvima koje ima 10 čvorova.



Slika 3

Svako drvo koje zadovoljava prva dva svojstva vrlo lako se predstavlja pomoću jednodimenzionalnog niza:

(i) vrednost pridružena nekom čvoru odgovara prvom elementu niza;

(ii) ako je vrednost pridružena nekom čvoru drveta zapisana u *i*-tom elementu niza, onda je vrednost pridružena njegovom levom nasledniku u (2**i*)-tom elementu niza, a vrednost pridružena desnom nasledniku u (2**i*+1)-om elementu niza.

Intuitivno, u niz se upisuje vrednost pridrižena korenu, zatim vrednost pridružena čvorovima sa sledećih nivoa sleva nadesno. Tako se drvo koje ima *n* čvorova predstavlja pomoću niza koji ima *n*-elemenata. Drvo prikazano na slici 3(a) predstvlja se pomoću sledećeg niza 4, 1, 3, 2, 16, 9, 10, 14, 8, 7.

Vrlo lako se računa i element niza u kome je zapisana vrednost pridružena čvoru roditelja datog čvora: roditelj čvora zapisanog u *i*-tom elementu niza ($i>0$) je zapisan u *j*-tom elementu niza gde je *j*=Š(*i*-1)/2Ć.

Za formiranje odnosno predstavljanje heap-stabla koje se formira u tuku sortiranja, koristi se niz koji se sortira. Drvo se formira tako sto se u njega dodaje jedan po jedan element. U proizvoljnom trenutku niz možemo podeliti u dva dela. Prvi deo čine elementi koji su uključeni u drvo, i taj deo niza predstavlja drvo sastavljeno od tih elemenata. Preostali elementi niza čine drugi deo niza, i ti elementi još uvek nisu uključeni u drvo. Na slici 3 je prikazana situacija posle uključivanja prvih 5(b), odnosno prvih sedam (c) elemenata. Kako sledeći element niza dodajemo u stablo? Za početak taj element se postavlja na prvo slobodno mesto u stablu. To je:

- krajnje levo slobodno mesto u poslednjem nivou, ili
- krajnje levo mesto u sledećem nivou, ako je poslednji nivo potpono popunjen.

U predstavljanju drveta pomoću niza taj element dolazi upravo na mesto na kome se trenutno nalazi. Znači, prvi deo niza se povećao za jedan element. Nakon dodavanja tog elemente, stablo ne mora da zadovoljava sva svojstva koja su na početku nabrojana. Može se desiti da vrednost u novododatom čvoru nije manja od vrednosti u čvoru roditelja. Zato se vrši korekcija pomeranjem novododate vrednosti duž puta koji vodi od tog čvora prema korenu, sve dok je ona veća od vrednosti u čvoru roditelja ili dok ne stigne do korena drveta.

Na slici 3(d) je prikazano dodavanje 8. elementa niza (broj 14) drvetu. Taj element će se nalaziti na početku, na krajnje levom mestu u četvrtom nivou. Njegov roditelj je broj 1. Kako je 1 manje od 14 ta dva broja zamenjuju mesta. Nakon te zamene dobijamo drvo kao na slici 3(e). Trenutno je roditelj broja 14 broj 4, pa i ta dva elementa zamenjuju mesta. Posle ovoga dobijamo drvo prikazano na slici 3(f). U tom drvetu roditelj broja 14 (broj 6) veći, te se tog trenutka prekida postupak dodavanja broja 14 u drvo.

Preostale dve slike (1(g) i 1(h)) prikazuje izgled drveta po dodavanju devetog i desetog člana niza. Tako bi funkcija za formiranje drveta za dati niz mogla da se napiše na sledeći način:

```
NapraviHeapŠl_Ć:=
 ModuleŠšpoml=l,n=LengthŠlĆ,i,j,kć,
  ForŠi=2,i<=n,i++,
   j=i;
   WhileŠ(j>1)&&(pomlŠŠjĆĆ>pomlŠŠCielingŠ(j-1)/2ĆĆĆ),
     k=pomlŠŠjĆĆ;
     pomlŠŠjĆĆ=pomlŠŠCelingŠ(j-1)/2ĆĆĆ;
     pomlŠŠCeilingŠ(j-1)/2ĆĆĆ=k;
     j=CeilingŠ(j-1)/2Ć
   Ć
  Ć;
  ReturnŠpomlĆ;
Ć
```

Algoritam sortiranja. Kada se drvo formira potrebno je izvršiti sortiranje. To obavljamo određivanjem jednog po jednog elementa sortiranog niza, od poslednjeg ka prvom. Na poslednjem mestu u sortiranom nizu stajaće najveći element niza. Najveći element niza se nalazi u korenu formiranog drveta (tj. u pvom elementu niza). Kako je ceo niz u tom trenutku iskorišćen za predstavljanje stabla, to je i element koji je na poslednjem mestu takođe u stablu i treba ga zadržati u ostatku drveta. Jedino slobodno mesto u stablu je trenutno oslobođeni koren. Znači, za početak ćemo zameniti prvi i poslednji element niza. Na taj način smo niz podelili na dva dela:

- poslednji element niza,koji je istovremeno i najveći element niza,
- svi elementi osimposlednjeg koji obrazuju binarno drvo.

Međutim nakon premeštanja poslednjeg elementa u koren stabla, dobijamo drvo koje ne zadovoljava sva navedena svojstva. U suštini svojstvo (iii) ne mora biti zadovoljeno. Da bismo to otklonili, vršimo određene izmene u drvetu. U korenu mora stajati najveća vrednost. Najveća vrednost u levom poddrvetu se nalazi u levom nasledniku korena. Slično važi i za desno poddrvo. Zato u korenu treba staviti najveću od sledeće tri vrednosti:

- vrednost koja je trenutno u korenu,
- vrednost u levom nasledniku i desnom nasledniku.

Ukoliko najveća vrednost nije ona koja se nalazi u korenu, tada koren i čvor u kome se nalazi najveća vrednost razmenjuju vrednost. Nakon te zamene čvor u koji je prešla vrednost iz korena je mesto na kome ne mora biti zadovoljeno svojstvo (iii). Stoga opisani postupak ponavljamo sve dok se vrednost koja je na početku bila u korenu kreće kroz drvo, odnosno dok postoji naslednik sa većom vrednošću. Prema tome, Evo kako će izgledati funkcija koja *Heap* (stablo) prevodi u sortirani niz:

```
IzHeapaUNizŠl_Ć:=
  ModuleŠšpoml=l,n=LengthŠlĆ,i,j,kć,
    ForŠi=n,i>1,i--,
      poml=ZameniŠpoml,i,1Ć;
       j=1;  LabelŠlabĆ;  k=j;
       IfŠ(2*k<i)&&(pmlŠŠ2*kĆĆ>pomlŠŠjĆĆ),j=2*kĆ;
       IfŠ(2*k+1<i)&&(pomlŠŠ2*k+1ĆĆ>pomlŠŠjĆĆ),
          j=2*k+1
       Ć;
       IfŠj!=k,poml=ZameniŠpoml,j,kĆĆ;
       IfŠj!=k, GotoŠlabĆĆ;
    Ć;
    ReturnŠpmlĆ;
  Ć
```

Pri tom je korišćena funkcija *Zameni*Š*l_,i_,j_*Ć koja u nizu *l* vrši zamenu *i*-tog i *j*-tog elementa. Ubacili smo je da bi program učinili kraćim i preglednijim. A evo i definicije:

```
ZameniŠl_,i_,j_Ć:=
 ModuleŠšpoml=l,kć,
   k=pomlŠŠiĆĆ;pomlŠŠiĆĆ=pomlŠŠjĆĆ; pomlŠŠjĆĆ=k;
   ReturnŠpomlĆ;
 Ć
```

Sada nam samo ostaje da napravimo funkciju koja će objediniti predhodne funkcije u jedan program:

```
HeapSortŠl_Ć:=
  ModuleŠšpoml=lć,
    ReturnŠIzHeapaUNizŠNapraviHeapŠpomlĆĆ Ć;
  Ć
```

1.8. SHELL-SORT

Ovaj način sortiranja nije efikasan poput postupaka *quick* ili *heap* ali pobuđuje pažnju. Na primer, nekim drugim postupkom sortiramo podnizove polaznog niza koji se dobijaju uzimanjem, na primer, svakog četvrtog elementa (imamo četiri takva podniza: svaki četvrti počev od nultog, prvog, drugog ili trećeg). Zatim, u tako modifikovanom nizu, sortiramo podnizove koje dobijamo izdvajanjem svakog drugo (takvih ima dva). Konačno, u trećem prolazu sortiramo ceo niz, tj. podniz koji se dobija od polaznog uzimanjem svih njegovih elemenata.

Dat je postupak sortiranja niza 44, 55, 12, 42, 94, 18, 06, 67:
44, 55, 12, 42, 94, 18, 06, 67
44, 18, 06, 42, 94, 55, 12, 67,
Posle drugog prolaza:
06, 18, 12, 42, 44, 55, 94, 67,
i konačno, posle trećeg prolaza:
06, 12, 18, 42, 44, 55, 67, 94.
Postavlja se pitanje: da li je potrebno toliko prolaza i šta se dobija ovim postupkom. Analize pokazuju da se u prvim prolazima sortiraju vrlo mali podnizovi (imaju malo elemenata), a da kasnije raste broj elemenata u podnizovima koje sortiramo. Međutim podnizovi su uglavnom sortirani tako da je broj premeštanja elemenata niza relativno mali, čime se opet nešto dobija.
U našem primeru uzeli smo da je rastojanje između elemenata koji se sortiraju bilo 4, pa 2, zatim 1, tj. ono se prepolovljavalo. Jasno može se eksporimentisati sa bilo kojim opadajućim nizom rastojanja između elemenata koji se sortiraju,tako da poslednje rastojanje bude 1. Dakle ako su rastojanja redom $h1,h2,...hp$ dovoljno je uzeti je h1=1 i $hi+1<hi$ $(i=1,...T-1)$, i imaćemo ispravan postupak za sortiranje. Sortiranje podnizova obavljamo umetanjem sledećeg elementa podniza u sortirani deo na odgovarajuće mesto. Neka smo sortirali deo podniza:

$$ai, ai+d, ai+2d,...,ai+j*d.$$

Element š$ai+(j+1)*d$ć dodajemo tako što ga pomeramo duž već sortiranog dela, sve dok je manji od svog predhodnika. Naravno, potrebno je kontrolisati da li smo stigli do početka podniza. Da bismo izbegli tu kontrolu, dodajemo jedan lažni element na početak niza:

$$ai-d=ai+(j+1)*d.$$

Na taj način pomeranje elemenata elementa $ai+(j+1)*d$ će biti prekinuto onog trenutka kada dođe neposredno iza $ai-d$ (tj. na i-to mesto). Zaista, $i-d$ je negativno. Stoga, pre sortiranja niz a pomerimo za toliko mesta tako da $i-d$ ne može biti negativno. To znači da ga je dovoljno pomeriti za najveću moguću vrednost koraka ($h1$ mesta).

Tako naša funkcija može biti zapisana na sledeći način:
```
shellSortŠl1_,d1_Č:=
 ModuleŠšd=d1,l=l1,i,j,k,pom,nć,
   n=LengthŠlČ;
   WhileŠd>0,
    ForŠk=0,k<d,k++,
     ForŠi=k,i<n-1,i+=d,
```

```
    ForŠj=i+d,j<n,j+=d,
      IfŠlŠŠiĆĆ>lŠŠjĆĆ,
       pom=lŠŠiĆĆ; lŠŠiĆĆ=lŠŠjĆĆ; lŠŠjĆ=pom
  ĆĆĆ Ć;
  d=QuotientŠd,2Ć
 Ć;
 ReturnŠlĆ    Ć
```

Na kraju se mogu dati rezultati dobijeni ispitivanjem brzina rada svakog od navedenih sortiranja. U *MATHEMATICA* je to moguće uraditi korišćenjem funkcije *Timing*. Za formiranje proizvoljnog niza realnih brojeva može se iskoristiti funkcija *Random* koja bira pseudoslučajne brojeve i formira niz. U ovom slučaju je uzeto da su elementi datih nizova realni brojevi. Ispitivanje je vršeno za nizove sa po 10000, 15000 i 20000 elemenata.

	Broj elemenata niza		
	10000	15000	20000
Selectsort	0.1	0.11	0.11
ModSelectsort	0.05	0.11	0.11
Insertion	0.05	0.11	0.11
ModInsertion	0.05	0.1	0.11
Stecksort	0.05	0.1	0.11
Shellsort	0.06	0.11	0.11
Heapsort	0.06	0.11	0.11
Bubllesort	/	0.05	0.06
MobBubllesort	0.05	0.11	0.11
Quicksort	0.05	0.1	0.11
Shellsort	/	0.05	0.11

Slika 5

Rezultati u tabeli na slici 5 predstavljaju vreme izvršenja (dato u sekundama) datih sortiranja.

2. REPETITIVNA PRIMENA FUNKCIJE KAO ARGUMENTA

Često puta je potrebno da se abstrahuje šablon izračunavanja, postavi kao argument i koristi u seriji različitih okruženja. Neke od opisanih funkcija u ovom poglavlju su sadržane u prethodnoj glavi. Cilj ovog poglavlja je da se detaljno izuči problem višestruke primene funkcionalnog argumenta. Ovaj problem je izučavan u radu Š10Ć.

Najpoznatiji primer repetitivne primene funkcija koje predstavljaju argumente jeste tzv. *funkcionalni maping* (*function mapping*) Š1Ć, Š2Ć, Š3Ć, Š9Ć, Š11Ć, Š12Ć. Generalno, funkcionalni mapping, ili skraćeno *mapper*, je

funkcija koja ima sposobnost da sekvencijalno primenjuje drugu funkciju *f* na nekoliko listi koje sadrže aktuelne argumente za *f*. Svaka lista argumenata odgovara jednom parametru za *f*. Ovakvih listi bi trebalo da bude onoliko koliko argumenata zahteva funkcija *f*. Mapper radi prema sledećem algoritmu: odabira jedan element iz svake liste argumenata i predaje ove elemente funkciji *f*. Rezultati ovakvih primena grupišu se na određeni način. Procesiranje izraza koji su dobijeni primenom funkcionalnog argumenta zavisi od konkretnog mapera. Zatim se maper pomera na ostatak svake liste argumenata. Proces se prekida kada se iskoriste evi elementi iz listi argumenata. Poznat je veliki broj mapera u različitim programskim jezicima: *MATHEMATICA* Š7Ć, Š13Ć, *LISP* Š1Ć, Š2Ć, Š3Ć, Š9Ć, Š11Ć, Š12Ć, *APL* Š8Ć, *HASKELL* Š4Ć, Š5Ć.
Takođe, izučava se repetitivna primena funkcionalnog argumenta na delovima listi ili izraza: primene na specificiranim delovima, kao i primena na specificiranim nivoima izraza ili liste.
Sledeći slučaj mapera jeste iterativna primena funkcije kao argumenta, koja je bazirana na izračunavanjima koja zavise od metoda fiksne tačke.
Takođe, konstrukcija tabela vrednosti neke funkcije *f*, koja predstavlja argument mapera, bazirana je na repetitvnom pozivu funkcije *f*. Argumenti funkcije *f* mogu biti obezbeđeni na više različitih načina, a mogu se konstruisati tabele različitih dimenzija.
Konačno, izučava se repetitivna primena funkcionalnog argumenta koja je zasnovana na distributivnosti i linearnosti.
COMMON LISP Š3Ć, Š6Ć, Š12Ć i *WALTZ LISP* Š2Ć podržavaju dva opšta tipa mapera: element maperi i rep (*tail*) maperi. Ovi maperi se razlikuju prema načinu na koji se obezbeđuju argumenti funkcionalnom parametru. Element mapperi odabiraju jedan element iz svake liste elemenata, dok rep maperi (tail maperi) koriste čitav ostatak svake liste argumenata.

Generička definicija element mappera u *LISP*u je (vidi Š2Ć):

```
(defun ELEMENT-MAPPER (f arg1 ... argN)
 (process-func
  '((funcall f (car (nth 1 arg1))...(car (nth 1 argN)))
             ... ... ...
   (funcall f (car (nth k arg1))...(car (nth k argN)))
 )))
```

Poznata funkcija *mapcar* je element maper u kome je *process-func=list*. Maperi *mapc* i *mapcan* jesu element maperi u kojima je *process-func=progn* i *process-func=nconc*, respektivno.
Generička definicija rep mappera u *LISP*u je (vidi Š2Ć):

```
(define (TAIL-MAPPER f arg1 ... argN)
  (process-func
   '( (funcall f (nth 1 arg1)...(nth 1 argN))
              ...  ...  ...
     (funcall f (nth k arg1)...(nth k argN)))))
```

Funkcija *maplist* je rep maper u kome je *process-func=list*. Maperi *mapc* i *mapcan* jesu rep maperi u kojima je *process-func=progn* i *process-func=nconc*, respektivno.

U *MATHEMATICA* su ugrađeni jedino element-mapperi.

2.1. ELEMENT MAPPERI

Kada posedujete listu argumenata, često puta je potrebno da se neka funkcija primeni na svaki od njeniih elemenata posebno. Element maperi u *MATHEMATICA* su *Map* (za jedno argumentne funkcije) i *MapThread* (za višeargumentne funkcije).

MapŠf,ša,b,...ć Ć	primenjuje funkciju *f* na svaki element u listi š*a,b,*...ć, i produkuje listu šfŠaĆ,fŠbĆ,...ć.

Funkcija *Map* se može primeniti na proizvoljni izraz, ne samo na liste, zbog toga što su liste u *MATHEMATICA* jesu parcijalni slučaj opšte forme izraza, čija je glava jednaka *List*. Koristeći *head*Š*x,y,...*Ć kao prototip izraza, možemo pisati

*Map*Š*f, head*Š*x,y,...*ĆĆ=*head*Š*f*Š*x*Ć,*f*Š*y*Ć, ..Ć.

MapThreadŠf,šša$_1$,a$_2$,...ć,šb$_2$,b$_2$, ...ć,...ćĆ	rezultat je lista š*f*Ša_1,b_1,...Ć, *f*Ša_2,b_2, ...Ć,...ć

Ova funkcija je analogna *LISP* maperu *mapcar*.

ThreadŠfŠargsĆ Ć	primenjuje *f* na svaku listu koja se pojavljuje u *args*
SelectŠlist, fĆ	selektuje elemente liste koristeći funkciju *f* kao kriterijum

Izraz *ThreadŠfŠargsĆĆ* je ekvivalentan sa *MapThread*Š*f,args*Ć.

Izraz *SelectŠlist,fĆ* primenjuje *f* na svaki element liste, i za krajni rezultat ostavlja samo one koji daju rezultat *True*. Objekat *list* može da ima proizvoljnu glavu, ne samo *List*, tako da *Select*Š*expr,f*Ć selektuje elemente iz *expr* za koje funkcija *f* daje rezultat *True*. Funkcija *Select* je analogna *LISP* funkciji *subset*, u slučaju jednoargumentnih funkcija.

ScanŠf, exprĆ	primenjuje *f* na svaki deo izraza *expr,* ali ne konstruiše novi izraz

2.2. PRIMENA FUNKCIJA NA DELOVIMA LISTI I IZRAZA

Jedan od problema u repetitivnoj primeni funkcija kao argumenata jeste da se njihova primena ograniči na određene elemente liste ili izraza. Taj problem se može rešiti kombinacijom mapinga i upravljanjem toka izvršenja programa.

2.2.1. Primena na selektovanim elementima

Delovi liste ili izraza nad kojima se se primenjuje funkcionalni argument mogu biti eksplicitno zadati u listama koje sadrže indekse.

MapAtŠf,expr,šši_1,j_1,...ć, ši_2, j_2,...ć,...ćĆ	primenjuje *f* na delove izraza *expr* u pozicijama *expr*ŠŠi_1,j_1,...ĆĆ, *expr*ŠŠi_2,j_2,...ĆĆ,…

Takođe, delovi liste ili izraza u kojima se funkcija primenjuje mogu bii specifrani proizvoljnim svojstvom.

ThreadŠfŠargsĆ, hĆ	primenjuje *f* na svaki objekat sa glavom *h* koji se pojavljuje u *args*
ThreadŠfŠargsĆ, h, nĆ	primenjuje *f* na svaki objekat sa glavom h koji se pojavljuje u prvih *n* elemenata u *args* sa glavom *h*
ThreadŠfŠargsĆ, h, -nĆ	primenjuje *f* na poslednjih *n* elemenata u *args* sa glavom *h*
ThreadŠfŠargsĆ, h, šm,nćĆ	primenjuje *f* na argumentima od pozicija *m* do *n*

2.2.2. Primena na selektovanim nivoima izraza

Specifikacija nivoa dozvoljava da koristite funkciju *Map* na željenim nivoima izraza na koji želite da primenite funkciju. Specifikacija nivoa je opisana u prethodnoj glavi.

MapŠf, expr, levelĆ	primenjuje *f* na delove izraza *expr* koji su određeni argumentom (default vrenost za *level* je š1ć)
MapThreadŠf, šexpr1, expr2, ...ć, levelĆ	primenjuje *f* na delove izraza *expr*1, ... koji su određeni sa *level*
MapAllŠf, exprĆ ili f //Ž expr	primenjuje *f* na svaki podizraz u *expr*. Izraz je ekvivalentan sa *MapŠf,expr, š0,InfinityćĆ*
MapAllŠf, expr, Heads->TrueĆ	primenjuje *f* unutar glava u delovima izraza *expr*
ApplyŠf, expr, levelĆ	primenjuje *f* u nivoima izraza *expr* koji su određeni parametrom *level*
MapIndexedŠf, expr, levelĆ	primennjuje *f* na delovima izraza *expr* koji su određeni argumentom *level*, uzimajući listu indeksa za svaki deo kao drugi argumentza *f*
ScanŠf, expr, levelĆ	primenjuje *f* na delovima izraza *expr* u nivoima koji su određeni izrazom *level*, ali ne konstruiše novi izraz.

2.2.3. Primena funkcije na nepoznatom delu liste ili izraza

Ovi operatori primenjuju funkcionalni argument na nepoznati deo date liste ili izraza. Nije unapred poznat broj primena funkcionalnog argumenta.

SelectŠexpr, f, nĆ	selekcija prviht *n* elemenata u *expr* za koje funkcija *f* daje *True*

2.3. REPETITIVNA PRIMENA NA FUNKCIONALNOM ARGUMENTU

U ovom slučaju aplikativni operatori primenjuju argument tipa *Function,* repetitivno, koristeći prethodni rezultat kao argument. Funkcinalne operacije *Nest, Nestlist, Fold* i *FoldList* imaju neku funkciju *f* kao jedan od svojih argumenata, i primenjuju je repetitivno. U svakom koraku se koristi rezultat iz prethodnog koraka kao novi argument za *f*.

NestŠf, expr, nĆ	rezultat je izraz dobijen primenom *f* na izraz *expr* *n* puta
NestListŠf, expr, nĆ	rezultat je lista rezultata primene *f* na *expr* od 0 do *n* puta
FoldListŠf, x, ša,b,...ćĆ	kreira listu š*x*, *f*Š*x*,*a*Ć, *f*Š*f*Š*x*,*a*Ć,*b*Ć,...ć
FoldŠf, x, ša,b,...ćĆ	poslednji element liste koja je generizana izrazom *FoldList*Š*f, x,* š*a*,*b*,...ćĆ

Izračunavanja koja repetitivno primenjuju neku funkciju ili proces na njen nepoznati izlaz sve dok ulaz ne postane jednak izlazu, nazivaju se izračunavanja bazirana na metodu fiksne tačke (*fixed-point computations*). Funkcionalne operacije *FixedPoint* i *FixedPointList* su bazirane na ovakvom stilu izračunavanja.

FixedPointŠf, exprĆ	počevši od *expr*, primenjuje se *f* sve dok se rezultat menja
FixedPointListŠf, exprĆ	generiše listu repetitivnih primena funcije *f,* počevši od *expr,* sve dok se rezultat menja
FixedPointŠf, expr, SameTest->compĆ	prekinuti primenu kada primena funkcije *comp* na dve uzastopna rezultata daje *True*

2.4. TABELE VREDNOSTI

U ovom slučaju liste ili izrazi koji sadrže argumente za funkciju zadatu kao argument, nisu eksplicitno zadati. Argumenti za funkcionalni argument u višedimenzionalnim tabelama su specificirani koristeći standardnu notaciju.

ArrayŠf,nĆ	generiše listu š*f*Š1Ć,*f*Š2Ć,...ć dužine *n*
ArrayŠf, dims, originĆ	generiše listu koristeći *origin* da odredi indeks (default vrednost je 1)
ArrayŠf, dims, origin, hĆ	koristi glavu *h* umesto *List*, za svaki nivo
Array Šf,šn_1, n_2, ... ćĆ	generiše $n_1 \times n_2$... ugnežđenu listu sa elementima *f*ŠŠi_1,i_2,...ĆĆ
TableŠf, šimaxćĆ	generiše listu od *imax* vrednosti funkcije *f*
TableŠf,ši,imaxćĆ	generiše listu od vrednosti za *f* pri čemu *i*

	uzima vrednosti od 1 do *imax*
TableŠf, ši,imin,imaxćĆ	počinje sa *i=imin*
TableŠf, ši,imin,imax,dićĆ	koristi korake *di*
TableŠf,ši,imin,imax,dić, šj,jmin,jmax,đć,...Ć	generiše višedimenzionalnu tabelu
InnerŠf, list1, list2, gĆ	generalisani unutrašnji proizvod
OuterŠf, list1, list2,...Ć	generalisani spoljašnji proizvod. Uzima sve moguće kombinacije elemenata iz *list*1, *list*2,... i kombinuje ih pomoću *f*
FoldListŠf,x,ša$_1$, a$_2$,...ćĆ	rezultat je lista *šx, fŠx,a$_1$Ć, fŠfŠx,a$_1$Ć, a$_2$Ć,...ć.*
FoldŠf, x, ša$_1$, a$_2$, ... ćĆ	poslednji element izraza *FoldListŠf,x,ša$_1$,a$_2$,...ćĆ*
MapIndexedŠf, exprĆ	primeniti *f* na elemente izraza *expr*, pri čemu se pozicije svakog elementa koriste kao drugi argument za *f*

2.5. DISTRIBUTIVNOST I ASOCIJATIVNOST

Funkcija *Distribute* dozvoljava da se kontrolišu neke osobine funkcijionalnog argumenta, kao na primer distributivnost i linearnost.

DistributeŠfŠa$_1$+b$_1$+..., a$_2$+b$_2$+..., ...ĆĆ	distribuira *f* u odnosu na + i daje rezultat *fŠa$_1$,a$_2$,...Ć+fŠb$_1$,b$_2$,...Ć+...*
DistributeŠfŠargsĆ,gĆ	distribuira *f* u odnosu na sve argumente sa glavom *g*
DistributeŠexpr, g, f, gp, fpĆ	distribuira *f* u odnosu na *g*, zamenjujući ih sa *fp* i *gp*, respektivno.

2.6. O PORETKU U KOME SE ARGUMENTI KORISTE

U svim prethodno opisanim slučajevima repetitivne primene funkcije *f,* koja se koristi ka argument, redosled prosleđivanja argumenata funkciji *f* određivan je implicitno. Podrazumevan je redosled s leva na desno. Pored toga, moguće je da se argumenti obezbeđuju u nekom drugom poretku, na primer zbog povećanja efikasnosti. Poredak je u najvećem broju slučajeva nebitan, ali u nekim slučajevima može da bude kritičan. Takav slučaj se sreće kada evaluacija sadrži bočni efekt. Primena funkcinalnog argumenta *f* s desna na levo, u slučaju jednoargumentne funkcije *f*, izučavana je u Š1Ć. Koristeći ovu ideju, možemo modifikovati generičku definiciju element mapera i tail maper tako da budu sposobni da kontrolišu poredak uzimanja argumenata. Mogu se uzeti tri opšta principa za implementaciju ove ideje:

1. modifikacija procesne funkcije koja generiše rezultat, bez promene redosleda primenje *f*;
2. modikacija redosleda primene funkcije *f*, bez izmene procesne funkcije;
3. modifikacija i redosleda primene *f kao* i procesne funkcije.

2.7. DEFINICIJA REP MAPERA U *MATHEMATICA*

Kao što je poznato u *MATHEMATICA* ne postoje rep-maperi. U *LISP*-u postoje. Po uzoru na rep-mapere iz *LISP*-a, ovde su definisani neki rep-maperi u *MATHEMATICA*.

Rep-maper *Tailmapp* je klasičan rep-maper *maplist* iz *LISP*-a. On, dakle, primenjuje funkciju *f* na svaki rep liste *l*.

Maper *Tailmultimapp* primenjuje *n* puta funkciju *f* na *n*-ti rep liste *l*.

Funkcional *Tailmultimappredos* primenjuje funkcionalni arument na repove liste *l* onoliko puta koliko je to specifirano elementima liste *redosled*.

```
Tailmapp[f_, l_] := If[l== {}, Return[{}],
                     Return[ Prepend[Tailmapp[f, Rest[l]], Apply[f, List[l]]]]]
Tailmapp[Length, {1, 2, 3, 4, 5, 6}]
{6, 5, 4, 3, 2, 1}
Tailmapp[Rest, {1, 2, 3, 4, 5, 6}]
{{2, 3, 4, 5, 6}, {3, 4, 5, 6}, {4, 5, 6}, {5, 6}, {6}, {}}
Tailmultimapp[f_, l_] := Block[{pom = l, res = {}},
                              Do[res = Append[res, Primeni[f, pom, i]];
                                 pom = Rest[pom], {i, 1, Length[l]}];
                              Return[res]]

Primeni[f_, l_, n_] := Block[{pom = l},
                              Do [pom = Apply[f, List[pom]],
                                  {i, 1, n}]; Return[pom]]

Tailmultimapp[First, {1, {1, 2}, {{1, 2}, 2}}]
{1, 1, 1}
```

```
Tailmultimappredos[f_, l_, redosled_] := Block [{pom = l, upo = redosled, res = {}},
                                             Do[
  res = Append[res, Primeni[f, pom, First[upo]]];
                                                 pom = Rest[pom];
                                                 upo = Rest[upo] , {Length [l]}];
                                              Return[res]]
Tailmultimappredos[Rest, {1, 2, 3, 4, 5}, {4, 3, 2, 1, 0}]
{{5}, {5}, {5}, {5}, {5}}
```

LITERATURA

[1] I. Danicic, *LISP programming*, Blackwell Scientific Publications, Oxford, London, Edinburgh, Boston, Melbourne 1985.

[2] J. Foderaro, *The Franz LISP manual*, Franz, Inc., Alameda, California, 1985.

[3] L.W. Henessey, *Common LISP*, McGraw-Hill Book Company, 1989.

[4] P. Hudak and J.H. Fasel, *A gentle introduction to Haskell*, ACM SIGPLAN Notices **27 No 5** (1992), 1—53.

[5] P. Hudak *at all*, *Report on the programming language Haskell*, ACM SIGPLAN Notices **27 No 5** (1992), 1—157.

[6] E. Hyvonen and J. Seppanen, *Introduction to LISP and funkcional programming*, Moskva, "Mir", 1990.

[7] N. Krejić i Đ. Herceg, *Matematika i MATHEMATICA*, Računari u univerzitetskoj praksi, Novi Sad, 1993.

[8] K.N. Samuek, *Programming Languages, an Interpreter-based Approach,* Addison-Wesley publishing company, Inc., 1990.

[9] J.D. Smith, *An Introduction to Scheme*, Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[10] P.S. Stanimirović, S. Rančić and M.B. Tasić, *Repetitive applications of funkcijas as arguments in programming languages,* Proceedings of VIII Conference on Logic and Computer Science, LIRA '97, Novi Sad 1.9.-4.9.1997, 231—238.

[11] W.R. Stark, *LISP, Lore and Logic*, Springer-Verlag, New York, Berlin, Heidelberg, London, Paris, Tokyo 1990.

[12] R. Wilensky, *Common LISPcraft*, Norton, New York, 1986.

[13] S.Wolfram, *Mathematica: a system for doing mathematics by computer*, Addison-Wesley Publishing Co, Redwood City, California, 1991.

3. KORIŠĆENJE TEHNIKE PRETRAŽIVANJA SA VRAĆANJEM

Pretraživanje sa vraćanjem je postupak nalaženja rešenja nekog problema, ne po eksplicitnim pravilima (odnosno relacijama-jednačinama koje povezuju ulazne i izlazne podatke), već nizom proba. Taj niz proba obično se prirodno izražava kroz rekurziju i zahteva rešavanje niza podzadataka. Nakon nalaženja jednog rešenja prvog podzadatka rešavamo skup preostalih podzadataka, pri čemu rešenje već rešenog podzadatka ima uticaja na rešenje preostalih. Ako ne možemo da rešimo ostatak, onda je rešenje prvog podzadatka neodgovarajuće i zato treba probati sa drugim rešenjem. U engleskom jeziku je uobičajen naziv *backtrack* (ili *backtracking*) za ovaj postupak. U nastavku izaganja će termini *backtrack* i *bektrek* značti isto.

3.1. POSTAVLJANJE KRALJICA NA ŠAHOVSKU TABLU

Ilustrujmo postupak na primeru sledećeg zadatka: postaviti 8 kraljica na šahovsku tablu 8x8 tako da se one uzajamno ne napadaju.
Prvo rešenje bi bilo: postaviti 8 kraljica na 8 proizvoljno izabranih polja od ukupno 64 polja i proveriti da li se napadaju. To je moguće izvesti na ukupno

$$\binom{64}{8} = 4\,426\,165\,368$$

načina.
U sledećoj modifikaciji iskoristimo činjenicu da dve kraljice ne mogu stajati u istoj koloni. Zato u svaku kolonu postavimo po jednu kraljicu (znači za svaku kraljicu postoji 8 mogućih pozicija) i proverimo da li se uzajamno napadaju. U tom slučaju bismo imali ukupno 8^8= 16,777,216 mogućnosti što je značjno poboljšanje, ali i dalje vrlo neefikasno.
Dve kraIjice u dvema kolonama ne mogu zauzimati istu poziciju. Stoga pozicije pojedinih kraljica po kolonama moraju biti različiti brojevi u opsegu 1-8 (ili 0-7, zavisno kako numerišemo prvo poIje u koloni). Stoga tih 8 pozicija za 8 kolona obrazuju jednu permutaciju pa je dovoljno da isprobamo sve permutacije od 8 elemenata, a toje ukupno 8! = 40320.
Konačno dolazimo do poslednjeg poboljšanja. Ono se sastoji u sledećem: postavimo kraljicu na neko polje u prvoj koloni i rešavajmo problem: postaviti 7 kraljica u 7 preostalih kolona tako da se:
- medusobno ne napadaju;
- ne napadaju sa već postavljenom kraljicom u prvoj koloni.

Po istoj šemi se i novi problem može podeliti na dva podproblema:

- postaviti kraljicu u drugoj koloni tako da se ne napada sa već postavljenom kraljicom u prvoj koloni;
- postaviti 6 kraljica u poslednjih 6 kolona tako da se međusobno ne napadaju i da se ne napadaju sa već postavljenim dvema kraljicama.

Postupak se može nastaviti do:

- postavljanja svih 8 kraijica;
- do trenutka kada u sledećoj koloni ne budemo mogli da postavimo kraljicu.

U prvom slučaju rešavanje je okončano. U drugom slučaju se vraćamo nazad i neku već postavljenu kraljicu premeštamo na neko drugo mesto, vodeći računa:

- da se kraljica ne napada sa ostalim postavljenim;
- da ne ponovimo neko rešenje koje smo već probali.

Nameće se kao prirodno da prvo treba pomeriti poslednju postavljenu kraljicu, pa ako to ne uspe, prethodnu, itd.

Svako polje na šahovskoj tabli je određeno dvema koordinatama:

- rednim brojem vrste u kojoj se nalaze (prva koordinata);
- rednim brojem kolone u kojoj se nalaze (druga koordinata).

Obe koordinate se kreću u opsegu od 0 do 7. Tako je polje sa koordinatama (0,0) gornje levo polje, dok je polje sa koordinatama (7,7) donje desno polje table.

Postavljanjem kraljice na i-to, polje ($0 \leq i < 8$) u koloni kn ($0 \leq kn < 8$), jedna vrsta i dve dijagonale postaju zauzete. Da je vrsta ili dijagonala zauzeta, znači da u toj vrsti ili dijagonali više ne možemo postaviti nijednu kraljicu. Tako ćemo, u trenutku kada postavimo kraljicu na neko polje, određenim indikatorima naglasiti da su odgovarajuća vrsta, i odgovarajuće dijagonale zauzete. Za to su nam potrebna 3 niza:

- jedan sa 8 elemenata (za vrste);
- dva sa po 15 elemenata (za dijagonale).

Dijagonale smo označili sa *dij45* i *dij35*, u zavisnosti od toga da li zaklapaju ugao od 45 ili 135 stepeni sa pozitivnim smerom x-ose koordinatnog sisterna, koji smo postavili tako da jedno teme (donje levo) table bude u koordinatnom početku.

Za dijagonale koje zaklapaju ugao od 45 stepeni sa pozitivnim smerom važi: sva polja koja pripadaju jednoj dijagonali imaju isti zbir koordinata. Taj zbir se kreće od 0 (za gornje levo polje, odnosno, za dijagonalu koja prolazi kroz gornje levo) do 14 (za donje desno polje).

Slično važi i za dijagonale koje zaklapaju ugao od 135 stepeni: razlika prve i druge koordinate je ista za sva polja na jednoj dijagonali. Razlika se kreće od -7 (gornje desno polje) do 7 (donje levo polje).

Tako dolazimo do ršenja koje možemo ispisati u sledećem obliku:

```
Stampaj:=BlockŠši,jć,
 ForŠi=1,i<9,
  ForŠj=1,j<9,
    IfŠkpŠŠjĆĆ==i,
      sahtablaŠŠj,iĆĆ=1, sahtablaŠŠj,iĆĆ=0Ć;
    j++
  Ć;
  i++
 Ć;
 PrintŠTableFormŠsahtablaĆĆ;

 PostaviŠkn_Ć:=BlockŠšić,
  ForŠi=1,i<9,
    IfŠ(vrŠŠiĆĆ==0)&&(dij45ŠŠi+kn-1ĆĆ==0)&&
               (dij135ŠŠkn-i+8ĆĆ==0),
      IfŠkn<8,
        kpŠŠknĆĆ=i; vrŠŠiĆĆ=1;
        dij45ŠŠi+kn-1ĆĆ=1;dij135ŠŠkn-i+8ĆĆ=1;
        PostaviŠkn+1Ć;  vrŠŠiĆĆ=0;
        dij45ŠŠi+kn-1ĆĆ=0;  dij135ŠŠkn-i+8ĆĆ=0,
          (* else *)
        kpŠŠknĆĆ=i;    Stampaj;   1
      Ć
    Ć;
    i++
  Ć;
  ReturnŠ"Ovo su sva resenja"ĆĆ

 Kraljice:= BlockŠši,pomć,
  vr=šć; kp=šć;dij45=šć;dij135=šć;pom=šć;sahtabla=šć;
  ForŠi=1,i<9,kp=AppendŠkp,0Ć;i++Ć;
  ForŠi=1,i<9,vr=AppendŠvr,0Ć;i++Ć;
  ForŠi=1,i<16,dij45=AppendŠdij45,0Ć;
            dij135=AppendŠdij135,0Ć;i++
    Ć;
    ForŠi=1,i<9,
      ForŠj=1,j<9,
        pom=AppendŠpom, 0Ć; j++
      Ć;
      sahtabla=AppendŠsahtabla,pomĆ;pom=šć;   i++
    Ć;
  PostaviŠ1ĆĆ
```

3.2. PROBLEM KRALJICA ZA ŠAHOVSKU TABLU DIMENZIJE N ×N

Navedenu funkciju možemo uopštiti za slučaj šahovske table *n*x*n*, za proizvoljno *n*. Potrebno je izbaciti liniju kojom se proverava rezultat izvršavanja rekurzivnog poziva funkcije š*Postavi*ć. Tako bi funkcija za nalaženje svih rešenja mogla da se zapiše u sledećem obliku:

```
StampajŠn_Ć:=BlockŠši,jć,
  ForŠi=1,i<n+1,
    ForŠj=1,j<n+1,
      IfŠkpŠŠjĆĆ==i,
        sahtablaŠŠj,iĆĆ=1,  sahtablaŠŠj,iĆĆ=0Ć;
        j++
      Ć;
    i++
  Ć;
  PrintŠTableFormŠsahtablaĆ
 Ć;
 PostaviŠn_,kn_Ć:=BlockŠšić,
   ForŠi=1,i<n+1,
     IfŠ(vrŠŠiĆĆ==0)&&(dij45ŠŠi+kn-1ĆĆ==0)&&
                (dij135ŠŠkn-i+nĆĆ==0),
      IfŠkn<n,
        kpŠŠknĆĆ=i;       vrŠŠiĆĆ=1;
        dij45ŠŠi+kn-1ĆĆ=1; dij135ŠŠkn-i+nĆĆ=1;
        PostaviŠn,kn+1Ć;   vrŠŠiĆĆ=0;
        dij45ŠŠi+kn-1ĆĆ=0; dij135ŠŠkn-i+nĆĆ=0,
             (* else *)
        kpŠŠknĆĆ=i;  StampajŠnĆ;     1
       Ć
      Ć;
     i++
     Ć;
   ReturnŠ"Ovo su sva resenja"ĆĆ
KraljiceŠn_Ć:= BlockŠši,pomć,
   vr=šć;   kp=šć;   dij45=šć;   dij135=šć;
   pom=šć;   sahtabla=šć;
   ForŠi=1,i<n+1,kp=AppendŠkp,0Ć;i++Ć;
   ForŠi=1,i<n+1,vr=AppendŠvr,0Ć;i++Ć;
   ForŠi=1,i<2*n,dij45=AppendŠdij45,0Ć;
              dij135=AppendŠdij135,0Ć;    i++
   Ć;
   ForŠi=1,i<n+1,
```

```
    ForŠj=1,j<n+1,
      pom=AppendŠpom, 0Ć;     j++
      Ć;
    sahtabla=AppendŠsahtabla,pomĆ; pom=šć;   i++
    Ć;
  PostaviŠn,1ĆĆ
```

3.3. NEREKURZIVNO REŠENJE PROBLEMA KRALJICA

Pokušajmo, na kraju, da proizvedemo nerekurzivno rešenje problema kraljica. Kao što se vidi, jedini argument funkcije je redni broj kolone u koju postavljamo kraljicu. Uvodemo promenljivu koja određuje redni broj kolone u kojoj se trenutno nalazimo. Tako izabranu promenljivu:

- uveaćavamo na mestu gde trenutno stoji rekurzivni poziv,
- umanjujemo neposredno iza mesta gde stoji rekurzivni poziv.

Dobijamo sledeću nerekurzivnu funkciju:

```
StampajŠn_Ć:=BlockŠši,jć,
 ForŠi=1,i<n+1,
  ForŠj=1,j<n+1,
   IfŠkpŠŠjĆĆ==i,
    sahtablaŠŠj,iĆĆ=1, sahtablaŠŠj,iĆĆ=0Ć; j++
   Ć;
   i++
  Ć;
  PrintŠTableFormŠsahtablaĆĆ;
  PrintŠ"**************************************"ĆĆ
PostaviŠn_Ć:=BlockŠši,jć,
 j=1;  kpŠŠjĆĆ=1;
 WhileŠj>0,
  ForŠi=kpŠŠjĆĆ,i<n+1,
    IfŠ(vrŠŠiĆĆ==0)Đ&Đ&(dij45ŠŠi+j-1ĆĆ==0)Đ&Đ&
                  (dij135ŠŠj-i+nĆĆ==0),
      IfŠj<n,
        kpŠŠjĆĆ=i; vrŠŠiĆĆ=1;
        dij45ŠŠi+j-1ĆĆ=1; dij135ŠŠj-i+nĆĆ=1;
        BreakŠĆ,
      (* else *)
        kpŠŠjĆĆ=i; rn++; StampajŠnĆĆĆ; i++
       Ć;
      IfŠi<n+1,
        j=j+1; kpŠŠjĆĆ=1,(* else *)
        j=j-1;
        IfŠj>0,
          i=kpŠŠjĆĆ;  vrŠŠiĆĆ=0;
```

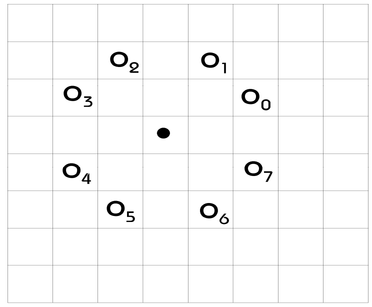
```
            dij45ŠŠi+j-1ĆĆ=0; dij135ŠŠj-i+nĆĆ=0;
            kpŠŠjĆĆ=kpŠŠjĆĆ+1
           Ć
         Ć
   Ć     Ć
KraljiceŠn_Ć:= BlockŠši,pomć,
   rn=0;  vr=šć;  kp=šć; dij45=šć; dij135=šć; pom=šć;
   sahtabla=šć;
   ForŠi=1,i<n+1,kp=AppendŠkp,0Ć;i++Ć;
   ForŠi=1,i<n+1,vr=AppendŠvr,0Ć;i++Ć;
   ForŠi=1,i<2*n,dij45=AppendŠdij45,0Ć;
            dij135=AppendŠdij135,0Ć; i++
     Ć;
   ForŠi=1,i<n+1,
     ForŠj=1,j<n+1,
       pom=AppendŠpom, 0Ć;   j++
     Ć;
     sahtabla=AppendŠsahtabla,pomĆ;  pom=šć;    i++
   Ć;
   PostaviŠnĆĆ
```

3.4. OBILAZAK ŠAHOVSKE TABLE SKAKAČEM

Drugi problem sličnog tipa je određivanje da li postoji maršuta po kojoj bi se kretao skakač po šahovskoj tabli proizvoljnih dimenzija tako da svako polje poseti tačno jedanput.

Zadatak se rešava putem proba. Sa skoro svakog polja na tabli skakač može skočiti na 8 različitih polja (slika 7. 1). Numerišimo moguće poteze kao što je prikazano na slici:

- nulti je dva polja desno, jedno polje naviše;
- prvi je jedno polje desno, dva naviše, i na kraju
- sedmi je dva polja desno, jedno naniže.



Slika 7.1

Koristimo izraz "sa skoro svakog" zato što sa nekih polja blizu kraja table nije moguće izvesti svaki od prikazanih poteza, jer se izlazi sa table.

U toku obilaska table krećemo sa nekog polja i u svakom koraku:

- izvedemo skok na jedan od mogućih načina;
- probamo da li od polja na kome se nalazimo nakon skoka možemo obići ostatak table.

U slučaju neuspeha u rešavanju ostatka problema, probamo sa sledećim mogućim nastavkom od polja na kome se trenutno nalazimo. Postupak se završava nakon nalaženja rešenja ili po iscrpljivanju svih mogućnosti.

Ovde navodimo funkciju za nalaženje svih rešenja.

```
NastaviŠm_,n_,cm_,cn_,pn_Ć:=
 BlockŠši,j,nm,nnć,
   DoŠnm=cm+poxŠŠiĆĆ;   nn=cn+poyŠŠiĆĆ;
     IfŠ(nm<=0)đđ(nm>m),ContinueŠĆĆ;
     IfŠ(nn<=0)đđ(nn>n),ContinueŠĆĆ;
     IfŠpotŠŠnm,nnĆĆ!=0,ContinueŠĆĆ;
     potŠŠnm,nnĆĆ=pn;
     IfŠpn==m*n,
       PrintŠTableFormŠpotĆĆ;
       PrintŠ"**********************************"Ć,
         (* else *)
       NastaviŠm,n,nm,nn,pn+1Ć
     Ć;
     potŠŠnm,nnĆĆ=0,ši,1,8ćĆ
 Ć
ObiđiŠm_,n_,sm_,sn_Ć:= BlockŠši,j,pom=šćć,
  pox=š2,1,-1,-2,-2,-1,1,2ć;
  poy=š1,2,2,1,-1,-2,-2,-1ć; pot=šć;
  ForŠi=1,i<=m,
    ForŠj=1,j<=n, pom=AppendŠpom,0Ć;  j++ Ć;
    pot=AppendŠpot,pomĆ;   pom=šć;  i++
   Ć;
```

```
   potŠŠsm,snĆĆ=1;   NastaviŠm,n,sm,sn,2Ć
Ć
```

3.5. PROBLEM STABILNIH BRAKOVA

Neka imamo dva disjunktna skupa *A* i *B* sa istim brojem elemenata (*n*). Potrebno je naći skup od *n* parova oblika (*a, b*) takvih ispunjavaju uslove $a \in A$ i $b \in B$ i još neki dodatni uslov. Za izbor parova postoje razni kriterijumi a jedan od njih je *pravilo stabilnih brakova*.

Pretpostavimo daje *A* skup muškaraca, a *B* skup žena. Svaki muškarac ima svoju rang-listu žena. Obratno, svaka žena ima svoju rang-listu muškaraca. Ako među njima postoje jedan muškarac i jedna žena koji nisu u braku, ali se svako od njih nalazi na rang-listi drugog pre stvarnog supružnika, skup brakova se naziva nestabilnim**.** Zadatak je da se na osnovu rang-listi formira *n* stabilnih brakova.

Problem rešavamo tako što nađemo suprugu prvom muškarcu, a potom tražimo još *n*-1 stabilan brak. Naravno, taj problem opet razdvajamo na problem nalaženja supruge prvom od preostalih *n*-1 muškaraca i na problem proizvodnje *n*-2 stabilna braka.

Suprugu određenom muškarcu tražimo sistemom proba, krećući se po rang-listi tog muškarca. Za suprugu uzimamo prvu osobu sa rang-liste koja nije u braku, tako da po dodavanju tog para skupu brakova i dalje imamo skup stabilnih brakova. Ukoliko to ne možemo da izvedemo, moramo da menjamo neki od prethodno generisanih brakova.

Pretpostavke su sledeće:

(1) *mf* matrica u kojoj je *mf*Š*zn*ĆŠ*i*Ć *i*-ti muškarac na rang-listi žene sa rednim brojem *zn*;

(2) *zf* matrica u kojoj e *zf*Š*mn*ĆŠ*i*Ć *i*-ta žena na rang-listi muškarca sa rednim brojem *mn*;

(3) *mi* predstavlja niz u kome je *mi*Š*zn*Ć muškarac koji je u braku sa ženom zn (uzima se -1 ako žena *zn* još uvek nema supruga);

(4) *zi* je niz u kome *zi*Š*mn*Ć predstavlja ženu koja je u braku sa muškarcem *mn* (-1 ako muškarac *mn* još uvek nema ženu).

3.6. ZADATAK OPTIMALNOG IZBORA

Navedimo primer još jednog problema koji se može rešiti korišćenjem bektreka, a koji se malo razlikuje od već navedenih. Problem pripada klasi problema nalaženja optimalnog rešenja.

Problem se može formulisati na sledeći način: Dat je skup od n objekata. Poznate su težine i cene svih objekata. Potrebno je izdvojiti podskup objekata tako da ukupna težina ne prelazi zadati broj, a ukupna cena bude maksimalna.
Problem se svodi na nalaženje svih podskupova koji zadovoljavaju prvi uslov i određivanje maksimalnog od tih podskupova po drugom kriterijumu. Generisanje podskupova koji zadovoljavaju prvi uslov vršimo tako što za svaki od objekata pojedinačno probamo:

- varijantu kada taj objekat nije sadržan u maksimalnom skupu, ali i
- varijantu kada taj objekat jeste u maksimalnom skupu.

Medutim, to bi zahtevalo ukupno 2^n pokušaja, što eksponencijalno raste. Stoga je potrebno odbaciti neke varijante. Odbacivanje varijanti vršimo po sledećim kriterijumima:

(i) Što se tiče uključivanja nekog elementa, sasvim je prirodno postaviti ograničenje da dati element treba uključiti samo ako nakon njegovog uključivanja dobijeni podskup ima sumu težina manju od zadatog ograničenja.

(ii) Neki element ne treba uključiti u dosad generisani podskup ako posle toga ostaje dovoljno elemenata i može se dostići cena veća od dosad poznate najveće cene.

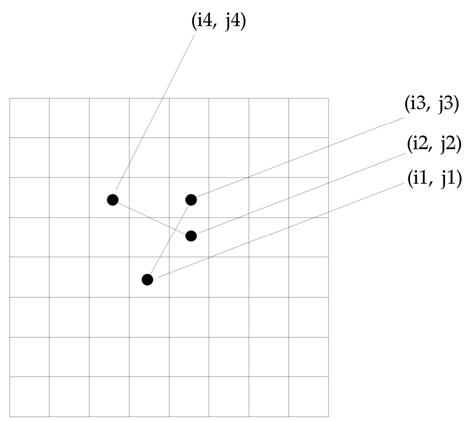
Težina do sada generisanog podskupa i ukupna cena ostatka skupa objekata (još neanalizirani objekti) zajedno sa generisanim podskupom biće argumenti funkcije koju rekurzivno pozivamo za sledeći objekt.

3.7. ODREĐIVANJE NAJDUŽE PROSTE MARŠUTE SKAKAČA

Neka se skakač nalazi na polju (i, j) na šahovskoj tabli dimenzija $m \times n$. Neka se kreće po tabli po pravilima šahovske igre. Izvršimo spajanje središta polja koje je skakač obišao (naravno, spajamo susedna polja, tj. polja na kojima se nalazio pre i posle pojedinih skokova). Dobijenu liniju ćemo zvati maršruta. Kazaćemo da je maršruta prosta ukoliko sama sebe ne seče, odnosno ako ne postoje dve nesusedne duži koje imaju zajedničkih tačaka (seku se).
Potrebno je da se odredi najduža prosta maršruta koju može napraviti skakač kad kreće od polja (i,j) na tabli dimenzija $m \times n$. Iskoristićemo tehniku bektreka. Od polja (i,j) skakač može preći u najpovoljnijem slučaju na neko od 8 polja, kao što je prikazano na slici 7.1. Numerišimo moguće skokove brojevima od 0 do 7. Ideja je da sa svakim od mogućih poteza pokušamo da produžimo maršrutu.

Kada napravimo skok sa polja (*i*1, *j*1) na polje (*i*2, *j*2), maršruta se produžava za liniju koja spaja središta ta dva polja. Ova linija razdvaja neke parove polja (označimo ih sa (*i*3, *j*3) i (*i*4, *j*4)) na šahovskoj tabli za koju važi sledeće:

- od jednog do drugog polja skakač bi mogao stići jednim potezom po pravilima šhovske igre;
- ako bi u nastavku skakač napravio skok sa polja (*i*3,*j*3) na polje (*i*4, *j*4) (ili obratno), maršruta više ne bi bila prosta, jer bi se duž koja spaja središta ta dva polja sekla sa duži koja spaja središta polja (*i*1, *j*1) i polja (*i*2, *j*2).

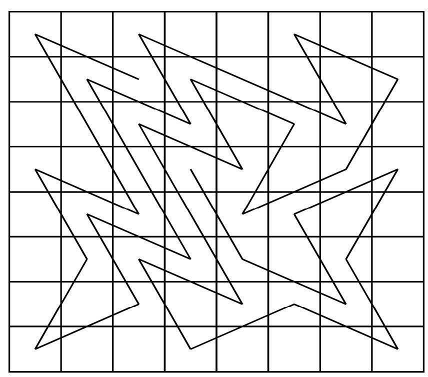


Slika 7.2

Znači, par polja (*i*3, *j*3) i (*i*4, *j*4) ne može biti susedan u nizu polja koje će skakač obići. Primer parova polja sa tim svojstvom prikazan je na slici 7.2. Posle izvođenja pojedinih skokova za svaki par polja (*i*3, *j*3) i (*i*4, *j*4) sa gore navedenim svojstvima beležimo da je neizvodljiv skok sa jednog u paru na drugo. Kako ćemo to izvesti? Kako je moguć skok sa polja (*i*3, *j*3) na polje (*i*4, *j*4), to postoji neki potez (jedan od onih koje smo označili brojevima od 0 do 7) kojim od prvog stižemo do drugog polje. Neka je redni broj tog poteza *k*. Znači beležimo da sa polja (*i*3, *j*3) nije moguće izvesti potez sa rednim brojern k. Ako se u nastavku nađemo na polju (*i3*, *j3*), to ćemo potez sa rednim brojem k preskočti, odnosno nećemo analizirati. Slično važi i za skok sa polja (*i*4, *j*4) na polje (*i*3, *j*3) (s tim što će biti zabranjen potez sa nekim drugim rednim brojem).

U početnom trenutku ne postoje nikakva ograničenja ovog tipa, odnosno smatra se da je moguć bilo koji potez na bilo kom polju (naravno, osim poteza kojim bi se skočilo van table).

Par polja ($i3$, $j3$) i ($i4$, $j4$), odnosno par koji se sastoji od polja ($i3$, $j3$) i poteza k jednoznačno je određen polaznim poljem (polje ($i1$, $j1$)) i rednim brojem poteza kojim smo stigli na polje ($i2$, $j2$). Naime, par ($i3$-$i1$, $j3$-$j1$) je konstantan za bilo koje polazno polje ($i1$, $j1$) i redni broj poteza kojim smo od ($i1$, $j1$) stigli do polja ($i2$, $j2$). Stogaje dovoljno odrediti parove ($i3$-$i1$, $j3$-$j1$) i redne brojeve poteza k koje ne treba analizirati: dovoljnoje napisati pomoćni program (što je brže) ili analizu staviti na papir.
Ako bismo se odlučili za program, dovoljno je fiksirati polazno i ciljno polje za potez sa rednim brojem l. Neka su to polja ($i1$, $j1$) i ($i2$, $j2$). Zatim analiziramo sva polja koja se nalaze dovoljno blizu ovih dvaju polja i polja blizu linije koja spaja ta dva polja. Izraz *dovoljno blizu* podrazumeva da se do njih može stići jednim potezom skakača, ili da su bliža od polja do kojih se može stići jednim potezom. Za svako od tih izabranih polja analiziramo svaki od osam (8) mogućih poteza i provervamo da li bi se izvođenjem tog poteza presekla linija koja spaja ($i1$, $j1$) i ($i2$, $j2$). Tako bi taj pomoćni program imao sledeći zapis:



Slika 7.3

Sada možemo da zapišemo i funkciju koja određuje najdužu maršrutu. Kao što smo rekli, od polja na kome se nalazimo probamo da produžimo maršrutu izvodeći jedan od mogućih 8 poteza. Ako je moguće izvesti potez sa rednim brojem l, onda:

- izvodimo potez i na taj način prelazimo sa polja ($i1$, $j1$) na polje ($i2$,$j2$);
- za određen broj polja iz okoline polja ($i1$, $j1$) zabranjujemo neke skokove (to su polja i potezi koje smo odredili prethodno opisanim programom).

Na slici 7.3 prikazana je jedna prosta maršruta, pri čemu skakač kreće sa polja (3, 3).

3.8. ODREĐIVANJE SKUPOVA SLOBODNIH ZA SUMU

Kažemo da je skup S slobodan za sumu ako važi: ako $x,y \in S$ onda $x+y \notin S$. Napišimo funkciju koja za zadati broj k određuje najveći ceo broj n, tako da se skup š1,2,..., *nć* može razbiti u k disjunktnih podskupova S_1, S_2 , S_k koji su slobodni za sumu.

Ako broj n označimo sa $f(k)$, onda važi:

$$f(1)=1,\ f(2)=3,\ f(3)=13,\ f(4)=44,\ f(k+1) \le 3\,f(k)+1,\ f(k) \le k!$$

Računanje funkcije f možemo izvesti korišćenjem tehnike bektreka. Ako je $k>1$, brojeve jedan i dva rasporedimo u prvi i drugi podskup. Za sve brojeve veće od dva određujemo skupove Si $(1 \le i \le k)$ kojima ne mogu biti dodati zato što nakon dodavanja ne bi bili slobodni za sumu. Tako nakon dodavanja brojeva 1 i 2 u prvi i drugi skup, broj četiri ne može biti stavljen u drugi skup. Takođe je u svakom trenutku poznata dosad određena najveća vrednost broja n, tj. najveće n takvo da je š1,...,*nć* razbijen na k podskupova slobodnih za sumu.

Po raspoređivanju brojeva od 1 do i u neke od skupova, raspoređujemo broj $i+1$. Tražimo prvi od k skupova kome broj $i+1$ može biti dodat.

Ako nađemo takav skup (nekaje to skup S_j), onda:

- broj $i+1$ dodajemo skupu S_j;
- za sve elemente $x \in S_j$ notiramo da broj $x+i+1$ ne može biti raspoređen u skup S_j.

Ako se nakon ovoga desi da neki broj $l \le n$ ne može biti raspoređen ni u jedan od k skupova, onda broj $i+1$ ne može biti raspoređen u skup S_j. Zato tražimo sledeći skup kome može biti dodat broj $i+1$.

Ako ne postoji skup kome može biti dodat broj $i+1$, vraćamo se nazad i tražimo drugi skup kome može biti dodat broj i.

LITERATURA

Š1Ć D. Urošević, *Algoritmi u programskom jeziku C,* Mikro knjiga, Beograd, 1996.

Š2Ć M. Živković, *Algoritmi,* Matematički fakultet, Beograd, 2000.

4. IZRAČUNAVANJE DETERMINANTI

Programski paket *MATHEMATICA* je pogodan za numeričko i simboličko izračunavanje determinanti.

4.1. IZRAČUNAVANJE NEKIH OSNOVNIH DETERMINANTI

I. Izračunajmo sledeću determinantu reda $2n$:

$$D_{2n}=\begin{vmatrix} a & 0 & & & & & 0 & b \\ 0 & a & \ddots & & & \iddots & b & 0 \\ & \ddots & \ddots & 0 & 0 & \iddots & \iddots & \\ & & 0 & a & b & 0 & & \\ & & 0 & b & a & 0 & & \\ & \iddots & \iddots & 0 & 0 & \ddots & \ddots & \\ 0 & b & \iddots & & & \ddots & a & 0 \\ b & 0 & & & & & 0 & a \end{vmatrix}.$$

Ovde se može primeniti uopštena *Laplace-ova* teorema na centralni minor, tj.

$$\begin{aligned} D &= (-1)^{n+n+1+n+n+1}\begin{vmatrix} a & b \\ b & a \end{vmatrix}\cdot D_{2n-2} \\ &= (a^2-b^2)D_{2n-2} = (a^2-b^2)^2 D_{2n-4} \\ &= \ldots = (a^2-b^2)^{n-1}D_2 = (a^2-b^2)^n \end{aligned}$$

```
Determ36[n_, a_, b_] := Module[{k = n},
              If[k == 1, Return[a*a - b*b],
                         Return[(a*a - b*b) * Determ36[k - 1, a, b]]]]
Determ36[3, a, b]
(a^2 - b^2)^3
```

II. Izračunajmo sada sledeće tri determinante:

$$\begin{vmatrix} 1 & 1 & 1 & \ldots & 1 & 1 \\ 1 & 0 & 1 & \cdots & 1 & 1 \\ 1 & 1 & 0 & \cdots & 1 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & 1 & \cdots & 0 & 1 \\ 1 & 1 & 1 & \cdots & 1 & 0 \end{vmatrix}; \quad \begin{vmatrix} 1 & 2 & 3 & \cdots & n-1 & n \\ 2 & 3 & 4 & \cdots & n & n \\ 3 & 4 & 5 & \cdots & n & n \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ n-1 & n & n & \cdots & n & n \\ n & n & n & \cdots & n & n \end{vmatrix}; \quad \begin{vmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & 1-x & 1 & \cdots & 1 & 1 \\ 1 & 1 & 2-x & \cdots & 1 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & 1 & \cdots & (n-1)-x & 1 \\ 1 & 1 & 1 & \cdots & 1 & n-x \end{vmatrix}.$$

Kod prve determinante reda n, od svake vrste oduzmemo prvu vrstu. Tada dobijamo

$$D_n = \begin{vmatrix} 1 & 1 & 1 & \dots & 1 & 1 \\ 0 & -1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & -1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & -1 \end{vmatrix} = (-1)^{n-1} \cdot 1 = (-1)^{n-1}.$$

U slučaju druge determinante od n-te kolone oduzmemo (n-1)-vu kolonu, zatim od (n-1)-ve kolone (n-2)-gu, itd., dobijamo

$$D_n = \begin{vmatrix} 1 & 1 & 1 & \dots & 1 & 1 \\ 2 & 1 & 1 & \cdots & 1 & 0 \\ 3 & 1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ n-1 & 1 & 0 & \cdots & 0 & 0 \\ n & 0 & 0 & \cdots & 0 & 0 \end{vmatrix} = n \cdot 1 \cdot (-1)^{\delta},$$

gde je δ broj inverzija u permutaciji n, n-1, ... , 3, 2, 1. Taj broj je očigledno

$$1 + 2 + \cdots + (n-1) = \frac{(n-1)n}{2} = \binom{n}{2}.$$

Dakle, $D_n = n \cdot (-1)^{\binom{n}{2}}$.

Treća determinanta je (n+1)-vog reda. Ako od svake vrste oduzmemo prvu vrstu, determinanta postaje

$$D_{n+1} = \begin{vmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \\ 0 & -x & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1-x & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & n-2-x & 0 \\ 0 & 0 & 0 & \cdots & 0 & n-1-x \end{vmatrix}$$

$$= 1 \cdot (-x) \cdot (1-x) \cdot (2-x) \cdots (n-1-x)$$

$$= (-1)^n x(x-1) \cdots (x-n+1).$$

```
Determ314[n_] :=
 Module[{i, j, a},
  a = Table[If[i == j, 0, 1], {i, 1, n}, {j, 1, n}];
  a[[1, 1]] = 1;
  Return[Det[a]]
 ]

Determ314[2]
-1
```

III. Izračunajmo sada determinantu

$$D_n = \begin{vmatrix} a & b & 0 & \cdots & 0 & 0 & 0 \\ c & a & b & \cdots & 0 & 0 & 0 \\ 0 & c & a & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & c & a & b \\ 0 & 0 & 0 & \cdots & 0 & c & a \end{vmatrix},$$

gde su $a, b, c \in R.$

Razvijanjem determinante po prvoj vrsti dobijamo diferencnu jednačinu $D_n = aD_{n-1} - bcD_{n-2}$. Ako su r_1 i r_2 rešenja kvadratne jednačine $r^2 - ar + bc = 0$, a $a^2 - 4bc$ je njena diskriminanta, imamo sledeća tri slučaja:

$D_n = C_1 r_1^n + C_2 r_2^n$ za $a^2 - 4bc > 0$,

$D_n = (C_1 n + C_2) r_1^n$ za $a^2 - 4bc = 0$,

$D_n = \rho^n (C_1 \cos n\varphi + C_2 \sin n\varphi)$ za $a^2 - 4bc < 0$, gde je $\rho = \sqrt{\alpha^2 + \beta^2}$, φ - glavna vrednost kompleksnog broja $\alpha + i\beta$. Ovde su rešenja kvadratne jednačine kompleksni brojevi $\alpha \pm i\beta$.

U svim slučajevima konstante C_1 i C_2 određujemo iz početnih uslova: $D_1 = a$, $D_2 = a^2 - bc$.

```
Determ316[n_] :=
  Block[{i, j, a, b, c},
    q = Table[If[i == j, a,
          If[j - i == 1, b,
             If[i - j == 1, c, 0]
          ]  ] ,
          {i, 1, n}, {j, 1, n}];
    Return[Det[q]]
 ]
```

```
Determ316[3]
```

$a^3 - 2abc$

IV. Sada izračunavamo determinante sledećih matrica:

$$\begin{bmatrix} 2 & 1 & 0 & \dots & 0 & 0 & 0 \\ 1 & 2 & 1 & \dots & 0 & 0 & 0 \\ 0 & 1 & 2 & \dots & 0 & 0 & 0 \\ \vdots & & & & & & \\ 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 & 2 \end{bmatrix}, \qquad \begin{bmatrix} 5 & 3 & 0 & \dots & 0 & 0 & 0 \\ 2 & 5 & 3 & \dots & 0 & 0 & 0 \\ 0 & 2 & 5 & \dots & 0 & 0 & 0 \\ \vdots & & & & & & \\ 0 & 0 & 0 & \dots & 2 & 5 & 3 \\ 0 & 0 & 0 & \dots & 0 & 2 & 5 \end{bmatrix},$$

$$\begin{bmatrix} \alpha+\beta & \alpha\beta & 0 & \dots & 0 & 0 & 0 \\ 1 & \alpha+\beta & \alpha\beta & \dots & 0 & 0 & 0 \\ \vdots & & & & & & \\ 0 & 0 & 0 & \dots & 1 & \alpha+\beta & \alpha\beta \\ 0 & 0 & 0 & \dots & 0 & 1 & \alpha+\beta \end{bmatrix}.$$

Na osnovu prethodnog primera imamo $D_n = 2D_{n\text{-}1}$ Đ $D_{n\text{-}2}$, tj. D_n Đ $2D_{n\text{-}1} + D_{n\text{-}2} = 0$. Kako kvadratna jednačina r^2 Đ $2r + 1 = 0$ ima dvostruki koren $r_1 = r_2 = 1$, za prvu determinantu dobijamo

$$D_n = (C_1 n + C_2)1^n = C_1 n + C_2.$$

Kako je $D_1 = 2$, $D_2 = 3$, to je $2 = C_1 + C_2$, $3 = 2C_1 + C_2$, tj. $C_1 = 1$, $C_2 = 1$. Dakle, $D_n = n + 1$.

Za drugu determinantu je $D_n = 5D_{n\text{-}1}$ Đ $6D_{n\text{-}2}$, tj. $D_n = C_1 2^n + C_2 3^n$, odakle se zbog $D_1 = 5$, $D_2 = 19$ dobija $C_1 = -2$, $C_2 = 3$. Dakle, $D_n = 3^{n+1}$ Đ 2^{n+1}.

Pri izračunavanju treće determinante primenjujemo postupak kao u prethodna dva slučaja. Tako dobijamo da je $D_n = (\alpha + \beta)D_{n-1} - \alpha\beta D_{n-2}$, tj. diferencnoj jednačini pridružujemo kvadratnu jednačinu r^2 Đ$(\alpha+\beta)r+\alpha\beta=0$. Rešenja kvadratne jednačine su $r_1=\alpha$, $r_2=\beta$.
Ako je $\alpha=\beta\neq 0$ onda je $D_n=(C_1+ nC_2)\alpha^n$, odakle se zbog početnih uslova $D_1 = \alpha + \beta$, $D_2 = \alpha^2 + \alpha\beta + \beta^2$, dobija $C_1 = 1$, $C_2 = 1$, tj. $D_n = (n + 1)\alpha^n$.
Ako je $\alpha = \beta = 0$, onda je $D_n = 0$.
Za $\alpha \neq \beta = 0$ ili $\beta \neq \alpha = 0$ imamo $D_n = \alpha^n$, tj. $D_n = \beta^n$.

Neka je sada $\alpha \neq 0$, $\beta \neq 0$, $\alpha \neq \beta$. Tada je $D_n = C_1\alpha^n + C_2\beta^n$.
Iz početnih uslova $D_1 = \alpha + \beta$, $D_2 = \alpha^2 + \alpha\beta + \beta^2$ određujemo konstante C_1 i C_2. Iz sistema $\alpha + \beta = C_1\alpha + C_2\beta$, $\alpha^2 + \alpha\beta + \beta^2 = C_1\alpha^2 + C_2\beta^2$, dobijamo da je $C_1 = \alpha/ (\alpha - \beta)$, $C_2 = \beta/ (\beta - \alpha)$. Najzad, $D_n=\dfrac{\alpha^{n+1}-\beta^{n+1}}{\alpha-\beta}$.

```
Determ317[n_] :=
 Module[{i, j, a},
   a = Table[If[i == j, 2, If[Abs[j - i] == 1, 1, 0]],  {i, 1, n}, {j, 1, n}];
    Return[Det[a]]
 ]

Determ317[2]
3
```

V. Razvijanjem determinante

$$M=\begin{bmatrix} 1 & -a_1 & -a_2 & \dots & -a_{n-1} & -a_n \\ a_1 & 1 & 0 & \dots & 0 & 0 \\ \vdots & & & & & \\ a_{n-1} & 0 & 0 & \dots & 1 & 0 \\ a_n & 0 & 0 & \dots & 0 & 1 \end{bmatrix} \in M_{n+1}(R).$$

po prvoj vrsti dobija se $n + 1$ determinanata, od kojih svaka ima po jednu vrstu sa svim nulama izuzev u prvoj koloni, tj.

$$\det M = 1\cdot\begin{vmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & & & \\ 0 & 0 & \dots & 1 \end{vmatrix} + a_1\begin{vmatrix} a_1 & 0 & \dots & 0 \\ a_2 & 1 & \dots & 0 \\ \vdots & & & \\ a_n & 0 & \dots & 1 \end{vmatrix} - a_2\begin{vmatrix} a_1 & 1 & \dots & 0 \\ a_2 & 0 & \dots & 0 \\ \vdots & & & \\ a_n & 0 & \dots & 1 \end{vmatrix} + \dots.$$

$$= 1 + a_1^2 + a_2^2 + \dots + a_n^2.$$

```
Determ326[n_, v_] := Module[{i, j},
       h = Table[If[i == j, 1,
                                 If[i == 1, -v[[j - 1]],
                                        If[j == 1, v[[i - 1]], 0]]],
    {i, 1, n + 1}, {j, 1, n + 1}];
         Return[Det[h]]]
```

```
General::spell1 :
 Possible spelling error: new symbol name "Determ326" is similar to existing symbol "Determ36".
```

```
f := {a1, a2, a3}
f
{a1, a2, a3}
Determ326[3, f]
1 + a1^2 + a2^2 + a3^2
h
{{1, -a1, -a2, -a3}, {a1, 1, 0, 0}, {a2, 0, 1, 0}, {a3, 0, 0, 1}}
```

VI. Za sledeće determinante n-tog reda se može upotrebiti *MATHEMATICA* za verifikaciju sledećih formula:

$$\begin{vmatrix} a_1+x & x & \dots & x \\ x & a_2+x & \dots & x \\ \vdots & & & \\ x & x & \dots & a_n+x \end{vmatrix} = a_1a_2\dots a_n + (a_1a_2\dots a_{n-1} + a_1\dots a_{n-2}a_n + \dots + a_2a_3\dots a_n)x;$$

$$\begin{vmatrix} a_1+x & a_2 & \dots & a_n \\ a_1 & a_2+x & \dots & a_n \\ \vdots & & & \\ a_1 & a_2 & \dots & a_n+x \end{vmatrix} = x^n + (a_1 + \dots + a_n)\, x^{n-1};$$

$$\begin{vmatrix} 3 & 2 & 0 & 0 & \dots & 0 & 0 & 0 \\ 1 & 3 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 2 & 3 & 2 & \dots & 0 & 0 & 0 \\ \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & 3 & 1 \\ 0 & 0 & 0 & 0 & \dots & 0 & 2 & 3 \end{vmatrix} = 2^{n+1} \text{ Đ } 1; \qquad \begin{vmatrix} 1 & 2 & 3 & \dots & n \\ -1 & 0 & 3 & \dots & n \\ -1 & -2 & 0 & \dots & n \\ \vdots \\ -1 & -2 & -3 & \dots & 0 \end{vmatrix} = n!\,;$$

$$\begin{vmatrix} 1 & 1 & \dots & 1 & -n \\ 1 & 1 & \dots & -n & 1 \\ \vdots \\ 1 & -n & \dots & 1 & 1 \\ -n & 1 & \dots & 1 & 1 \end{vmatrix} = (-1)^{(n+1)n/2}\,(n+1)^{n-1}.$$

```
determ328[a_, x_] :=
     Module[{i, j, n = Length[a]},
     d = Table[a[[i]], {i, n}, {j, n}]; d = Transpose[d];
          For[i = 1, i <= n, i++,
                d[[i, i]] = d[[i, i]] + x;
             ];
     Print[MatrixForm[d]];
     Return[Simplify[Det[d]]];
       ]
```

```
determ328[{a1, a2, a3}, x]
```

a1 + x	a2	a3
a1	a2 + x	a3
a1	a2	a3 + x

4.2. IZRAČUNAVANJE VANDERMONDOVE DETERMINANTE

V. Determinanta oblika

$$V_n(x_1, x_2, \dots, x_n) = \begin{vmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ 1 & x_2 & x_1^2 & \dots & x_2^{n-1} \\ \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^{n-1} \end{vmatrix}$$

naziva se Vandermonde-ova determinanta. Za ovu determinntu važi

$$V_n(x_1, x_2, ..., x_n) = \prod_{n \geq i \geq j \geq 1} (x_i - x_j).$$

Zaista

$$V_2(x_1, x_2) = \begin{vmatrix} 1 & x_1 \\ 1 & x_2 \end{vmatrix} = x_2 - x_1,$$

$$V_3(x_1, x_2, x_3) = \begin{vmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \end{vmatrix} = \begin{vmatrix} 1 & 0 & 0 \\ 1 & x_2 - x_1 & x_2^2 - x_1 x_2 \\ 1 & x_3 - x_1 & x_3^2 - x_1 x_3 \end{vmatrix} \quad \begin{pmatrix} K_2 \to K_2 \to x_1 K_1, \\ K_3 \to K_3 \to x_1 K_2 \end{pmatrix}$$

$$= \begin{vmatrix} x_2 - x_1 & x_2^2 - x_1 x_2 \\ x_3 - x_1 & x_3^2 - x_1 x_3 \end{vmatrix} = (x_2 - x_1)(x_3 - x_1) \begin{vmatrix} 1 & x_2 \\ 1 & x_3 \end{vmatrix}$$

$$= (x_2 \text{ Đ } x_1)(x_3 \text{ Đ } x_1)(x_3 \text{ Đ } x_2).$$

Isti postupak može biti primenjen i kod determinanata višeg reda. Pretpostavimo da je formula (3.1) tačna za nĐ1. Sada u determinanti reda n oduzmimo od svake kolone prethodnu kolonu pomnoženu sa x_1:

$$V_n(x_1, x_2, ..., x_n) = \begin{vmatrix} 1 & 0 & 0 & ... & 0 \\ 1 & x_2 - x_1 & x_2^2 - x_2 x_1 & ... & x_2^{n-1} - x_2^{n-2} x_1 \\ \vdots & & & & \\ 1 & x_n - x_1 & x_2^2 - x_n x_1 & ... & x_n^{n-1} - x_n^{n-2} x_1 \end{vmatrix}$$

$$= (x_2 \text{ Đ } x_1)(x_3 \text{ Đ } x_1) \cdots (x_n - x_1) \begin{vmatrix} 1 & x_2 & x_2^2 & ... & x_2^{n-2} \\ 1 & x_3 & x_3^2 & ... & x_3^{n-2} \\ \vdots & & & & \\ 1 & x_n & x_n^2 & ... & x_n^{n-2} \end{vmatrix}$$

$$= (x_2 \text{ Đ } x_1)(x_3 \text{ Đ } x_1) \cdots (x_n - x_1) V_{n-1}(x_2, x_3, ..., x_n).$$

Prema indukcijskoj hipotezi se zatim dobija

$$V_n(x_1, x_2, ..., x_n) = (x_2 - x_1)...(x_n - 1) \prod_{n \geq i \geq j \geq 2} (x_i - x_j) = \prod_{n \geq i \geq j \geq 1} (x_i - x_j),$$

čime je dokaz završen.

```
Determ322[n_, v_] := Module[{i, j, h},
       h = Table[Power[v[[i]], j - 1], {i, 1, n}, {j, 1, n}];
       Return[Det[h]]]
e := {1, 2, 3}
e
Determ322[3, e]
p := {x1, x2, x3}
p
Determ322[3, p]
Factor[%]
```

LITERATURA

[1] G.V. Milovanović i R.Ž. Đorđević, *Matematika za studente tehničkih fakulteta*, I deo, Nauka, Beograd, 1992.

[2] D.S. Mitrinović i D. Ž. Đoković, *Polinomi i matrice*, ICS, Beograd, 1975.

5. NEKE PRIMENE LINEARNOG PROGRAMIRANJA

5.1. OPTIMALNI PROGRAM PROIZVODNJE

Sastaviti optimalni program proizvodnje znači izabrati, između velikog broja proizvoda, onaj asortiman proizvoda koji će obezbediti maksimalno ekonomske efekte iz ograničene količine proizvodnih resursa.

Pretpostavimo da preduzeće može proizvoditi n različitih tipova proizvoda: P_1, P_2,..., P_n. Za proizvodnju preduzeće koristi m različitih vrsta mašina, r različitih kategorija radnika i g različitih vrsta sirovina u ograničenim količinama.

Poznat je dohodak koji se ostvaruje po jedinici svakog proizvoda, pa je problem kako preduzeće da programira proizvodnju da bi ostvarilo najveći mogući dohodak u poslovanju.

Matematički model ćemo formulisati korišćenjem sledećih simbola:

x_j - količina j-tog proizvoda koju treba proizvesti prema optimalnom programu proizvodnje, a koju treba odrediti pomoću matematičkog modela;

c_j - dohodak po jedinici j-tog proizvoda;

a_{ij} - vreme koje je potrebno da se na i-toj mašini proizvede jedinica j-tog proizvoda;

a_{i0} - kapacitet i-te mašine izra*žen u vremenskim jedinicama;*
b_{kj} - *vreme potrebno radniku* k-te kategorije da obradi, odnosno proizvede, jedinicu i-tog proizvoda;
b_{k0} - raspoloživi fond radnog vremena radnika k-te kategorije;
s_{vj} - količina v-te sirovine koja je potrebna za proizvodnju jedinice j-tog proizvoda;
s_{v0} - raspoloživa količina v-te vrste sirovine;
e_j - količina j-tog proizvoda koja se može prodati na trzištu.
Pomoću uvedenih simbola formulišemo matematički model. On ima funkciju kriterijuma $z_0=\sum_{j=1}^{n} c_j x_j$ koja označava dohodak od celokupne proizvodnje, pa treba naći njenu maksimalnu vrednost uz sledeće ograničavajuće faktore:

$$\sum_{j=1}^{n} a_{ij} x_j \le a_{i0}, \qquad i = 1,2,...,m$$

$$\sum_{j=1}^{n} b_{kj} x_j \le b_{k0}, \qquad k = 1,2,...,r$$

$$\sum_{j=1}^{n} s_{vj} x_j \le s_{v0}, \qquad v = 1,2,...,g$$

$$0 \le x_j \le e_j, \qquad j = 1,2,...,n.$$

```
DodajŠa_,b_Ć:=ModuleŠšm=a,i,n=LengthŠbĆć,
        ForŠi=1,i<=n,i++,m=AppendToŠm,bŠŠiĆĆĆĆ ;mĆ

OptimalniProgramŠc_,a_,a0_,b_,b0_,s_,s0_,e_Ć:=
 ModuleŠšm=-a,v=-a0,n=LengthŠcĆ,ić,
  m=DodajŠm,-bĆ;  m=DodajŠm,-sĆ;
   m=DodajŠm,-IdentityMatrixŠnĆĆ;
   v=DodajŠv,-b0Ć; v=DodajŠv,-s0Ć;
   ForŠi=1,i<=n,i++, v=AppendToŠv,-eŠŠiĆĆĆĆ;
   LinearProgrammingŠ-c,m,vĆ
 Ć
```

5.2. OPTIMIZACIJA UTROŠKA MATERIJALA

Optimalni utrošak materijala je takav utrošak materijala koji obezbeđuje ostvarenje određene proizvodnje uz najmanji ukupni otpadak. Kod formiranja odgovarajućeg matematičkog modela poćićemo od sledećih pretpostavki: preduzeće treba da proizvede m različitih delova u određenim količinama, za proizvodnju tih delova koristi se materijal istih dimenzija.

Broj varijanti za obradu materijala unapred je poznat, a isto tako poznata je količina otpadaka koja se javlja pri svakoj varijanti, kao i količina pojedinih delova koja se dobija obradom jedinice materijala po svakoj varijanti.
Za potrebe formiranja matematičkog modela uvodimo sledeće oznake:

x_j - količina materijala određene dimenzije koja će biti obrađena po j-toj varijanti;

a_j - količina otpadaka od jedinice datog materijala koji je obrađen po j-toj varijanti;

a_{ij} - količina delova i-te vrste koja se dobija od jedne jedinice materijala obrađenog po j-toj varijanti;

b_i - ukupna količina i-tog dela koju treba obezbediti za proizvodnju;

s - raspoloživa količina datog materijala.

Modelom će se odrediti količina materijala date dimenzije koja će biti obrađena po j-toj varijanti, ali tako da se željene količine delova proizvedu uz minimalni ukupni otpadak.

Funkcija kriterijuma modela $z_0 = \sum_{j=1}^{n} a_j x_j$ označava ukupni otpadak pri obradi materijala po svim varijantama, pa treba naći njenu minimalnu vrednost uz sledeći sistem ograničenja:

$$\sum a_{ij} x_j = b_i, \quad i = 1,2,...m$$

$$\sum_{j=1}^{n} x_j \leq s \ , \qquad x_j \geq 0, \qquad j = 1,2,...,n$$

Formirali smo model za slučaj kada preduzeće za proizvodnju koristi materijal istih dimenzija.

```
UtrosakMaterijalaŠa0_,a_,b_,s_Ć:=
   ModuleŠšm=a,v=b,n=LengthŠa0Ć,s0=šć,ić,
     ForŠi=1,i<=n,i++,s0=AppendToŠs0,-1ĆĆ;
     m=DodajŠm,šs0ćĆ;   v=DodajŠv,š-sćĆ;
     LinearProgrammingŠa0,m,vĆ
   Ć
```

Pretpostavićemo da preduzeće za proizvodnju koristi isti materijal u k raznih dimenzija. Svaka dimenzija materijala može se obraditi prema raznim varijantama. Ako sa n_v označimo broj varijanti obrade v-tog materijala, onda će ukupan broj varijanti obrade materijala svih dimenzija biti jednak N, pri čemu je $N = \sum_{v=1}^{k} n_v$.

Sa oznakama i parametrima, čije je značenje isto kao i u prethodnom modelu, formiramo sledeći model. Treba naći minimalnu vrednost funkcije kriterijuma

$$z_0 = \sum_{j=1}^{N} a_j x_j \qquad \text{uz zadovoljenje sledećeg sistema ograničenja:}$$

$$\sum_{j=1}^{N} a_{ij} x_j = b_i, \quad i = 1,2,\ldots,m, \qquad \sum_{j=1}^{n_v} x_j \le s_v, \quad v = 1,2,\ldots,k,$$

$$x_j \ge 0, \qquad j = 1,2,\ldots,N.$$

```
UtrosakMaterijalaRazniŠa0_,a_,b_,s_,n_Ć:=
  ModuleŠšm=a,v=b,vn,k=LengthŠnĆ,i,nula,j,l=LengthŠa0Ćć,
      vn=SumŠnŠŠiĆĆ,ši,kćĆ;
      nula=TableŠ0,škć,šlćĆ;
      ForŠi=1,i<=k,i++,
        ForŠj=1,j<=nŠŠiĆĆ,j++,nulaŠŠi,jĆĆ=-1ĆĆ;
      m=DodajŠm,nulaĆ;
      v=DodajŠv,-sĆ;
      LinearProgrammingŠa0,m,vĆ
  Ć
```

5.3. IZBOR SASTAVA MEŠAVINE

Problem svodi na određivanje količina pojedinih sirovina koje će biti utrošene za proizvodnju gotovog proizvoda odgovarajućeg kvaliteta, ali tako da troškovi nabavke sirovina budu minimalni.

U prvom slučaju, u kome se od više sirovina prozvodi jedan proizvod, polazimo od sledećih pretpostavki: za dobijanje gotovog proizvoda može se koristiti *n* vrsta sirovina; gotov proizvod mora da sadrži *m* raznih elemenata u određenim količinama; treba proizvesti ukupno *b* jedinica gotovog proizvoda; poznate su nabavne cene pojedinih sirovina.

Matematički model ćemo formirati korišćenjem sledećih simbola:

x_j - količina *j*-te sirovine koja će se utrošiti za proizvodnju b jedinica gotovog proizvoda;

p_j - nabavna cena jedinice *j*-te sirovine;

a_{ij} - količina *i*-tog elementa u jedinici j-te sirovine;

a_{i0} - propisana (minimalna ili maksimalna) količina *i*-tog elementa u gotovom proizvodu;

b - količina gotovog proizvoda koju treba proizvesti;

s_j - dozvoljena količina *j*-te sirovine u gotovom proizvodu.

Model se sastoji od funkcije kriterijuma (min); $z_0=\sum_{j=1}^{n} p_j x_j$ i ograničavajućih faktora:

$$\sum_{j=1}^{n} a_{ij} x_j = a_{i0}, \qquad i = 1,2,...,m;$$

$$\sum_{j=1}^{n} x_j = b, \; 0 \le x_j \le s_j, \qquad j = 1,2,...,n.$$

```
MesavinaŠp_,a_,a0_,b_,s_Ć:=
    ModuleŠšm=a,v=a0,n=LengthŠpĆ,pom,ić,
      pom=šć;
      ForŠi=1,i<=n,i++,pom=AppendToŠpom,1ĆĆ;
      m=DodajŠm,špomćĆ;
      m=DodajŠm,-IdentityMatrixŠnĆĆ;
      v=DodajŠv,šbćĆ;    v=DodajŠv,-sĆ;
      LinearProgrammingŠp,m,vĆ
    Ć
MesavinaProcentiŠp_,a_,a0_,s_Ć:=MesavinaŠp,a,a0,1,sĆ
```

5.4. PROBLEMI ISHRANE

Problem se sastoji u sledećem: kako sačiniti program ishrane većeg broja istorodne stoke (iste vrste i iste starosne grupe) tako da izabrana hrana sadrži u dovoljnoj količini sve vrste neophodnih hranljivih sastojaka, a da izdaci za hranu budu minimalni. Savremena stočarska proizvodnja toliko je napredovala da može precizno formulisati brojne zahteve proizvođačima stočne hrane u pogledu kvaliteta te hrane. Ti zahtevi mogu se svesti na sledeće:

a) Sastavijanje obroka je specifično za svaku vrstu stoke, s tim što obrok mora odgovarati biološkim potrebama organizma i mogućnostima iskorišćavanja hrane.

b) Pri sastavljanju obroka poznate su kvantitativne potrebe stoke za pojedinim hranljivim sastojcima.

c) Proizvođač stočne hrane mora poznavati funkciju pojedinih sastojaka hrane u organizmu stoke. To znači da hranljivi sastojci imaju različite funkcije: služe za proizvodnju energije, za stvaranje novih telesnih materija, za obavljanje bioloških funkcija u organizmu itd.

d) Moraju se poznavati sastavi pojedinih elemenata hrane koja će se koristiti u obroku. Jedino tako je moguće učiniti takav izbor i kombinaciju hraniva koji će obezbediti sve ove zahteve.

Odgovarajući matematički model ćemo formirati na osnovu sledećih pretpostavki. Za pripremu krmne smese mogu se koristiti n vrsta hraniva.

Poznate su količine *m* vrsta hranljivih sastojaka koje dnevni obrok mora da sadrži. Na kraju, poznate su i količine pojedinih hranljivih sastojaka sadržane u jedinici svakog hraniva, kao i tržišna cena jedinice hraniva.
Model ćemo formirati korišćenjem sledećih simbola:

x_j - količina *j*-tog hraniva koja će se upotrebiti za dnevni obrok jednog grla stoke;

p_j - nabavna cena jedinice *j*-tog hraniva;

a_{ij} - količina *i*-tog hranljivog sastojka u jedinici *j*-tog hraniva;

a_{i0} - propisana količina *i*-tog hranljivog sastojka koju jedno grlo stoke mora da unese u organizam za jedan dan;

q_j - dozvoljena količina *j*-tog hraniva u jednom obroku za jedno grlo stoke.

Optimalni sastav krmne smeše odredićemo pomoću sledećog modela: treba naći minimalnu vrednost funkcije kriterijuma $z_0=\sum_{j=1}^{n} p_j x_j$ koja mora zadovoljiti sledeći sistem ograničenja:

$$\sum_{i=1}^{n} a_{ij} x_j \geq a_{i0}, \quad i=1,2,\ldots,m, \quad 0 \leq x_j \leq q_j, \quad j=1,2,\ldots,n.$$

```
IshranaŠp_,a_,a0_Ć:=
  ModuleŠšm=a,v=a0,n=LengthŠqĆć,
    m=DodajŠm,-IdentityMatrixŠnĆĆ;v=DodajŠv,-qĆ;
    LinearProgrammingŠp,m,vĆ
  Ć
```

5.5. PRIMENA LINEARNOG PROGRAMIRANJA U POLJOPRIVREDI

Pretpostavimo da poljoprivredno dobro raspolaže sa *h* hektara obradive površine, da na njoj može zasejati *n* vrsta poljoprivrednih kultura, da raspolaže sa *m* vrsta poljoprivrednih mašina i da se za proizvodnju koristi *g* vrsta semena, zaštitnih sredstava i đubriva. Pored toga, poznati su prosečni prinosi svih poljoprivrednih kultura po jednom hektaru, te prodajna cena po jedinici svake kulture i direktni troškovi obrade jednog hektara pod određenom kulturom. Odrediti na kojoj površini zasejati svaku od poljoprivrednih kultura da bi poljoprivrodno dobro ostvarilo maksimalan čist prihod.
Matematički model ćemo formirati korišćenjem sledećih simbola:

x_j - broj hektara na kojima će biti zasejana *j*-ta poljoprivredna kultura;

q_j - prosečni prinos *j*-te kulture po jednom hektaru;

p_j - prodajna cena po jedinici *j*-te kulture;

t_j - direktni troškovi obrade jednog hektara pod j-tom kulturom;

a_{ij} - vreme potrebno i-toj poljoprivrednoj mašini za obradu jednog hektara pod j-tom kulturom;

a_{i0}^k - raspoloživo vreme rada i-te poljoprivredne mašine u k-toj sezoni;

b_j^k - broj radnika koje treba angažovati u k-toj sezoni po jednom hektaru pod j-tom kulturom;

b_0^k - broj radnika sa kojima raspolaže poljoprivredno dobro za k-tu sezonu;

s_{vj} - količina v-tog materijala (semena, zaštitnog sredstva, veštačkog đubriva) potrebna po jednom hektaru pod j-tom kulturom;

s_{v0} - raspoloživa količina v-tog materijala;

h - raspoloživa obradiva površina u hektarima;

$\underline{h_j}$ - minimalna površina pod j-tom kulturom;

$\overline{h_j}$ - maksimalna površina pod j-tom kulturom.

Model se sastoji od funkcije kriterijuma $z_0 = \sum_{j=1}^{n} (q_j p_j - t_j) x_j$ koja označava ukupan čist prihod poljoprivrednog dobra, pa treba naći njenu maksimalnu vrednost uz sledeće ograničavajuće faktore:

$$\sum_{j=1}^{n} a_{ij} x_j \le a_{i0}^k, \quad i = 1,2,\dots,m, \quad k = 1,2,\dots,r$$

$$\sum_{j=1}^{n} b_j^k x_j \le b_0^k, \quad k = 1,2,\dots,r$$

$$\sum_{j=1}^{n} s_{vj} x_j \le s_{v0}, \quad v = 1,2,\dots,g$$

$$\sum_{j=1}^{n} x_j = h,$$

$$\underline{h_j} \le x_j \le \overline{h_j}, \quad j = 1,2,\dots,n.$$

```
PoljoprivredaŠq_,p_,t_,a_,a0_,b_,b0_,s_,s0_,h_,hmin_,hmax_Ć:=
   ModuleŠšr=DimensionsŠa0ĆŠŠ1ĆĆ,m=šć,v=šć,i,qpt,pom,
        n=LengthŠqĆ,vnć,
  vn=DimensionsŠa0Ć;
  ForŠi=1,i<=vnŠŠ1ĆĆ,i++,
    ForŠj=1,j<=vnŠŠ2ĆĆ,j++,v=AppendToŠv,-a0ŠŠi,jĆĆĆ
  Ć Ć;
  qpt=TableŠqŠŠiĆĆ pŠŠiĆĆ-tŠŠiĆĆ,ši,nćĆ;
  ForŠi=1,i<=r,i++,m=DodajŠm,-aĆĆ;
  m=DodajŠm,-bĆ;  m=DodajŠm,-sĆ;   pom=šć;
  ForŠi=1,i<=n,i++,pom=AppendToŠpom,1ĆĆ;
  m=DodajŠm,špomćĆ; m=DodajŠm,IdentityMatrixŠnĆĆ;
  m=DodajŠm,-IdentityMatrixŠnĆĆ;
```

```
  v=DodajŠv,-b0Ć; v=DodajŠv,-s0Ć; v=AppendToŠv,hĆ;
  v=DodajŠv,hminĆ;     v=DodajŠv,-hmaxĆ;
  LinearProgrammingŠ-qpt,m,vĆ
Ć
```

LITERATURA

[1] J. Petrić, *Operaciona istraživanja*, Naučna knjiga, Beograd, 1989.

[2] O. Todorović, *Operaciona istraživanja*, Prosveta, Niš, 1992.

6. VIŠEKRITERIJUMSKA OPTIMIZACIJA

Opšta formulacija višekriterijumske optimizacije (*VKO*) je:

(max) Š$f_1(x)$, $f_2(x)$,... , $f_p(x)$Ć, $p \geq 2$,

p.o. $g_i(x)$<=0, $i = 1, \ldots, m$;

x_j>=0, $j=1,\ldots,n$,

gde su $f_1(x),\ldots, f_p(x)$ i $g_1(x),\ldots, g_m(x)$ realne funkcije od n promenljivih $x = (x_1, x_2,\ldots, x_n)$.

U ovom zadatku se traži resenje x koje maksimizira svih p funkcija cilja. Zato se zadatak *VKO* naziva i zadatkom vektorske optimizacije. Radi jednostavnosti,ovde se razmatraju samo problemi maksimizacije. Poznato je da se zadatak minimizacije jednostavno prevodi u zadatak maksimizacije množenjem kriterijumske funkcije sa -1. Sve nadalje izložene definicije i metode moguće je prilagoditi da važe za rešavanje zadatka minimizacije.

Kažemo da je $X \subset R^n$ *skup dopustivih* rešenja, ako je:

$$X = \{x \in R^n \big| g_i(x) \leq 0,\ \ i = 1,\ldots,m; \quad x_j \geq 0,\ j = 1,\ldots,n\}.$$

Svakom dopustivom rešenju $x \in X$ odgovara skup vrednosti kriterijumskih funkcija, tj. vektor $f(x)=(f_1(x), f_2(x), \ldots, f_p(x)$. Na taj način se skup dopustivih rešenja preslikava u *kriterijumski skup*, tj. S=š$f(x)$đ $x \in X$ć.

U daljem tekstu biće korišćeni sledeci pojmovi:

- Marginalna rešenja zadatka *VKO* se određuju optimizacijom svake od funkcija cilja pojedinačno nad zadatim dopustivim skupom, tj. rešavanjem p jednokriterijumskih zadataka:

  (max) $f_k(x)$, $k=1,\ldots,p,$

  p.o. $x \in X$.

  Marginalna rešenja ćemo obeležavati sa

  $$x^{(k)^*} = (x_1^{(k)^*}, x_2^{(k)^*},\ldots, x_n^{(k)^*})$$

  i svako od njih je rešenje dobijeno optimizacijom k-te funkcije cilja nad zadatiom dopustivim skupom X.

- Idealne vrednsti funkcija cilja $f_k^* = f_k(x^{(k)*})$, $k=1,..., p$ predstavljaju vrednosti funkcija cilja za marginalna rešenja.
- Idealne vrednosti funkcija cilja određuju *idealnu tačku* u kriterijumskom prostoru, tj. idealnu vrednost vektorske funkcije $f^*=(f_1^*, f_2^*, ..., f_p^*)$.
- Ako postoji rešenje x* koje istovremeno maksimizira sve funkcije cilja, tj. x^*=šx đ $f_k(x) = f_k^*$, $k = 1,...,p$ć, onda se takvo rešenje naziva *savršeno rešenje*.

U najvećem broju slučajeva karginalna rešenja se razlikuju i savršeno rešenje ne postoji. Kada ono postoji, tada se u suštini ne radi o problemu *VKO*.

6.1. PARETO OPTIMALNOST

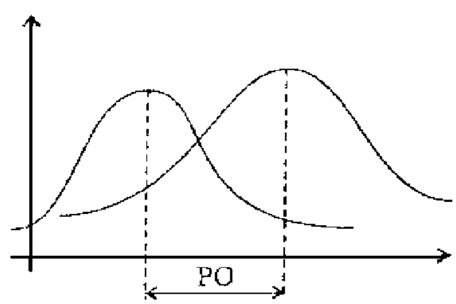
Činjenica da zadaci *VKO* po pravilu nemaju savršeno rešenje upućuje na preispitivanje koncepta optimalnosti i definicije optimalnog rešenja. Ključnu ulogu u tome ima *koncept Pareto optimalnosti*. To je proširenje poznatog koncepta optimalnosti koji se koristi u klasičnoj jednokriterijumskoj optimizaciji.

Pareto optimum se definiše na sledeći način:

- Dopustivo rešenje x^* predstavlja *Pareto optimum* zadatka VKO ako ne postoji neko drugo dopustivo rešenje x takvo da važi:

  $$(\forall k = 1,...,p) \quad f_k(x) \geq f_k(x^*),$$

  pri čemu bar jedna od nejednakosti prelazi u strogu nejednakost ">". Drugim rečima, x je Pareto optimum ako bi poboljšanje vrednosti bilo koje funkcije cilja prouzrokovalo pogoršanje vrednosti neke druge funkcije cilja (slika 1). Za Pareto oprimum koriste se sledeći sinonimi: *efikasno*, *dominantno* i *nedominirano* rešenje.



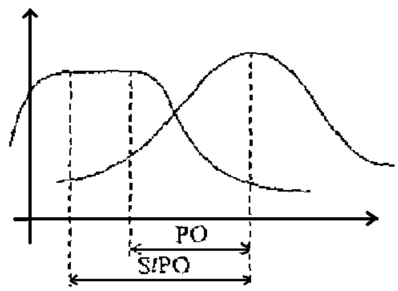
slika 1

Pored Pareto optimuma definišu se slabi i strogi (jaki) Pareto optimumi.

Dopustivo rešenje x^* je *slabi Pareto optimum* ako ne postoji neko drugo dopustivo rešenje x takvo da važi:

$$(\forall k = 1,..., p) \quad f_k(x) > f_k(x^*).$$

Drugim rečima, x^* je slabi Pareto optimum ako nije moguće istovremeno poboljšati sve funkcije cilja (slika 2).



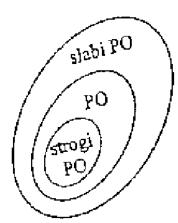
slika 2

Pareto optimalno rešenje x^* je *strogi Pareto optimum* ako postoji broj β>0 takav da za svaki indeks $k\in$š1,.., pć i svako x za koje ne postoji savršeno rešenje i zadovoljava uslov $f^k(x) > f^k(x^*)$, postoji bar jedno $l\in$š1,..., pćĐšk$ć$ takvo da je $f^l(x) < f^l(x^*)$ i da važi:

$$\frac{f^k(x) - f^k(x^*)}{f^l(x^* - f^l(x)} \geq \beta \,.$$

Strogi Pareto optimum izdvaja ona Pareto rešenja čije promene ne prouzrokuju prevelike relativne promene u funkcijama cilja.

Odnos između opisanih optimuma je takav da svaki skup strožijih Pareto optimuma predstavlja podskup slabijih optimuma, tj. svaki Pareto optimum je istovremeno i slabi Pareto optimum, a svaki strogi Pareto optimum je i Pareto optimum. Odnos svih skupova dat je na slici 3.



slika 3

6.2. METODE ZA REŠAVANJE ZADATAKA VKO

Prema pristupu rešavanju zadatka metode *VKO* klasifikuju se u sledećce tri grupe:

1) *A posteriori metodi* u kojima se donosilac odluke informiše o dominantnim (Pareto optimalnim) rešenjima matematičkog modela, a on na osnovu njih donosi konačnu odluku.
2) *A priori metodi* u kojima se informacije o odnosu do prema kriterijumima ugrađuju u matematički model ili metodu *a priori*, tj. pre bilo kakvog rešavanja modela, a konačna odluka se odnosi na osnovu tako dobijenog rešenja.
3) *Interaktivne metode* u kojima donosilac odluke aktivno učestvuje tokom rešavanja modela. U njima se iterativno kombinuju metode iz prethodne dve grupe, tj. donosilac odluke prvo daje preliminarne informacije o svojim preferencijama, a zatim kada dobije rešenje, može promeniti informacije ili rešenje. Ovaj postupak se iterativno ponavlja sve dok donosilac odluke bude konačno zadovoljan dobijenim rešenjem.

Od apriori metoda obradićemo relaksiranu leksikografsku metodu i metodu ε ograničenja i objasniti princip grupe metoda nazvanih metode rastojanja. Predstavnik interaktivnih metoda leste metod interaktivnog kompromisnog programiranja.

6.2.1. Leksikografska višekriterijumska optimizacija

Često se do optimalnog rešenja dolazi posle uzastopnog donošenja odluka. Prvo se nađe optimalno rešenje za najvažniju funkciju cilja. Ako je optimalno rešenje jedinstveno, tada je problem rešen. Međutim, ako optimalno rešenje nije jedinstveno, tada se na skupu svih optimalnih rešenja optimizuje funkcija cilja koja je druga po važnosti. Ako je optimalno rešenje sada jedinstveno, problem je rešen; ako nije, optimizuje se funkcija cilja treća po važnosti na skupu optimalnih rešenja prve i druge funkcije cilja itd.
Dakle posmatrajmo problem minimizacije date uređene sekvence ciljnih funkcija:
$Q_1(x),...,Q_l(x)$ i skup ograničenja oblika nejednakosti:

$$F:\ f_i(x) \leq 0,\ \ i=1,...,p.$$

Treba rešiti sledeći skup konveksnih uslovnih nelinearnih programa

$$\begin{array}{lll} \text{(min)} & Q_k(x), & x \in R^n \\ \text{p.o.} & Q_{k-1}(\text{x}) \leq a_{k-1}, & \\ & \dots\ \dots & \\ & Q_1(x) \leq a_1, & \end{array}$$

$$x \in F, \quad k=1,...,l$$

gde su a_i, $i=1,...,k$-1 optimalne vrednosti predhodno postavljenih i rešenih problema (L_i), $i=1,...,k$-1.
Jedan od načina za implementaciju višekriterijumske optimizacije dat je funkcijom *MultiLex*ŠĆ.

```
MultiLexŠq_List,constr_ListĆ:=
 ModuleŠšres=šć,f=constr,Lis=šć,l=LengthŠqĆ ,k=1ć,
  Lis=VariablesŠqĆ;
  WhileŠk<=l,
     rez=FirstŠConstrainedMaxŠqŠŠkĆĆ,f,LisĆĆ;
     AppendToŠf,qŠŠkĆĆ>=rezĆ; AppendToŠres,rezĆ;
     k=k+1;
  Ć ;
  ReturnŠresĆ;
 Ć
```

Za gore naveden problem rešenje se dobija pozivom funkcije
MultiLexŠš8x1+12x2,14x1+10x2,x1+x2ć,
š8x1+4x2<=600,2x1+3x2<=300,4x1+3x2<=360, 5x1+10x2>=600,x1>=0,x2>=0ćĆ
pri čemu se dobija rešenje:
š1200,1220,110ć

Mogu se naglasiti sledeće prednosti koje proizilaze iz simboličke implementacije leksikografski višekriterijumskih problema:

(1) Mogućnost izbora i zamene kako izabrane ciljne funkcije tako i funkcija koje predstavljaju zadata ograničenja, iz liste koja predstavlja unutrašnju formu postavljenog problema, u listu formalnih parametara procedure kojom se implementira uslovni optimizacioni metod.
(2) Jednostavna konstrukcija unutrašnje forme koja predstavlja uslovni program (L_k). Ovo svojstvo proizilazi iz simboličke manipulacije sa listama, kojom je omogućeno dodavanje na početak ograničenja tipa nejednakost $Q_i(x) \leq a_i$, $1 \leq i \leq l$ u listu ograničenja tipa nejednakost, koja je do tada formirana.
(3) Simbolička obrada dozvoljava da se funkcije koriste kao objekti prvog reda. Između ostalog, mogu da se koriste nizovi funkcija, čiji se elementi kasnije mogu selektovati i primenjivati na zadatu listu argumenata. Ovakve strukture nisu podesne za konstrukciju u proceduralnim programskim jezicima.

6.2.2. Metod težinskih koeficijenata

Metod težinskih koeficijenata je najstariji metod za *VKO*. Ovaj metod uvode koristi težinske koeficijente w_i za sve kriterijumske funkcije $f_i(x)$, i=1,...,n, pa se problem optimizacije svodi na sledeću skalarnu optimizaciju

$$(\max)\qquad \sum_{i=1}^{n} w_i f_i(x)$$

$$\text{p.o.}\qquad g_i(x)<=0,\quad i=1,2,\ldots,m,$$
$$x_j>=0,\ j=1,\ldots,n.$$

Često se metod težinskih koeficijenata koristi tako što se zadaju vrednosti ovih koeficijenata. Međutim, to uvek izaziva određene teškoće i primedbe na ovakav postupak, jer se unosi subjektivan uticaj na konačno rešenje preko zadatih vrednosti težinskih koeficijenata.
Funkcijom *Compositions*Š*n,k*Ć možemo odrediti "*k*-dimenzionalne tačke", čiji zbir koordinata daje *n*. Kada tu listu podelimo brojem *n*, možemo dobiti tačke u intervalu Š0,1Ć čije koordinate mogu predstavljati koeficijente w_i. Na taj način smo obezbedili automatsko određivanje koeficijenata w_i potrebnih za realizaciju metoda. Ostavljena je i mogućnost da sami izaberemo koeficijente w_i. To se postiže tako što pri pozivu funkcije *MultiW*Š Ć, kojom se implementira metoda težinskih koeficijenata, četvrtu koordinatu zadamo da bude 0, a petu koordinatu, koja je lista koeficijenata w_i, zadamo kao nenultu listu, tj. zadamo koeficijente w_i u obliku liste. Ukoliko je četvrta koordinata različita od 0 to je znak algoritmu da sam generiše koeficijente za w_i.
Sledi program kojim je implementirana metoda težinskih koeficijenata.

Ulazne veličine:
q_, pr_List - ciljna funkcija i lista njenih parametara;
PO_List - lista ograničenja;
*kor_* - korak podele intervala Š0,1Ć;
*w1*_List - lista težinskih koeficijenata u intervalu Š0,1Ć.
Lokalne promenljive:
fun - formirana funkcija;
rmax - maksimum funkcije pod datim ograničenjima.

```
MultiWŠq_List,pr_List,PO_List,kor_,w1 _ListĆ:=
 BlockŠšq0=q,prom=pr,fun,k=kor,i=0,L1=šć,w=šćć,
   l=LengthŠq0Ć;
   IfŠk==0, w=w1;k=LengthŠw1Ć-1,   w=CompositionsŠk,1Ć/kĆ;
   WhileŠi<=k,i+i+1;
     fun=SimplifyŠSumŠwŠŠi,jĆĆ*q0ŠŠjĆĆ,šj,1,lćĆĆ;
```

```
    rmax=ConstrainedMaxŠfun,PO,promĆ; L1=AppendŠL1,rmaxĆ;
  Ć;
  PrintŠL1Ć
Ć
```

Primer. Neka su kriterijumske funkcije $f_1=x_1+x_2$, $f_2=2x_1-x_2$. Zadatak je da se odredi rešenje koje maksimizira obe funkcije, tj.
(max) Šf_1, f_2Ć uz ograničenja

$$-3x_1+5x_2 \le 9,$$
$$3x_1+2x_2 \le 12,$$
$$5x_1-4x_2 \le 9,$$
$$x_1,x_2 >= 0.$$

Potražimo prvo rešenja problema ne zadavajući koeficijente w_i.

MultiWŠšx+y,2x-yć,šx,yć, š-3x+5y<=9,3x+2y<=12,5x-4y<=9,x>=0,y>=0ć, 6,šćĆ

```
šš9/2,šx->3,y->3/2ćć,š9/2,šx->3,y->3/2ćć,š9/2,šx->3,y->3/2ćć,
  š9/2,šx->3,y->3/2ćć,9/2,šx->3,y->3/2ćć,š9/2,šx->3,y->3/2ćć, š5,šx->2,y->3ććć.
```

Sada zadajemo sami vrednosti za w_i.

MultiWŠšx+y,2x-yć,šx,yć, š-3x+5y<=12,5x-4y<=9,x>=0,y>=0ć, 0,

šš1,0ć, š0.9,0.1ć, š0.875,0.125ć, š0.8,0.2ć,š0,1ććĆ

```
šš5,šx->2,y->3ćć,š4.6,šx->2,y->3ćć,š4.5,šx->3,y->1.5ćć,š4.5,šx->3,y->1.5ćć,
  š9/2,šx->3,y->3/2ććć
```

6.2.3. Relaksirana leksikografska metoda

Ovaj metod je iterativni postupak u kome se u svakoj iteraciji rešavaju odgovarajući jednokriterijumski zadaci optimizacije. Pretpostavlja se da su od strane donosioca odluke dati prioriteti kriterijuma i da su u skladu sa njima dodeljeni indeksi kriterijumima. U relaksiranoj leksikografskoj metodi se po svakom od p kriterijuma rešava jednokriterijumski zadatak optimizacije. Pri tome se u narednoj iteraciji ne postavlja kao ograničenje zahtev da rešenje bude optimalno po kriterijumu višeg prioriteta, več se ono relaksira tako da se zahteva da rešenje bude u okolini optimalnog rešenja dobijenog u prethodnoj iteraciji. Na taj način, svaki kriterijum utiče na konačno rešenje.
donosilac odluke zadaje redosled kriterijuma po značajnosti. Pored toga, svakom kriterijumu, uzimajući poslednji, donosilac odluke deljuje se vrednost α_k, k=1,..., p-1, za koju kriterijum višeg prioriteta sme da odstupi

od svoje optimalne vrednosti. Metoda obuhvata izvršavanje sledećih p koraka:

1. (max) f_1(x), p.o. $x \in X$.

Ako je rešenje ovog problema jednako f_1^*, u sledećem koraku se rešava problem

2. (max) f_2(x), p.o. $x \in X$; $f_1(x) \geq f_1^* - \alpha_1$.

Proces se analogno nastavlja. U p-tom koraku rešava se problem

3. (max) f_p(x), p.o. $x \in X$; $f_l(\mathrm{x}) \geq f_l^* - \alpha_l$, $l = 1,...,p-1$.

Za rešenje polaznog modela se usvaja rezultat dobijen u poslednjem koraku, a vrednosti funkcija cilja za dobijeno rešenje se moraju posebno računati. Rešenje dobijeno ovim metodom obezbeđuje slabi Pareto optimum, a ako je rešenje jedinstveno, ono je i Pareto optimalno. Ako je dopustiva oblast konveksna, podešavanjem parametara α_k, k=1,..., p-1, može se dobiti bilo koje Pareto optimalno rešenje.

Primer. Primenom relaksirane leksikografske metode rešićemo sledeći zadatak *VKO* (funkcije cilja su relaksirane po prioritetu):

$$(\max) \quad \check{S}f_1(x),\ f_2(x),\ f_3(x)\acute{C}$$

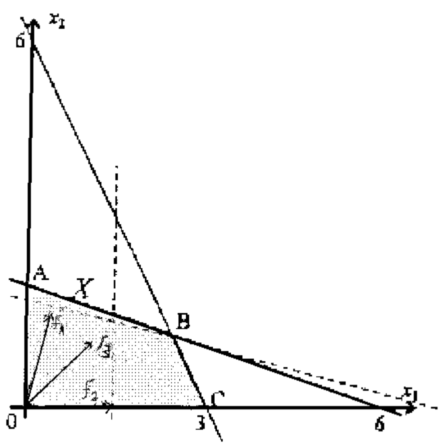
$$\text{p.o.} \quad 2x_1+x_2 \leq 6,\ x_1+3x_2 \leq 6,\ \ x_1, x_2 \geq 0$$

ako je zadato:

$$2\, f_1(x)=x_1+4x_2, \qquad \alpha_1=1$$

$$f_2(x)=x_1, \qquad \alpha_2=1$$

$$f_3(x)=x_1+x_2.$$



Slika 4

Korak 1. (max) $f_1(x) = x_1+4x_2$ p.o. $x \in X$.

Ovaj zadatak ima jedinstveno optimalno rešenje u tački x^{1*}=(0,2), pri čemu je f_1^*=8.

Korak 2. (max) $f_2(x)=x_1$ p.o. $x \in X$, $x_1+4x_2 \geq 7$.
Dobija se rešenje: $x^{2*}=(2.43, 1.14)$, $f_2^*=2.43$
Korak 3. (max) $f_3(x)=x_1+x_2$ p.o. $x \in X$, $x_1+4x_2 \geq 7$, $x_1 \geq 1.43$.
Rešenje ovog zadatka se usvaja kao konačno. To je rešenje: $x_1^*=3.6$; $x_2^*=1.2$.
Vrednosti funkcija cilja u ovoj tački su: $f^*=(7.2, 2.4, 3.6)$.

Relaksirana leksikografska metoda je veoma osetljiva na izbor koeficijenata α_k, u tolikoj meri da se "pogrešnim" izborom mogu dobiti neprihvatljiva rešenja. Tako u prethodnom primeru imamo slučaj da se konačno resenje poklapa sa marginalnim rešenjem funkcije cilja koja ima najniži prioritet, dok je vrednost najznačajnijeg kriterijuma smanjena. Kod primene ove metode se preporučuje da donosilac odluke kritički preispita dobijena rešenja, uporedi ih sa marginalnim i da po potrebi koriguje zadate koeficijente.

Jedan od načina za implementaciju relaksirane leksikografske metode dat je funkcijom *MultiRelax*Š Ć:
Ulazne veličine:
q - lista ciljnih funkcija;
parcf - parametri ciljnih funkcija;
constr - lista ograničenja.

```
MultiRelaxŠq_,parcf_,constr_Ć:=
 ModuleŠšls=šć,con=constr,rez=šć,i,lć,
  l=LengthŠqĆ;
  ForŠi=1,i<=l,i++,
   ls=ConstrainedMaxŠqŠŠiĆĆ,con,VariablesŠqĆĆ;
   rez=FirstŠlsĆ;
   IfŠi<l, AppendToŠcon,qŠŠiĆĆ>=rez-parcfŠŠiĆĆĆ  Ć;
  Ć;
  rez=LastŠlsĆ;  ReturnŠšq/.rez,rezćĆ
 Ć
```

6.2.4. Metod ε ograničenja

U ovom metodu donosilac odluke izdvaja kriterijum $f_q(x)$ koji ima najviši prioritet i njega maksimizira, dok ostale funkcije cilja ne smeju imati vrednosti manje od unapred zadatih ε_k, $k=1,\ldots,p$, $k \neq p$. Drugim rečima, rešava se sledeći jednokriterijumski zadatak:

$$
\begin{array}{ll}
\text{(max)} & f_q(x) \\
\text{p.o.} & g_i(x) \leq 0, \quad i=1,\ldots,m,
\end{array}
$$

$$f_k(x) \geq \varepsilon_k, \quad k=1,...,p, \quad k \neq p,$$
$$x_j \geq 0, \; j = 1,...,n,$$

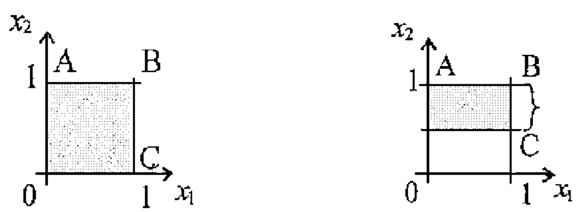
čije se rešenje usvaja kao rešenje polaznog zadatka *VKO*. Ako postavljeni zadatak nema dopustivo rešenje, potrebno je smanjiti vrednosti $\varepsilon_{k.}$ Slično prethodnoj metodi, rešenje dobijeno ovim postupkom osetljivo je na izbor parametara koje daje donosilac odluke. Zato se preporučuje da donosilac odluke aktivno učestvuje u eksperimentima na modelu i da u slučaju potrebe koriguje zadate koeficijente.

Primer. Rešiti sledeći zadatak *VKO:*

(max) Š$f_1(x)=x_1$, $f_2(x)=x_2$Ć p.o. $0\leq x_j\leq 1$, j=1,2

metodom ε ograničenja, ako prvi kriterijum ima najviši prioritet, dok drugi ne sme imati vrednost manju od 0.5.
Skup dopustivih rešenja za polazni zadatak prikazan je na slici. Na početku rešavamo sledeći zadatak linearnog programiranja:

(max) $f_1(x)=x_1$ p.o. $0 \leq x_1 \leq 1$; $1/2\leq x_2 \leq 1$,



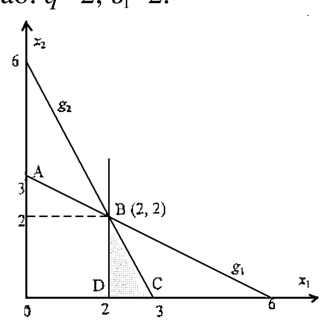
slika 5

čije je rešenje višestruko (slika 5) $x_1^*=1$, $x_1^*\in$Š1/2,1Ć, a vrednosti funkcija cilja su $f_1^*=1$ i u zavisnosti od izbora višestrukog rešenja $f_2^*\in$Š1/2,1Ć.

Primer. Rešiti zadatak *VKO*:

(max) Š$f_1(x)=x_1$, $f_2(x)=x_2$Ć p.o. $x_1+2x_2\leq 6$; $2x_1+x_2\leq 6$, $x_1, x_2 \geq 0$.

ako je donosilac odluke zadao: q=2, ε_1=2.



slika 6

Model transformišemo na sledeći način:

$$(\max)\ \ f_2(x)=x_2 \quad \text{p.o.} \quad x_1+2x_2 \le 6\ ; \quad 2x_1+x_2 \le 6,\ \ x_1 \ge 2,\ x_2 \ge 0.$$

Rešenje ovog modela je jedinstveno (tačka B na slici) i Pareto optimalno:

$$x_1^*=2,\ \ x_2^*=2,\ \ f_1^*=2,\ \ f_2^*=2.$$

Sledećom funkcijom *MultiEpsilon*Š Ć biće dat jedan način implementacije ε ograničenja.

Ulazne veličine:

q - lista ciljnih funkcija

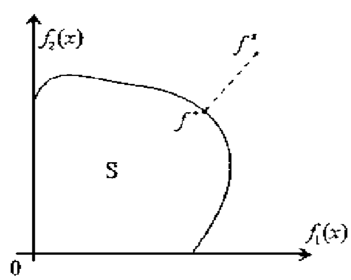
parcf - parametri ciljnih funkcija

constr - lista ograničenja.

```
MultiEpsilonŠq_,parcf_,constr_Ć:=
  ModuleŠšlisrez=šć,con=constr,rez=šć,i,l,j=1ć,
    l=LengthŠqĆ;
    ForŠi=2,i<=l,i++,
        AppendToŠcon,qŠŠiĆĆ>=parcfŠŠi-1ĆĆĆ
    Ć;
    lisrez=ConstrainedMaxŠqŠŠjĆĆ,con,VariablesŠqĆ Ć;
    rez=LastŠlisrezĆ;  ReturnŠšq/.rez,rezćĆ
 Ć
```

6.2.5. Metodi rastojanja

Metodi rastojanja čine grupu za rešavanje zadataka *VKO* čija je osnovna ideja da se u kriterijumskom prostoru traži tačka koja je najbliža nekoj unapred određenoj tački koja se želi dostići ili ka kojoj treba težiti ako ona nije dopustiva. Drugim rečima, minimizira se rastojanje između željene tačke i dopustive oblasti (slika 7). Razlike između pojedinih metoda ove



slika 7

grupe se baziraju na algoritmu prema kome željena tačka određuje, na načinu prema kome se rastojanje od nje "meri", da li se uvode težinski koeficijenti, itd.

U opštem slučaju, *donosilac odluke* za svaki od p kriterijuma zadaje željene vrednosti ili određuje način kako će se one izračunati. Na taj način se u p-dimenzionalnom kriterijumskom prostoru definiše tačka $f^z=(f_1^z,...,f_p^z)$ koja po pravilu ne pripada dopustivoj oblasti S. U slučaju da je $f^z \in S$ zadatak se rešava rešavanjem sistema jednačina koje su definisane skupom ograničenja.

U metodu rastojanja rešava se sledeći opšti zadatak:

$$\text{(min)} \quad d(f^z\text{-}f(x)) \ \text{ p.o. } x \in X$$

gde d(*.*) označava rastojanje definisano pogodnom metrikom.

U kontekstu merenja rastojanja u kriterijumskom prostoru ovde ćemo ponoviti definicije metrika koje se najčešće koriste u metodama rastojanja:

l_1 (pravougaona) metrika: $d_{l1}(f^z, f(x))=\sum_{k=1}^{p}\left|f_k^{\,z}-f_k(x)\right|$,

l_2 (Euklidova) metrika: $d_{l2}(f^z, f(x))=\sqrt{\sum_{k=1}^{p}(f_k^z-f_k^2(x))}$,

l_∞ (Čebiševljeva) metrika: $d_{l\infty}(f^z, f(x))=\max_{1\le k\le p}\left|f_k^z-f_k(x)\right|$.

Kriterijumima je moguće dodeliti težinske koeficijente, tako da prethodne formule za rastojanje između željenog i traženog rešenja dobijaju oblik:

$$d_{l1}(f^z, f(x))=\sum_{k=1}^{p}w_k\left|f_z^k-f_k(x)\right| \quad \text{za } \textit{pravougaonu} \text{ metriku,}$$

$$d_{l2}(f^z, f(x))=\sqrt{\sum_{k=1}^{p}w_k(f_k^z-f_k^2(x))} \quad \text{za } \textit{Euklidovu} \text{ metriku,}$$

$$d_{l\infty}(f^z, f(x))=\max_{1\le k\le p} w_k\left|f_k^z-f_k(x)\right| \quad \text{za } \textit{Čebiševljevu} \text{ metriku.}$$

Bez obzira na oblik polaznih funcija cilja, zadaci minimizacije rastojanja su problemi nelinearnog programiranja, za koje, u opštem slučaju, nije jednostavno pronaći optimalno rešenje. Izuzetno se u slučaju l_1 metrike i linearnih funkcija cilja i ograničenja zadatak minimizacije rastojanja može formulisati kao problem *LP*.

Primer. Zadatak *VKO*

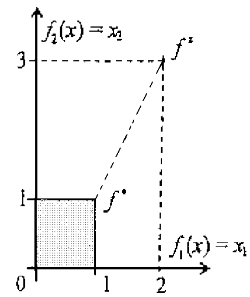
$$\text{(max)} \quad \text{Š}f_1(x)=x_1,\ f_2(x)=x_2\text{Ć} \quad \text{p.o.} \quad 0\le x_j \le 1,\ j=1,2$$

rešiti metodom rastojanja koristeći pravougaonu metriku, ako se žele dostići vrednosti kriterijuma: $f_1^{\,z}=2$ i $f_2^{\,z}=3$.

Zbog specifičnog oblika funkcija cilja, dopustivi skup i kriterijumski skup će imati isti oblik (slika 8). Mada je sa slike očigledno da je dopustiva tačka

f^*=(1,1) najbliža željenoj, pokazaćemo to analitičkim putem. Rešava se sledeći zadatak:

(min) $\pi(x)$= đ 2 Đ x_1đ + đ 3 Đ x_2đ p.o. $x_1 \leq 1$, $x_2 \leq 1$, $x_1, x_2 \geq 0$.



slika 8

Kada smo u rešavanju konkretnog zadatka sigurni da vrednosti kriterijumskih funkcija neće biti veće od zadatih željenih vrednosti, možemo slobodno da zanemarimo operatore apsolutne vrednosti jer je

đ$f_k^{\,z}$ Đ f_k(x)đ=$f_k^{\,z}$ Đ $f_k(x)$.

U konkretnom primeru to znaći da bismo rešavali zadatak (min) Š5-x_1-x_2Ć na datom dopustivom skupu. Rešenje ovog zadatka je x^*=(1,1) i ono je istovremeno rešenje polaznog nelinearnog modela. Međutim, opisani postupak zanemarivanja apsolutnih vrednosti u opštem slučaju ne bi bio korektan. Zato se koristi sledeći način za oslobađanje od apsolutnih vrednosti.

Najpre se definišu promenljive y_k=f_k^z-f_k(x), k=1,...,p, koje predstavljaju odstupanja funkcija cilja od željenih vrednosti. U ovom slučaju uvode se smene y_1=2-x_1 i y_2=3-x_2**,** tako se dobija model:

(min) $F(x,$y) = đ y_1 đ + đ y_2 đ p.o. $x_1 + y_1 = 2$, $x_2 + y_2 = 3$, 0≤ x_1≤1, 0 ≤ x_2≤ 1.

Treba primetiti da je $F(x,y)$ funkcija od y, a time i implicitno i funcija od x.
Odstupanje y_k može biti pozitivno ili negativno. Kada je pozitivno, pokazuje za koliko je $f_k(x)$ veća od željene vrednosti f^z i zato se naziva prebačaj i obeležava sa y_k^+. Kada je odstupanje negativno, ono pokazuje za koliko je $f_k(x)$ manje od željene vrednosti $f^{\,z}$ i zato se naziva *podbačaj* i obeležava sa y_k^-. Jasno je da važi y_k=y_k^+ - y_k^- i y_k^+ * y_k^- *=0,* jer za jedan kriterijum ne možemo istovremeno imati i prebačaj i podbačaj.
Prema tome, u konkretnom zadatku uvodimo sledeće smene:

$$y_1 = y_1^- - y_1^+ \text{ i } y_2 = y_2^- - y_2^+.$$

Apsolutne vrednosti mogu se napisati kao:

$$\text{đ } y_1 \text{ đ } = y_1^+ + y_1^- \ \ i \ \ \text{đ } y_2 \text{ đ} = y_2^+ + y_2^-, \text{ jer je } y_1^+, y_1^-, \ y_2^+, y_2^- \geq 0.$$

Na ovaj način se polazni zadatak metode rastojanja transformiše u sledeći oblik:

(min) $F(\mathrm{x,y}) = y_1^+ + y_1^- + y_2^+ + y_2^-$

p.o. x_1 Ð $y_1^+ + y_1^- = 2$, x_2 Ð $y_2^+ + y_2^- = 3$,

$\mathrm{x}_1, \mathrm{x}_2 \leq 1$, $\mathrm{x}_1, \mathrm{x}_2, \mathrm{y}_1^+, \mathrm{y}_1^-, \mathrm{y}_2^+, \mathrm{y}_2^- \geq 0$.

Ovo je klasičan zadatak LP čije se rešenje može dobiti simpleks algoritmom, i ono je:

$$y_1^+ = 0, \ \ \mathrm{y}_1^- = 1, \ \ \mathrm{y}_2^+ = 0, \ \ \mathrm{y}_2^- = 2, \ \ f_1^* = \mathrm{x}_1^* = 1, \ \ f_2^* = \mathrm{x}_2^* = 1.$$

Primer. Dat je zadatak *VLP*

(max) Š$f_1(\mathrm{x}),\ f_2(\mathrm{x}),\ f_3(\mathrm{x})$Ć

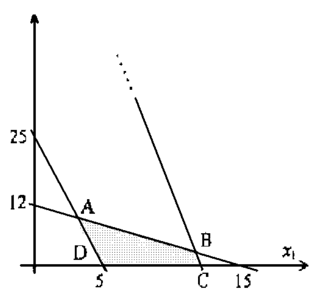
p.o. $35\, x_1 + 7\, x_2 \geq 175$; $136\, x_1 + 170\, x_2 \leq 2400$, $20x_1 + 3x_2 \leq 240$,

$x_1, x_2 \geq 0$,

gde su:

$$f_1(\mathrm{x}) = 4\, x_1 + 5\, x_2, \ \ f_2(\mathrm{x}) = x_1 + x_2, \ \ f_3(\mathrm{x}) = 40\, x_1 + 6\, x_2.$$

Rešiti problem metodom rastojanja koristeći ravougaonu metriku, ako se želi dostići idealna tačka.



slika 9

U ovom problemu donosilac odluke je odredio da su željene vrednosti kriterijumskih funkcija idealne vrednosti tih funkcija. Zato je najpre potrebno odrediti marginalna rešenja i odgovarajuće idealne vrednosti funkcije:

$f_1^* = 60$ za višestruko rešenje koje pripada duži *AB* sa krajnjim tačkama:

$$x_1^{(1)*} = 3.09, \ \ x_2^{(1)*} = 9.52 \ \text{ i } \ x_1^{(1)**} = 11.59, \ \ x_2^{(1)**} = 2.73.$$

$f_2^* = 14.32$ za: $\mathrm{x}_1^{(2)*} = 11.59$, $\mathrm{x}_2^{(2)*} = 2.73$ (tačka *B*),

$f_3^* = 480$ za višestruko rešenje koje pripada duži *BC* sa krajnjim tačkama:

$$\mathrm{x}_1^{(3)**} = 11.59, \ \ \mathrm{x}_2^{(3)**} = 2.73 \ \text{ i } \ \mathrm{x}_1^{(3)**} = 12, \ \ \mathrm{x}_2^{(3)**} = 0.$$

Zadatak *VKO* prilagođen za rešavanje metodom rastojanja sada ima oblik:

$$(\min)\ \pi(x) = \text{đ}\, f_1^* - f_1(x)\ \text{đ} + \text{đ}\, f_2^* - f_2(x)\ \text{đ} + \text{đ} f_3^* - f_3(x)\text{đ}$$

pri istim ograničenjima.

S obzirom da sve kriterijume trebamo maksimizirati, a po definiciji marginalnih rešenja je $f_k^* \geq f_k(x)$, može se staviti da je $\text{đ}\, f_k^* - f_k(x)\ \text{đ} = f_k^* - f_k(x)$, k=1,2,3. Tada se umesto polaznog zadatka može rešavati zadatak minimizacije funkcije

$$\pi(\boldsymbol{x}) = 554.32 - 45\, x_1 - 12\, x_2,$$

pri istim ograničenjima. Pokazaćemo kako se ovaj zadatak rešava opštim postupkom koji koristi definiciju prebačaja i podbačaja. U tom slučaju matematićki model ima oblik :

$$
\begin{array}{ll}
(\min) & F(x,y) = y_1^+ + y_1^- + y_2^+ + y_2^- + y_3^+ + y_3^- \ , \\
\text{p.o.} & 4x_1 + 5x_2\ \text{Đ}\ y_1^+ + y_1^- = 60, \\
 & \mathrm{x}_1 + \mathrm{x}_2\ \text{Đ}\ \mathrm{y}_2^+ + \mathrm{y}_2^- = 14.32\ , \\
 & 40x_1 + 6x_2\ \text{Đ}\ y_3^+ + y_3^- = 480, \\
 & 35x_1 + 7x_2 \geq 175, \\
 & 136\, x_1 + 170\, x_2 \leq 2400\ , \\
 & 20\mathrm{x}_1 + 3\mathrm{x}_2 \leq 240\ , \\
 & x_1, x_2, y_1^+, \mathbf{y}_1^-, y_2^+, y_2^-, y_3^+, y_3^- \geq 0.
\end{array}
$$

Rešavanjem ovog zadatka dobija se jedinstveno rešenje :

$$F^* = 0,\ \ x_1^* = 11.59,\ \ x_2^* = 2.73.$$

To što je vrednost funkcije cilja jednaka nuli ukazuje da je dobijeno rešenje savršeno jer nema ni prebačaja ni podbačaja vrednosti funkcija cilja od idealne tačke. Može se primetiti da u tački *B* sve funkcije cilja dostižu svoju najveću vrednost.

Ulazne veličine:
q - lista ciljnih funkcija
constr - lista ograničenja.

```
idealŠq_,constr_Ć:=
  ModuleŠšlisrez=šć,con=constr,rez=šć,i,lć,
   l=LengthŠqĆ;
   ForŠi=1,i<=l,i++,
    AppendToŠlisrez,FirstŠConstrainedMaxŠqŠŠiĆĆ,
      con,VariablesŠqĆĆĆĆ;
   Ć;
   ReturnŠlisrezĆ;   Ć

MultiDistŠq_,constr_,w1_Ć:=
 ModuleŠšw=w1,param=šć,con=constr,fun=šć,l,a=šć,
       lisrez=šć,yy=šćć,
```

```
l=LengthŠqĆ;
ForŠi=1,i<=l,i++,AppendToŠyy,qŠŠiĆĆ Ć Ć;
IfŠw==šć,fun=idealŠq,conĆ ,fun=wĆ;
ForŠi=1,i<=l,i++,
   AppendToŠcon,qŠŠiĆĆ-$Š2i-1Ć+$Š2iĆ<=funŠŠiĆĆĆ;
   AppendToŠcon,qŠŠiĆĆ-$Š2i-1Ć+$Š2iĆ>=funŠŠiĆĆĆ;
Ć;
ForŠi=1,i<=2l,i++, AppendToŠcon,$ŠiĆ>=0Ć Ć;
AppendToŠyy,ArrayŠ$,2l,1ĆĆ;
lisrez=ConstrainedMinŠArrayŠ$,2l,1,PlusĆ,con,VariablesŠyyĆĆ;
ReturnŠLastŠlisrezĆĆ Ć;
```

6.2.6. Interaktivno kompromisno programiranje

Interaktivni metodi za rešavanje zadataka *VKO* podrazumevaju aktivno učešće donosioca odluke u procesu modeliranja problema, generisanja mogučih rešenja, njihove analize i konačnog usvajanja. Razvoj pogodnih metoda zahteva korišćenje odgovarajućih matematičkih modela i poznavanje čovekovog ponašanja u procesu odlučivanja. Algoritam koji će ovde biti prikazan kao predstavnik ovih metoda razvijen je na osnovu sledećih ideja.

Uočeno je da donosilac odluke lakše može da poredi alternative ako se uz vrednosti kriterijuma iskaže i koliko je koje rešenje blizu svoje idealne vrednosti. Pri tome je pogodno da se rastojanja predstave na skali od 0 do1.

Kako god da izražava svoje stavove prema konkretnim rešenjima ili kriterijumima, u velikom broju zadataka donosilac odluke se eksplicitno ili implicitno ponaša kao da kriterijumima dodeljuje težinske koeficijente. Nevolja je što donosilac odluke ne može unapred da zna kakve će posledice imati određivanje težina, odnosno, ne može svoje osećaje i namere da iz prvog pokušaja iskaže pomoću brojeva koji predstavljaju težinske koeficijente. Zato mu treba pomoći da iterativno dođe do onih vrednosti koje mu najviše odgovaraju.

Posmatrajmo zadatak *VLP*

$$(\max) \qquad [f_1(\mathrm{x}),\dots,f_p(\mathrm{x})] \qquad \text{p.o.} \quad x\in X=\{x\in R^n,\ x\geq 0,\ Ax\leq b,\ b\in R^m\}$$

gde su $f_1(x),\dots,f_p(x)$ linearne funkcije.

U ovom algoritmu se umesto rastojanja od idealne tačke, koje je korišćeno u predhodnim modelima, koristi koncept stepena bliskosti. Stepen bliskosti d_k koji ima rešenje x_k optimalnoj vrednosti po kriterijumu k je:

$$d_k(x)=\frac{f_k(x)-f_k{}^L}{f_k{}^U-f_k{}^L},\ \ k=1,\dots p,$$

gde su:

f_k(x) - vrednost kriterijumske funkcije za rešenje x,
f_k^U - maksimalna moguća vrednost kriterijumske funkcije f_k na dopustivom skupu X,
f_k^L - minimalna moguća vrednost kriterijumske funkcije f_k na dopustivom skupu X.
Očigledno da $d_k(x)$ uzima vrednosti između 0 i 1. Stepen bliskosti konkretnog rešenja pokazuje koliko je to rešenje blisko maksimalno mogoćoj vrednosti kriterijumske funkcije f_k na dopustivom skupu X.
Umesto originalnog problema sada se rešava sledeći:

$$\text{(max)} \qquad d^{\,p+1}(x)=\sum_{k=1}^{p} w_k d_k(x) \quad \text{p.o.} \quad x \in X.$$

Kriterijumska funkcija $d^{p+1}(x)$ ovako definisanog problema naziva se agregatna ili kompozitna funkcija. Ona predstavlja *ukupnu bliskost* između vrednosti kriterijuma za rešenje x i idealne tačke.
Oslanjajući se na rezultate teorije igara, razvijen je sledeći proces rešavanja zadataka višekriterijumskog linearnog programiranja. Počinje se rešavanjem $2p$ jednostavnih problema linearnog programiranja kojima se nalaze maksimalne (f_k^U) i minimalne (f_k^L) vrednosti za svaku kriterijumsku funkciju, $f_k(x)$, k=1,...,p, na dopustivom skupu X. Zapamte se marginalna rešenja i izračunaju njihovi stepeni bliskosti po svakom kriterijumu. Jasno je da je stepen bliskosti marginalnog rešenja x^{k*} po kriterijumu k jednak jedinici dok je po drugim kriterijumima, po pravilu manji od jedan. U saradnji sa donosiocem odluke analiziraju se tekuća rešenja i po potrebi generišu nova.

Algoritam interaktivnog kompromisnog programiranja (*IKP*) ima sledeće korake:

1. ***Inicijalizacija***. Odrediti:

a) f_k^U, k=1,...,p, kao marginalna rešenja originalnog zadatka, tj. rešiti p jednokriterijumskih zadataka:

$$\text{(max)} \qquad f_k \qquad \text{p.o. } x \in X.$$

Rešenja su x^{kU} i f_k^U, tj. vektor $(f_1^U,...,f_p^U)$ je idealna tačka.

b) f_k^L, k=1,..., p, kao rešenja p jednokriterijumskih zadataka:
(min) f_k p.o. $x \in X$. Rešenja su x^{kL} i f_k^L, a vektor $(f_1^L,..., f_p^L)$ se naziva *anti-idealna tačka*.

2. Usvojiti rešenja $x^{\,jU}$ kao početna rešenja x^j, j=1,...,p, i odrediti stepene bliskosti po formuli

$$d_k(x^j)=\frac{f_k(x^j)-f_k^{\,L}}{f_k^{\,U}-f_k^{\,L}}, \quad k=1,..., p.$$

Rezultati se mogu urediti u tabelu oblika:

f	x				f^{U}
	x^1	x^2	...	x^p	
f_1	$d_1^{\,1}$	$d_1^{\,2}$	...	$d_1^{\,p}$	$f_1^{\,U}$
f_2.	$d_2^{\,1}$	$d_2^{\,2}$	...	$d_2^{\,p}$	$f_2^{\,U}$
...	...	...	...	...	...
f_p	$d_p^{\,1}$	$d_p^{\,2}$	...	$d_p^{\,p}$	$f_p^{\,U}$

gde su $d_k^j = d_k(x^j)$ stepeni bliskosti rešenja x^j maksimalno mogućoj vrednosti f_k^U kriterijuma k, k=1,...p, j=1,...,p.

3. Rešiti sledeći zadatak *LP* kojim se nalaze optimalne težine stepena bliskosti:

$$(\max)\ d^{p+1} \quad \text{p.o.} \quad \sum_{k=1}^{p} w_k d_k^{\,j} \geq d^{\,p+1},\ j=1,...,p.$$

   Rešenje ovog zadatka su težinski koeficijenti w_k dodeljeni stepenima bliskosti kojima bi odgovarala maksimalna vrednost kompozitne funkcije, odnosno ukupnog stepena bliskosti izračunatog kao otežani zbir pojedinih stepena bliskosti. Treba primetiti da su promenljive koje se određuju u ovom zadatku w_k, k=1,..., p, i da one ne zavise od x niti su eksplicitno sadržane u k kriterijumskoj funkciji.

4. Formirati novu kompozitnu funkciju koristeći optimalne težinske koeficijente dobijene na koraku 3 algoritma, a zatim rešiti sledeći zadatak linearnog programiranja da bi se dobilo novo kompromisno rešenje koje maksimizira kompozitnu funkciju:

$$(\max) \quad d^{\,p+1}(x) = \sum_{k=1}^{p} w_k d_k(x) \ \text{ p.o. } \ x \in X.$$

   Rešenje zadatka je $x^{\,p+1}$.

5. Odrediti stepene bliskosti d_k^{p+1}, k=1,..., p, idealnoj tački koje odgovaraju rešenju x^{p+1}. Dodati ovu kolonu tabeli koja je pripremljena u koraku 2 i dobiti sledeću tabelu.

f_1	x					f^{U}
	x^1	x^2	...	x^p	x^{p+1}	
f_1	$d_1^{\,1}$	$d_1^{\,2}$	...	$d_1^{\,p}$	$d_1^{\,p+1}$	$f_1^{\,U}$
f_2.	$d_2^{\,1}$	$d_2^{\,2}$	...	$d_2^{\,p}$	$d_2^{\,p+1}$	$f_2^{\,U}$
...	...	...	...	...	...	...
f_p	$d_p^{\,1}$	$d_p^{\,2}$	...	$d_p^{\,p}$	$d_p^{\,p+1}$	$f_p^{\,U}$

6. Zatražiti od donosioca odluke da kaže da li postoji rešenje koje je za njega bolje u odnosu na sva druga rešenja u tabeli. Ako postoji, to je prihvatljivo rešenje i algoritam se završava. U suprotnom donosilac odluke treba da kaže koje je rešenje iz tabele za njega najmanje prihvatljivo. Zatim, to rešenje zameniti novim nađenim u koraku 5 i vratiti se na korak 3.

Sledi mogući način implementacije metode interaktivnog kompromisnog programiranja.

Ulazne veličine:
lisfun - lista funkcija
lisogr - lista ograničenja
Lokalne veličine:
fu - maksimalna rešenja pod ograničenjima
fl - minimalna rešenja pod ograničenjima
fures - vrednosti u kojima funkcija dostiže maksimum
w1 - lista očekivanih vrednosti

```
inkomŠlisfun_,lisogr_Ć:=
  ModuleŠšlll=šć,ll=šć,l,fu=šć, fl=šć,fures=šć,d1,d2,d,v=1ć,
   l=LengthŠlisfunĆ;
   ForŠi=1,i<=l,i++,
     AppendToŠfu,
       FirstŠConstrainedMaxŠlisfunŠŠiĆĆ,
          lisogr,VariablesŠlisfunŠŠiĆĆĆĆĆĆ;
     AppendToŠfures,
       LastŠConstrainedMaxŠlisfunŠŠiĆĆ,
         lisogr,VariablesŠlisfunŠŠiĆĆĆĆĆĆ;
     AppendToŠfl,
       FirstŠConstrainedMinŠlisfunŠŠiĆĆ,
          lisogr,VariablesŠlisfunŠŠiĆĆĆĆĆĆ;
   Ć;
   d1=(lisfunŠŠ1ĆĆ-flŠŠ1ĆĆ)/(fuŠŠ1ĆĆ-flŠŠ1ĆĆ);
   d2=(lisfunŠŠ2ĆĆ-flŠŠ2ĆĆ)/(fuŠŠ2ĆĆ-flŠŠ2ĆĆ);
   WhileŠv!=0,
     ClearŠd,q,w1,w2,x1,x2,sĆ;ll=šć;
     q=ConstrainedMaxŠd,
       šw1(d1/.furesŠŠ2ĆĆ)+w2(d2/.furesŠŠ2ĆĆ)>=d,
        w1(d1/.furesŠŠ1ĆĆ)+w2(d2/.furesŠŠ1ĆĆ)>=d,
        w1+w2>=1,w1+w2<=1,w1>=0,w2>=0ć,šw1,w2,dćĆ;
     d=d1(w1/.qŠŠ2ĆĆ)+d2(w2/.qŠŠ2ĆĆ);
     s=ConstrainedMaxŠd,lisogr,VariablesŠdĆĆ;
     AppendToŠfures,sŠŠ2ĆĆĆ;   AppendToŠlll,d1Ć;
     AppendToŠlll,d2Ć;   AppendToŠlll,dĆ;
```

```
    ForŠi=1,i<=l+1,i++,
      AppendToŠll,šlllŠŠ1ĆĆ/.furesŠŠiĆĆ,
            lllŠŠ2ĆĆ/.furesŠŠiĆĆćĆĆ;
      PrintŠ"Ponuđena resenja su"Ć;    PrintŠllĆ;
      PrintŠ"Ako resenja ne odgovaraju unesi
      vrednost najlosijeg, u protivnom unesi 0"Ć;
      v=InputŠĆ;    IfŠv!=0,furesŠŠvĆĆ=furesŠŠ3ĆĆĆ;
      fures=DeleteŠfures,3Ć;      lll=šć;
    Ć;
    PrintŠ"Dobijena resenja su"Ć;    PrintŠfuresĆ;
    ReturnŠllĆ
  Ć
Ć
```

inkomŠš3x1-x2,-5x1+2x2ć,š2x1+3x2<=21,-2x1+x2<=3,2x1+x2<=11,4x1+x2<=20,x1>=0,x2>=0ćĆĆ

Ponuđena rešenja su
šš1,0ć,š0,1ć,š1/12, 59/62ćć
Ako rešenja ne odgovaraju unesi vrednost najlošijeg, u protivnom unesi nulu.
Ponuđena rešenja su
šš1,0ć,š1/12, 59/62ć,š7/18, 20/31ćć
Ako rešenja ne odgovaraju unesi vrednost najlošijeg, u protivnom unesi nulu.
Dobijena rešenja su
šš1,0ć,š1/12,59/62ć,š7/18,20/31ćć

LITERATURA

[1] M. Vujošević, M. Stanojević, N. Mladenović, *Metode optimizacije,* Društvo Operacionih Istraživača, Beograd 1996.

[2] T.H. Corman, C.E. Leiserson and R.L. Rivest, *Introduction to Algorithms,* MIT Press, Cambridge, Massachusetts, London, England, 1990.

7. AUTOMATSKO SVOĐENJE JEDNAČINA U PAKETU MATHEMATICA

7.1. BULOVA ALGEBRA - DEFINICIJA I NOTACIJA

Prvu striktno aksiomatsku definiciju *Boole*-ove algebre postavio je E.V. Huntington 1904. godine. *Huntington*-ova aksiomatizacija, koja je najbliža originalnom *Boole*-ovom sistemu i koja se često uzima kao definicija *Boole*-ove algebre, data je sledećom definicijom:

Boole-ova algebra je uređena šestorka $(B, \oplus, \otimes, \varnothing, 0{,}1)$, gde je B skup, $\oplus$ binarna operacija (obično se naziva *unija* ili *disjunkcija*) u skupu B, $\otimes$ je takođe binarna operacija u B (obično se naziva *presek* ili *konjunkcija*), $\varnothing$ je unarna operacija skupa B (*komplement* ili *negacija*) a 0 i 1 su elementi skupa B tako da su zadovoljeni sledeći uslovi:

(i) Operacije $\oplus$ i $\otimes$ su asocijativne:
$a \oplus (b \oplus c)=(a \oplus b) \oplus c$
$a \otimes (b \otimes c)=(a \otimes b) \otimes c,\ \forall a, b, c\in B$

(ii) Operacije $\oplus$ i $\otimes$ su komutativne:
$a \oplus b=b \oplus a$
$a \otimes b=b \otimes a,\ \forall a, b\in B$

(iii) Operacije $\oplus$ i $\otimes$ su distributivne u odnosu jedna na drugu:
$a \oplus (b \otimes c)=(a \oplus b) \otimes (a \oplus c)$
$a \otimes (b \oplus c)=(a \otimes b) \oplus (a \otimes c),\ \forall a, b, c\in B$

(iv) Za $\forall a\in B$ važi:
$a \oplus 0=a$ i $a \otimes 1=a$

(v) Za $\forall a\in B$, $\exists a'\in B$ tako da važi:
$a \oplus a'=1$ i $a \otimes a'=0$

U kontekstu *Boole*-ove *logike*, na primer, $\oplus$ je jednako $\wedge$, $\otimes$ je $\vee$ i $\varnothing$ je $\rceil$. Osim toga, u *Boole*-ovoj logici, elementi 0 i 1 označavaju *kontradikciju* (tj. $p\vee\rceil p$) i *tautologiju* (tj. $p\wedge\rceil p$), redom. Sa ovako pripremljenim terminima i oznakama sada možemo da govorimo o jednoj od najznačajnijih pretpostavki (koja je i dokazana) u *Boole*-ovoj algebri - *Robbins*-ovoj pretpostavci.

7.2. ROBINSOVA PRETPOSTAVKA

Skoro 30 godina posle svoje prvobitne aksiomatizacije *Boole*-ove algebre, *Huntington* otkriva sledeće tri znatno uprošćene aksiome koje se zasnivaju **samo** na binarnoj operaciji $\oplus$:

$x \oplus y=y \oplus x$	Komutativnost
$(x \oplus y) \oplus z=x \oplus (y \oplus z)$	Asocijativnost
$\bar{x} \oplus y \oplus \bar{x} \oplus y=x$	Huntington-ova jednačina

Ubrzo posle toga, *Herbert Robbins* je pretpostavio da ako bi *Huntington*-ova jednačina bila zamenjena sledećom jednačinom, koja je u svakoj *Boole*-ovoj algebri **ekvivalentna** *Huntingto**n***-ovoj jednačini

$\bar{\bar{x}} \oplus y\ \bar{x} \oplus \bar{y}=x$ Robbins-ova jednačina,

onda bi tako dobijena aksiomatizacija bila takođe karakteristika *Boole*-ove algebre. Ovo je *Robbins*-ova pretpostavka. *Robbins* i *Huntington* nisu mogli da nađu dokaz ili kontraprimer ove pretpostavke.
Algebre koje zadovoljavaju uslove komutativnosti, asocijativnosti i Robinsove jednačine poznate su kao *Robbins*-ove *algebre.* Nije teško pokazati da je svaka *Boole*-ova istovremeno i *Robbins*-ova algebra pa se prirodno nameće pitanje: da li je svaka *Robbins*-ova algebra takođe i *Boole*-ova? Drugim rečima, da li se *Huntington*-ova jednačina može izvesti iz uslova komutativnosti, asocijativnosti i *Robbins*-ove jednačine?

7.3. KOMPJUTERSKI DOKAZ ROBINSOVE PRETPOSTAVKE

Dokaz koji rešava Robbins-ov problem pronašao je Bill McCune, 10. oktobra 1996. godine. Kao pomoć koristio je program *EQP*. Vreme potrebno za takvo mehaničko nalaženje dokaza bilo je oko 8 dana na RS/6000 računaru uz iskorišćenje 30 megabajta memorije. Na nesreću, 16 linija dokaza generisanih *EQP*-om su veoma kompleksne i teško čitljive. Osim toga, *EQP* ne daje nikakve informacije kako je svaka linija dobijena iz prethodnih pa je zbog toga veoma teško ručno proveriti ovako mehanički generisane *EQP* dokaze. U sledećem izlaganju biće objašnjeno kako *MATHEMATICA* 3.0 može biti iskorišćena za proveru i objašnjenje ovakvog dokaza *Robbins*-ove pretpostavke.

7.3.1. Razlika EQP i Mathematica 3.0 notacije

Glavni problem u dokazu generisanim *EQP*-om je što on koristi jednodimenzionalnu notaciju ′š*n*ć(š*x*ć)′ umesto *dvodimenzionalne* Boole-ove notacije $\bar{x}$ za označavanje operatora komplementa (ili negacije). Zbog ovoga je skoro nemoguće pratiti veoma kompleksne izraze koji se pojavljuju u kompjuterskom dokazu Robbins-ove pretpostavke. Kako Mathematica 3.0 dozvoljava korišćenje pomenute Boole-ove dvodimenzionalne notacije, mnogo je lakše uz pomoć nje sagledati sam tok dokaza generisanog *EQP*-om.
Da bi se u Mathematici 3.0 dobili simboli komplementa i disjunkcije, tj. prešlo na dvodimenzionalnu notaciju, potrebno je uraditi sledeće:
1. Komplement
- označiti ćeliju u kojoj prikazujemo ovakvu notaciju;
- iz menija *CellĐDisplay As* izabrati opciju *Standard Form;*
- selektovati izraz čiju negaciju želimo;

- kombinacijom tastera *Ctrl* i 7 ulazimo u mod *Above*;
- kucanjem donje crte (tasteri Shift i -) konačno nadvlačimo željeni izraz
- iz moda Above izlazimo pomoću tastera sa oznakom strelice na desno.

2. Disjunkcija

- kao u prethodnom slučaju, prebacimo ćeliju u *Standard Form*
- pritiskom na taster *ESC* prelazimo u specijalan mod za unošenje simbola
- unosimo niz karaktera "*c, +*" i ponovnim pritiskom na *ESC* dobijamo željeni simbol

Primer razlike u notaciji EQP-a i Mathematice 3.0

$n(n(x \oplus y) \oplus n(y \oplus n(x))) = y$ Š***EQP***Ć

$\overline{\overline{x \oplus y} \oplus \overline{y \oplus \overline{x}}} == y$ Š***Mathematica 3.0***Ć

Listing programa u Mathematici 3.0 kojim se svaka od 16 linija dokaza Robbins-ove pretpostavke proverava i razjašnjava, nalazi se u dodatku. Na kraju, dolazimo do konačnog cilja ovog izlaganja: *Napisati funkciju u Mathematici* 3.0 *koja za proizvoljan izraz sastavljen od kombinacije bilo kojeg izraza iz dokaza Robbins-ove pretpostavke kao i disjunkcije i negacije, vraća uprošćen (sveden) izraz na osnovu pravila navedenih u dokazu i pri tom formira listu korišćenih zamena pomoću kojih se došlo do tako uprošćenog izraza.*

Jedno od mogućih rešenja dato je u nastavku teksta.

7.3.2. Dodatak

Attributes[CirclePlus] = {Orderless, Flat};

j[1] = $\overline{\overline{x\oplus y}\oplus\overline{y\oplus\overline{x}}}$**;**

j[2] = $\overline{y\oplus\overline{y\oplus\overline{x}\oplus\overline{x\oplus y}}}$**;**

j[3] = $\overline{y\oplus\overline{x\oplus y\oplus\overline{y\oplus\overline{x}}}}$**;**

j[4] = $\overline{\overline{y\oplus\overline{x}}\oplus\overline{x\oplus y\oplus y\oplus\overline{y\oplus\overline{x}}}}$**;**

j[5] = $\overline{\overline{y\oplus z}\oplus\overline{z\oplus\overline{y\oplus\overline{x}}\oplus\overline{x\oplus y\oplus y\oplus\overline{y\oplus\overline{x}}}}}$**;**

j[6] = $\overline{y\oplus\overline{x\oplus y\oplus y\oplus\overline{y\oplus\overline{x}}\oplus\overline{y\oplus\overline{x}}}}$**;**

j[7] = $\overline{\overline{y\oplus\overline{x}}\oplus\overline{x\oplus y\oplus y\oplus y\oplus\overline{y\oplus\overline{x}}\oplus\overline{y\oplus\overline{x}}}}$**;**

j[8] = $\overline{z\oplus\overline{z\oplus\overline{y\oplus z}\oplus\overline{y\oplus\overline{x}}\oplus\overline{x\oplus y\oplus y\oplus\overline{y\oplus\overline{x}}}}}$**;**

j[9] = $\overline{\overline{u\oplus\overline{y\oplus z}}\oplus\overline{u\oplus z\oplus\overline{z\oplus\overline{y\oplus z}\oplus\overline{y\oplus\overline{x}}\oplus\overline{x\oplus y\oplus y\oplus\overline{y\oplus\overline{x}}}}}}$**;**

j[10] = $\overline{x\oplus\overline{x\oplus x\oplus x\oplus\overline{x\oplus\overline{x}}}}$**;**

j[11] = $\overline{\overline{y\oplus\overline{x\oplus\overline{x}}}\oplus\overline{x\oplus y\oplus\overline{x\oplus x\oplus x\oplus\overline{x\oplus\overline{x}}}}}$**;**

j[12] = $\overline{\overline{x\oplus\overline{x}}\oplus\overline{x\oplus x\oplus x\oplus x\oplus\overline{x\oplus\overline{x}}}}$**;**

j[13] = $\overline{x\oplus\overline{x\oplus x\oplus x\oplus x\oplus\overline{x\oplus\overline{x}}\oplus\overline{x\oplus\overline{x}}}}$**;**

j[14] = $\overline{\overline{y\oplus\overline{x\oplus\overline{x}}}\oplus\overline{x\oplus y\oplus\overline{x\oplus x\oplus x\oplus x\oplus\overline{x\oplus\overline{x}}\oplus\overline{x\oplus\overline{x}}}}}$**;**

j[15] = $\overline{x\oplus\overline{x\oplus\overline{x\oplus x\oplus x\oplus\overline{x\oplus\overline{x}}}\oplus\overline{x\oplus x\oplus x\oplus x\oplus\overline{x\oplus\overline{x}}\oplus\overline{x\oplus\overline{x}}}}}$**;**

j[16] = $\overline{x\oplus x\oplus x\oplus x\oplus\overline{x\oplus\overline{x}}\oplus\overline{x\oplus\overline{x}}}$**;**

```
uprosti[izr_] := (     sl = True;
                       zavisnost = {};
                       p := izr;
                       While[sl, For[sl = False; k = 1, k < 17, k++,
                             If[MemberQ[p, j[k], Infinity] ∨ p == j[k],
                                AppendTo[zavisnost, zav[[k]]];
                                sl = True;
                                Print[p];
                                Print["smena: ", smene[[k]]];
                                p = p /. smene[[k]]]
                       ]   ];
                       zavisnost = Union[Flatten[zavisnost]];
                       If[Last[zavisnost] == Null, zavisnost = Rest[RotateRight[zavisnost]], zavisnost];
                       Print["Koriscene su jednacine ", zavisnost];
                       p              );
```

Primenom funkcije *uprosti* se određuje lista jednačina koje su korišćene u transformaciji izraza.

Primer. U primeru je opisana primena funkcije *uprosti* Š*izr_*Ć:

uprosti Š $\overline{\overline{(j[12]\oplus j[7])}\oplus j[6]}$ Ć

Korišćene su jednačine š***1,4,6,10***ć

y

uprosti Š $\overline{x\oplus j[15]}$ Ć

Korišćene su jednačine š***1,2,4,7,9,11,14,15***ć

$\overline{x\oplus\overline{x}}$

uprosti Š*j*Š***13***Ć ⊕ *j*Š***16***ĆĆ

Korišćene su jednačine š***1,4,12,14,15***ć

$\overline{x\oplus\overline{x}}\oplus\overline{x\oplus x\oplus x\oplus\overline{\overline{x\oplus\overline{x}}}}$

uprosti Š $\overline{j[8]\oplus j[9]}\oplus j[2]$ Ć

Korišćene su jednačine š***1,5,8***ć

$\overline{u\oplus\overline{\overline{y\oplus z}}}\oplus\overline{x\oplus y}$

LITERATURA

[1] Mathematica in Education and Research, **l.7** No.1, 1998.

8. LOKACIJSKI PROBLEMI

8.1. UVOD

Lokacijski problemi čine posebnu klasu zadataka optimizacije. U opštem slučaju, zadatak lokacijskog problema je određivanje položaja (lokacije) nekih novih objekata u postojećem prostoru u kome se već nalaze drugi relevantni objekti. Novi objekti su obično neka vrsta centara koji pružaju usluge, i nazivaju se *snabdevači*. Postojeći objekti su korisnici usluga ili klijenti, i mi ćemo ih zvati *korisnici*.

Moguće su različite klasifikacije lokacijskih problema. kao kriterijumi klasifikacije obično se koriste sledeći:

a) broj novih objekata koje treba razmestiti: jedan ili više;

b) karakter objekata: da li je željen ili neželjen;

c) mogući položaj objekta: da li postoji predodređeni diskretni skup potencijalnih lokacija ili se objekat može postaviti u bilo koju tačku datog kontinualnog skupa;

d) karakter prostora u koji se locira objekat: da li je u pitanju ravan, mreža ili nešto treće;

e) metrika koja se koristi za računanje rastojanja između dve tačke u posmatranom prostoru.

Kako na rešenje lokacijskog problema značajno utiče način na koji se računa rastojanje između dva tačke u posmatranom prostoru, najpre ćemo izložiti pristupe tom zadatku.

8.1.1. Metrika

Metrika je način na koji se određuje rastojanje između dva elementa nekog skupa. U lokacijskim problemima je potrebno odrediti rastojanje između dve tačke u posmatranom prostoru na osnovu poznavanja njihovih koordinata. U tu svrhu se koriste tzv. l_p metrike gfe je p realan broj takav da je $1<p<\infty$. Ove metrike se u opštem slučaju defihišu nad tačkama prostora R^n (tj. skupova svih uređenih n-torki realnih brojeva).

Neka su u prostoru R^n zadate dve tačke $A = (x_1^A, \dots ,x_n^A)$ i $B = (x_1^B, \dots ,x_n^B)$.

Rastojanje između tačaka A i B u l_p metrici za $1<p<\infty$ se računa po obrazcu:

$$d_{lp}(A,B)=\sqrt[p]{\sum_{j=1}^{n}\left|x_j^A - x_j^B\right|^p} ,$$

dok je za $p=\infty$:

$$d_{l\infty}=\lim_{p\to\infty} d_{šlpć}(A,B).$$

Teorijski ima beskonačno mnogo metrika tipa l_p . Najznačajnije su i najviše se koriste u praksi sledeće:

- l_1 (pravougaona) metrika: $d_{l1}(A,B)=\sum_{j=1}^{n}\left|x_j{}^A - x_j^B\right|$.

praŠa_,b_Ć:=AbsŠaŠŠ1ĆĆ-bŠŠ1ĆĆĆ+AbsŠaŠŠ2ĆĆ-bŠŠ2ĆĆĆ;

- l_2 (Euklidova) metrika:

$$d_{l2}(A,B)=\sqrt{\sum_{j=1}^{n}(x_j{}^A - x_j{}^B)^2}$$

euŠa_,b_Ć:=SqrtŠ(aŠŠ1ĆĆ-bŠŠ1ĆĆ)Č2+(aŠŠ2ĆĆ-bŠŠ2ĆĆ)Č2Ć;

- l_∞ (Čebiševljeva) metrika:

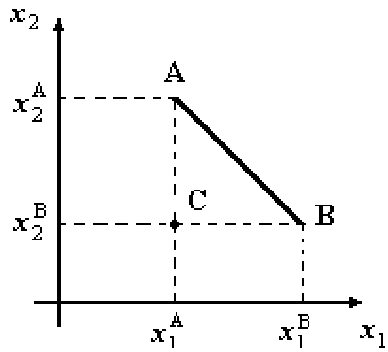
$$d_{1\infty}(A,B)=max_{1<j<n}\, đx_j^A\text{-}x_j^B đ$$

cebŠa_,b_Ć:=MaxŠAbsŠaŠŠ1ĆĆ-bŠŠ1ĆĆĆ,AbsŠaŠŠ2ĆĆ-bŠŠ2ĆĆĆĆ;

Tumačenje ovih metrika i razlike između njih daćemo na primeru prostora R^2 sa slike 1. U l_1 metrici, rastojanje između tacaka A i B jednako je zbiru dužina kateta AC i BC, tj.

$$d_{l1}(A,B)=x_1{}^B\text{-}x_1{}^A+x_2{}^A\text{-}x_2{}^B.$$

Ovo rastojanje se naziva i Menhetn rastojanje jer podseća na rastojanje u ovoj njujorškojv cetvrti u kojoj su ulice i avenije pod pravim uglom.



slika 1

U l_2 metrici rastojanje između tačaka A i B je jednako hipotenuzi trougla ABC, tj.

$$d_{l2}(A,B)=\sqrt{(x_1{}^B - x_1^A)^2 + (x_2^A - x_2^B)^2}\ .$$

Ovo rastojanje se naziva Euklidsko jer se u klasičnoj Euklidskoj geometriji rastojanje između dve tačke definiše upravo na ovaj način. Na kraju, u l_∞ metrici je rastojanje između tačaka A i B jednako dužoj od kateta AC i BC. U konkretnom slučaju je

$$d_{l\infty}(A,B)=x_1^B-x_1^A$$

U praktičnim primerima, obično se usvaja da za $p \geqslant 7$ važi $d_{lp} \sim d_{l\infty}$.

Za bilo koje dve tačke važi da je $d_{l_1} \geqslant d_{l_2} \geqslant d_{l\infty}$, odnosno u opštem slučaju se može dokazati sledeće tvrđenje: ako $p,q\in R$ i $p,q>1$ tada je $p>q$ **<=>** $\boldsymbol{d_{lp}\leqslant d_{lq}}$. Izbor metrike l_p metrike zavisi od više faktora, od kojih su najznačajniji:

1. Priroda problema: korišćenje određenje metrike daje manje ili više dobru aproksimaciju realnog rastojanja u zavisnosti od prirode prostora u kome se tačke nalaze. Na primer, ako je moguče kretati se pravolinijski između dve tačke, tačno rastojanje između njih se dobija Euklidovom metrikom, ali u gradovima u kojima su ulice pod pravim uglom, dužina puta najbolje će se aproksimirati pravougaonom metrikom.
2. Računska složenost: ako je za neki model bitno da se lakše reši primenom određene metrike, a preciznost dobijenog rezultata nije od presudnog značaja (tj. ako se određena odstupanja mogu tolerisati), tada će ta biti upotrebljenja metrika kojom se dobija rešenje na najjednostavniji način.

Pri rešavanju lokacijskih problema važno je naglasiti koja metrika se koristi, pogotovu kada se one razlikuju od Euklidove. U daljem tekstu isključivo će se razmatrati problemi lokacije u ravni, tj. u prostoru R^2.

8.2. DISKRETNI LOKACIJSKI PROBLEMI

Kod diskretnih lokacijskih problema su tačke na koje se novi objekat može postaviti elementi nekog konačnog skupa. Neka je dato m tačaka $A_1, \dots ,A_m$ u ravni. U ovim tačkama se nalaze postojeći objekti (korisnici) i r potencijalnih lokacija $B_1,\dots,B_r$ na koje je moguće postaviti novi željeni objekat (snabdevači). Suma otežanih (težinskih) rastojanja od potencijalne lokacije snabdevača B_k do korisnika je data izrazom

$$W_k=\sum_{i=1}^{m} w_i d(A_i, B_k)$$

U kome su:

w_i - težinski koeficijent i-te zadate tačke (npr. broj stanara neke zgrade, važnost tačke itd.);

$d(A_i,B_k)$ - rastojanje između i-te zadate tačke i k-te potencijalne lokacije u odgovarajućoj metrici.
Rešenje problema se sastoji u tome da se odredi ona lokacija B_k^* za koju je suma otežanih rastojanja minimalna, tj.

$$W_{k*} = \min{}_{1<k<r}\, \check{s} W_k \,\acute{c}.$$

Postavljeni zadatak se rešava tako što se za svaku potencijalnu lokaciju $B_k \in$ š$B_1,...,B_r$ ć izračunava suma otežanih rastojanja, a za rešenje se bira ona tačka za koju je ova suma najmanja. Za merenje rastojanja se može usvojiti bilo koja metrika, a najčešće se koristi Euklidova.

Primer 8.1. Date su koordinate pet postojećih objekata: $A(1,2)$, $B(2,5)$, $C(3,4)$, $D(6,0)$ i $E(5,5)$. Moguće su tri lokacije za novi objekat: $N_1(2,3)$, $N_2(3,2)$ i $N_3(6,3)$. Težinski koeficijenti su: $w_A=w_D=2$, $w_B=w_C=1$, a $w_E=3$. Koristeći Euklidovu metriku odrediti koordinate novog objekta.

```
primer81Šlp_,lm_,lt_Ć:=
  ModuleŠšd,i,ind=1,ras,ras2ć,
    d=LengthŠlmĆ;  ras=eusumarastojanjaŠlmŠŠ1ĆĆ,lp,ltĆ;
    ForŠi=2,i<=d,
        i++, ras2=eusumarastojanjaŠlmŠŠiĆĆ,lp,ltĆ;
        IfŠras>ras2, ind=i; ras=ras2 Ć;
    Ć;
    ReturnŠlmŠŠindĆĆĆ;
  Ć;
```

Primer 8.2. Rešiti predhodni zadatak koristeći Čebiševljevu metriku.

```
primer82Šlp_,lm_,lt_Ć:=
  ModuleŠšd,i,ind=1,ras,ras2ć,
       d=LengthŠlmĆ;
       ras=cebsumarastojanjaŠlmŠŠ1ĆĆ,lp,ltĆ;
       ForŠi=2,i<=d,i++,
         ras2=cebsumarastojanjaŠlmŠŠiĆĆ,lp,ltĆ;
         IfŠras>ras2,ind=i; ras=ras2 Ć;
         Ć;
       ReturnŠlmŠŠindĆĆĆ;
       Ć;
```

primer82Ššš1,2ć,š2,5ć,š3,4ć,š6,0ć,š5,5ćć,šš2,3ć,š3,2ć,š6,3ćć, š2,1,1,2,3ćĆ

š2,3ć

Primer 8.3. Trgovinsko preduzeće ima 5 prodavnica nameštaja u jednom regionu. Imajući u vidu troškove dopremanja robe, rukovodstvo je odlučilo da sagradi objekat koji će biti skladište robe za ove prodavnice. Koordinate ovih lokacija su: $S_1(2,1)$, $S_2(6,5)$ i $S_3(4,7)$. Koordinate prodavnica su: $P_1(1,6)$, $P_2(2,7)$, $P_3(1,0)$, $P_4(6,7)$ i $P_5(6,1)$, a na osnovu njihovog poslovanja dodeljeni su im težinski koeficijenti: 2, 3, 3, 2 i 5 respektivno. Potrebno je odrediti lokaciju skladišta tako da suma otežanih rastojanja između njega i svih prodavnica bude minimalna. Za merenje rastojanja između objekata koristiti l_1 metriku.

```
primer83Šlp_,lm_,lt_Ć:=
 ModuleŠšd,i,ind=1,ras,ras2ć,
  d=LengthŠlmĆ;
  ras=prasumarastojanjaŠlmŠŠ1ĆĆ,lp,ltĆ;
  ForŠi=2,i<=d,i++,
    ras2=prasumarastojanjaŠlmŠŠiĆĆ,lp,ltĆ;
    IfŠras>ras2,ind=i; ras=ras2Ć;
  Ć;
  ReturnŠlmŠŠindĆĆĆ;
 Ć;
```

8.3. KONTINUALNI LOKACIJSKI PROBLEMI

8.3.1. Veberov problem

Neka je dato m tačaka u ravni $A_1,\ldots,A_m$, gde je $A_i=(a_1^{\,i},a_2^{\,i})$, i=1,...,m. Potrebno je naći tačku $X=(x_1,\ x_2\ )$ za koju je suma otežanih rastojanja do datih tačaka minimalna, tj. treba rešiti sledeći problem bezuslovne optimizacije:

$$\text{(min)} \qquad f(X)=\sum_{i=1}^{m} w_i d_i(X)\,,$$

gde su:

w_i - težinski koeficijenti tačke A;

$d_i(X)=d(A_i,X)$ - rastojanje tačke A_i od lokacije X.

Ovako postavljen problem se naziva problem tipa *minisum* ili minisum problem.

Najčešći je slučaj da se usvoji Euklidova metrika. Tada matematički model ima sledeći oblik:

$$\text{(min)} \qquad f(X)=\sum_{i=1}^{m} w_i \sqrt{\left(x_1-a_1^i\right)^2+\left(x_2-a_2^i\right)^2}\ .$$

Ovaj zadatak bezuslovne nelinearne optimizacije u opštem slučaju nije moguće rešiti analitički. *Vajsfeldov algoritam* je iterativni numerički metod za rešavanje Veberovog problema za euklidsku metriku, i njime se dobija približno rešenje problema.

8.3.2. Vajsfeldov algoritam za rešavanje Veberovog problema

Dato je m tačaka $A_i=(a_1^i,a_2^i)$, njihove težine w_i, $i=1,...,m$ i koeficijent kriterijuma zuastavljanja numeričkog postupka $\varepsilon>0$; potrebno je da se odredi optimalna lokacija (određena koordinatama) nove tačke, koristeći kao kriterijum sumu otežanih rastojanja (*minisum* problem). S obzirom na to da ovaj algoritam jako sporo konvergira kada se optimalno rešenje poklapa sa jednom od zadatih tačaka, najpre se proverava da li je neka od postojećih tačaka optimalna lokacija za novi objekat. Tek ako se utvrdi da nije, prelazi se na iterativni deo algoritma.

Algoritam je određen sledećim koracima.

1) izračunati međusobna rastojanja između svih m tačaka;

$$(\forall\ r,l \in \text{š}1,...,m\text{ć})\ \ d(A_r,A_l)=\sqrt{\left(a_1^r-a_1^l\right)^2+\left(a_2^r-a_2^l\right)^2}\ .$$

2) Proveriti da li za neku tačku $r\in\text{š}1,...,m\text{ć}$ važi:

$$c_r=\sqrt{\left(\sum_{i=1,i\neq r}^{m}\frac{w_i\left(a_1^r-a_1^i\right)}{d\left(A_r,A_i\right)}\right)^2+\left(\sum_{i=1,i\neq r}^{m}\frac{w_i\left(a_2^r-a_2^i\right)}{d\left(A_r,A_i\right)}\right)^2}<w_r$$

Ako je ovaj uslov ispunjen za neko r, tada prekinuti algoritam. Rešenje se nalazi u tački A_r. U suprotnom, prelazi se na sledeći korak.

3) Stavimo da je $k=0$ i odredimo početno rešenje $X^0=(x_1^0,x_2^0)$ po formuli:

$$x_j^0=\frac{\sum_{i=1}^{m}w_i a_j^i}{\sum_{i=1}^{m}w_i},\quad j=1,2.$$

4) Izračunati rastojanja između $X^k=(x_1^k,x_2^k)$ i zadatih tačaka:

$$(\forall i\in\text{š}1,...,m\text{ć})\ \ d(X^k,A_i)=\sqrt{\left(x_1^k-a_1^i\right)^2+\left(x_2^k-a_2^i\right)^2}\ .$$

5) Računamo po iterativnoj formuli:

$$x_j^{k+1}=\frac{\sum_{i=1}^{m}\frac{w_i a_j^i}{d\left(X^k,A_i\right)}}{\sum_{i=1}^{m}\frac{w_i}{d\left(X^k,A_i\right)}},\quad j=1,2.$$

6) Ako je $\forall\ j\in\text{š}1,2\text{ć}$, $\ \text{đ}x_j^{k+1}-x_j^k\text{đ}<\varepsilon$ => *KRAJ*.

Algoritam se prekida, X^{k+1} se usvaja kao "dovoljno dobro" rešenje. U suprotnom, uzeti $k := k+1$ i ići na korak 4).

```
euveberŠlp_,lt_,e_Ć:=
 ModuleŠšd,i,r,xn,xn1,p1xn,p1xn1,p2xn1ć,
  d=LengthŠlpĆ;
  ForŠi=1,i<=d,i++,
    r=zksumaŠlp,lt,i,dĆ;
    IfŠr<=ltŠŠiĆĆ, ReturnŠlpŠŠiĆĆĆ Ć;
  Ć;
  p1xn=SumŠltŠŠiĆĆFirstŠlpŠŠiĆĆĆ,ši,dćĆ/SumŠltŠŠiĆĆ,ši,dćĆ;
  p2xn=SumŠltŠŠiĆĆLastŠlpŠŠiĆĆĆ,ši,dćĆ/SumŠltŠŠiĆĆ,ši,dćĆ;
  xn=šp1xn,p2xnć;
  WhileŠTrue,
   p1xn1=SumŠltŠŠiĆĆFirstŠlpŠŠiĆĆĆ/euŠxn,lpŠŠiĆĆĆ,ši,dćĆ
         /SumŠltŠŠiĆĆ/euŠxn,lpŠŠiĆĆĆ,ši,dćĆ;
   p2xn1=SumŠltŠŠiĆĆLastŠlpŠŠiĆĆĆ/euŠxn,lpŠŠiĆĆĆ,ši,dćĆ
         /SumŠltŠŠiĆĆ/euŠxn,lpŠŠiĆĆĆ,ši,dćĆ;
   xn1=šNŠp1xn1Ć,NŠp2xn1Ćć;
   IfŠAbsŠxnŠŠ1ĆĆ-xn1ŠŠ1ĆĆĆ<e and AbsŠxnŠŠ2ĆĆ-xn1ŠŠ2ĆĆĆ<e,
        ReturnŠxn1Ć,  xn:=xn1Ć;
 Ć Ć;
```

Primer 8.5. U jednoj opštini se nalaze četri naseljena mesta. Potrebno je sagraditi osnovnu školu u koju bi išla deca iz sva četri naselja. Da bi se odredila lokacija nove škole, u obzir se uzimaju položaji tih naselja i broj stanovnika. Lokacije mesta (koordinate zadate u kilometrima) i broj stanovnika (u hiljadama) je dat u sledećoj tabeli:

Mesta	M1	M2	M3	M4
X	3	8	10	12
Y	7	1	5	1
Broj stanovnika	80	30	25	40

Potrebno je odrediti mesto gradnje nove škole, tako da ukupan put svih đaka od kuće do škole bude minimalan. Smatra se da je put od naselja do škole pravolinijski i da je rezultat dovoljno dobar ako je razlika obe koordinate između dve uzastopne iteracije manja od 20 metara.

Rešavanje Veberovog problema sa pravougaonom metrikom.

Potrebno je rešiti sledeći problem:

$$
\begin{aligned}
(\min)\quad f(x) &= \sum_{i=1}^{m} w_i d_i(x) = \sum_{i=1}^{m} w_i \left(\left| x_1 - a_1^i \right| + \left| x_2 - a_2^i \right| \right) \\
&= \sum_{i=1}^{m} w_i \left| x_1 - a_1^i \right| + w_i \left| x_2 - a_2^i \right| = f_1(x_1) + f_2(x_2).
\end{aligned}
$$

Kako je minimum zbira u ovom slučaju jednak zbiru minimuma, rešenje polaznog modela se može dobiti rešavanjem sledeća dva zadatka:

$$(\min) f_1(x_1) = \sum_{i=1}^{m} w_i \left| x_1 - a_1^i \right| , \qquad (\min) f_2(x_2) = \sum_{i=1}^{m} w_i \left| x_2 - a_2^i \right| .$$

Na taj način rešavanja originalnog problema sa dve promenljive svedeno je na rešavanje dva nezavisna, ali po strukturi indentična zadatka sa po jednom promenljivom. Znači, prvo se rešava problem za jednu koordinatu: (min) $f_1(x_1)$, odakle se dobija x_1^*, a zatim za drugu: (min) $f_2(x_2)$, odakle se dobija x_2^*. Tačka $X^*=(x_1^*, x_2^*)$ predstavlja rešenje polaznog Veberovog problema.
Neka su, kao i u predhodnom slučaju, zadate tačke $A_i=(a_1^i,a_2^i)$ i težinski koeficijenti tih tačaka w_i, $i=1,...,m$.

Algoritam za određivanje j-te koordinate sastoji se u sledećem:

1) Sortirati koordinate tačaka A_i, $i=1,...,m$ u neopadajući niz. Nadalje ćemo smatrati da indeks i raste po sortiranom redosledu, tj. $a_j^1<a_j^2<...<a_j^m$. Moguća su ovde dva slučaja:
2) Ako za neko $k\in$š$1,..., m$ć važi $\sum_{i=1}^{k-1} w_i < \frac{1}{2}\sum_{i=1}^{m} w_i < \sum_{i=1}^{k} w_i$, pri čemu je za $k=1$ leva strana nejednakosti jednaka je nuli, tada je tražena jednačina koordinata x-$j^*=a_j^k$.
3) Ako je za neko $k\in$š$1,...,m$ć važi $\frac{1}{2}\sum_{i=1}^{m} w_i < \sum_{i=1}^{k} w_i$, tada je rešenje višestruko, tj. tražena koordinata x_j^* može da ima bilo koju vrednost iz intervala Ša_j^k, a_j^{k+1}Ć.

Primenjujući ovaj algoritam za $j=1$, a zatim za $j=2$ dobijaju se obe koordinate tražene tačke X^*.

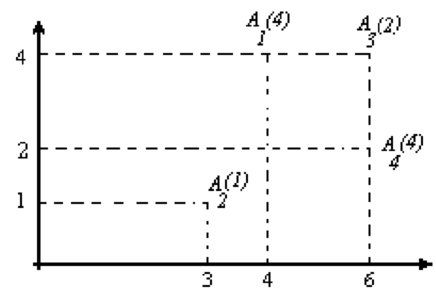
Sledi implementacija algoritma.

```
praveberŠlp_,lt_Ć:=
 ModuleŠši,j,pt,pp,d,pr=šć,dr=šć,lt1,lt2,s1,s2ć,
  d=LengthŠlpĆ;lt1=lt;lt2=lt;
  ForŠi=1,i<=d,i++,
    pr=AppendŠpr,FirstŠlpŠŠiĆĆĆĆ;
    r=AppendŠdr,LastŠlpŠŠiĆĆĆĆ;
  Ć;
  ForŠi=1,i<=d-1,i++,
    ForŠj=i+1,j<=d,j++,
      IfŠprŠŠiĆĆ>prŠŠjĆĆ,pp=prŠŠjĆĆ;
        prŠŠjĆĆ=prŠŠiĆĆ;   prŠŠiĆĆ=pp;
        pt=lt1ŠŠjĆĆ;lt1ŠŠjĆĆ=lt1ŠŠiĆĆ;  lt1ŠŠiĆĆ=pt;
      Ć;
```

```
      IfŠdrŠŠiĆĆ>drŠŠjĆĆ,
         pp=drŠŠjĆĆ;drŠŠjĆĆ=drŠŠiĆĆ;
         drŠŠiĆĆ=pp;  pt=lt2ŠŠjĆĆ;
         lt2ŠŠjĆĆ=lt2ŠŠiĆĆ;   lt2ŠŠiĆĆ=pt;
        Ć;
     Ć;
  Ć;
  m=SumŠltŠŠiĆĆ,ši,dćĆ;    s1=šlt1ŠŠ1ĆĆć;s2=šlt2ŠŠ1ĆĆć;
  ForŠi=2,i<=d,i++,
    s1=AppendŠs1,s1ŠŠi-1ĆĆ+lt1ŠŠiĆĆĆ;
    s2=AppendŠs2,s2ŠŠi-1ĆĆ+lt2ŠŠiĆĆĆ;
  Ć;
  ForŠi=1,i<=d,i++,
   IfŠs1ŠŠiĆĆ==m/2,
      PrintŠ"x je između",prŠŠiĆĆ,"i",prŠŠi+1ĆĆĆ   Ć;
      IfŠs2ŠŠiĆĆ==m/2,
      PrintŠ"y je između",drŠŠiĆĆ,"i",drŠŠi+1ĆĆĆ;
    Ć;
    IfŠi=!=1,IfŠs1ŠŠi-1ĆĆ<m/2 and m/2<s1ŠŠiĆĆ,
      PrintŠ"x= ",prŠŠiĆĆĆĆ;
    IfŠs2ŠŠi-1ĆĆ<m/2 and m/2<s2ŠŠiĆĆ,
           PrintŠ"y= ",drŠŠiĆĆĆĆ;,
    IfŠm/2<s1ŠŠ1ĆĆ,PrintŠ"x= ",prŠŠ1ĆĆĆĆ;
    IfŠm/2<s2ŠŠ1ĆĆ,PrintŠ"y= ",drŠŠ1ĆĆĆĆ;
   Ć  Ć
Ć;
```

Primer 8.6. Rešiti zadatak iz predhodnog primera koristeći pravougaonu metriku uz izmene da je težina tačke A_4 jednaka 3. U ovom slučaju je:

$$A_1(4,4),\ w_1=4;\quad A_2(3,1),\ w_2=1;\ A_3(6,4),\ w_3=2;\quad A_4(6,2),\ w_4=3.$$



slika 2.

Problem se rešava pozivom funkcije

praveberŠšš4,4ć,š3,1ć,š6,4ć,š6,2ćć,š4,1,2,3ćĆ

x je između 4 i 6

y=4

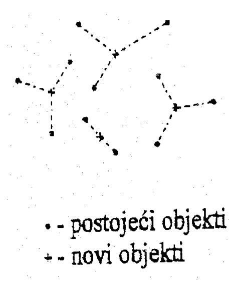
Primer 8.7. U ovom primeru rešavamo sledeći Veberov problem koristeći pravougaonu metriku:

lokacije		L1	L2	L3	L4	L5	L6
koordinate	x	500	400	200	500	300	100
	y	200	300	400	500	100	200
tež.koeficijenti		0.08	0.04	0.22	0.10	0.12	0.44

Rešenje se generiše izrazom
praveberŠšš500,200ć,š400,300ć,š200,400ć,š500,500ć,š300,100ć, š100,200ćć,š0.08,0.04,0.22,0.1,0.12,0.44ćĆ
x=200
y=200

8.4. LOKACIJSKO-ALOKACIJSKI PROBLEM

Neka je dato m tačaka $A_i=(a_1^i,a_2^i)$ koje predstavljaju lokacije korisnika (postojeći objekti) i njihovi težinski koeficijenti w_i, $i=1,...m$. Potrebno je odrediti lokacije za p snabdevača (to su novi objekti) $X_k=(x_1^k,x_2^k)$, $k=1,...p$, tako da suma otežanih rastojanjaod njih do korisnika bude minimalna (lokacijski problem), kao i odrediti koji sanbdevač će biti prodružen kom korisniku (alokacijski, tj. asignacijski problem).
Ovom problemu odgovara sledeći realni zadatak: u nekom naselju postoji m stambenih zgrada, a potrebno je rasporediti p prodavnica tako da budu što bliže stanovnicima naselja. Osim mesta (lokacija) na kojima bi prodavnice trebalo sagraditi, ovaj zadatak podrazumeva i to da je potrebno odrediti u kojim će se prodavnicama snabdevati stanovnici kojih zgrada. Jedan mogući slučaj je prikazan na slici 3.



slika 3.

Matematički model ovog problema ima sledeći oblik:

$$\text{(min)} \qquad f(C,X)=\sum_{i=1}^{m}\sum_{k=1}^{p} c_{ik} w_i d_i\left(X_k\right),$$

$$\text{p.o.} \qquad \sum_{k=1}^{p} c_{ik}=1,\ c_{ik}>0,\ i=1,...m,\ k=1,...,p\ ,$$

gde su:

m - broj postojećih objekata (korisnika);

p - broj novih objekata (snabdevača);

$C=(c_{ik})_{m\mathrm{X}p}$ - matrica u kojoj je $c_{ik}\in$Š0,1Ć alokacijska promenljiva koja predstavlja deo potreba koje i-ti korisnik zadovoljava kod k-tog snabdevača;

$X=(X_1, ..., X_p)$ Đ vektor u kome je $X_k=(x_1^k, x_2^k)$ lokacija k-tog snabdevača koju treba odrediti;

$d_i(X_k)=d(A_i,X_k)$ - rastojanje od i-tog korisnika do k-tog snabdevača;

w_i - težinski koeficijent i-tog korisnika (npr. broj stanara zgrade).

Razmotrićemo dve varijante ovog problema:

1) Kapaciteti snabdevača su neograničeni, tj. proizviljan broj korisnika može se dodeliti svakom od snabdevača; tada se dobija rešenje u kome promenljiva c_{ik} uzima vrednosti iz skupa š0,1ć.
2) kapaciteti snabdevača su ograničeni; optimalne vrednosti promenljive c_{ik} mogu imati bilo koju realnu vrednost iz intervala Š0,1Ć. U ovom slučaju, polazni matematički model je potrebno proširiti sledećim skupom ograničenja: $\sum_{i=1}^{m} c_{ik} q_i \le Q_k$, $k=1,...p,$

   gde su:

q_i - potrebe korisnika u tački A_i;

Q_k- kaoacitet snabdevača u X_k.

Bez obzira da li se radi o prvoj ili drugoj varijanti, ovaj problem je analitički teško rešiti. Zbog toga sa predlaže jedna heuristička metoda za unapred zadat broj snabdevača p.

8.4.1. Kuperov algoritam

U slučaju kada kapacitet snabdevača nije ograničen, Kuperov algoritam ima sledeće korake:

1) Proizvoljno izabrati p tačaka X_1,...,X_p.
2) Rešiti alokacijski problem, tj. svakom korisniku dodeliti najbližeg od snabdevača lociranih u tačkama X_1,...,X_p. Ovaj postupak se svodi na

rešavanje p diskretnih lokacijskih problema. (Izračunava se matrica rastojanja $D=(d_i(X_k))_{mXp}$ i nalazi minimum po svakoj vrsti. Tako se odrede elementi matrice $C=(c_{ik})$, $c_{ik}\in$š0,1ć).

3) Za svako $k\in$š1,...,pć rešiti Veberov problem u odnosu na sve korisnike vezane za snabdevača X_k. Tako dobijena tačka postaje novo X_k. Rešiti alokacijski problem kao u koraku 2).
4) Ako nema promena u alociranju u odnosu na predhodnu iteraciju, prekinuti postupak. Poslednje dobijeno rešenje je i konačno rešenje problema. U suprotnom, vratiti se na korak 3).

```
kuperŠlp_,lt_,bn_Ć:=
 ModuleŠši,po=šć,a,bć,
  ForŠi=1,i<=bn,i++,
    po=AppendŠpo,ši*100,i*200ćĆ;
  Ć;
  WhileŠTrue,
    a=napraviŠlp,lt,poĆ; b=napraviŠlp,lt,aŠŠ1ĆĆĆ;
    IfŠaŠŠ2ĆĆ==bŠŠ2ĆĆ,ReturnŠbŠŠ1ĆĆĆ,po=bŠŠ1ĆĆĆ;
 Ć Ć;
```

8.5. LOKACIJSKI PROBLEMI NA MREŽAMA

Slično modelima optimizacije za rešavanje lokacijskih problema u ravni, formulišu se i različiti modeli za rešavanje problema lokacije na mrežama. U ovim modelima se koristi pojam rastojanja između dva čvora mreže. To rastojanje je po definiciji dužina najkraćeg puta između posmatranih čvorova. Treba primetiti da je, u nesimetričnim mrežama, pri računanju dužine puta važno koji čvor početni, a koji krajnji čvor puta. Prema tome, za jedan par čvorova mogu postojati dva rastojanja.

Čvorovima se pripisuju brojevi koji se koriste kao težinski koeficijenti pri računanju otežanih rastojanja. Kao kriterijum može da se koristi otežano rastojanje od novog objekta do najdaljeg čvora ili zbir otežanih rastojanja od novog objekta do čvorova mreže. U prvom slučaju radi se o problemima tipa *minimax* koji se u teoriji grafova nazivaju problemi određivanja centra grafa. U drugom slučaju u pitanju su problemi tipa *minisum*, odnosno problemi određivanja *medijane grafa*.

Lokacijski problemi na mrežama mogi da se klasifikuju i prema tome da li je novi objekat moguče postaviti samo u čvor grafa ili je to moguče učiniti i na grani grafa. Na kraju, zadaci sa mogu uopštiti za slučajeva kada se razmešta više objekata.

Mrežase može predstavljati grafom

$$G = (N, L), N = \text{š}1, \ldots ,n\text{ć}, l = \text{š}(i,j)\ \text{đ}\ i{\in}N,\ \ j{\in}N\text{ć},$$

pri čemu se granama $(i, j){\in}L$ pridružuju dužine $c_{ij}{\geqslant}0$, a svakom čvoru $i{\in}N$ težina w_i. Rastojanje između čvorova i i j označavaćemo sa $d(i,j)$.

8.5.1. Problem lokacije službe za hitne intervencije u čvoru mreže

Pretpostavimo da je za jednu službu koja pruža hitne usluge stanovništvu nekoliko naselja potrebno odabrati lokaciju i da zbog nekih razloga ta služba treba da se nalazi u naselju. Kao kriterijum je moguće koristiti:

a) Rastojanje službe do najdaljeg čvora (što može da odgovara vremenu putovanja od mesta službe do mesta gde je potrabna pomać, npr. vatrogasna služba);
b) Rastojanje od najdaljeg čvora do službe (kada klijent sam treba da dođe u službu za pružanje pomoći);
c) Rastojanje od mesta službe do najdaljeg čvora i nazad (kada se putuje od službe do mesta za pomoć i nazad kao u primeru hitne pomoći koja treba da ode do bolesnika i doveze ga u bolnicu).

Svako rastojanje može se otežati brojem stanovnika naselja ili verovatnoćom da će baš to naselje zatražiti uslugu. Zadatak je naći lokaciju službe za pomoć tako da otežano rastojanje bude što je moguće manje.
Ako se kao kriterijum koristi rastojanje od mesta službe do najdaljeg čvora, tada se zadatak lokacije formuliše na sledeći način:
Odrediti čvor i_0^* takav da je

$$\xi_0(i_0^*) = \min_{i\in N} \text{Š}\max_{j\in N}\ w_i\, d(i,j)\text{Ć}.$$

Čvor i_0^* se naziva *spoljašnji centar* težinskog grafa G, a vrednost $\rho_0{=}\xi_0\,(i_0^*)$ *spoljašnji radijus* grafa. Jedan graf može imati više spoljašnjih centara.
U slučaju da se kao kriterijum koristi rastojanje od najdaljeg čvora do mesta novog objekta, tada se zadatak lokacije formuliše na sledeći način:
Odrediti čvor i_t^* takav da je

$$\xi_t(i_t^*) = \min_{i\in N} \text{Š}\max_{j\in N}\ w_j\, d(j,i)\text{Ć}.$$

Čvor i_t se naziva *unutrašnji centar* težinskog grafa G, a vrednost $\rho_{\,t}{=}\xi_t\,(i_t^*)$ *unutrašnji radijus* grafa. Jedan graf može imati više unutrašnjih centara.
Ukoliko se računa ukupno rastojanje od mesta lokacije da najdaljeg čvora i nazad, lokacijski problem se sastoji u sledećem:
Odrediti čvor $\boldsymbol{i_{ot}^*}$ takav da je

$$\xi_{ot}(i_{ot}^*) = \min_{i\in N} \text{Š}\max_{j\in N}\, w_j\,(d(j,i) + d(j,i)\text{Ć}.$$

Čvor i_{ot} naziva se *unutrašnjo-spoljašnji centar* težinskog grafa G, a vrednost $\rho_{ot}=\xi_{ot}(i_{ot}^{*})$ unutrašnjo-spoljašnji radijus grafa.

Primetimo da u mrežama za koje važi da je $c_{ij}=c_{ji}$ za svako $(i,j)\in L$ važi i $d(i,j)=d(j,i)$ pa se unutrašnji, spoljašnji i unutrašnjo-spoljašnji centri grafa poklapaju.

Algoritam za određivanje spoljašnje centra grafa ima sledeće korake:

1) Naći rastojanja između svaka dva čvora u mreži i formirati matricu rastojanja $D = d(i,j)$ čiji su elementi rastojanja između čvorava i i j, za svako $i,j = 1,...n$.
2) Svaku kolonu j matrice D pomnožiti težinom čvora w_j, tj. formirati matricu otežanih rastojanja D'.
3) Naći maksimalni element $\xi_0(i)$ za svaki red matrice D', tj. odrediti
$$\xi_0(i)=\max_{j\in N} \text{š} w_j\, d(i,j)\text{ć},\ \ i = 1,...,n.$$
4) Od svih $\xi_0(i)$ određenih u predhodnom koraku naći najmanje, tj.
$$\xi_0(i_0^{*}) = \min_{j\in N}\ \xi_0(i_0^{*}).$$

Za nalaženje unutrašnjeg i unutrašnjo-spoljašnjeg centra i poluprečnika grafa ovaj algoritam se neznatno modifikuje.

Sledi implementacija algoritma.

```
centriŠm_,lt_Ć:=
 ModuleŠšmt,s,ds,i,u,du,tm,dd,tdd,us,dusć,
   mt=mnkolonaŠm,ltĆ;s=minmaxredŠmtĆ; ds=LengthŠsĆ;
   ForŠi=1,i<=ds,i++,
     PrintŠ"Spoljasnji centar je cvor "Ć; PrintŠsŠŠiĆĆĆ;
   Ć;
   u=minmaxkolŠmtĆ;  du=LengthŠuĆ;
   ForŠi=1,i<=du,i++,
     PrintŠ"Unutrasnji centar je cvor "Ć;  PrintŠuŠŠiĆĆĆĆ;
     tm=TransposeŠmĆ; dd=tm+m; tdd=mnkolonaŠdd,ltĆ;
     us=minmaxredŠtddĆ; dus=LengthŠusĆ;
     ForŠi=1,i<=dus,i++,
          PrintŠ"Unutrasnjo-spljasnji centar je cvor "Ć;
          PrintŠusŠŠiĆĆĆ;
     Ć;
  Ć;
```

8.5.2. Problem lokacije skladišta (snabdevača)

Pretpostavimo da od nekoliko naselja treba izabrati jedno u koje će se smestiti zajedničko skladište. Iz ovog skladišta snabdevaju se sva naselja tako da su troškovi transporta od izabranog do nekog drugog naselja proporcionalni njihovom rastojanju. Ukupni troškovi su jednaki otežanoj sumi ovih rastojanja gde se težinski koeficijenti pripisuju naseljima, tj.

čvorovima grafa. I u ovom slučaju je, zavisno od načina računanja rastojanja, moguče postaviti sledeća tri optimizaciona zadatka.

1. Ako se kao kriterijum koristi rastojanje od mesta skladišta do mesta potrošača, tada se zadatak lokacije formuliše na sledeći način:

Odrediti čvor i_0^* za koji je:

$$\lambda_0\left(i_0^*\right) = \min_{\{i \in N\}}[\sum_{i \in N} w_j d(i, j)].$$

Čvor i_0^* naziva se *spoljašnja medijana* težinskog grafa G.

2. U slučaju da se kao kriterijum koristi rastojanje od potrošača do skladišta, tada se zadatak lokacije formuliše na sledeši način:

Odrediti čvor i_t^* takav da je

$$\lambda_t\left(i_t^*\right) = \min_{\{i \in N\}}[\sum_{j \in N} w_j d(j, i)].$$

Čvor i_t^* se naziva *unutrašnja medijana* težinskog grafa *G*.

3. Kada se računa ukupno rastojanje od skladišta do potrošača i nazad, zadatak lokacije je:

Odrediti čvor i_{0t}^* takav da je:

$$\lambda_{0t}\left(i_{0t}^*\right) = \min_{\{i \in N\}}[\sum_{j \in N} w_j d(j, i) + d(i, j)].$$

Čvor i_{0t}^* se naziva *unutrašnjo-spoljašnja medijana* težinskog grafa G.

```
medijanaŠm_,lt_Ć:=
 ModuleŠštm,dd,tdd,dm,i,j,pom,a,k,s=šć,indć,
  tm=TransposeŠmĆ;dd=tm+m; tdd=mnkolonaŠdd,ltĆ;
  dm=LengthŠtddĆ;
  ForŠi=1,i<=dm,i++,
    pom=šć;
    ForŠj=1,j<=dm,j++,
       pom=AppendŠpom,PartŠtddŠŠjĆĆ,iĆĆ Ć;
       a=SumŠpomŠŠkĆĆ,šk,dmćĆ; s=AppendŠs,aĆ;
     Ć;
    ind=1;
    ForŠi=2,i<=dm,i++, IfŠsŠŠindĆĆ>sŠŠiĆĆ,ind=iĆ;
  Ć;
  ForŠi=1,i<=dm,i++,
       IfŠsŠŠiĆĆ==sŠŠindĆĆ,
         PrintŠ"Medijana je cvor:"Ć;PrintŠiĆ
ĆĆ Ć
```

Zadaci određivanja medijane mogu da se formulišu i za slučaj kada je dopušteno da se skladište postavi u bilo koju tačku grafa. Pokazuje se, međutim, da se tačka koja odgovara medijani grafa i tada nalazi u nekom od čvorova. Na taj način se problem lokacije na mrežama tipa minisum svode na diskretne lokacijske probleme. Prethodno je potrebno samo izračunati rastojanje između čvorova i njihove odgovarajuće sume.

8.6. PRIMERI

U sledećim primerima ilustruje se primena prethodno datih programa.

1. Potrebno je organizovati sistemski pregled za predškolsku decu u opštini koja ima 5 vrtića (označeni sa *V*), čije su lokacije određene koordinatama *X* i *Y*. Ekipa zdravstvenog osoblja koja vrši sistematski pregled treba da odabere jednu od 4 zdravstvene ustanove (označene sa *Z*) u kojoj će pregled biti obavljen tako da ukupni troškovi prevoza dece (suma otežanih rastojanja) budu što manji. Kako se radi o urbanoj sredini, treba koristiti pravougaonu metriku. Koordinate zdravstvenih ustanova, vrtiša i broj dece (sa oznakama *D*) u njima dati su u tabelama:

V	V1	V2	V3	V4	V5
X	15	58	40	66	80
Y	40	30	8	14	44
D	155	92	134	112	48

Z	Z1	Z2	Z3	Z4
x	32	26	70	60
y	40	25	30	8

Rešenje se dobija pozivom funkcije:
primer83Ššš15,40ć,š58,30ć,š40,8ć,š66,14ć,š80,44ćć,
šš32,40ć,š26,25ć,š70,30ć,š60,8ćć,š155,92,134,112,48ćĆ
Za rešenje se dobija š32,40ć.

2. Odrediti lokaciju novog objekta koji treba da snabdeva korisnike koji stanuju u 5 postojećih objekata. Koordinata ovih objekata su: *A*(1,2), *B*(3,1), *C*(7,2), *D*(7,7), *E*(2,6). Na osnovu broja stanara u njima, ovim objektima su dodeljeni sledeći težinski koeficijenti: 2 za *A* i *D*, 3 za *B*, 4 za *C*, a za *E* je 5. Novi objekat trebalo bi postaviti na jednoj od sledeće 3 moguće lokacije: *N*1(6,1), *N*2(5,6) ili *N*3(1,5), tako da suma otežanih rastojanja od njega do postojećih objekata bude što je moguće manja. Za merenje rastojanja koristiti l_2 metriku.

Pozivom
primer81Ššš1,2ć,š3,1ć,š7,2ć,š7,7ć,š2,6ćć,šš6,1ć,š5,6ć,š1,5ćć,š2,3,4,2,5ćĆ
dobija se rešenje je š5,6ć.

3. Istraživačka ekspedicija ispitivaće reon u kome se nalazi 5 nalazišta, a u blizini se nalazi samo jedan izvor vode. Članovi ekspedicije su odlučili da postave logor na jednom mestu, a da odatle ujutru odlaze do mesta na kojima se vrše iskopavanja, a uveče se vraćaju. Za svako od nalažišta je određeno: koliko će dana biti potrebno vršiti istraživanja. Takođe je određeno koliko će puta biti potrebno za to vreme otići do izvora da bi se logor snabdeo vodom. Ovi podaci, kao i koordinate nalazišta i izvora (u *km*) dati su u tabeli:

nalazište		A	B	C	D	E	izvor
koordinate	x	1	7	3	9	14	4
	y	6	7	2	3	6	4
broj odlazaka		2	6	1	7	10	18

Članovi ekspedicije treba da odrede lokaciju logora tako da se što je moguće manje izgubi vremena na putu do nalazišta, ali i da izvor vode bude što bliži. Kako je moguće kretati se pravolinijski, za računanje udaljenosti koristićemo Euklidovu metriku, a za težinske koeficijente tačaka broj odlazaka do njih. Smatra se da su koordinate logora dovoljno precizne ako se za koeficijent zaustavljanja usvoji ε=0.25 (250*m*).

Pozivom
euveberŠšš1,6ć,š7,7ć,š3,2ć,š9,3ć,š14,6ć,š4,4ćć,š2,6,1,7,10,18ć, 0.25Ć
dobijamo rešenje š6.47628, 4.60516ć.

4. U naselju od 6 solitera potrebno je sagraditi prodavnicu široke potrošnje. Koordinate solitera, kao i broj stanara u svakom od njih, dati su u sledećoj tabeli:

soliter		S1	S2	S3	S4	S5	S6
koordinate	x	5	20	40	20	40	70
	y	20	30	35	10	15	20
broj stanara		180	60	90	120	50	200

a) Odrediti lokaciju prodavnice tako da jedini kriterijum bude taj da suma otežanih rastojanja između nje i solitera bud minimalna. Koristiti pravougaonu metriku, a za težine usvojiti broj stanara solitera.

Rešenje se bobija pomoću:
praveberŠšš5,20ć,š20,30ć,š40,35ć,š20,10ć,š40,15ć,š70,20ćć,
š180,60,90,120,50,200ćĆ

x=20
y=20

b) Naknadno je urbanističkim planom utvrđeno da se prodavnica mora nalaziti na jednoj od sledeće tri lokacije: $P1(15,15)$, $P2(30,25)$, $P3(50,30)$. Koristeći isti kriterijum i metriku kao u a), utvrditi na kojoj od ove tri lokacije sagraditi prodavnicu.

Rezultat se dobija pomoću poziva funkcije:
primer83Ššš5,20ć,š20,30ć,š40,35ć,š20,10ć,š40,15ć,š70,20ćć,
šš15,15ć,š30,25ć,š50,30ćć,š180,60,90,120,50,200ćĆ
Rešenje je š30,25ć.

LITERATURA

[1] D. Urošević, *Algoritmi u programskom jeziku C,* Mikro knjiga, Beograd, 1996.

[2] M. Vujošević, M. Stanojević, N. Mladenović, *Metode optimizacije,* Društvo operacionih istraživača, Beograd 1996.

9. IZRAZI GENERISANI REGULARNIM GRAMATIKAMA

9.1. Uvod

Regularne gramatike definisane su uređenom četvorkom (V_N, V_T, S, V_P), gde su:
V_N - neterminalni alfabet;
V_T -terminalni alfabet, za koga važi $V_T \cap V_A = \emptyset$;
$S \in V_N$ - početni simbol ili aksioma, koja zadovoljava $S \in V_A$;
$V_P \subset V_A \times (V_N \cup V_T)$ - skup pravila izvođenja.
Sva pravila izvođenja u regularnoj gramatici su oblika:
$\alpha \to \beta$, $|\alpha| \le |\beta|$, $\alpha \in V_N$, $\beta \in \{aB, a\}$, $a \in V_T$, $B \in V_N$.
Jezik L je regularan ako je generisan regularnom gramatikom.
U tekstu koji sledi, razvijen je softver za karakterizaciju regularnih gramatika. Takođe, implementiran je algoritam za konstrukciju regularnih izraza koji mogu biti generisani ovim gramtikama. Konačno, razvijen je softver koji za zadati izraz ispituje da li može biti generisan datom

regularnom gramatikom. Ako reč može biti genrisana gramatikom, potrebno je da se generiše odgovarajući skup transformacija koje proizvode dati izraz.

9.2. Softver

Formalni parametri *VN_, VT_, AKS_* i *VP_* funkcije *Regular* predstavlja regularnu gramatiku. Konačno, formalni parametar *MaxLen* predstavlja maksimalnu dužinu generisanog regularnog (neterminalnog ili terminalnog) izraza. Proveravamo sledeći glavni kriterijum za karakterizaciju gramatike (*VN,VT,AKS,VP*) da bi bila regularna:

1. $VN \cap VT=\emptyset$;
2. prvi element u svakom pravilu $\alpha \to \beta$ je element iz *VN*;
3. za đαđ = k mora da zadovoljava

$$\alpha_i \in VT,\ \ i=1,\dots,k\text{-}1,\ \ \alpha_k \in VN \cap VT.$$

Za automatsku karakterizaciju regularnih gramatika koristi se sledeći kod:

```
RegularŠVN_,VT_,AKS_, VP_,MaxLen_Ć:=
 BlockŠšp=True,d,rul,start,aux=šć,auxlist,ls=šć,
   firstsymbol,firstsymbol2, sl, change, i,j,h,
     n,w,k, listrules=šć, startrules=šć ć,
   IfŠIntersectionŠVT,VNĆ==šć,       (*then*)
     ForŠi=1,i<=LengthŠVPĆ,i++,
      rul=VPŠŠiĆĆ/.Rule->List;
      aux=CharactersŠToStringŠrulŠŠ2ĆĆĆĆ;
      k=LengthŠauxĆ;
      ForŠj=1,j<= k,j++,
          auxŠŠjĆĆ=ToExpressionŠauxŠŠjĆĆĆ
      Ć;
      PrependToŠaux, rulŠŠ1ĆĆĆ;
      p=p && MemberQŠVN,FirstŠauxĆĆ;
      ForŠj=2,j<=k-1,j++,
         p=p && MemberQŠVT,auxŠŠjĆĆĆĆ;
         p=p&&(MemberQŠVN,auxŠŠkĆĆĆ
              đđMemberQŠVT,auxŠŠkĆĆĆ)
     Ć,                        (*else*)
     p= False
 Ć;
```

U slučaju da je *p=False* gramatika (*VN,VT,AKS,VP*) nije regularna; u suprotnom, ako je *p=True*, gramatika je regularna. Sva pravila iz liste *VP* u formi $\alpha \to \beta$, gde je α=*AKS*, su predstavljene listom početnih pravila, koja se mogu generisati primenom sledećeg koda:

```
ForŠj=1,j<=LengthŠVPĆ,j++,
```

```
  firstsymbol=ToStringŠ(VPŠŠjĆĆ/.Rule->List)ŠŠ1ĆĆĆ;
  IfŠfirstsymbol==ToStringŠAKSĆ,
        AppendToŠstartrules,VPŠŠjĆĆĆ
Ć  Ć;
```

Svi neterminalni i terminalni izrazi dužine 1, ... ,*MaxLen* mogu biti generisani primenom sledećeg koda. Unutar ciklusa
ForŠw=1,w<=LengthŠstartrulesĆ, w++,
koristimo sve elemente iz liste startnih pravila koja su sposobna da generišu regularni izraz.

```
ForŠw=1,w<=LengthŠstartrulesĆ, w++,
 ls=ListŠstartrulesŠŠwĆĆĆ;
 ForŠj=0, j<=MaxLen,j++,
  h=LengthŠlsĆ;  auxlist=šć;
  ForŠi=1, i<=h, i++,
    startsymbol=(lsŠŠiĆĆ/.Rule->List)ŠŠ1ĆĆ;
    start=(lsŠŠiĆĆ/.Rule->List)ŠŠ2ĆĆ;
    aux=CharactersŠToStringŠstartĆĆ;
    sl=LastŠauxĆ;    aux=DropŠaux,-1Ć;
    listrules=šć;
    ForŠk=1, k<=LengthŠVPĆ,k++,
     firstsymbol2=ToStringŠ(VPŠŠkĆĆ/.Rule->List)ŠŠ1ĆĆĆ;
       IfŠsl==firstsymbol2,
          AppendToŠlistrules,VPŠŠkĆĆĆĆ;
       Ć;
       ForŠn=1, n<=LengthŠlistrulesĆ, n++,
         sl=ToExpressionŠslĆ;
         change= sl/.listrulesŠŠnĆĆ;
        change=ToExpressionŠStringJoinŠauxĆ<>ToStringŠchangeĆĆ;
         AppendToŠauxlist,RuleŠfirstsymbol,changeĆĆ
        Ć;
     Ć;
     ls=auxlist;     k=LengthŠlsĆ;
     ForŠl=1,l<=k,l++,
        PrintŠ(lsŠŠlĆĆ/.Rule->List)ŠŠ2ĆĆĆ
     Ć
   Ć;
 Ć
```

Primer.
InŠ1Ć:=RegularŠšA,B,Cć,ša,b,c,eć,A,
šA->bB,B->cC,C->eć,10Ć
Regular grammar

bcC
bce
InŠ2Ć:=RegularŠšA,B,C,Dć,ša,b,c,eć,A,
šA->aB,B->cD,A->bD,D->aBć,15Ć
Regular grammar
acD
acaB
acacD
acacaB
acacacD
acacacaB
acacacacD
acacacacaB
acacacacacD
acacacacacaB
acacacacacacD
acacacacacacaB
acacacacacacacD
acacacacacacacaB
acacacacacacacacD
acacacacacacacacaB
baB
bacD
bacaB
bacacD
bacacaB
bacacacD
bacacacaB
bacacacacD
bacacacacaB
bacacacacacD
bacacacacacaB
bacacacacacacD
bacacacacacacaB
bacacacacacacacD
bacacacacacacacaB
bacacacacacacacacD

Rešićemo još jedan problem. Naime, neka je data proizvoljna reč. Utvrditi kada reč može biti generisana regularnom gramatikom. Ako je to moguće, formirati listu pravila koja generišu reč. U sledećoj proceduri, koristi se parametar *Chain* koji reprezentuje posmatranu reč.
U cilju pronalaženja skupa pravila izvođenja koja generišu lanac, koristimo bottom-up raščlanjivanje koje je definisano sledećim pravilima:

Pravilo 1. Ako je poslednji simbol datog lanca terminalni karakter $a \in V_T$, i ako postoji izvođenje $A \rightarrow a$, $A \in V_N$, zameniti simbol a sa A.
Pravilo 2. U suprotnom, proveriti postojanje pravila $\alpha \rightarrow bB$, gde je $\alpha \in V_N$, $b \in V_T$, $B \in V_N$ i bB kao poslednja dva simbola u lancu. Ako je to slučaj, zameniti bB u lancu sa neterminalnim simbolom α.

Ako oba ova pravila nisu primenljiva, reč ne može biti generisana datom gramatikom regularnom gramatikom. U suprotnom, nastavljamo primenu pravila sve dok starti lanac ne bude transformisan u neterminal $A \in V_N$. Kada je netrminal A jednak aksiomi, onda dati lanac može biti generisan datom gramatikom. Inače, lanac ponovo ne može biti generisan datom gramatikom.
Tokom primene Pravila 1 i Pravila 2, pamtimo redni broj primenjene transformacije u listu označenu sa *deriv*. Takođe, sve transformacije datog lanca su zapamćene u listi koja predstavlja vrednost promenljive *transform*.

```
Regular1ŠVN_,VT_,AKS_, VP_,Chain_Ć:=
 BlockŠšp=True,d,d2,drv,aux=šć,i,j,k,brl,sim="",
    rulelist=šć,auxchain, transform=šć,auxtransform=šć,
    end=False,deriv=šć,rulderiv=šć,
     auxderiv=šć, auxrulderiv=šćć,
   d=LengthŠVPĆ;
   ForŠi=1,i<=d,i++,         (*then*)
     drv=VPŠŠiĆĆ;   drv=drv/.Rule->List;
     rulelist=AppendToŠrulelist,drvĆ;
     auxchain=ToStringŠChainĆ;
     d=LengthŠrulelistĆ;
     ForŠi=1,i<=d,i++,
       drv=rulelistŠŠiĆĆ;
       IfŠToStringŠdrvŠŠ2ĆĆĆ==StringTakeŠauxchain,-1Ć,
       AppendToŠtransform,StringDropŠauxchain,-1Ć
                       <>ToStringŠdrvŠŠ1ĆĆĆĆ;
       AppendToŠderiv,ListŠiĆĆ;
       AppendToŠrulderiv,ListŠLastŠtransformĆĆĆ;
       IfŠLastŠtransformĆ==ToStringŠAKSĆ,
          end=True;  brl=LengthŠtransformĆ,
          IfŠStringLengthŠLastŠtransformĆĆ==1,
            transform=DropŠtransform,-1Ć;
            deriv=DropŠderiv,-1Ć;
            rulderiv=DropŠrulderiv,-1Ć
   Ć Ć  Ć  Ć;
```

```
  WhileŠ(!end)&&(transform!=šć),
   d2=LengthŠtransformĆ;
   ForŠi=1,i<=d2,i++,
     ForŠj=1,j<=d,j++,
       drv=rulelistŠŠjĆĆ;
       IfŠToStringŠdrvŠŠ2ĆĆĆ==StringTakeŠtransformŠŠiĆĆ,-2Ć,
         AppendToŠauxtransform,
            StringDropŠtransformŠŠiĆĆ,-2Ć
                        <>ToStringŠdrvŠŠ1ĆĆĆĆ;
         AppendToŠauxderiv,PrependŠderivŠŠiĆĆ,jĆĆ;
         AppendToŠauxrulderiv,
                  PrependŠrulderivŠŠiĆĆ,
                   LastŠauxtransformĆĆ
         Ć;
         IfŠLastŠauxtransformĆ==ToStringŠAKSĆ,
           end=True; brl=LengthŠauxtransformĆ,
           IfŠStringLengthŠLastŠauxtransformĆĆ==1,
             auxtransform=DropŠauxtransform,-1Ć;
             auxderiv=DropŠauxderiv,-1Ć;
             auxrulderiv=DropŠauxrulderiv,-1Ć
   Ć Ć Ć Ć Ć;
   transform=auxtransform;
   deriv=auxderiv;  rulderiv=auxrulderiv;
   auxtransform=šć;  auxderiv=šć;    auxrulderiv=šćĆ;
   IfŠend,
     PrintŠ"Chain has the following derivation:"Ć;
     d=LengthŠderivŠŠbrlĆĆĆ;
     ForŠi=1,i<=d,i++,
       sim=sim<>(rulderivŠŠbrlĆĆ)ŠŠiĆĆ;
       sim=sim<>" (rule ";
       sim=sim<>ToStringŠVPŠŠ(derivŠŠbrlĆĆ)ŠŠiĆĆĆĆĆ
                <>") => "
     Ć;
     sim=sim<>auxchain,
     ReturnŠ"Chain has no derivation"Ć
   Ć
 Ć
```

Primer.

InŠ1Ć:=Regular1ŠšA,B,C,Dć,ša,b,c,eć,A,
šA->aB,B->cD,A->bD,D->aB,B->bć, acacacabĆ
Chain has the following derivation:
OutŠ1Ć=A (rule A->aB) => aB (rule B->cD) => acD
(rule D->aB) => acaB (rule B->cD) => acacD

(rule D->aB) => acacaB (rule B->cD) => acacacD
(rule D->aB) => acacacaB (rule B->b) => acacacab

Dobijene međuvrednosti u promenljivima *transform* i *deriv*:
transform = šaaabaCć deriv = šš5ćć
transform = šaaabCć deriv = šš4,5ćć
transform = šaaaBć deriv = šš3,4,5ćć
transform = šaaSć deriv = šš2,3,4,5ćć
transform = šaSć deriv = šš1,2,3,4,5ćć
transform = šSć deriv = šš1,1,2,3,4,5ćć

LITERATURA

[1] N. Blachman, *Mathematica: A Practical Approach,* Englewood Cliffs, New Jersey, Prentice-Hall, 1992.

[2] D. Gries, *Compiler Construction for Digital Computers*, John Wiley & Sons, Inc., New York, London, Sydney, Toronto, 1971.

[3] R. Maeder, *Programming in Mathematica, Third Edition,* Redwood City, California, Adisson-Wesley,1996.

[4] R. Sedgewick, *Algorithms in C*, Addison-Wesley Publishing Company, Reading, MA, 1990.

[5] J.P. Tremblay and P.G. Sorenson, *The Theory and Practice of Compiler Writ*ing, McGraw-Hill Book Company, New York, 1985.

[6] S. Wolfram, *Mathematica: a System for Doing Mathematics by Computer*, Addison-Wesley Publishing Co, Redwood City, California, 1991.

[7] S. Wolfram, *Mathematica Book*, Version 3.0, Addison-Wesley Publishing Co, Redwood City, California, 1997.

10. TJURINGOVA MAŠINA U MATHEMATICA

10.1. UVOD

Tjuringova mašina M je konačni uređaj čije se operacije vrše na pokretnoj traci. Ova traka je beskonačna sa obe strane i podeljena na pojedinačne ćelije. U svakom trenutku ćelija trake sadrži jedan simbol iz konačne liste ulaznih simbola Σ ili blanko karaktera **#.** Formalno, Tjuringova mašina je definisana uređenom petorkom (Q, q_0, Σ, Γ, Δ), gde je:

Q	neprazan konačan skup stanja
q_0	početno stanje
Σ	ulazni alfabet
Γ	alfabet trake, definisan sa γ=Σ∩š#ć
#	za neke blanko karaktere #∉σ
Δ	Γ×*Q* →Σ×Q×š*L*,*R*ć je parcijalna funkcija kretanja

Akcija koju *M* preduzima u svakom trenutku zavisi od trenutnog stanja *M* i od simbola koji je prethodno skaniran. Ova zavisnost je definisana u *M*-ovoj funkciji kretanja Δ. Ova funkcija je definisana skupom pravila *T*, u formi šq_i,s_jć→šq_k,s_l,αć,gde je q_i∈Γ, q_k∈Σ, s_j, s_l∈Q i α∈š*L*,*R*ć. Pravilo šq_i,s_jć→šq_k,s_l,αć određuje akciju kad je *M* u stanju q_i i skanira simbol s_j. U ovom slučaju, mašina uzima novo stanje q_k, u skaniranoj ćeliji upisuje simbol s_l i pomera traku kao što sledi:

- Ako je α=*R*, pomera glavu za čitanje za jednu ćeliju udesno,
- a ako je α=*L*, pomera glavu za jednu ćeliju ulevo.

M prestaje sa radom kad je u stanju q_i, skanira simbol s_j tako da ne postoji pravilo šq_i,s_jć→šq_k,s_l,αć∈*T*. Više o Tjuringovoj mašini može se pročitati u Š2Ć, Š3Ć, Š4Ć, Š6Ć. Ovde ćemo opisati implementaciju Tjuringove mašine u paketu *MATHEMATICA* 4.0. Uglavnom se koriste mogućnosti simboličkog procesiranja u *MATHEMATICA*. Osim simboličkih mogućnosti paketa koristi se snaga funkcionalnog programiranja kao i reprezentacija dvo-dimenzionalnih grafičkih slika.

10.2. IMPLEMENTACIJA

Sadržaj trake predstavljen je globalnim nizom *TAPE*Š*1*Ć*, TAPE*Š*2*Ć*,...* u kojem se popunjavaju samo oni elementi koji su potrebni u određenom prenutku.Vrednost *TAPE*Š*i*Ć predstavlja sadržaj *i*-te ćelije trake. Niz *TAPE* može biti definisan stringom *s*, i popunjen koristeći funkciju *InitTape*Š*s*Ć. Globalne promenljive *MINPOS* i *MAXPOS* označavaju minimalni i maksimalni indeks, respektivno, između popunjenih kvadrata. Sve ostale ćelije (drugačije nazvane *empty* ćelije) sadrže prazan karakter #. Takođe, globalna promenljiva *POS* označava indeks trenutne skenirane ćelije.
U funkciji *InitTape*Š*s*Ć koristimo mapping funkciju *Map* i potpunu funkciju za konstrukciju niza *TAPE :*

```
InitTapeŠs_StringĆ := (
   MAXPOS = 0;
   ClearŠTAPEĆ;
   TAPEŠ_Ć := "#";
```

MapŠ(TAPEŠ++MAXPOSĆ = #) &, CharactersŠsĆĆ;
MINPOS = POS = 1;   )

Odmah posle inicijalizacije trake, misleći na funkciju *InitTapeŠsĆ,* minimalni indeks *MINPOS* kao i trenutna pozicija *POS* su obe jednake 1. Ove vrednosti mogu se modifikovati na osnovu zahteva korisnika eksplicitno.

Primer 10.1. Traka je definisana stringom ″1101″ može se popuniti sledećim izrazom
InŠ1Ć:= InitTapeŠ"1101"Ć
Sadržaj trake može biti prikazan na sledeći način:
InŠ2Ć:= ?TAPE
Global žTAPE
TAPEŠ1Ć = "1"
TAPEŠ2Ć = "1"
TAPEŠ3Ć = "0"
TAPEŠ4Ć = "1"
TAPEŠ_Ć := "#"

Alfabet $\Sigma\subseteq\Gamma$, skup satnja Q i trenutno stanje kao i funkcija prelaza Δ su definisana sledećim pravilom

(10.1) T=ššq_{im},s_{jm}ć → šq_{km},s_{lm}ć, αć, m = 1,..., nć.

Skupovi Q, Σ i Γ su implicitno definisani sa

$$Q=\bigcup_{m=1}^{n} q_{i_m} \cup \bigcup_{m=1}^{n} q_{k_m}\text{ , } \Sigma=\bigcup_{m=1}^{n} s_{j_m} \cup \bigcup_{m=1}^{n} s_{l_m}\text{ , } \Gamma=\Sigma\cup\#.$$

Funkcija prelaza Δ definisana je pravilima koja su sadražana u T:

$$\Delta(q_1,s_1)=\begin{cases}(q_2,s_2,\alpha), & \{q_1,s_1\}\to\{q_2,s_2,\alpha\}\in T\\ \{\}, & \{q_1,s_1\}\to\{q_2,s_2,\alpha\}\notin T\end{cases}$$

Funkcija Δ je definisana funkcijom *InitDelta*Š*T*Ć*,* i označena je *MATHEMATICA* funkcijom *DELTA*. Takođe, izračunavanja funkcijom *InitDelta*Š*T*Ć definišemo početno stanje q_0 , označeno globalnom promenljivom *STATE*. U funkciji *InitDelta*Š*T*Ć koristi se funkcija *Scan* koja primenjuje datu funkciju na svaki element liste *T*.
InitDeltaŠT_Ć := (
ClearŠDELTAĆ;
DELTAŠ_, _Ć := šć;

```
ScanŠ(DELTAŠ#ŠŠ1,1ĆĆ, #ŠŠ1,2ĆĆĆ= #ŠŠ2ĆĆ) &, TĆ;
STATE = TŠŠ1,2,1ĆĆ; )
```

Primer 10.2. Neka je skup *T* definisan listom pravila

InŠ3Ć:=T=ššA,"0"ć->šA,"1",Rć,šA,"1"ć->šA,"1",Rć,
šA,"#"ć->šB,"#",Lććć

Posle primene funkcije *InitDelta*Š*T*Ć dobijamo sledeće inicijalno stanje q_0, označeno sa *STATE*, i sledeću definiciju funkcije *DELTA*:

InŠ4Ć:=InitDeltaŠTĆ;
STATE
OutŠ4Ć=A
InŠ5Ć:= ?DELTA

```
GlobalžDELTA
DELTAŠA, "Đ#"Ć = šB, "Đ#", Lć
DELTAŠA, "0"Ć = šA, "1", Rć
DELTAŠA, "1"Ć = šA, "1", Rć
DELTAŠ_, _Ć := šć
```

Pomoćnom funkcijom *TapeToList*Š*s*Ć konstruišemo listu čiji su elementi karakteri sadržani u nizu *s* indeksiranih promenljivih, sa početkom u poziciji *MINPOS* a krajem u poziciji *MAXPOS*. Jasno je da je formalni parametar *s* globalni niz *TAPE*.

```
TapeToListŠs_Ć :=
 ModuleŠšl = šć, ić,
  ForŠi=MINPOS, i<=MAXPOS, i++, AppendToŠl, sŠiĆĆĆ;
  ReturnŠlĆ;
 Ć
```

U svakom trenutku, svi relevantni elementi koji određuju skup stanja mašine *M* su utvrđeni u uređenoj petorci

(10.2) š*STATE, MINPOS, POS, MAXPOS*, *TapeToList*Š*TAPE*Ćć

pri čemu:

STATE - označava trenutno stanje;

MINPOS i *MAXPOS* Đ startna i krajnja pozicija popunjenog dela trake, respektivno;

POS - pozicija trenutno skanirane ćelije i,

*TapeToList*Š*TAPE*Ć Đ predstavlja listu

š*TAPE*Š*MINPOS*Ć,...,*TAPE*Š*MAXPOS*Ćć

odgovarajućih karaktera koji su sadržani u popunjenim delovima trake.

Konstruisan je globalni niz *History* koji sadrži elemente u formi (10.2). On se može formirati koristeći funkcije *Move* i *Compute*. Formalni parametar funkcije *Move* je lista š*s,x,d*ć, koja predstavlja desnu stranu izabranog

pravila *ša,bć*→*šs,x,dć* iz *T*. Pri tome je: s∈*Q* je novo stanje; *x*∈Σ je simbol koji će biti ispisan na traci u trenutnoj poziciji *POS*; d∈*šL,Rć* je simbol koji definiše pomeranje glave za čitanje.
Saglasno funkciji *Move*Š*šs,x,dć*Ć definisali smo listu u formi (10.2) i primenili na kraj liste *History*.

```
MoveŠšs_, x_, d_ćĆ := (
 PrintŠ"radi Move za ", šs, x, dćĆ;
 TAPEŠPOSĆ = x; POS = POS + IfŠd === R, 1, -1Ć;
 MINPOS = MinŠPOS, MINPOSĆ; MAXPOS = MaxŠPOS, MAXPOSĆ;
 STATE = s;
 AppendToŠHistory,šSTATE,MINPOS,POS,MAXPOS,
       TapeToListŠTAPEĆćĆ;
 True
MoveŠšćĆ = False;
```

Formalni parametri sledeće funkcije *Compute* su:
T - skup pravila (10.1); *t* - string koji određuje sadržaj trake.
Vrednosti za formalne parametre *s, x, d* u svakoj ćeliji funkcije *Move* su definisane kao vrednosti funkcije *DELTA*:

```
ComputeŠT_, t_Ć := (
 InitTapeŠtĆ;
 InitDeltaŠTĆ;
 History = ššSTATE,MINPOS,POS,MAXPOS,TapeToListŠTAPEĆćć;
 WhileŠMoveŠDELTAŠSTATE, TAPEŠPOSĆĆĆĆ    );
```

Primer 10.3. Razmotrimo skup pravila definisanih u Primeru 10.2. Procedura *Compute*Š*T,"*110101010*"*Ć proizvodi vrednosti globalne promenljive *History* koje su jednake

```
 ššA,1,1,9,š"1","1","0","1","0","1","0","1","0"ćć,
  šA,1,2,9,š"1","1","0","1","0","1","0","1","0"ćć,
  šA,1,3,9,š"1","1","0","1","0","1","0","1","0"ćć,
  šA,1,4,9,š"1","1","1","1","0","1","0","1","0"ćć,
  šA,1,5,9,š"1","1","1","1","0","1","0","1","0"ćć,
  šA,1,6,9,š"1","1","1","1","1","1","0","1","0"ćć,
  šA,1,7,9,š"1","1","1","1","1","1","0","1","0"ćć,
  šA,1,8,9,š"1","1","1","1","1","1","1","1","0"ćć,
  šA,1,9,9,š"1","1","1","1","1","1","1","1","0"ćć,
šA,1,10,10,š"1","1","1","1","1","1","1","1","1","#"ćć,
šG,1,9,10,š"1","1","1","1","1","1","1","1","1","#"ććć
```

Napisaćemo funkciju *DumpHistory* koja daje grafičku reprezentaciju liste *History*. Za tu svrhu napisane su nekoliko pomoćnih funkcija. Funkcijom *cellc*Š*i,j,k*Ć konstruisana su dva jednostavna grafika.

Prvi grafik sadrži četiri spojene linije koje konstruišu kvadrat. To je učinjeno standardnom funkcijom *Line*:

```
LineŠšši,jć, ši+1,jć, ši+1,j+1ć, ši,j+1ć, ši,jććĆ
```

Drugi grafik je definisan funkcijom *Text*Š*k,ši+0.5,j+0.5ć*Ć i predstavlja tekst koji odgovara štampanom tekstu u formi *k*, centriran od pozicije određene sa š*i*+0.5, *j*+0.5ć i upisan u centar prvog grafičkog objekta.

```
cellcŠi_, j_, k_Ć := š
   LineŠšši,jć, ši+1,jć, ši+1,j+1ć, ši,j+1ć, ši,jććĆ,
   TextŠk, ši+0.5, j+0.5ćĆć
```

Funkcija *Headc*Š*i,j,s*Ć generiše jdnostavan grafik koji predstavlja listu dva elementa. Prvi element jednak je označen sa *s* i predstavlja grafičku naredbu koja određuje koji će grafički objekat biti prikazan, ako je moguće u datoj boji. Drugi element je ispunjeni pravougaonik, orjentisan paralelno osama, i određen parametrima *i* i *j*.

```
HeadcŠi_,j_,s_RGBColorĆ :=
 šs, RectangleŠši,j+1ć,ši+1,j+2ćĆć
```

U funkciji *line*Š*s,mp,p,MP,tape*Ć formalni parametri predstavljaju:

s - trenutno stanje,

mp, p i *MP* - minimalni indeks ,trenutni indeks i maksimalni indeks zauzetih ćelija, i

tape - lista koja odgovara globalnom nizu *TAPE*, a koja je generisana izrazom *TareToList*Š*TAPE*Ć.

Preciznije, funkcija *line*Š*s,mp,p,MP,tape*Ć koristi pet parametra, koji definišu aktuelno stanje mašine. Funkcija *line*Š*s,mp,p,MP,tape*Ć takođe generiše listu dva grafička objekta. Prvi od tih objekata generisan je funkcijom *Headc*Š*p,j++,Color*Š*s*ĆĆ, gde je početna vrednost za *j* jednaka 0. Drugi element je lista grafičkih objekata *cellc*Š*i,j,k*Ć*,* gde *i* uzima vrednosti iz *mp* do *MP* i tekst *k* je definisan u ovom slučaju na sledeći način:

Ako *i* nije jednako indeku aktuelne ćelije, onda se upisuje sadržaj trake *tape*ŠŠ*i-mp*+1ĆĆ u skaniranoj ćeliji; u suprotnom, upisuje listu š*tape*ŠŠ*i-mp*+1ĆĆ*,s*ć.

```
lineŠšs_, mp_, p_, MP_, tape_ćĆ :=
 š HeadcŠp, (j++), ColorŠsĆĆ,
   TableŠ cellcŠi,j,
    IfŠi!=p, tapeŠŠi-mp+1ĆĆ,
     štapeŠŠi-mp+1ĆĆ,sćĆ Ć, ši, mp, MPć
   Ćć
```

Rezultat funkcije *Color* je grafička naredba *RGBColor*. Na primer, može biti definisana kao sledeći niz dodeljivanja:

```
NeedsŠ"GraphicsžColorsž"Ć;
ColorŠ_Ć := Gray;
ColorŠAĆ = Blue;
ColorŠBĆ = Red;
ColorŠCĆ = Cyan;
ColorŠDĆ = Pink;
ColorŠEĆ = Magenta;
ColorŠFĆ = Yellow;
ColorŠGĆ = Green;
ColorŠHĆ = Brown;
```

Procedura *DumpHistory* daje tabelarnu reprezentaciju akcija Turingove mašine koje su smeštene u listu *History*. U ovoj proceduri primenjujemo funkciju *line* na svaki element koji je sadržan u listi određenoj povratnom listom *History*.

```
DumpHistory := (
 j = 0;
 ShowŠGraphicsŠline /Ž ReverseŠHistoryĆĆ,
     AspectRatio -> Automatic,
     ImageSize -> š600,600ćĆ      );
```

Primer 10.4. Razmotrimo skup pravila *T* kao u Primeru 10.2:

InŠ5Ć:=T=ššA,"0"ć->šA,"1",Rć,
šA,"1"ć->šA,"1",Rć,šA,"#"ć->šB,"#",Lććć

Pretpostavimo da je sadržaj trake definisan stringom "110101010". Koristimo izraz *ComputeŠT,"*110101010*"*Ć i proceduru *DumpHistory* da reprezentujemo istoriju Tjuringove mašine:

InŠ6Ć:= ComputeŠT, "110101010"Ć
DumpHistory;

Ispunjeni deo trake kao i aktuelna stanja reprezentovana su sledećom tabelom:

š1,Ać	1	0	1	0	1	0	1	0	#
1	š1,Ać	0	1	0	1	0	1	0	#
1	1	š0,Ać	1	0	1	0	1	0	#
1	1	1	š1,Ać	0	1	0	1	0	#
1	1	1	1	š0,Ać	1	0	1	0	#
1	1	1	1	1	š1,Ać	0	1	0	#
1	1	1	1	1	1	š0,Ać	1	0	#
1	1	1	1	1	1	1	š1,Ać	0	#
1	1	1	1	1	1	1	1	š0,Ać	#
1	1	1	1	1	1	1	1	1	š# ,Ać
1	1	1	1	1	1	1	1	š1,Gć	#

U ovoj tabeli, sadržaj skaniranog kvadrata je reprezentovan sa dvo-elementnom listom. Prvi element u takvoj reprezentaciji je sadržaj trake, dok je drugi element aktuelno stanje. Napomenimo da je stanje mašine takođe reprezentovana bojom koja je definisana funkcijom *Color*. Boje nisu predstavljene u tabeli. Takođe, u grafičkoj reprezentaciji datoj sa procedurom *DumpHistory* samo su obojeni kvadrati u nekom trenutku skanirani.

LITERATURA

[1] N. Blachman, *Mathematica: A Practical Approach*, Englewood Cliffs, New Jersey: Prentice-Hall, 1992.

[2] N. Cutland, *Computability: An Introduction to Recursive Functions Theory*, Cambridge University Press,Cambridge, London, Newyork, Sydney, 1986.

[3] M.D. Davies and E.J. Weyuker, *Computability, Complexity, and Languages*, Academic Press, New York, London, Paris, 1983.

[4] D. Gries, *Compiler Construction for Digital Computers*, John Wiley & Sons, Inc. New York, London, Sydney, Toronto 1971.

[5] R. Maeder, *Programming in Mathematica*, Third Edition, Redwood City, California: Adisson-Wesley 1996.

[6] J.P. Tremblay and P.G. Sorenson, *The Theory and Practice of Compiler Writing*, McGraw-Hill Book Company, New York,1985.

[7] S. Wolfram, *Mathematica: a System for Doing Mathematics by Computer*, Addison-Wesley Publishing Co, Redwood City, California, 1991.

[8] S. Wolfram, *Mathematica Book,* Version 3.0, Addison-Wesley Publishing Co, Redwood City, California, 1997.

11. IMPLEMENTACIJA DVOFAZNOG SIMPLEKS METODA

11.1. ODREĐIVANJE POČETNOG REŠENJA

Razmatramo problem linearnog programiranja

$$\begin{array}{ll} \text{(min)} & c^T x+d, \\ \text{p.o.} & Ax=b, \quad x\geq 0, \end{array} \qquad (11.1.1)$$

gde je matrica A iz prostora $R^{m\times(m+n)}$, $c\in R^{n+m}$ i sistem $Ax=b$, $x\in R^{n+m}$, $b\in R^m$ je potpunog ranga (rang(A)=m). Pretpostavljamo da je b=š$b_1,...,b_m$ć, $d\in R$ i c

= š$c_1,...,c_{m+n}$ć. Kanonički oblik problema (11.1.1) možemo zapisati u tabelarnom obliku

$x_{N,1}$	$x_{N,2}$		$x_{N,n}$	-1	
$-a_{11}$	$-a_{12}$	...	$-a_{nn}$	b_1	$-x_{B,1}$
$-a_{21}$	$-a_{22}$	...	$-a_{2n}$	b_2	$-x_{B,2}$
...	...	...	...	...	...
$-a_{m1}$	$-a_{m2}$	...	$-a_{mn}$	b_m	$-x_{B,m}$
c_1	c_2	...	c_n	$-d$	

(11.1.2)

gde je $x_{N,1},...\ ,\ x_{N,n}$ skup nebazičnih promenljivih i $x_{B,1},...,x_{B,m}$ su bazične promenljive. Transformisani koeficijenti matrice A (posle rešavanja sistema po bazičnim promenljivim) su označeni sa a_{ij}, zbog jednostavnosti. Ako je problem linearnog programiranja dat u opštem obliku on se prevodi u standardni oblik dodavanjem izravnavajućih promenljivih.
Ako sve nebazične promenljive $x_{N,1},...\ ,x_{N,n}$ izjednačimo sa nulom, tada je jednačinama $x_{B,1} = b_1,...\ ,x_{B,m} = b_m$ određeno bazično rešenje. Ako je $b_1 \geq 0$, ... ,$b_m \geq 0$, onda dobijamo odmah bazično dopustivo rešenje. Ako uslov $b \geq 0$ nije zadovoljen, problemu linearnog programiranja (11.1.1) pridružujemo dvofazni simpleks problem, definisan sa

$$\begin{array}{ll} \text{(min)} & e^T w, \\ \text{p.o.} & Ax+w = \text{b}, \\ & x \geq 0,\ w \geq 0, \end{array} \qquad (11.1.3)$$

gde je e = š1,...,1ć $\in R^m$ i $w \in R^m$ je vektor veštačkih promenljivih. Poznato je da ako je (x^*,w^*) optimalno rešenje za (11.1.3), tada potreban i dovoljan uslov da (11.1.1) ima dopustivo rešenje je $w_i^* = 0$, $i = 1,...,m$. Međutim, nedostatak ovog pristupa je uvođenje veštačkih promenljivih zbog čega je dimenzija problema (11.1.3) mnogo veća u odnosu na (11.1.1). Drugi pristup ovom problemu je opisan u Š14Ć. Razmatraćemo taj algoritam jer ne zahteva uvođenje veštačkih promenljivih i kako ne povećava dimenziju problema, pogodan je za primenu u kompjuterskim programima.
U nastavku ovog odeljka uvodimo modifikaciju dvofaznog simpleks metoda i proučavamo problem određvanja početnog bazičnog rešenja u simpleks algoritmu. Da bi ubrzali proces generisanja prvog bazično dopustivog rešenja, razmatramo dva problema:
- problem početnog izbora bazičnih i nebazičnih promenljivih i
- problem zamene bazičnih i nebazičnih promenljivih u opštem simpleks metodu.

Nakon toga opisujemo simboličku implementaciju dvofaznog simpleks metoda i modifikacije dvofaznog simpleks metoda u programskom paketu

MATHEMATICA. Na kraju odeljka navodimo ilustrativne numeričke primere i uporedno ispitujemo ponašanje algoritama na konkretnim primerima.

11.2. SIMBOLIČKA IMPLEMENTACIJA SIMPLEKS METODA

11.2.1. Uvod

Posmatra se problem linearnog programiranja u opštem obliku:

$$\text{(max)} \qquad Q(x)=Q(x_1,\dots,x_n)=\sum_{i=1}^{n} c_i x_i+d,$$

$$\text{p.o.} \quad : \quad \begin{aligned} &\sum_{k=1}^{n} p_{ik}x_k+r_i \le \sum_{k=1}^{n} q_{ik}x_k+s_i, \ i\in I_1,\\ &\sum_{k=1}^{n} p_{ik}x_k+r_i \ge \sum_{k=1}^{n} q_{ik}x_k+s_i, \ i\in I_2,\\ &\sum_{k=1}^{n} p_{ik}x_k+r_i == \sum_{k=1}^{n} q_{ik}x_k+s_i, \ i\in I_3,\\ &x_j\ge 0, \ j=1,\dots, n=I_1\cup I_2\cup I_3. \end{aligned} \tag{11.2.1}$$

Ovakav problem se predstavlja unutrašnjom reprezentacijom

$$\sum_{i=1}^{n} c_i x_i+d, \text{ š } \begin{aligned} &\sum_{k=1}^{n} p_{ik}x_k+r_i \le \sum_{k=1}^{n} q_{ik}x_k+s_i, \ i\in I_1,\\ &\sum_{k=1}^{n} p_{ik}x_k+r_i \ge \sum_{k=1}^{n} q_{ik}x_k+s_i, \ i\in I_2,\\ &\sum_{k=1}^{n} p_{ik}x_k+r_i == \sum_{k=1}^{n} q_{ik}x_k+s_i, \ i\in I_3, \text{ ć} \end{aligned} \tag{11.2.2}$$

Motivi za razvoj softvera za simboličku implementaciju simpleks metoda su sledeći:

A. Poboljšati implementacije simpleks metoda koje su napisane u proceduralnuim jezicima, što je dostupno u Š4Ć.

B. Poboljštai tačnost implementacije simpleks metoda koja je na raspolaganju u funkcijama *ConstrainedMin, ConstrainedMax* i *LinearProgramming*. Poznato je da linearno programiranje jeste osnova za mnoge probleme sadržane u matematičkom programiranju. Međutim, implementacija ovih metoda sa tačnošću 10^{-6} zahteva implementaciju simpleks metoda sa tačnošću koja je veća od 10^{-6}, što je nemoguće uraditi pomoću standardnih funkcija u *MATHEMATICA*.

S druge strane, implementacija problema linearnog programiranja (11.2.1) u proceduralnim programskim jezicima je dobro poznat problem. Mogu se sugerisati sledeće osnovne prednosti simboličke implementacije simpleks metoda Š11Ć:

(1L) Mogućnost selektovanja proizvoljne ciljne funkcije, koja nije definisana potprogramom. Takođe, moguće je da se definišu ograničenja u listi formalnih parametara implementatione procedure, bez suvišne leksičke i sintaksne analize.

(2L) Mogućnost simboličke obrade podataka u jeziku *MATHEMATICA* u transformaciji datih ograničenja zadatih nejednakostima u odgovarajuća ograničenja koja su zadata jednakostima, uvođenjem izravnavajućih promenljivih Š11Ć. U stvari, u radu Š11Ć je opisana primena simboličkog procesiranja u transformaciji opšteg linearnog programa u odgovarajuću standardnu formu.

(3L) Mogućnost jednostavne eliminacije zavisnih promenljivih u datom linearnom sistemu, kao i mogućnost jednostavne zamene zavisnih i nezavisnih promenljivih.

(4L) Postoje standardne funkcije u *MATHEMATICA* za rešavanje sistema linearnih jednačina.

Takođe, u odnosu na odgovarajuće ugrađene funkcije raspoložive u paketu *MATHEMATICA*, napominjemo sledeću prednost:

(5L) U priloženom softveru, linearno programiranje je implementirano sa proizvoljnom tačnošću.

11.2.2. Implementacija

Korak 1. Unutrašnja forma problema (11.2.1) sadrži dva različita dela. Prvi deo je proizvoljna linearna ciljna funkcija determinisana odgovarajućim izrazom u *MATHEMATICA*, koji je označen parametrom *cilj*. Drugi deo je lista ograničenja, koja se naziva *lisogr*. U odnosu na unutrašnju formu koja je potrebna u funkcijama *ConstrainedMin* i *ConstrainedMax*, izostavljena je lista promenljivih koje se korišćene u ciljnoj funkciji i datim ograničenjima. Međutim, ovakva unutrašnja forma je iskorišćena samo zbog jednostavnijeg korišćenja programa. Koristeći standardne funkcije *Variables* i *Union,* sledećom funkcijom *IzdvojPromenljiveŠlisĆ* je moguće izdvojiti listu promenljivih iz izraza koji su sadržani u datoj listi *lis*.

```
IzdvojPromenljiveŠlis_Ć :=
 ModuleŠšduz, lista=šćć,
   duz = LengthŠlisĆ;
   DoŠlista=UnionŠlista, VariablesŠlisŠŠiĆĆĆĆ,ši,duzćĆ;
   ReturnŠlistaĆ
 Ć
```

Sada se izrazom *prom=IzdvojPromenljiveŠAppendŠjed,lisogrĆĆ* izdvaja lista upotrebljenih promenljivih u problemu (11.2.1).
To predstavlja suštinu Prednosti (1L).

Korak 2. Proizvoljno ograničenje tipa nejednakost može se jednostavno transformisati u odgovarajuće ograničenje zadato jednakošću, dodavanjem ili oduzimanjem pozitivne izravnavajuće promenljive. Ovaj cilj se može dostići u sledećim koracima.

Korak 2.1. Za svako i=1,... ,m transformisati listu ograničenja

$$\sum_{k=1}^{n} p_{ik}x_k+r_i\le\sum_{k=1}^{n} q_{ik}x_k+s_i,\quad i\in I_1$$

$$\sum_{k=1}^{n} p_{ik}x_k+r_i\ge\sum_{k=1}^{n} q_{ik}x_k+s_i,\quad i\in I_2$$

$$\sum_{k=1}^{n} p_{ik}x_k+r_i==\sum_{k=1}^{n} q_{ik}x_k+s_i,\quad i\in I_3$$

u odgovarajuću listu ograničenje oblika

$$\begin{array}{ll}\text{š}\ \ \text{š}\ \sum_{k=1}^{n} p_{ik}x_k+r_i-(\sum_{k=1}^{n} q_{ik}x_k+s_i),\ \le\text{ć},\ i\in I_1 & \\ \quad\text{š}\ \sum_{k=1}^{n} p_{ik}x_k+r_i-(\sum_{k=1}^{n} q_{ik}x_k+s_i),\ \ge\text{ć},\ i\in I_2 & \qquad (11.2.3)\\ \quad\text{š}\ \sum_{k=1}^{n} p_{ik}x_k+r_i-(\sum_{k=1}^{n} q_{ik}x_k+s_i),\ ==\text{ć},\ i\in I_3\ \ \text{ć} & \end{array}$$

U tu svrhu je definisana sledeća funkcija:

```
SrediŠlis_Ć :=
 ModuleŠšlista=šć, gl, izrć,
   DoŠgl=HeadŠlisŠŠiĆĆĆ; izr=lisŠŠiĆĆ/.gl->Subtract;
      AppendToŠlista, ListŠizr,gl ĆĆ,
         ši,LengthŠlisĆć
   Ć;
   ReturnŠlistaĆ
```

Ć

Koristeći izraz *lista =SrediŠlisogr* lista datih ograničenja transformiše se u listu čiji su elementi oblika (11.2.3).

Korak 2.2. U sledećoj funkciji formalni parametar *lis* predstavlja neko ograničenje oblika (12.2.3), dok je prom neka pozitivna izravnavajuća promenljiva. Rezultat je leva strana ekvivalentnog ograničenja tipa jednakost, koja je jednog od sledećih oblika:

$$\sum_{k=1}^{n} p_{ik}x_k + r_i - \left(\sum_{k=1}^{n} q_{ik}x_k + s_i\right) - prom, \;\; i\in I_{1,}$$

$$\sum_{k=1}^{n} p_{ik}x_k + r_i - \left(\sum_{k=1}^{n} q_{ik}x_k + s_i\right) + prom, \;\; i\in I_2,$$

$$\sum_{k=1}^{n} p_{ik}x_k + r_i - \left(\sum_{k=1}^{n} q_{ik}x_k + s_i\right), \; i\in I_3.$$

```
DodajPromenljivuŠlis_, prom_Ć :=
 ModuleŠšć,
   SwitchŠlisŠŠ2ĆĆ,
      LessEqual, ReturnŠPlusŠlisŠŠ1ĆĆ,promĆĆ,
      GreaterEqual, ReturnŠSubstractŠlisŠŠ1ĆĆ,promĆĆ,
      Equal, ReturnŠlisŠŠ1ĆĆĆ
 Ć  Ć
```

Korak 2.3. Za svako i=1,... ,m u glavnoj funkciji, transformisati i-to zadato ograničenje dodavanjem ili oduzimanjem pozitivne izravnavajuće promenljive $ŠiĆ.

```
lista=SrediŠlisogrĆ;
 DoŠitajed=DodajPromenljivuŠlistaŠŠiĆĆ, $ŠiĆ Ć;
     AppendToŠsistem,EqualŠitajed,0ĆĆ,
     ši,LengthŠlisogrĆć
Ć;
```

Ovim je opisana prednost (2L).

Korak 3. Rešiti sledeći standardni linearni program:

$$\text{(max)} \qquad Q(x) = \mathrm{Q}(x_1,\dots, x_{\mathrm{n}}) = \sum_{i=1}^{n} c_i x_i - d,$$

p.o.:

$$\sum_{k=1}^{n} a_{ik}x_k - b_i - \text{\$Š}i\text{Ć} == 0,\ i\in I_1$$

$$\sum_{k=1}^{n} \mathrm{a_{ik}x_k} - b_i + \text{\$Š}i\text{Ć} == 0,\ i\in I_2$$

$$\sum_{k=1}^{n} \mathrm{a_{ik}x_k} - b_i == 0,\ i\in I_3.$$

Početni izbor bazičnih i nebazičnih promenljivih rešen je upotrebom funkcije *Solve* iz *MATHEMATICA*. Neka je $x_{\mathrm{N},1},\ldots ,x_{\mathrm{N,n}}$ lista nebazičnih promenljivih i $x_{\mathrm{B},1},\ldots,x_{\mathrm{B,m}}$ lista bazičnih promenljivih. Zbog suštine funkcije *Solve*, proizvoljno *i*-to ograničenje u generisanoj simpleks tabeli nije oblika $\sum a_{ij}x_{\mathrm{N,j}}-b_i=-x_{\mathrm{B},i}$, koji se koristi u algoritmima koji su opisani u Š11Ć, Š14Ć, već oblika $\sum (-a_{ij}x_{\mathrm{N,j}} + b_i) = x_{\mathrm{B},i}$, $i=1,\ldots , m$. Ova modifikacija zahteva da se inkorporira nekoliko modifikacija u klasičnim algoritmima maksimizacije, koji su opisani u Š11Ć, Š14Ć.

Korak S1. U slučaju $b_1,\ldots ,b_m \geq 0$, odgovarajuće rešenje se naziva *bazično dopustivo rešenje* (*basic feasible solution*), i potrebno je izvršiti korake S1A-S1D. Inače, izvršiti Korak S2.

Korak S1A. Ako je $c_1,\ldots ,c_{\mathrm{n}} \leq 0$, tada je bazično rešenje istovremeno i optimalno. Rezultat je lista š$Q(x)$, š$x_1, \ldots ,x_n$ćć u kojoj je $x=(x_1,\ldots ,x_{\mathrm{n}})$ i $x_{\mathrm{N},1}=0,\ldots ,x_{\mathrm{N,n}}=0$, $x_{\mathrm{B},1}=b_1,\ldots ,x_{\mathrm{B},m}=b_m$. Inače preći na sledeći korak.

Korak S1B. Izaberi maksimalno $c_j>0$. (Selekcija prvog c_j rešava problem cikliranja).

Korak S1C. Ako je $a_{1j},\ldots ,a_{mj} \geq 0$, tada *STOP*. Maksimum je $+\infty$.

Korak S1D. Izračunati

$$\min_{1<=i<=m} \text{š}\ \frac{b_i}{-a_{ij}}\ ,\ \mathrm{a_{ij}}<0\text{ć} = \frac{b_p}{-a_{pj}}$$

i zameniti bazičnu i nebazičnu promenljivu $x_{\mathrm{N},j}$<->$x_{\mathrm{B},p}$.

Korak S2. Selektovati poslednji $b_i<0$.

Korak S3. Ako je $a_{i1},\ldots ,a_{in}\leq 0$ tada prekinuti algoritam. Linearni program ne može biti rešen.

Korak S4. Izračunati

$$\min_{k>i} š \frac{b_i}{-a_{ij}} , a_{ij}<0 ć \cup š \frac{b_k}{-a_{kj}} ć = \frac{b_p}{-a_{pj}}$$

i zameniti bazičnu i nebazičnu promenljivu x_j<->x_p.

Korak S5. Ako je b_1,... ,$b_m \geq 0$ preći na Korak S1, inače preći na Korak S2.

Implementacija koraka Korak S1-S4 opisana je kako sledi.

Korak 3.1. Napomenimo da su u glavnoj funkciji izvršene sledeće inicijalizacije:

```
prom=IzdvojPromenljiveŠAppendŠjed,ciljĆĆ;
res1=SolveŠsistemĆ;  r=LengthŠFirstŠres1ĆĆ;
zavprom=TakeŠprom,rĆ;  nezprom=ComplementŠprom,zavpromĆ;
```

Funkcija *Solve* demonstrira svojstvo (4L) simboličke implementacije. U toku implementacije koraka S1A-S1D koriste se sledeći formalni parametri:
*sistem_*: lista datih ograničenja;
zavprom_, nezprom_: lista bazičnih i nebazič promenljivih, respektivno;
fja_: ciljna funkcija.

Korak 3.2. Implementacija Koraka S1.
Sledećom funkcijom se izdvajaju slobodni članovi sadržani u datom izrazu *izr*.

```
SlClanŠizr_Ć :=
 ModuleŠšclan = izr, promć,
   prom = IzdvojPromenljiveŠizrĆ;
   DoŠclan=clan/. promŠŠiĆĆ->0, ši,LengthŠpromĆćĆ;
   ReturnŠclanĆ
 Ć
```

Funkcija *Solve* daje rešenje u matričnoj formi ššx_1->v_1ćć, ... ššx_n->v_nćć.
Funkcijom *SrediResenje* ova lista se transformiše u pogodniji oblik
ššx_1, v_1ćć, ... ššx_n, v_nćć.

```
 SrediResenjeŠlisprav_Ć:=
   ModuleŠšduz,tab=šćć,
       duz=LengthŠlispravĆ;
       DoŠAppendToŠtab, lispravŠŠiĆĆ/.Rule->ListĆ,
           ši,duzćĆ;
       ReturnŠtabĆ
  Ć
```

Sada se implementacija Koraka S1 može opisati na sledeći način.

```
(* Implementacija Koraka S1A *)
```

```
    res=FirstŠSolveŠsistem,zavpromĆĆ; (* Prednost 4L*)
    vrf=fja/.res;
    DoŠc=CoefficientŠvrf,nezpromŠŠiĆĆĆ;
      IfŠc>0,AppendToŠcoef,ListŠnezpromŠŠiĆĆ,cĆĆĆ,
        ši,LengthŠnezpromĆć
    Ć; (* coef=ššx_{N,i},c_ićđ c_i>0, i=1,...,nć *)
    IfŠcoef==šć,      (*then*)
       (* c_1,... ,c_n≤0  *)
     DoŠvrf=vrf/.nezpromŠŠiĆĆ->0, ši,LengthŠnezpromĆćĆ;
     DoŠ DoŠresŠŠjĆĆ=resŠŠjĆĆ/.nezpromŠŠiĆĆ->0,
           ši,LengthŠnezpromĆć  (* x_{N,i}=0,i=1,...,n *)
        Ć,            šj,LengthŠresĆć
     Ć; (* Generise res=šx_{B,i}->b_i,i=1,...,mć  *)
     DoŠAppendToŠres,RuleŠnezpromŠŠiĆĆ,0ĆĆ,
        ši,LengthŠnezpromĆć
     Ć;
     (* res=šx_{B,i}->b_i,i=1,... ,m,x_{N,i}->0, i=1,... ,nć*)
     del=PositionŠres,$Š_Ć->_Ć; res=DeleteŠres,delĆ;
       (* Brisanje slack varijabli $ŠiĆ,i∈I_1∪I_2 *)
     ReturnŠ ListŠvrf, SortŠresĆ, FalseĆ
          (* Vrati rezultat i False *)
     Ć,     (*else*)
(* Implementacija Koraka S1B *)
     max=coefŠŠ1,2ĆĆ; poz=1;
     ForŠi=2, i<=LengthŠcoefĆ, i++,
        IfŠcoefŠŠi,2ĆĆ>max, max=coefŠŠi,2ĆĆ; poz=iĆ;
     Ć;  (* max=maxšc_j>0, j=1,...,nć *)
     zp=coefŠŠpoz,1ĆĆ;   (* zp=x_šN,maxć *)
(* Implementacija Koraka S1C *)
     prav=SrediResenjeŠresĆ; param=šć;
     (* Nađi a_ij<0, 1≤i≤m *)
     DoŠ c=CoefficientŠpravŠŠi,2ĆĆ,zpĆ;  (* c=a_ij *)
        b=SlClanŠpravŠŠi,2ĆĆĆ;        (* b=b_i*)
        IfŠc<0 && b!=0,
           AppendToŠparam,ListŠpravŠŠi,1ĆĆ,b,-cĆĆ
        Ć,            ši,LengthŠpravĆć
     Ć;  (* param=ššx_{B,i},b_i,-a_ijćđ a_ij<0ć *)
     IfŠparam==šć,      (*then*)
        (* a_1j,... ,a_mj≥0 *)
      ReturnŠ ListŠ+Infinity, FalseĆ
      Ć,                (*else*)
(* Implementacija Koraka S1D *)
      np=paramŠŠ1,1ĆĆ;
      minkol=paramŠŠ1,2ĆĆ/paramŠŠ1,3ĆĆ;
```

```
    DoŠkol=paramŠŠi,2ĆĆ/paramŠŠi,3ĆĆ;
      IfŠkol<minkol,minkol=kol;np=paramŠŠi,1ĆĆĆ,
         ši,2,LengthŠparamĆć
    Ć;
    ReturnŠListŠReplacePartŠzavprom,zp,
        FirstŠFirstŠPositionŠzavprom,npĆĆĆ
            Ć,    True
  Ć Ć     Ć
```

Korak 4. Koraci S2-S4 su implementirani kako sledi. Koristeći standardnu funkciju *Solve* moguće je da se reši dati sistem linearnih jenačina (označen sa sistem) u odnosu na bazične promenljive *zavprom*:
tab=FirstŠSolveŠsistem,zavpromĆĆ;
U glavnoj funkciji koristi se lista *tab=SrediResenje*Š*tab*Ć*;*
Ovim je opisana prednost (3L).

```
(* Implementacija Koraka S2 *)
tab=SrediResenjeŠFirstŠSolveŠsistem,zavpromĆĆĆ;
duz=LengthŠtabĆ;  indb=0;
DoŠIfŠSlClanŠtabŠŠi,2ĆĆĆ<0, indb=i;np=tabŠŠi,1ĆĆĆ,
     ši,1,duzć
Ć;
IfŠindb==0,   (* Basicno resenje postoji *)
 ReturnŠListŠsistem,zavprom,nezprom,FalseĆĆ,
 (* Basično resenje ne postoji *)
 zp=0;
 DoŠIfŠCoefficientŠtabŠŠindb,2ĆĆ,nezpromŠŠiĆĆ Ć>0,
   zp=nezpromŠŠiĆĆĆ,      ši,1,LengthŠnezpromĆć
 Ć;
(* Implementacija Koraka S3 *)
 IfŠzp==0,ReturnŠListŠ"Program nije resiv",0ĆĆ,
(* Implementacija Koraka S4 *)
 IfŠindb==duz,          (*then*)
  np=tabŠŠduz,1ĆĆ,      (*else*)
  np=tabŠŠindb,1ĆĆ;
  mn=SlClanŠtabŠŠindb,2ĆĆĆ/-CoefficientŠtabŠŠindb,2ĆĆ,zpĆ;
  DoŠv=SlClanŠtabŠŠi,2ĆĆĆ/-CoefficientŠtabŠŠi,2ĆĆ,zpĆ;
     IfŠv<mn, mn=v; np=tabŠŠi,1ĆĆĆ,      ši,indb,duzć
Ć    Ć Ć  Ć; ć
```

Zamena *np*=x_j<->x_p=*zp* u Korak 4 može se implementirati na sledeći način:

```
newzavprom=UnionŠComplementŠzavprom,nplĆ,ListŠzpĆĆ;
newnezprom=UnionŠComplementŠnezprom,zplĆ,ListŠnpĆĆ;
```

Ovim je opisan jedan deo prednosti (3L).
Ciklus simpleks metoda je implementiran u glavnoj funkciji:

```
SimpleksMaxŠcilj_,lisogr_Ć:=
ModuleŠšlista,prom,r,sistem=šć,jed=šć,itajed,radi,
       zavprom,nezprom,inic,res1,bit=0ć,
  lista=SrediŠlisogrĆ;
  DoŠitajed=DodajPromenljivuŠlistaŠŠiĆĆ, ĐŠiĆ Ć;
     AppendToŠsistem,EqualŠitajed,0ĆĆ;
     AppendToŠjed,itajedĆ,
       ši,LengthŠlisogrĆć
  Ć;
  prom=IzdvojPromenljiveŠAppendŠjed,ciljĆĆ;
  res1=SolveŠsistemĆ;
  IfŠres1==šć,ReturnŠ"Proverite ogranicenja!"Ć Ć;
  res1=FirstŠres1Ć;  r=LengthŠres1Ć;  zavprom=šć;
  DoŠAppendToŠzavprom,res1ŠŠi,1ĆĆĆ,ši,rćĆ;
    (* zavprom=TakeŠprom,rĆ; *)
  nezprom=ComplementŠprom,zavpromĆ;
  PrintŠ"zavprom= ",zavprom,"nezprom= ",nezpromĆ;
  radi=True; inic=ListŠsistem,zavprom,nezprom,radiĆ;
  WhileŠradi==True,
     inic=PreSimŠinicŠŠ1ĆĆ,inicŠŠ2ĆĆ,inicŠŠ3ĆĆ Ć;
     radi=LastŠinicĆ;
     bit++; PrintŠ"Broj iteracijaPreSim = ",bitĆ;
  Ć;
  IfŠradi==False,
    zavprom=inicŠŠ2ĆĆ; nezprom=inicŠŠ3ĆĆ;radi=True;
    WhileŠradi==True,
      inic=SimMaxŠsistem,zavprom,nezprom,ciljĆ;
      radi=LastŠinicĆ;  bit++;
      IfŠradi,
         zavprom=inicŠŠ1ĆĆ;
         nezprom=ComplementŠprom,zavpromĆ
  ĆĆ   Ć;
  ReturnŠDropŠinic,-1ĆĆ
Ć
```

Sledi opis prednosti (5L). Poznato je da *MATHEMATICA* radi sa proizvoljnom tačnošću jedino sa celim brojevima. Prema tome, prirodan način da se dostigne proizvoljna preciznost je sledeći algoritam.

Korak A. Transformisati koeficiene zadatih ograničenja i u ciljnoj funkciji u cele brojeve, množeći svaki od njih brojem 10^p, gde je p maksimalni broj decimalnih cifara koje su upotrebljene u ciljnoj funkciji i ograničenjima. lista decimalnih cifara može se konstruisati pomoću funkcije *RealDigits* u *MATHEMATICA*.

Korak B. Primeniti gore opisane funkcije na transformisanim ograničenjima i ciljnoj funkciji.

Korak C. Podeliti dobijenu ekstremnu vrednost ciljne funkcije i optimalne vrednosti upotrebljenih promenljivih sa 10^p.

LITERATURA

[1] E.D. Andersen, J. Gondzio, C. Meszaros and X. Xu, *Implementation of interior point methods for large scale linear programming*, Technical report, HEC, Universite de Geneve, 1996.

[2] M.D. Ašić and V.V. Kovačević-Vujčić, *Ill-conditionednes and interior-point methods*, Technical Report 901-98, Laboratory of Operations Research, Faculty of Organizational Sciences, November 1998.

[3] M.A. Bhatti, *Practical optimization methods*, Springer, New York, 2000.

[4] R. Bixby, *Implementing the simplex method*; the initial basis, ORSA Journal on Computing, **4** (1992), 267—284.

[5] J. Czyzyk, S. Mehrotra and S.J. Wright, *PCx User Guide*, Optimization Technology Center, Technical Report 96/01, 1996.

[6] G.B. Dantzig, *Lineare Programmierung und Erweiterungen*, Springer-Verlag Berlin, Heidelberg -New York, 1966.

[7] J. Gondzio, *HOPDM (Version 2.12): A fast LP solver based on a primal-dual interior point method*, European Journal of Operations Research, **85** (1995), 221—225.

[8] P.S. Stanimirović, M.B. Tasić and M. Ristić, *Symbolic implementation of the Hooke-Jeeves method,* YUJOR, **9** (1999), 285—301.

[9] N. Stojković and P.S. Stanimirović, *On the elimination of excessive constraints in linear programming, SYMOPIS*, Beograd, November 4-6 (1999), 207—210.

[10] N. Stojković and P.S. Stanimirović, *Two direct methods in linear programming,* European Journal of Operational Research, **131** (2001), 417—439.

[11] P.S. Stanimirović and I.B. Stanković, *Symbolic implementation of simplex method,* XIII Conference on Applied Mathematics, Budva, 25.05.-29.05. 1998.

[12] O. Todorović, *Operaciona istraživanja*, Prosveta, Niš, 1992.

[13] R.J. Vanderbei, *LOQO: an interior point code for quadratic programming, Statistics and Operations Research*, Princeton University, Technical Report SOR-94-15, 1998.

[14] S. Vukadinović i S. Cvejić, *Matematičko programiranje*, Univerzitet u Prištini, Priština 1996, 1983.

[15] S. Wolfram, *Mathematica: a system for doing mathematics by computer*, Addison-Wesley Publishing Co, Redwood City, California, 1991.

[16] S. Wolfram, *Mathematica Book*, *Version 3.0*, Wolfram Media and Cambridge University Press, 1996.

[17] S.J Wright, *Primal-dual interior-point methods*, SIAM, Philadelphia, 1997.

[18] Y. Zhang, *Solving large-scale linear programs by interior-point methods under the MATLAB environment*, Optimization Methods and Software, **10** (1998), 1-31.

12. IMPLEMENTACIJA UNARNIH PAR-FUNKCIJA U *MATHEMATICA*

12.1. UVOD

Implementacioni detalji za semigrupe unarnih funkcija su proučene u Š6Ć. Unarne funkcije operišu nad parom $<a,b>$ i imaju parove kao rezultate. Iz tog razloga, unarne funkcije se takođe zovu unarne parfunkcije. U Š6Ć i Š7Ć je razmatrana algebra sa binarnim operacijama <,>, dve unarne operacije L, R i sa sledeće tri aksiome:

$$A_1: L(<a,b>)= a;$$
$$A_2: R(<a,b>)= b;$$
$$A_3: <L(x),R(\mathrm{x})> = x.$$

Par funkcija je funkcija ove algebre u samu sebe, koja može biti definisana upotrebom <,>, L i R. U Š6Ć je dokazano da semigrupa zadovoljava tačno 69 dobijenih jednakosti, predstavljenih u Š6Ć i Š7Ć, je izomorfna semigrupi par funkcija. U Š6Ć je korišćen jezik *MAPLE* za implementaciju unarne par funkcije i u automatskoj verifikaciji jednakosti koje su potrebne za opisivanje semigrupa.

Opisaćemo odgovarajuće tipove podataka i programski jezik za implementaciju unarne parfunkcije. Glavna ideja je razviti odgovarajući model za semigrupu unarne parfunkcije. Za implementaciju je izabran programski jezik *MATHEMATICA*, a za odgovarajuće tipove podataka se koriste liste. Liste, kao često korišćene i veoma prikladne objekte u *MATHEMATICA*, su analogne parovima. Parovi su implementirani kao liste u odgovarajućoj formi A_3. Naš model funkcije *anglebracket*Š*a,b*Ć, koja predstavlja par <*a,b*>, je proširenje ugrađene *MATHEMATICA* funkcije *List*Š*a,b*Ć, koja se razmatra kao <*L*Š*a*Ć,*R*Š*a*Ć> i transformiće u *a*. Funkcije *L*Š*x*Ć i *R*Š*x*Ć su slične ugrađenim funkcijama *First*Š*x*Ć i *First*Š*Rest*Š*x*ĆĆ iz *MATHEMATICA*. Kako su parfunkcije definisane samo kao parovi <,> za funkcije *L* i *R*, funkcije *anglebracket*Š*a,b*Ć, *L*Š*a*Ć i *R*Š*a*Ć pronalaze semigrupe unarne parfunkcije. U tom slučaju, mi pokazujemo da je *MATHEMATICA* odgovarajuća za pronalačenje unarne parfunkcije. Takođe, *MATHEMATICA* je prihvatljiva i zbog simboličkog izračunavanja. Razvijeno je i nekoliko funkcija za verifikaciju 69 jednakosti koje generišu semigrupu unarnih parfunkcija. Ove jednakosti su predstavljene u Š6Ć and Š7Ć. Ovi rezultati su publikovani u Š5Ć.

12.2. IMPLEMENTACIJA PAR-FUNKCIJA

Parovi u formi <*x,y*> su implementirani pomoću listi, najfleksibilnijih i najjačih objekata u *MATHEMATICA*. Preciznije, posmatrani par <x,y> je implementiran kao lista š*x,yć*, koja ima odgovarajuća svojstva, izložena Aksiomom A_3. Još preciznije, moramo uključiti simplificaciju pravila, koja par <LŠ*a*Ć,RŠ*a*Ć> prevode u *a*.

$$< x, y >= \begin{cases} a, & x = L[a],\ y = R[a], \\ \{x, y\}, & x \neq L[a] \ ili \ y \neq R[a] \end{cases}$$

Proizvoljan par <*x,y*> je impletiran funkcijom *anglebracket*Š*x,y*Ć. Namena funkcije *anglebracket*Š*x,y*Ć je da izgradi listu š*x,yć*, ili ako su uslovi Aksiome A_3 ispunjeni, da izvrši odgovarajuće simplifikacije. Prema tome, ova funkcija je definisana kao sledeća modifikacija standardne funkcije *List*Š*x,y*Ć:

$$\text{anglebracketŠx,yĆ} = \begin{cases} a, & x = L[a],\ y = R[a], \\ List[x, y], & x \neq L[a] \ ili \ y \neq R[a] \end{cases}$$

$$= \begin{cases} a, & x = L[a],\ y = R[a], \\ \{x, y\}, & x \neq L[a] \ ili \ y \neq R[a] \end{cases}$$

Odgovarajuća funkcija je definisana na sledeći način:

```
anglebracketŠx_,y_Ć:=
 IfŠHeadŠxĆ===L && HeadŠyĆ===R && FirstŠxĆ===FirstŠyĆ,
    FirstŠxĆ,
    ListŠx,yĆ
```

Ć

Posmatrajmo, na primer, sledeći dijalog sa interpretatorom kojim se ilustruje funkcija *anglebracket*:

InŠ1Ć:= anglebracketŠa,bĆ

OutŠ1Ć= ša,bć

InŠ2Ć:= anglebracketŠanglebracketŠa,bĆ,cĆ

OutŠ2Ć= šša,bć,cć

InŠ3Ć:= anglebracketŠ*L*Š*x*Ć,*L*Š*x*ĆĆ

OutŠ3Ć= x

InŠ4Ć:= anglebracketŠ*L*Š*x*Ć,*L*Š*x*ĆĆ

OutŠ4Ć= š*L*Š*x*Ć,*L*Š*x*Ćć

Kada se liste koriste u modelovanju parova, tada funkci *L*Š*x*Ć i *L*Š*x*Ć mogu biti implementirane kao modifikacije standardnih funkcija *First*Š*x*Ć i *First*Š*Rest*Š*x*ĆĆ. Funkcija *L*Š*x*Ć je definisana kao sledeća modifikacija standardne funkcije First:

$$L\check{S}x\acute{C}= \begin{cases} First[x]=a, \quad x=\{a,b\}=List[a,b], \\ \qquad\qquad L[x], \quad x\neq\{a,b\} \end{cases}$$

Preciznije, ako je argument *x* lista, tada je vrednost *L*Š*x*Ć jednaka *First*Š*x*Ć; inače, rezultat je simbolički izraz *L*Š*x*Ć. Odgovarajuća funkcija u *MATHEMATICA* je:

LŠx_Ć:= FirstŠxĆ/; HeadŠxĆ===List

Ekvivalent funkcije *R*Š*x*Ć u *MATHEMATICA* je definisan sledećom modifikacijom funkcije FirstŠRestŠxĆĆ=PartŠx,2Ć:

$$L\check{S}x\acute{C}= \begin{cases} b=First[\mathrm{Re}\,st[x]], \quad x=\{a,b\}=List[a,b], \\ \qquad\qquad L[x], \quad x\neq\{a,b\} \end{cases}$$

Odgovarajuća funkcija je

RŠx_Ć:= FirstŠRestŠxĆĆ/; HeadŠxĆ===List

Sledi verifikacija aksioma A_1-A_3. Aksiome A_1 i A_2 mogu biti verifikovane koristeći definicije funkcija *L*Š*x*Ć, RŠ*x*Ć i *anglebracket*Š*x*,*y*Ć:

LŠ<x,y>Ć = LŠanglebracketŠx,yĆĆ

$$= \begin{cases} \qquad L[a], \quad x=L[a], y=R[a], \\ First[\{x,y\}], \quad x\neq L[a] \ ili \ y\neq R[a] \end{cases}$$

$$= \begin{cases} x, & x = L[a], y = R[a], \\ x, & x \neq L[a] \ ili \ y \neq R[a] \end{cases} = x.$$

RŠ<x,y>Ć= RŠanglebracketŠx,yĆĆ

$$= \begin{cases} R[a], & x = L[a], y = R[a], \\ First[\mathrm{Re}\,st[\{x, y\}]], & x \neq L[a] \ ili \ y \neq R[a] \end{cases}$$

$$= \begin{cases} y, & x = L[a], y = R[a], \\ y, & x \neq L[a] \ ili \ y \neq R[a] \end{cases} = y.$$

Aksioma A_3 sledi iz definicije funkcije *anglebracket*Š*x,y*Ć, i predstavlja uopštenje sledeće činjenice, koja proizilazi iz definicija funkcija *First* i *Rest*: za svaku listu x je x=*List*Š*First*Š*x*Ć,*First*Š*rest*Š*x*ĆĆĆ. Argument Waldovih funkcija *L*(*x*) i *R*(*x*) jesu promenljive čija se imena zadaju malim slovima ili parovi takvih varijabli, koji su konstruisani operatorom <,>. Prema tome, argumenti funkcija *L*Š*x*Ć i *L*Š*x*Ć, koje su napisane u *MATHEMATICA*, jesu promenljive čija su imena zadata malim slovima ili dvo-elementne liste takvih promenljivih, koje su definisane pomoću funkcije *List*ŠĆ. Prema tome, prirodno je da se posmatraju dva slučaja u toku verifikacije Aksiome A_3. U prvom slučaju, x predstavlja proizvoljnu promenljivu zadatu malim slovom. Tada možemo pisati *x*=*a*, gde je *a* nepoznata promenljiva. Dobijamo sledeće:

<*L*(*x*),*R*(*x*)> = *anglebracket*Š*L*Š*a*Ć,*R*Š*a*ĆĆ = *a* = *x*,

što predstavlja verifikacju Aksiome A_3 za ovaj slučaj.

U drugom slučaju se pretpostavlja da x predstavlja proizvoljnu listu oblika x=š*a*,*b*ć, gde su *a* i *b* nepoznate promenljive ili parovi promenljivih zadatih malim slovima.
Aksioma A_3 se može verifikovati i u tom slučaju:

<*L*(*x*),*R*(*x*)>= *anglebracket*Š*L*Š*x*Ć, *L*Š*x*ĆĆ=
*anglebracket*Š*a*, *b*Ć = š*a*,*b*ć = *x*.

Parfunkcije *I*,*S*,*B*,*M*,*D*,*U*,*C*,*E*,*A* i *Q* su defiisane sledećim funkcijama. S obzirom da imena *C*,*D*,*E* imaju posebno značenje u *MATHEMATICA* (vidi Š8Ć,Š9Ć), moraju se ukloniti svi atributi vezani za imena *C*,*D* i *E* izrazom

UnprotectŠC,D,EĆ

Takođe, s obzirom da se atributi kompleksne jedinice $I = \sqrt{-1}$ ne mogu biti promenjeni, koristimo ime i za funkciju *I*.

```
 iŠx_Ć:=x
SŠx_Ć:= anglebracketŠ LŠxĆ,LŠxĆ Ć
BŠx_Ć:= anglebracketŠ anglebracketŠ LŠxĆ, LŠLŠxĆĆ Ć,
                 RŠLŠxĆĆ     Ć
MŠx_Ć:= anglebracketŠ LŠLŠxĆĆ, LŠxĆ Ć
DŠx_Ć:= anglebracketŠ x,x Ć
UŠx_Ć:= anglebracketŠ
          anglebracketŠ RŠLŠxĆĆ, LŠLŠxĆĆ Ć,
          LŠxĆ
        Ć
CŠx_Ć:= anglebracketŠ
         anglebracketŠ
            anglebracketŠ LŠLŠxĆĆ, LŠRŠLŠxĆĆĆ Ć,
            RŠRŠLŠxĆĆĆ
         Ć,
         LŠxĆ
        Ć
EŠx_Ć:= anglebracketŠ
         anglebracketŠ LŠxĆ, LŠxĆ Ć,
         LŠxĆ   Ć

AŠx_Ć:= anglebracketŠ LŠLŠxĆĆ,
          anglebracketŠ RŠLŠxĆĆ, LŠxĆ Ć
        Ć
QŠx_Ć:= anglebracketŠ
          anglebracketŠ
             LŠLŠLŠxĆĆĆ,
             anglebracketŠ RŠLŠLŠxĆĆĆ, RŠLŠxĆĆ Ć
          Ć,
          LŠxĆ
        Ć
```

Ove funkcije su ilustrovane u sledećem dijalogu:

InŠ1Ć:= SŠaĆ

OutŠ1Ć=šRŠaĆ,LŠaĆć

InŠ2Ć:= SŠša,bćĆ

OutŠ2Ć=šb,ać

InŠ3Ć:= BŠaĆ

OutŠ3Ć=ššLŠaĆ,LŠRŠaĆĆć,RŠRŠaĆĆćć

InŠ4Ć:= BŠša,šb,cććĆ

OutŠ4Ć=šša,bć,cć

InŠ5Ć:= MŠaĆ

OutŠ5Ć=šLŠLŠaĆĆ,RŠaĆć

InŠ6Ć:= MŠšša,bć,cćĆ

OutŠ6Ć=ša,cć

InŠ7Ć:= DŠaĆ
OutŠ7Ć=ša,ać
InŠ8Ć:= UŠaĆ
OutŠ8Ć=šššRŠLŠaĆĆ,LŠLŠaĆĆć,šRŠaĆć
InŠ9Ć:= UŠšša,bć,cćĆ
OutŠ9Ć=ššb,ać,cć
InŠ10Ć:= CŠaĆ
OutŠ10Ć=šššLŠLŠaĆĆ,LŠRŠLŠaĆĆĆć,RŠRŠRŠLŠaĆĆĆĆć,RŠaĆć
InŠ11Ć:= CŠšša,šb,cćć,dćĆ
OutŠ11Ć=šššša,bć,cć,dć
InŠ12Ć:= EŠaĆ
OutŠ12Ć=ššLŠaĆ,LŠaĆć,šRŠaĆćć
InŠ13Ć:= EŠša,bćĆ
OutŠ13Ć=ššša,ać,bć
InŠ14Ć:= AŠaĆ
OutŠ14Ć=šLŠLŠaĆĆ, šRŠLŠaĆĆ,RŠaĆćć
InŠ15Ć:= AŠšša,bć,cćĆ
OutŠ15Ć=ša,šb,cćć
InŠ16Ć:= QŠaĆ
OutŠ16Ć=ššLŠLŠLŠaĆĆĆ, šRŠLŠLŠaĆĆĆ,RŠLŠaĆĆćć,RŠaĆć
InŠ17Ć:= QŠššša,bć,cć,dćĆ
OutŠ17Ć=šša,šb,cćć,dć

12.3. VERIFIKACIJA RELACIJA SEMIGRUPE

U ovoj sekciji se proučavaju funkcije koje mogu da se koriste u strukturnoj analizi semigrupe. Potrebno je da se reši problem sledećeg tipa: Data je semigrupa *S* i lista osobina *P*, šta se može reći u vezi ispunjenja osobina *P* u *S*? Lista od 69 jednačina koji pu potpunosti opisuju semigrupu unarnih parfunkcija date je u Š6Ć i Š7Ć. Ovde su date dve funkcije, myapply i eqtest, za verifikaciju ovih jednačina. Umesto funkcije *word2comp*, koja gradi string koji sadrži imena parfunkcija iz kompozicije, koristi se funkcija *myapply*Š*x*_Ć. Formalni parametar *x* predstavlja sekvencu imena unarnih parfunkcija. Rezultat ove funkcije je lista koja je generisana uzastopnim primenama ovih parfunkccija na nepoznatoj promenljivoj, koja je označena sa *arg*.

```
myapplyŠx_Ć:=ModuleŠšstr,d,i,res=argć,
  str=ToStringŠxĆ;  d=StringLengthŠstrĆ;
  DoŠres=ApplyŠToExpressionŠStringTakeŠstr,š-i,-ićĆĆ,
            ListŠresĆ
       Ć,
       ši,1,dć
  Ć;
```

ReturnŠresĆ
Ć

Koristeći funkciju *myapplyŠxĆ*, definisana je funkcija *eqtestŠx_,y_Ć* koja se koristi za verifikaciju individualnih jednačina iz Š6Ć i Š7Ć. Formalni parametri *x* i *y* označavaju levu i desnu stranu odabrane jednačine, respektivno.

eqtestŠx_,y_Ć:=IfŠmyapplyŠxĆ===myapplyŠyĆ,True,FalseĆ

Funkcija eqtest je ilustrovana sledećim izrazima:

InŠ1Ć:=eqtestŠi,SSĆ
OutŠ1Ć=True
InŠ2Ć:=eqtestŠR,LSĆ
OutŠ2Ć=True
InŠ3Ć:=eqtestŠM,SLBSĆ
OutŠ3Ć=True
InŠ4Ć:=eqtestŠE,BSLBDĆ
OutŠ4Ć=True
InŠ5Ć:=eqtestŠT,BSBSLBBSBLBSBLBDĆ
OutŠ5Ć=True
InŠ6Ć:=eqtestŠC,BSBLBLBSBBBSBBSBSLBBLDĆ
OutŠ6Ć=True

Skup jednačina može biti verifikovan ako se njihove individualne verifikacije (pomoću funkcije *eqtest*) postave u listu, i primeni listable funkcija *And* na ovu listu. Na primer, jednačine koje su individualno u izrazima InŠ1Ć-InŠ6Ć mogu biti verifikovane istovremeno:

**InŠ7Ć:=AndŠeqtestŠi,SSĆ,eqtestŠR,LSĆ,eqtestŠM,SLBSĆ,
eqtestŠE,BSLBDĆ, eqtestŠT,BSBSLBBSBLBSBLBDĆ,
eqtestŠC,BSBLBLBSBBBSBBSBSLBBLDĆ**
OutŠ7Ć=True

12.4. ZAKLJUČAK

Pokazano je da semigrupa unarnih parfunkcja iz Š6Ć i Š7Ć može biti zasnovana koristeći sledeće pojmove iz *MATHEMATICA*:

- liste kao standardni tipovi podataka,
- bazične funkcije *List*, *First* i *Rest*,
- veoma moćni aparat za simboličko procesiranje, i
- jednostavna manipulacija stringovima.

Pokazano je da semigrupa unarnih parfunkcija može biti implementirana koristeći nekoliko prirodnih generalizacija standardnih tipova podataka i standardnih funkcija iz *MATHEMATICA* Š5Ć.

LITERATURA

[1] N. Blachman, *MATHEMATICA: A Practical Approach,* Englewood Cliffs, New Jersey: Prentice-Hall 1992.

[2] T. Gray and J. Glynn, *Exploring Mathematics in MATHEMATICA*, Redwood City, California: Adisson-Wesley, 1991.

[3] H. Jurgensen, *Computers in semigroup,* Semigroup Forum **15** 1977 1--20

[4] R. Maeder, *Programming in MATHEMATICA, Third Edition,* Redwood City, California: Adisson-Wesley 1996.

[5] P.S. Stanimirović, *About the pairalgebra and unary pairfunctions*, Analele Universitatii din Oradea, Fascicola *MATHEMATICA*, **VII** (1999-2000),36—56.

[6] B. Wald, *The semigroup of unary pairfunctions, verification of the group relations,* MapleTech, **4** (1997), 51—54.

[7] B. Wald, *The theory of unary pairfunctions,* in Semantics of Programming Languages and Model Theory, **5** (1993), pp. 287—304.

[8] S. Wolfram, *MATHEMATICA: a System for Doing Mathematics by Computer,* Addison-Wesley Publishing Co, Redwood City, California, 1991.

[9] S. Wolfram, *MATHEMATICA Book, Version 3.0,* Addison-Wesley Publishing Co, Redwood City, California, 1996.