Sveučilište u Zagrebu

Prirodoslovno-matematički fakultet

Matematički odsjek

OSNOVE ALGORITAMA

Predavanja

Vedran Krčadinac

Akademska godina 2016./2017.

Sadržaj

Poglavlje 1

Pojam algoritma

Pojam algoritma objasnit ćemo kroz jednostavan primjer. Treba zbrojiti dva velika prirodna broja bez kalkulatora, recimo 6247345 i 98492. Sjetimo se postupka za zbrajanje naučenog u osnovnoj školi:

$$\begin{array}{rr} & {\scriptstyle 1\ \ 1\ \ \ \ \ \ \ } \\ & 6\,2\,4\,7\,3\,4\,5 \\ + & \quad 9\,8\,4\,9\,2 \\ \hline & 6\,3\,4\,5\,8\,3\,7 \end{array}$$

Ovdje nas ne zanima rezultat, nego postupak kojim do njega dolazimo – to je prvi primjer algoritma kojeg ćemo obraditi!

Pokušajmo precizno zapisati algoritam za zbrajanje. Kao prvo, na ovaj način možemo zbrojiti bilo koja dva prirodna broja. Na početku ih moramo zadati. Ulazni podaci algoritma za zbrajanje su dva prirodna broja zadani kao nizovi znamenaka: $(a_m, a_{m-1}, \ldots, a_2, a_1)$ i $(b_n, b_{n-1}, \ldots, b_2, b_1)$, gdje su $a_i, b_j \in \{0, 1, \ldots, 9\}$. Brojevi ne moraju imati jednako mnogo znamenaka, tj. može biti $m \neq n$. Međutim, možemo nadopisati vodeće nule onom broju koji ima manje znamenaka. Tako će zapis algoritma za zbrajanje biti jednostavniji.

Ako oba broja na ulazu imaju n znamenaka, rezultat može imati najviše $n+1$ znamenaka. Izlazne podatke čini niz znamenaka zbroja $(c_{n+1}, c_n, c_{n-1}, \ldots, c_2, c_1)$. U slučaju kada zbroj ima samo n znamenaka, stavljamo $c_{n+1} = 0$. Račun općenito izgleda ovako:

$$\begin{array}{rr} & a_n\ a_{n-1}\ \cdots\ a_2\ a_1 \\ + & \quad b_n\ b_{n-1}\ \cdots\ b_2\ b_1 \\ \hline & c_{n+1}\ c_n\ c_{n-1}\ \cdots\ c_2\ c_1 \end{array}$$

Zbrajanje počinjemo zdesna, od znamenaka jedinica. U prvom koraku računamo $a_1 + b_1$ i rezultat zapišemo na mjesto c_1. U uvodnom primjeru imamo $a_1 = 5$ i $b_1 = 2$, pa je $c_1 = 7$.

Ako algoritam za zbrajanje želimo analizirati do najjednostavnijih koraka, trebamo se zapitati otkud znamo da je $5 + 2 = 7$? Takve male zbrojeve možemo izračunati "na prste", no tijekom školovanja zapamtili smo sve moguće zbrojeve znamenaka. Možemo,

dakle, reći da znamenke zbrajamo prema ovoj tablici:

+	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	10	11
3	3	4	5	6	7	8	9	10	11	12
4	4	5	6	7	8	9	10	11	12	13
5	5	6	7	8	9	10	11	12	13	14
6	6	7	8	9	10	11	12	13	14	15
7	7	8	9	10	11	12	13	14	15	16
8	8	9	10	11	12	13	14	15	16	17
9	9	10	11	12	13	14	15	16	17	18

U drugom koraku računamo $4 + 9 = 13$, no na mjesto c_2 pišemo samo znamenku 3. Znamenku desetice prenosimo u treći korak ("pišem 3, pamtim 1"). Radi jednostavnijeg zapisa razdvojit ćemo tablicu zbrajanja znamenaka na tablicu zbrojeva (koje potpisujemo ispod trenutačnih znamenaka pribrojnika) i na tablicu prijenosa (koje pamtimo za sljedeći korak, odnosno zabilježimo iznad odgovarajućih znamenaka pribrojnika). Dakle, imamo ove dvije tablice:

Zbroj	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	0
2	2	3	4	5	6	7	8	9	0	1
3	3	4	5	6	7	8	9	0	1	2
4	4	5	6	7	8	9	0	1	2	3
5	5	6	7	8	9	0	1	2	3	4
6	6	7	8	9	0	1	2	3	4	5
7	7	8	9	0	1	2	3	4	5	6
8	8	9	0	1	2	3	4	5	6	7
9	9	0	1	2	3	4	5	6	7	8

Prijenos	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	0	0	1	1
3	0	0	0	0	0	0	0	1	1	1
4	0	0	0	0	0	0	1	1	1	1
5	0	0	0	0	0	1	1	1	1	1
6	0	0	0	0	1	1	1	1	1	1
7	0	0	0	1	1	1	1	1	1	1
8	0	0	1	1	1	1	1	1	1	1
9	0	1	1	1	1	1	1	1	1	1

Na unose u tablicama pozivat ćemo se s pomoću uglatih zagrada. Tako je npr. Zbroj$[4, 9] = 3$ i Prijenos$[4, 9] = 1$ jer je $4 + 9 = 13$.

U trećem koraku zbrajamo znamenke $a_3 = 3$, $b_3 = 4$ i dodajemo im prijenos iz prethodnog koraka. Taj prijenos može biti samo 0 ili 1, zato nam je jednostavnije modificirati tablice zbrajanja nego ih pozivati više puta. Ako je prijenos iz prethodnog koraka 0,

koristimo gore navedene tablice, a ako je prijenos 1 ove modificirane tablice:

ZBROJ2	0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9	0
1	2	3	4	5	6	7	8	9	0	1
2	3	4	5	6	7	8	9	0	1	2
3	4	5	6	7	8	9	0	1	2	3
4	5	6	7	8	9	0	1	2	3	4
5	6	7	8	9	0	1	2	3	4	5
6	7	8	9	0	1	2	3	4	5	6
7	8	9	0	1	2	3	4	5	6	7
8	9	0	1	2	3	4	5	6	7	8
9	0	1	2	3	4	5	6	7	8	9

PRIJENOS2	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	0	1	1	1
3	0	0	0	0	0	0	1	1	1	1
4	0	0	0	0	0	1	1	1	1	1
5	0	0	0	0	1	1	1	1	1	1
6	0	0	0	1	1	1	1	1	1	1
7	0	0	1	1	1	1	1	1	1	1
8	0	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1	1

Dakle, u trećem koraku pišemo znamenku $c_3 = \text{ZBROJ2}[3,4] = 8$ i pamtimo $\text{PRIJENOS2}[3,4] = 0$. Analogno nastavljamo dalje.

Zapišimo sada što radimo u i-tom koraku algoritma. Neka p označava prijenos iz prethodnog koraka. Ako je $p = 0$, znamenku c_i i prijenos za sljedeći korak uzimamo iz tablica ZBROJ i PRIJENOS, a u slučaju $p = 1$ koristimo tablice ZBROJ2 i PRIJENOS2. To možemo zapisati ovako:

ako je $p = 0$ onda
  $c_i = \text{ZBROJ}[a_i, b_i]$
  $p = \text{PRIJENOS}[a_i, b_i]$
inače
  $c_i = \text{ZBROJ2}[a_i, b_i]$
  $p = \text{PRIJENOS2}[a_i, b_i]$

Ovo ponavljamo za $i = 1, 2, \ldots, n$. U prvom koraku nema prijenosa pa stavljamo $p = 0$, a na kraju definiramo c_{n+1} kao prijenos iz n-tog koraka. Zapis cijelog algoritma izgleda ovako:

$p = 0$
za $i = 1, \ldots, n$ ponavljaj
  ako je $p = 0$ onda
    $c_i = \text{ZBROJ}[a_i, b_i]$
    $p = \text{PRIJENOS}[a_i, b_i]$
  inače
    $c_i = \text{ZBROJ2}[a_i, b_i]$
    $p = \text{PRIJENOS2}[a_i, b_i]$
$c_{n+1} = p$

Sada kad smo upoznali jedan primjer algoritma, možemo istaknuti neke opće značajke tog pojma. Općenito, *algoritam* je postupak za rješavanje neke klase problema. Pojedine probleme iz te klase nazivamo *instancama*. Na početku poglavlja prisjetili smo se algoritma za zbrajanje prirodnih brojeva kroz instancu $6247345 + 98492$ problema zbrajanja.

Svaki algoritam uzima neke *ulazne podatke*; oni određuju instancu problema na koji ga primjenjujemo. Na kraju rada vraća *izlazne podatke*, koji predstavljaju rješenje tog

problema. Kod algoritma za zbrajanje ulazni podaci su pribrojnici, a izlazni podaci njihov zbroj.

Algoritam *staje nakon konačno mnogo koraka*, za svaku instancu problema kojeg rješava. Nije teško napisati niz naredbi koji bi se izvodio beskonačno, ali takve "programe" ne smatramo opisima algoritama. Naš algoritam za zbrajanje staje nakon n koraka ako su na ulazu n-znamenkasti brojevi (ne računajući prvo i zadnje pridruživanje).

Svaki korak od kojeg se sastoji algoritam je *precizno i jednoznačno određen*. U praksi možemo imati jednostavnije ili kompliciranije korake, no bitno je da ih čovjek ili stroj koji izvodi algoritam može izvršavati točno i bez dvosmislenosti.

Slika 1.1: Al-Khwarizmi (oko 790 – 840 g.)[1].

Riječ "algoritam" potječe od perzijskog matematičara, astronoma i geografa iz 9. stoljeća al-Khwarizmija (puno ime u engleskoj transkripciji glasi *Muhammad ibn Musa al-Khwarizmi*). Napisao je knjigu u kojoj je opisao postupke za računanje u indijskom brojevnom sustavu. Original na arapskom nije sačuvan, a latinski prijevod proširio europom pod naslovom *Algoritmi de numero Indorum* ("Al-Khwarizmi o indijskim brojevima"). Prema tom naslovu postupke za sustavno rješavanje problema danas nazivamo algoritmima. Stariji naziv *algorizam* ponekad se koristi u užem, aritmetičkom smislu[2].

Al-Khwarizmi napisao je još jednu iznimno važnu knjigu, *Hisab al-jabr w'al-muqabala*, u kojoj se bavi rješavanjem linearnih i kvadratnih jednadžbi. Matematička disciplina *algebra* dobila je ime po drugoj riječi iz tog naslova. S današnjeg stanovišta problemi kojima se al-Khwarizmi bavi mogu se činiti trivijalnim. Međutim, tehniku njihovog rješavanja dugujemo upravo Al-Khwarizmiju: postavimo problem u obliku algebarskih jednadžbi i primjenjujemo unaprijed zadana pravila da bismo došli do rješenja. Trivijalizacija problema koji se svode na linearne ili kvadratne jednadžbe posljedica je duboke "algoritmizacije" matematike. Formula za rješenja kvadratne jednadžbe

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

[1]Slika je preuzeta s web stranice [19].

[2]U Klaićevom rječniku stranih riječi [14] algoritam se definira kao "pravilan postupak pri računanju".

zapravo je sažet opis algoritma kojim od koeficijenata jednadžbe $ax^2+bx+c=0$ dolazimo do njezinih rješenja.

Vratimo se algoritmima za osnovne aritmetičke operacije. U školi smo osim algoritma za zbrajanje naučili algoritme za oduzimanje, množenje, dijeljenje, možda čak i algoritam za računanje približne vrijednosti drugog korijena. Mnoga djeca ne vole matematiku upravo zbog inzistiranja na "mehaničkom" savladavanju takvih računskih rutina[3].

Cilj ove skripte nije učenje izabranih algoritama napamet, nego razvijanje sposobnosti razumijevanja i kreiranja novih algoritama. Za to nam je nužna vještina preciznog zapisivanja algoritama i "čitanja s razumijevanjem" algoritama koji je netko drugi napisao. Po mišljenju autora, da bi se to savladalo potrebno je upoznati neke važne algoritme i samostalno riješiti određen broj zadataka u kojima se traži razvijanje algoritma za neki problem. Čitatelje također potičemo da algoritme koje ćemo upoznati pokušaju implementirati u nekom programskom jeziku. Pritom treba savladati određene tehničke prepreke, ali na taj način alogitmi "ožive" i dobiva se mogućnost testiranja, isprobavanja s raznim ulaznim podacima i jednostavnog modificiranja algoritama. Izbor programskog jezika manje je važan. Koristit ćemo uglavnom naredbe i koncepte koji postoje u svima, ili barem u većini programskih jezika[4].

Zadaci

Zadatak 1.1. *Dokažite da zbrajanjem dvaju n-znamenkastih prirodnih brojeva dobivamo broj s najviše $n+1$ znamenaka.*

Zadatak 1.2. *Zapišite algoritam za množenje prirodnih brojeva naučen u osnovnoj školi! Koraci algoritma mogu biti složenijih od onih koje smo koristili u zapisu algoritma za zbrajanje, npr. možete koristiti zbrajanje višeznamenkastih brojeva kao poznatu operaciju.*

Zadatak 1.3. *Na sličan način zapišite algoritam za dijeljenje prirodnih brojeva naučen u osnovnoj školi.*

Zadatak 1.4. *Sjetite se i zapišite algoritam za "ručno" vađenje drugog korijena, u kojem se znamenke grupiraju po dvije.*

Zadatak 1.5. *Pokušajte izmisliti i zapisati alternativne algoritme za osnovne računske operacije, različite od onih koje ste naučili u školi.*

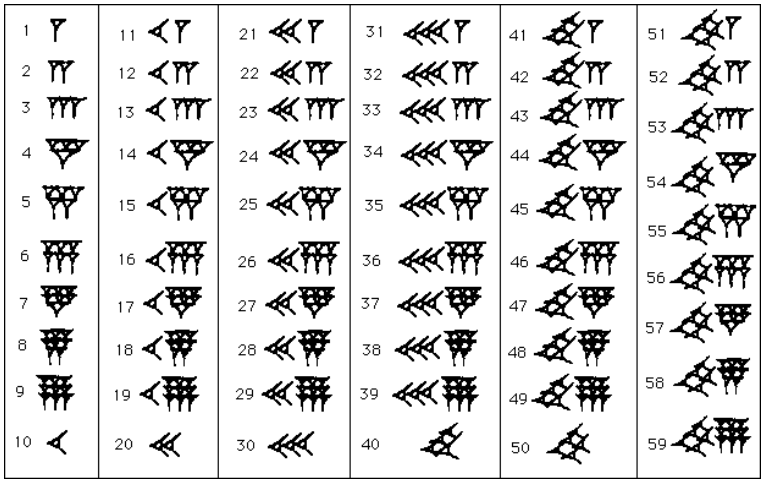
[3] U američkom školskom kurikulumu "Everiday mathematics" djecu se potiče da razviju vlastite postupke za računanje i upoznaje ih se s alternativnim algoritmima za osnovne aritmetičke operacije; vidi [25].

[4] Prethodnih akademskih godina na vježbama iz kolegija *Osnove algoritama* zadaci su se radili u programskim jezicima C, Pascal i Python.

Poglavlje 2

Brojevni sustavi

Razvoj sustava za zapisivanje brojeva možemo smatrati početkom matematike. Jedan od najranijih poznatih sustava razvili su Babilonci oko 2000 godine prije Krista. Današnji način zapisivanja brojeva potječe iz Indije i vjerojatno je nastao krajem 6. stoljeća. Ponekad se takvi brojevi nazivaju arapskim, jer su ih u srednjem vijeku Arapi donijeli u Europu. Taj je sustav zapisivanja brojeva omogućio razvoj algoritama za računanje o kojima smo govorili u prethodnom poglavlju. U drugom poznatom sustavu, rimskom aditivno-suptraktivnom sustavu, računanje je vrlo nespretno. Pokušajte direktno pomnožiti brojeve CXVII i XXIV!



Slika 2.1: Znamenke u babilonskom brojevnom sustavu[1].

U indijsko-arapskom sustavu koristi se deset znamenaka 0, 1, 2, 3, 4, 5, 6, 7, 8 i 9. Značenje znamenaka ovisi o položaju u zapisu broja, zato se takvi sustavi nazivaju *pozicionim*. Naprimjer, zapis 2562 znači $2\cdot 10^3+5\cdot 10^2+6\cdot 10+2$ (prva i zadnja znamenka 2 ne doprinose isto!). Istaknuta uloga broja 10 posljedica je toga što ljudi imaju 10 prstiju. Na isti način možemo zapisivati brojeve koristeći se bilo kojim skupom od b različitih simbola, gdje je $b \geq 2$ prirodan broj koji nazivamo *bazom*.

[1]Slika je preuzeta s web stranice [20].

Tijekom povijesti koristili su se i sustavi s bazom 20, a spomenuti babilonski sustav bio je pozicioni s bazom 60. Babilonci ipak nisu imali 60 različitih simbola, nego su znamenke zapisivali aditivno s pomoću dvaju simbola (vidi sliku 2.1). U računarstvu su posebno pogodni sustavi kojima je baza potencija od 2. Sustav s bazom $b = 2$ naziva se *binarnim sustavom*, sustav s bazom $b = 3$ *ternarnim sustavom* i tako dalje. Uobičajeni sustav s bazom $b = 10$ zovemo *dekadskim sustavom*, a koriste se i *oktalni sustav* ($b = 8$) te *heksadecimalni sustav* ($b = 16$). Naprimjer, binarni broj $(101101)_2$ znači $1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2 + 1$, a oktalni broj $(2763)_8$ predstavlja $2 \cdot 8^3 + 7 \cdot 8^2 + 6 \cdot 8 + 3$. U sustavima s bazom manjom od 10 kao znamenke koristimo podskup uobičajenih dekadskih znamenaka. U heksadecimalnom sustavu trebamo 16 simbola, pa se uz dekadske znamenke koristimo slovima A, B, C, D, E i F koja redom predstavljaju brojeve od 10 do 15. Heksadecimalni broj $(7A9FE)_{16}$ znači $7 \cdot 16^4 + 10 \cdot 16^3 + 9 \cdot 16^2 + 15 \cdot 16 + 14$.

Općenito, u sustavu s bazom b skup simbola identificiramo s brojevima koje predstavljaju: $\{0, 1, 2, \ldots, b-1\}$. Zapis $(x_k x_{k-1} \cdots x_1 x_0)_b$ sa znamenkama x_i iz tog skupa predstavlja broj $x_k \cdot b^k + x_{k-1} \cdot b^{k-1} + \ldots + x_1 \cdot b + x_0$. S pomoću ove definicije brojeve zapisane u bazi b lako možemo prevesti u uobičajeni dekadski zapis.

Primjer 2.1. *Zapišimo brojeve* $(1011011)_2$, $(123)_4$ *i* $(\mathrm{BABA})_{16}$ *u dekadskom sustavu.*

Rješenje. Po definiciji, vrijedi $(1011011)_2 = 2^6 + 2^4 + 2^3 + 2 + 1 = 91$. Kada u zapisu ne istaknemo bazu, podrazumijevat ćemo da je dekadski ($b = 10$). Analogno dobivamo $(123)_4 = 1 \cdot 4^2 + 2 \cdot 4 + 3 = 27$ i $(\mathrm{BABA})_{16} = 11 \cdot 16^3 + 10 \cdot 16^2 + 11 \cdot 16 + 10 = 47802$. □

"Valjanost" pozicionih sustava počiva na činjenici da se svaki prirodan broj može na jedinstven način zapisati u sustavu s bilo kojom bazom.

Teorem 2.2. *Neka je $b \geq 2$ zadani prirodan broj. Za svaki prirodan broj n postoji jedinstveni niz znamenaka $(x_k, \ldots, x_1, x_0)$, $x_i \in \{0, 1, \ldots, b-1\}$, $x_k \neq 0$, takav da je $n = x_k \cdot b^k + \ldots + x_1 \cdot b + x_0$.*

Za dokaz ovog teorema trebat će nam teorem o dijeljenju s ostatkom.

Teorem 2.3 (o dijeljenju s ostatkom)**.** *Neka su $m, n \in \mathbb{N}$ prirodni brojevi. Tada postoje jedinstveni brojevi $k, o \in \mathbb{N}_0$, $o < n$ takvi da je $m = k \cdot n + o$.*

Broj k zovemo *kvocijentom* ili *količnikom*, a broj o *ostatkom* dijeljenja. Zbog njihove jedinstvenosti možemo uvesti oznake $k = m$ div n i $o = m$ mod n. Naprimjer, 17 div $5 = 3$ i 17 mod $5 = 2$ jer je $17 = 3 \cdot 5 + 2$. Dokažimo sada teorem 2.2.

Dokaz. Trebamo dokazati egzistenciju i jedinstvenost prikaza broja n u bazi b. Egzistenciju ćemo dokazati konstruktivno, opisujući algoritam za nalaženje traženog prikaza. Označimo $n_0 = n$ i podijelimo taj broj s b. Kvocijent označimo n_1, a ostatak x_0:

$$n_0 = n_1 \cdot b + x_0.$$

Zatim podijelimo n_1 s b uz kvocijent n_2 i ostatak x_1:

$$n_1 = n_2 \cdot b + x_1.$$

Postupak analogno nastavimo dalje:

$$\begin{aligned} n_2 &= n_3 \cdot b + x_2, \\ n_3 &= n_4 \cdot b + x_3, \\ &\vdots \\ n_k &= 0 \cdot b + x_k. \end{aligned}$$

Zaustavljamo se kada kvocijent postane 0. Svi daljnji kvocijenti n_i i ostaci x_i bili bi 0. Opisani postupak ovako možemo zapisati u obliku algoritma:

$n_0 = n$
$i = 0$
dok je $n_i \neq 0$ ponavljaj
$\left[\begin{array}{l} x_i = n_i \bmod b \\ n_{i+1} = n_i \operatorname{div} b \\ i \leftarrow i + 1 \end{array}\right.$

Naredba $i \leftarrow i + 1$ u zadnjem retku povećava vrijednost varijable i za jedan.

Da bi ovo zaista bio dokaz egzistencije prikaza u bazi b, trebamo se uvjeriti da postupak staje nakon konačno mnogo koraka i definira traženi prikaz. Svaki sljedeći n_i dobivamo od prethodnog cjelobrojnim dijeljenjem s $b \geq 2$. Jasno je da će nakon konačno mnogo koraka rezultat biti 0 i tada algoritam staje. Još treba provjeriti da su ostaci $x_0, \ldots, x_k \in \{0, \ldots, b-1\}$ zaista znamenke od n u bazi b. Pretpostavimo da smo nakon k koraka dobili kvocijent $n_{k+1} = 0$. U prethodnom koraku definirali smo $n_k = n_{k+1} \cdot b + x_k = 0 \cdot b + x_k = x_k$. Supstitucijom u prethodne korake dobivamo redom

$$\begin{aligned} n_{k-1} &= n_k \cdot b + x_{k-1} = x_k \cdot b + x_{k-1}, \\ n_{k-2} &= n_{k-1} \cdot b + x_{k-2} = (x_k \cdot b + x_{k-1}) \cdot b + x_{k-1} = x_k \cdot b^2 + x_{k-1} \cdot b + x_{k-2}, \\ &\vdots \\ n_0 &= n_1 \cdot b + x_0 = (x_k \cdot b^{k-1} + \ldots + x_1) \cdot b + x_0 = x_k \cdot b^k + \ldots + x_1 \cdot b + x_0. \end{aligned}$$

Vidimo da $n = n_0$ ima prikaz $(x_k \cdots x_1 x_0)_b$. Sada još treba provjeriti jedinstvenost tog prikaza. Pretpostavimo da n ima još jedan prikaz $(y_l \cdots y_1 y_0)_b$. Tada je $n = (x_k \cdot b^{k-1} + \ldots + x_1) \cdot b + x_0 = (y_l \cdot b^{l-1} + \ldots + y_1) \cdot b + y_0$. Podijelimo li taj izraz cjelobrojno s b, zbog jedinstvenosti u teoremu o dijeljenju s ostatkom dobivamo da je $x_0 = y_0 = n \bmod b$ i $x_k \cdot b^{k-1} + \ldots + x_2 \cdot b + x_1 = y_l \cdot b^{l-1} + \ldots + y_2 \cdot b + y_1 = n \operatorname{div} b$. Posljednji izraz ponovo dijelimo s b i dobivamo $x_1 = y_1$ i tako dalje. Budući da su sve znamenke jednake ($x_i = y_i$, $i = 0, 1, 2, \ldots$), vodeća nenul znamenka se također podudara pa su zapisi jednake duljine ($k = l$). Ta se duljina može izraziti kao funkcija od n i od b, što ostavljamo čitateljima kao zadatak. □

Kroz ovaj dokaz ujedno smo dobili postupak kojim broj n zapisan u dekadskom sustavu možemo prevesti su sustav s bilo kojom bazom b. Algoritam iz dokaza izvodimo na papiru tako da desno od broja n povučemo vertikalnu crtu i uzastopno ga dijelimo s bazom. Ostatke dijeljenja pišemo desno od crte, a kvocijente lijevo, jedan ispod drugog.

Primjer 2.4. *Zapišimo broj $n = 3761$ u sustavu s bazom $b = 7$.*

Rješenje. Dijeljenjem sa 7 dobivamo kvocijent 537 i ostatak 2.

$$\begin{array}{r|l} 3761 & 2 \\ 537 & \end{array}$$

Zatim podijelimo 537 sa 7, kvocijent 76 napišemo ispod, a ostatak 5 desno od crte:

$$\begin{array}{r|l} 3761 & 2 \\ 537 & 5 \\ 76 & \end{array}$$

Nastavimo dijeliti sa 7, rezultate pisati ispod, a ostatke desno.

$$\begin{array}{r|l} 3761 & 2 \\ 537 & 5 \\ 76 & 6 \\ 10 & 3 \\ 1 & 1 \\ 0 & \end{array}$$

Kada dobijemo nulu, pročitamo ostatke odozdo prema gore i imamo zapis $n = (13652)_7$. □

Broj zapisan u bazi b_1 možemo prevesti u bazu b_2 tako da ga najprije zapišemo u dekadskom sustavu, a zatim prebacimo u bazu b_2 algoritmom koji smo upoznali. Ako su b_1 i b_2 potencije istog broja, to možemo učiniti efikasnije grupiranjem ili proširivanjem znamenaka.

Primjer 2.5. *Zapišimo binarni broj $(10110111101)_2$ u oktalnom sustavu.*

Rješenje. Najprije znamenke grupiramo po 3 zdesna na lijevo:

$$\begin{array}{cccc} 10 & 110 & 111 & 101 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 2 & 6 & 7 & 5 \end{array}$$

Istaknute nizove triju binarnih znamenaka prevodimo u po jednu oktalnu: $000 \to 0$, $001 \to 1$, $010 \to 2$, $\ldots$, $111 \to 7$. Vodeću grupu znamenaka 10 tretiramo kao 010. Tako dobivamo zapis $(2675)_8$. □

Primjer 2.6. *Zapišimo heksadecimalni broj $(2A9)_{16}$ u binarnom sustavu.*

Rješenje. Svaku heksadecimalnu znamenku prevodimo u niz od 4 binarne znamenke: $0 \to 0000$, $1 \to 0001$, $2 \to 0010$, $\ldots$, $9 \to 1001$, $A \to 1010$, $\ldots$, $F \to 1111$. Vodeće nule prvog po redu niza zanemarujemo. Tako dolazimo do zapisa $(1010101001)_2$. □

Primjer 2.7. *Zapišimo oktalni broj* $(6251)_8$ *u heksadecimalnom sustavu.*

Rješenje. Prvo oktalni broj pretvorimo u binarni:

$$\begin{array}{cccc} 6 & 2 & 5 & 1 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 110 & 010 & 101 & 001 \end{array}$$

Zatim grupiramo binarne znamenke po četiri i prevedemo ih u hekdadecimalne:

$$\begin{array}{ccc} 1100 & 1010 & 1001 \\ \downarrow & \downarrow & \downarrow \\ \mathrm{C} & \mathrm{A} & 9 \end{array}$$

Dobili smo broj $(\mathrm{CA9})_{16}$. □

Algoritmi za osnovne računske operacije koje smo naučili u školi funkcioniraju u sustavu s bilo kojom bazom. Postupci su potpuno analogni onima u dekadskom sustavu, samo treba prilagoditi tablice zbrajanja i množenja znamenaka. Naprimjer, u binarnom sustavu je $(1)_2 + (1)_2 = (10)_2$ ("pišem 0, pamtim 1").

Primjer 2.8. *Izračunajmo zbroj* $(12201)_3 + (2121)_3$ *u ternarnom sustavu.*

Rješenje. Treba nam tablica zbrajanja ternarnih znamenaka:

$+$	0	1	2
0	0	1	2
1	1	2	10
2	2	10	11

Postupak izgleda ovako:

$$\begin{array}{r} {\scriptstyle 1\ 1\ \ \ \ \ } \\ 1\,2\,2\,0\,1 \\ +\quad 2\,1\,2\,1 \\ \hline 2\,2\,0\,2\,2 \end{array}$$

Rezultat je broj $(22022)_3$. □

Primjer 2.9. *Izračunajmo razliku* $(421314)_5 - (34123)_5$ *u sustavu s bazom* 5.

Rješenje. I za oduzimanje koristit ćemo se tablicom zbrajanja znamenaka u bazi 5.

$+$	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	10
2	2	3	4	10	11
3	3	4	10	11	12
4	4	10	11	12	13

Postupak počinje isto kao u dekadskom sustavu:

$$\begin{array}{r} 4\,2\,1\,3\,1\,4 \\ -\quad 3\,4\,1\,2\,3 \\ \hline 1 \end{array}$$

Oduzimamo najdesnije znamenke $(4)_5 - (3)_5 = (1)_5$ i rezultat pišemo ispod crte. U idućem koraku trebali bismo oduzeti veću znamenku od manje. "Posuđujemo" jedinicu sa sljedeće pozicije i umjesto $(1)_5 - (2)_5$ računamo $(11)_5 - (2)_5$. U tablici zbrajanja pronalazimo $(2)_5 + (4)_5 = (11)_5$, pa je rezultat znamenka 4.

$$\begin{array}{r} \overset{}{}\,\overset{}{}\,\overset{}{}\,\overset{2}{}\,\overset{}{}\,\overset{}{} \\ 4\,2\,1\,\not{3}\,1\,4 \\ -\quad 3\,4\,1\,2\,3 \\ \hline 4\,1 \end{array}$$

Sada zbog posuđene jedinice ne računamo $(3)_5 - (1)_5$, nego $(2)_5 - (1)_5 = (1)_5$.

$$\begin{array}{r} \overset{}{}\,\overset{}{}\,\overset{}{}\,\overset{2}{}\,\overset{}{}\,\overset{}{} \\ 4\,2\,1\,\not{3}\,1\,4 \\ -\quad 3\,4\,1\,2\,3 \\ \hline 1\,4\,1 \end{array}$$

U iduća dva koraka također posuđujemo jedinice sa sljedećih pozicija.

$$\begin{array}{r} \overset{}{}\,\overset{3}{}\,\overset{1}{}\,\overset{2}{}\,\overset{}{}\,\overset{}{} \\ \not{4}\,\not{2}\,1\,\not{3}\,1\,4 \\ -\quad 3\,4\,1\,2\,3 \\ \hline 3\,3\,2\,1\,4\,1 \end{array}$$

Rezultat je $(332141)_5$. □

Primjer 2.10. *Izračunajmo umnožak* $(1011011)_2 \cdot (1101)_2$ *u binarnom sustavu.*

Rješenje. Množenje je u binarnom sustavu posebno jednostavno jer su jedine znamenke 0 i 1, pa nam tablice ne trebaju. Za svaku znamenku jedinice iz desnog faktora potpisujemo lijevi faktor pomaknut odgovarajući broj mjesta udesno.

$$\begin{array}{r} 1\,0\,1\,1\,0\,1\,1 \cdot 1\,1\,0\,1 \\ \hline 1\,0\,1\,1\,0\,1\,1 \\ \quad 1\,0\,1\,1\,0\,1\,1 \\ \qquad\qquad 0 \\ +\qquad 1\,0\,1\,1\,0\,1\,1 \\ \hline 1\,0\,0\,1\,0\,0\,1\,1\,1\,1\,1 \end{array}$$

Zbrajanjem potpisanih brojeva dobivamo umnožak $(10010011111)_2$. □

Primjer 2.11. *Izračunajmo umnožak* $(3122)_4 \cdot (232)_4$ *u sustavu s bazom* 4.

Rješenje. Za ovaj račun trebaju nam tablice zbrajanja i množenja u bazi 4.

+	0	1	2	3
0	0	1	2	3
1	1	2	3	10
2	2	3	10	11
3	3	10	11	12

·	1	2	3
1	1	2	3
2	2	10	12
3	3	12	21

Množimo $(3122)_4$ s prvom znamenkom drugog faktora 2, zdesna na lijevo. U tablici nalazimo produkt znamenaka $(2)_4 \cdot (2)_4 = (10)_4$, pišemo znamenku 0 i prenosimo 1 u sljedeći korak. U drugom koraku računamo $(2)_4 \cdot (2)_4 + (1)_4 = (11)_4$, pišemo 1 ponovo prenosimo 1. U trećem koraku pišemo $(2)_4 \cdot (1)_4 + (1)_4 = (3)_4$, a u četvrtom $(2)_4 \cdot (3)_4 = (12)_4$:

$$\begin{array}{r} {\scriptstyle 1\ 1\ \ \ \ } \\ 3122 \cdot 232 \\ \hline 12310 \end{array}$$

Na sličan način množimo $(3122)_4$ sa znamenkom 3 i rezultat zapisujemo jedno mjesto udesno.

$$\begin{array}{r} {\scriptstyle 1\ 1\ 1\ \ \ } \\ 3122 \cdot 232 \\ \hline 12310 \\ \ 22032 \end{array}$$

Na kraju ponovno množimo s 2 i zbrajamo tri produkta.

$$\begin{array}{r} 3122 \cdot 232 \\ \hline 12310 \\ \ 22032 \\ +\quad 12310 \\ \hline 2130230 \end{array}$$

Rezultat je $(2130230)_4$. □

Primjer 2.12. *U binarnom sustavu podijelimo brojeve* $(100011)_2 : (111)_2$.

Rješenje. Dijeljenje je jednostavnije u binarnom nego u dekadskom sustavu. Da bismo odredili prvu znamenku kvocijenta, moramo procijeniti koliko puta djelitelj "stane" u početni niz znamenaka djeljenika. U binarnom sustavu odgovor može biti samo 0 ili 1!

$$\begin{array}{r} 100011:111=1 \\ -\quad 111 \\ \hline \quad\ 11 \end{array}$$

Broj $(111)_2$ oduzimamo od prve četiri znamenke broja $(100011)_2$ i zatim "spuštamo" sljedeću znamenku. Dobili smo broj $(11)_2$ koji je manji od djelitelja, pa na rezultat

dijeljenja dodajemo 0 i spuštamo još jednu znamenku djeljenika.

$$\begin{array}{r} 100011:111=10 \\ -\ \ 111 \\ \hline 111 \end{array}$$

Sada djelitelj stane točno jednom u $(111)_2$ i vidimo da je kvocijent $(101)_2$:

$$\begin{array}{r} 100011:111=101 \\ -\ \ 111 \\ \hline 111 \\ -\ 111 \\ \hline 0 \end{array}$$

□

Osim prirodnih brojeva u sustavima s različitim bazama možemo zapisivati cijele, racionalne i realne brojeve. Broj nula u svim sustavima zapisujemo kao 0. Zanimljivo je da u babilonskom heksadecimalnom sustavu nije bilo posebnog simbola za nulu. Vrijednost broja morala se pogađati iz konteksta i razmaka među znamenkama. Negativne cijele brojeve zapisujemo na uobičajen način, sa znakom "$-$" ispred zapisa. Naprimjer, $-(1100)_2$ je binarni zapis broja -12.

Razlomci su omjeri cijelog i prirodnog broja i možemo ih tako zapisivati. Druga mogućnost je da proširimo pozicioni sustav i dozvolimo negativne potencije baze, analogno decimalnim brojevima[2] u dekadskom sustavu. Decimalni zapis 27.69 predstavlja broj $2 \cdot 10^1 + 7 \cdot 10^0 + 6 \cdot 10^{-1} + 9 \cdot 10^{-2}$. Općenito, zapis $(x_k \cdots x_1 x_0 . x_{-1} \cdots x_{-m})_b$ znači $x_k \cdot b^k + \ldots + x_1 \cdot b^1 + x_0 \cdot b^0 + x_{-1} \cdot b^{-1} + \ldots + x_{-m} \cdot b^{-m}$. Pritom su $k, m \in \mathbb{N}$, a $x_k, \ldots, x_0, \ldots, x_{-m} \in \{0, \ldots, b-1\}$ su znamenke.

Primjer 2.13. *Zapišimo razlomak* $91/16$ *u binarnom sustavu.*

Rješenje. Rješenje možemo dobiti tako da brojnik $91 = (1011011)_2$ i nazivnik $16 = (10000)_2$ zapišemo u binarnom sustavu i primijenimo algoritam za dijeljenje:

$$\begin{array}{r} 1011011:10000=101 \\ -\ 10000 \\ \hline 11011 \\ -\ 10000 \\ \hline 1011 \end{array}$$

Dobili smo kvocijent $(101)_2$ i ostatak $(1011)_2$, koji nastavljamo dijeliti "spuštanjem" nula.

[2]Prema pravopisu decimalne brojeve trebali bismo pisati sa zarezom umjesto s točkom, tj. kao $27,69$. U ovoj knjizi ipak ćemo se koristiti decimalnom točkom zbog čitljivijeg zapisa nizova decimalnih brojeva. Pisat ćemo 1.1, 1.2, 1.3... umjesto $1,1$, $1,2$, $1,3$....

Znamenke koje dobivamo bilježimo iza znaka točke.

$$\begin{array}{rl}
 & 1\,0\,1\,1\,0\,1\,1 : 1\,0\,0\,0\,0 = 1\,0\,1\,.\,1\,0\,1\,1 \\
- & 1\,0\,0\,0\,0 \\
\hline
 & \quad 1\,1\,0\,1\,1 \\
- & \quad 1\,0\,0\,0\,0 \\
\hline
 & \quad\;\; 1\,0\,1\,1\,0 \\
- & \quad 1\,0\,0\,0\,0 \\
\hline
 & \quad\;\;\; 1\,1\,0\,0\,0 \\
- & \quad 1\,0\,0\,0\,0 \\
\hline
 & \quad\;\;\;\; 1\,0\,0\,0\,0 \\
- & \quad 1\,0\,0\,0\,0 \\
\hline
 & \quad\;\;\;\;\; 0
\end{array}$$

Rezultat je $(101.1011)_2$. Alternativno, razlomak $91/16 = 5\frac{11}{16}$ možemo zapisati u binarnom sustavu prevodeći posebno cjelobrojni dio 5, a zatim "decimalni dio" $\frac{11}{16}$. Za cjelobrojni dio koristimo algoritam uzastopnog dijeljenja s bazom, koji smo ranije upoznali:

$$\begin{array}{r|l}
5 & 1 \\
2 & 0 \\
1 & 1 \\
0 &
\end{array}$$

Znamenke iza točke dobivamo "dualnim" algoritmom. Pomnožimo brojnik 11 s bazom 2 i cjelobrojno ga podijelimo s nazivnikom. Kvocijent $22 \text{ div } 16 = 1$ je prva znamenka i zapisujemo ga desno od crte, a ostatak $22 \bmod 16 = 6$ zapisujemo ispod i s njim nastavljamo postupak.

$$\begin{array}{r|l}
11 & 1 \\
6 &
\end{array}$$

Produkt $2 \cdot 6 = 12$ daje rezultat 0 i ostatak 12 pri dijeljenju s nazivnikom 16.

$$\begin{array}{r|l}
11 & 1 \\
6 & 0 \\
12 &
\end{array}$$

Nadalje, $2 \cdot 12 = 24$ daje rezultat 1 i ostatak 8 pri dijeljenju sa 16.

$$\begin{array}{r|l}
11 & 1 \\
6 & 0 \\
12 & 1 \\
8 & 1 \\
0 &
\end{array}$$

U zadnjem koraku $2 \cdot 8 = 16$ djeljiv je bez ostatka s nazivnikom pa algoritam staje. Znamenke cjelobrojnog dijela pišemo od decimalne točke ulijevo (redom kojim smo ih dobivali), a znamenke decimalnog dijela udesno: $(101.1011)_2$. □

U prethodnom primjeru razlomak ima konačni zapis. Kada razlomak do kraja skratimo i u nazivniku ostane prosti faktor koji ne dijeli bazu, dobivamo beskonačni periodični zapis.

Primjer 2.14. *Zapišimo razlomak* $7/15$ *u ternarnom sustavu.*

Rješenje. Koristimo se algoritmom uzastopnog množenja s bazom $b = 3$ i dijeljenja s nazivnikom 15, kojeg smo upoznali u prethodnom primjeru.

7	1
6	1
3	0
9	1
12	2
6	

U petom koraku dobili smo ostatak $3 \cdot 12 \bmod 15 = 6$, koji smo već imali u prvom koraku. To znači da će se u nastavku ponavljati niz ternarnih znamenaka 1012. Razlomak ima zapis $(0.11012101210121012\ldots)_3$, što kraće zapisujemo kao $(0.1\overline{1012})_3$ ili $(0.1\dot{1}01\dot{2})_3$. □

Pokazuje se da svi razlomci imaju beskonačne periodične zapise. "Konačni zapis" shvaćamo kao periodični u kojem se ponavlja znamenka 0. Iracionalni brojevi kao $\sqrt{2}$, π i e i imaju beskonačne neperiodične zapise u pozicionom sustavu s bilo kojom bazom. Uz određeni tehnički dogovor, svaki realan broj ima jedinstven zapis u sustavu s bazom b.

Problem nastaje upravo zbog brojeva s "konačnim zapisom". Naime, umjesto ponavljanja znamenke 0, broj možemo zapisati ponavljanjem najveće znamenke $b-1$. Naprimjer, broj 1 u dekadskom sustavu možemo zapisati kao $1.000\ldots$ ili $0.999\ldots$. Da bismo osigurali jedinstvenost zapisa drugu mogućnost zabranjujemo.

Teorem 2.15. *Neka je* $b \geq 2$ *prirodan broj. Za svaki realan broj* $x > 0$ *postoji jedinstveni cijeli broj* k *i niz znamenaka* $x_i \in \{0, \ldots, b-1\}$, $i \leq k$ *takav da je* $x_k \neq 0$, $x = \sum_{i \leq k} x_i \cdot b^i$ *te da ne postoji* m *takav da je* $x_i = b - 1$ *za sve* $i \leq m$.

Nulu zapisujemo kao 0, a negativne realne brojeve s pomoću znaka "−". Dokaz ovog teorema za $b = 10$ nalazi se u knjizi [21] na str. 35 (teorem 2).

Za računanje s realnim brojevima zapisanim na ovaj način koristimo iste algoritme kao za prirodne brojeve. Pritom zapis ograničavamo na konačan broj znamenaka kako bi algoritmi stali u konačno mnogo koraka. Kod zbrajanja pazimo da pribrojnike potpišemo s točkom na odgovarajućim mjestima, kod množenja je broj znamenaka iza točke u produktu jednak zbroju broja znamenaka iza točke u faktorima itd. Neke probleme koji nastaju zbog ograničenja na konačan broj znamenaka upoznat ćemo u poglavlju o prikazu brojeva u računalu.

Zadaci

Zadatak 2.1. *Opišite algoritam za zbrajanje brojeva u rimskom brojevnom sustavu.*

Zadatak 2.2. *Dokažite teorem o dijeljenju s ostatkom.*

Zadatak 2.3. *Napišite formulu za broj znamenaka prirodnog broja n u zapisu s bazom b.*

Zadatak 2.4. *Zapišite brojeve* $(100110111)_2$, $(1201221)_3$, $(42310)_5$, $(2147)_8$ *i* $(\mathrm{DEDA})_{16}$ *u dekadskom sustavu.*

Zadatak 2.5. *Zapišite dekadski broj* 5790 *u sustavima s bazom* 2, 3, 8 *i* 16.

Zadatak 2.6. *Grupiranjem znamenaka pretvorite heksadecimalni broj* $(\mathrm{8D6E})_{16}$ *u oktalni sustav.*

Zadatak 2.7. *Zbrojite direktno, bez pretvaranja u dekadski sustav* $(1111011)_2 + (10110)_2$, $(122)_3 + (211)_3 + (101)_3 + (120)_3 + (222)_3$ *i* $(\mathrm{BABA})_{16} + (\mathrm{DEDA})_{16}$.

Zadatak 2.8. *Oduzmite direktno, bez pretvaranja u dekadski sustav* $(110011011)_2 - (110110)_2$ *i* $(1302)_4 - (123)_4$.

Zadatak 2.9. *Pomnožite direktno, bez pretvaranja u dekadski sustav* $(1101)_2 \cdot (101)_2$ *i* $(21201)_3 \cdot (1022)_3$.

Zadatak 2.10. *Podijelite direktno, bez pretvaranja u dekadski sustav* $(110323)_5 : (401)_5$ *i* $(1101)_2 : (110)_2$.

Zadatak 2.11. *Precizno zapišite algoritme za zbrajanje, oduzimanje, množenje i dijeljenje u binarnom sustavu.*

Zadatak 2.12. *Zapišite brojeve* $(102.21)_3$, $(0.4)_7$ *i* $(3.03\overline{2})_5$ *kao razlomke i kao decimalne brojeve u sustavu s bazom* 10.

Zadatak 2.13. *Zapišite dekadski broj* 17.3125 *u binarnom i u ternarnom sustavu.*

Zadatak 2.14. *Precizno zapišite algoritam za pretvaranje razlomka manjeg od* 1 *u sustav s bazom b uzastopnim množenjem s bazom i dijeljenjem s nazivnikom, opisan u primjerima 2.13 i 2.14.*

Zadatak 2.15. *Proširivanjem i grupiranjem znamenaka pretvorite broj* $(123.4567)_8$ *u sustav s bazom* 2 *te brojeve* $(10111011.101)_2$ *i* $(443.5274)_8$ *u sustav s bazom* 16.

Zadatak 2.16. *Odredite bazu b takvu da broj* $(235)_{b+1}$ *bude dvostruko veći od broja* $(141)_b$.

Zadatak 2.17. *Odredite šesteroznamenkasti broj u sustavu s bazom* 5 *kojemu je zadnja znamenka* 2 *i koji se poveća dva puta kad zadnju znamenku premjestimo na prvo mjesto. Rezultat zapišite u sustavu s bazom* 5.

Zadatak 2.18. *Odredite bazu b u kojoj je moguć račun* $(11x)_b + (x3y)_b = (3y0)_b$, *za neke znamenke x i y.*

Zadatak 2.19. *Neka je $b \geq 2$ prirodan broj. Zapišite brojeve* $(bbb)_{b^2}$ *i* $(b.b)_{b^2}$ *u sustavu s bazom b.*

Zadatak 2.20. *Dokažite da se svaki prirodan broj n može na jedinstven način prikazati u obliku $n = x_k \cdot k! + x_{k-1} \cdot (k-1)! + \ldots + x_2 \cdot 2! + x_1 \cdot 1!$, pri čemu su znamenke $x_i \in \{0, 1, \ldots, i\}$, $i = 1, \ldots, k$ te vrijedi $x_k \neq 0$. Pišemo $n = (x_k x_{k-1} \cdots x_2 x_1)_!$. To je takozvani* faktorijelski brojevni sustav*.*

Zadatak 2.21. *Fibonaccijevi brojevi definirani su s $F_0 = 0$, $F_1 = 1$ i s rekurzijom $F_{k+1} = F_k + F_{k-1}$ za $k \in \mathbb{N}$. Dokažite da se svaki prirodan broj n može prikazati kao zbroj međusobno različitih Fibonaccijevih brojeva $n = F_{k_1} + F_{k_2} + \ldots + F_{k_r}$. Ako zahtijevamo da pritom vrijedi $k_{i+1} \leq k_i - 2$, $i = 1, \ldots, r-1$ i $k_r \geq 2$, takav prikaz je jedinstven. Ovaj način prikazivanja brojeva naziva se* Fibonaccijevim brojevnim sustavom*.*

Poglavlje 3

Načini zapisivanja algoritama

U prethodnom poglavlju upoznali smo algoritme za osnovne aritmetičke operacije i za prevođenje brojeva iz baze u bazu. Neke od tih algoritama nastojali smo precizno zapisati, dok smo druge opisali neformalno ili smo ih objasnili kroz primjere i zadatke. Taj je pristup prikladniji kod podučavanja. Ne bi bilo mudro školsku djecu učiti zbrajanje tako da im se na ploču napiše algoritam iz prvog poglavlja.

Potreba za preciznim zapisivanjem algoritama pojavila se s razvojem računskih strojeva. U prvoj polovici 19. stoljeća engleski matematičar, inženjer i izumitelj Charles Babbage projektirao je prvo univerzalno računalo, sposobno izvoditi različite proračune ovisno o "programu" koji se u njega unese. Babbage je stroj nazvao *Analytical engine* i do kraja života nije ga uspio izgraditi. Ipak, u suradnji s Adom Lovelace[1] razvijao je prve programe za svoj mehanički kompjuter. Mnogo kasnije programski jezik Ada nazvan je po prvoj programerki Adi Lovelace.

Sredinom 20. stoljeća izgrađena su prva elektronička računala. Tehnologiju elektronskih cijevi ubrzo su zamijenili tranzistori, a u 70-tim godinama poluvodički integrirani krugovi. S minijaturizacijom računala postaju sve brža i sve moćnija. Svjedoci smo da se taj proces nastavlja i danas.

Prva elektronička računala programirala su se u strojnom jeziku, naredbama koje izvodi procesor. Kao pomoć u programiranju razvijeni su tzv. *asembleri*, programi koji prevode simboličke naredbe u strojne i pomažu pri adresiranju memorije. Stoga se takvi programski jezici nazivaju i asemblerskim jezicima.

Prvi viši programski jezik bio je FORTRAN (od engleskog *FORmula TRANslator*), razvijen 1954. Još neki od ranih programskih jezika iz tog doba su LISP, Cobol, Algol i Basic. Da bi se programi pisani u tim jezicima mogli izvoditi na računalu, potrebno ih je najprije prevesti na strojni jezik. To se radi s pomoću posebnih programa prevoditelja ili kompilatora (eng. *compiler*). Jedna naredba višeg programskog jezika tipično se prevodi u više strojnih naredbi.

U 70-tim godinama 20. stoljeća razvijeni su mnogi specijalizirani programski jezici. Programski jezik Pascal razvijen je za učenje strukturiranog programiranja. Programski jezik C originalno je razvijen za sistemsko programiranje, zajedno s operacijskim sustavom Unix. Oba su postala vrlo popularna i C se do danas koristi za mnoge druge svrhe. Iz

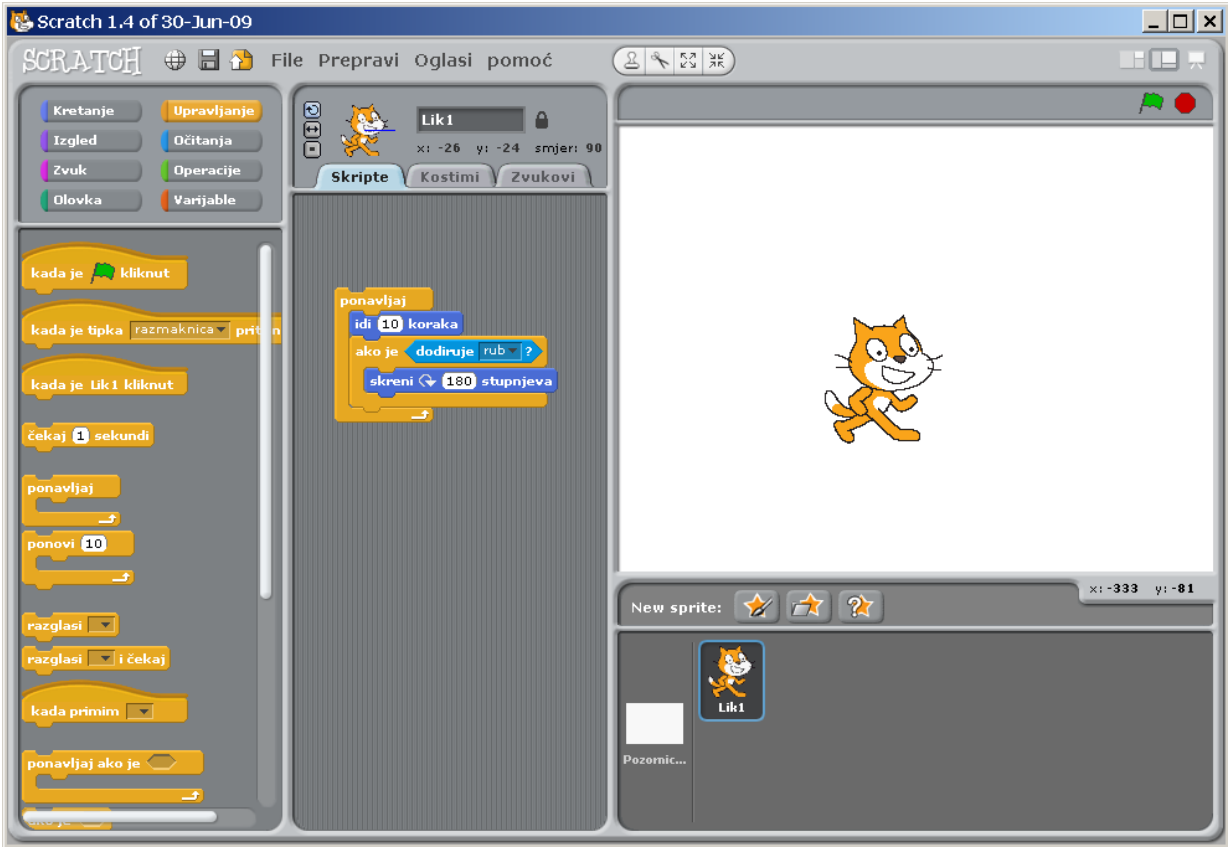
[1] Augusta Ada King, vojvotkinja od Lovelacea, bila je matematičarka i kći poznatog pjesnika lorda Byrona.

tog razdoblja potječu i prvi objektno orijentirani programski jezici Simula i Smalltalk.

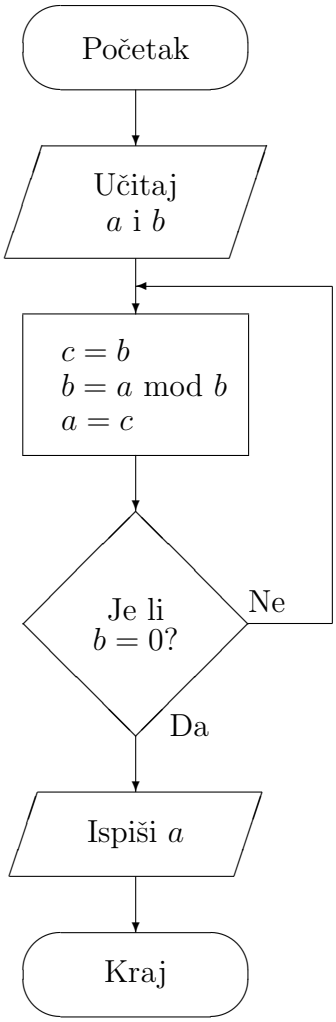
Razvoj novih programskih jezika i novih inačica postojećih nastavljen je u 80-tim i 90-tim godinama. Godine 1983. definiran je već spomenuti programski jezik Ada, s ciljem standardizacije softvera pisanog za Ministarstvo obrane Sjedinjenih američkih država. Iste godine razvijen je C++, objektno orijentirana inačica jezika C. Godine 1987. razvijen je skriptni jezik Perl, a 1991. Python. Takvi jezici ne prevode se s pomoću kompilatora, nego ih izvodi takozvani *interpreter*, program koji paralelno prevodi i izvodi naredbe. Također, 1991. godine razvijen je jezik Java koji je postao popularan za pisanje aplikacija za web. U 90-tim godinama sve je više programskih jezika namijenjenih za internet, naprimjer PHP razvijen 1995. godine.

Za buduće nastavnike važni su edukacijski programski jezici namijenjeni djeci. Jedan od najpoznatijih je LOGO, razvijen 1967. i poznat po takozvanoj "kornjačinoj grafici". Od 2007. godine slobodno je dostupan programski jezik Scratch [23] razvijen na MIT-u. Postoje inačice za razne operacijske sustave i na hrvatskom jeziku. Scratch također omogućuje kornjačinu grafiku, ali i puno više od toga. Zanimljiv je zato što se programira bez pisanja, slaganjem pravokutnika i okvira s naredbama (vidi sliku 3.1).

Vrlo rano uočena je potreba zapisivanja algoritama u jednostavnom i preglednom obliku, čitljivijem od kôda pisanog u nekom programskom jeziku. Jedan od načina su *dijagrami toka*, poput onog na slici 3.2. Dijagrami toka vizualno su atraktivni, ali zahti-



Slika 3.1: Programski jezik Scratch.



Slika 3.2: Primjer dijagrama toka.

jevaju puno prostora i zato ih nećemo koristiti u ovoj skripti. Algoritme ćemo zapisivati u takozvanom *pseudojeziku*.

Većina programskih jezika ima mnoštvo zajedničkih značajki. Ilustrirat ćemo ih na primjerima jezika FORTRAN, Pasca, C i Python. U ta četiri jezika, kao i u skoro svim ostalim, ulazni podaci, međurezultati i rezultati proračuna pohranjuju se u *varijablama*. Na početku programa potrebno je deklarirati varijable, tj. odrediti koji će se tip podataka u njima spremati (npr. cijeli brojevi – "integeri", realni brojevi, znakovi ili nizovi znakova i dr.). Programski jezik Python po tome je nešto drugačiji. U njemu se varijable ne deklariraju, ali ih je prije prve upotrebe potrebno *inicijalizirati*, tj. pridružiti im neku vrijednost. Time se ujedno određuje koji je tip podataka spremljen u pojedinoj varijabli. Deklaracija i inicijalizacija varijabli radi se u naša četiri "ogledna" programska jezika na sljedeći način.

Deklaracija / inicijalizacija varijabli i pridruživanje

FORTRAN

```
INTEGER m,n
REAL x,y
```

```
x = 3.14
y = SIN(x) - x**3
m = 49 / 21
n = MOD(49,21)
```

Pascal

```
var m,n:integer;
    x,y:real;
```

```
x := 3.14;
y := sin(x) - x*x*x;
m := 49 div 21;
n := 49 mod 21;
```

C

```
int m,n;
float x,y;
```

```
x = 3.14;
y = sin(x) - pow(x,3);
m = 49 / 21;
n = 49 % 21;
```

Python

```
m = n = 0
x = y = 0.0
```

```
x = 3.14
y = math.sin(x) - x**3
m = 49 / 21
n = 49 % 21
```

Ovdje smo ujedno vidjeli kako se varijablama pridružuju konstante ili rezultati nekih računskih operacija.

Kao što smo objasnili, svaki algoritam na početku uzima ulazne podatke, a na kraju vraća izlazne podatke koji predstavljaju rješenje problema. Zato svi programski jezici imaju naredbe za ulaz i za izlaz podataka.

Ulaz i izlaz podataka

FORTRAN

```
PRINT *,'Unesi prirodan broj'
READ *,n
```

Pascal

```
writeln('Unesi prirodan broj');
read(n);
```

C

```
printf("Unesi prirodan broj\n");
scanf("%d",&n);
```

Python

```
print("Unesi prirodan broj")
n = int(input())
```

Programi su nizovi naredbi i izvode se redom kojim su napisani. Često je potrebno dio programa izvesti ili ne izvesti, ovisno o nekom uvjetu. To nam omogućuje naredba za grananje.

Grananje

FORTRAN

```
IF ((n.EQ.0).AND.(m.NE.0)) THEN
  ⋮
ENDIF
```

Pascal

```
if (n=0 and m<>0) then
begin
 ⋮
end;
```

C

```
if (n==0 && m!=0)
{
⋮
}
```

Python

```
if n==0 and m!=0:
  naredba unutar if
     ⋮
naredba nakon if
```

Također, često je potrebno dio programa izvesti više puta. Zato u programskim jezicima postoje naredbe za ponavljanje, tzv. *petlje*.

Ponavljanje

FORTRAN

```
    DO 10 i=1,n
      ⋮
10  CONTINUE
```

```
10  IF (n.GT.0) THEN
       ⋮
      GOTO 10
    ENDIF
```

Pascal

```
for i:=1 to n do
begin
 ⋮
end;
```

```
while (n>0) do
begin
 ⋮
end;
```

C

```
for (i=0; i<n; ++i)
{
 ⋮
}
```

```
while (n>0)
{
⋮
}
```

Python

```
for i in range(n):
  naredba unutar for
     ⋮
naredba nakon for
```

```
while n>0:
  naredba unutar while
     ⋮
naredba nakon while
```

Naša četiri programska jezika dozvoljavaju indeksirane varijable, takozvane *nizove* ili *polja* (eng. *arrays*).

Nizovi

FORTRAN

```
    INTEGER a(0:99)

    DO 10 i=0,99
10     a(i) = 2*i+1
```

Pascal

```
var a:array [0..99] of integer;

for i:= 0 to 99 do
  a[i] := 2*i+1;
```

C

```
int a[100];

for (i=0; i<100; ++i)
  a[i] = 2*i+1;
```

Python

```
a = [0] * 100

a = [2*i+1 for i in range(100)]
```

Dijelovi programa mogu se izdvojiti u zasebne cjeline i kasnije pozivati preko imena. To su takozvani *potprogrami* ili *funkcije*.

Potprogrami

FORTRAN

```
    SUBROUTINE ispis(n)
    INTEGER n
    PRINT *,'Rezultat je ',n
    RETURN
    END
```

```
C   Aritmeticka sredina
    REAL FUNCTION sredina(x,y)
    REAL x,y
    sredina = (x+y)/2
    RETURN
    END
```

Pascal

```
procedure ispis(n:integer);
begin
  writeln('Rezultat je ',n);
end;
```

```
(* Aritmeticka sredina *)
function sredina(x,y:real):real;
begin
  sredina := (x+y)/2;
end;
```

C

```
void ispis(int n)
{ printf("Rezultat je %d\n",n);
}
```

Python

```
def ispis(n):
   print("Rezultat je", n)
```

C

```
/* Aritmeticka sredina */
float sredina(float x, float y)
{ return (x+y)/2;
}
```

Python

```
# Aritmeticka sredina
def sredina(x,y):
   return (x+y)/2.0
```

Pseudojezik objedinjuje zajedničke značajke ovih četiriju i mnogih drugih programskih jezika. Omogućuje nam manje formalno zapisivanje algoritama, prilagođeno ljudima umjesto računalima. Nadedbe ćemo pisati na hrvatskom, slično kao u zapisu algoritma za zbrajanje iz prvog poglavlja i algoritma iz dokaza teorema 2.2. U idućem poglavlju preciznije ćemo opisati pseudojezik i koristit ćemo ga u nastavku za zapisivanje raznih algoritama. Programe pisane u pseudojeziku lako je prevesti na neki pravi programski jezik. Savjetujemo čitateljima da to rade paralelno s čitanjem ove skripte.

Zadaci

Zadatak 3.1. *Istražite što radi algoritam prikazan dijagramom toka na slici 3.2.*

Zadatak 3.2. *Zapišite s pomoću dijagrama toka algoritam za zbrajanje iz prvog poglavlja te algoritam iz dokaza teorema 2.2.*

Zadatak 3.3. *U programskom jeziku Pascal napišite program "Hello world", koji ispisuje poruku na ekranu:*

```
PROGRAM hello(input, output);
BEGIN
  writeln('Hello, world!');
END.
```

Prevedite ga s pomoću nekog kompilatora za Pascal i pokrenite. Besplatno dostupni kompilatori za Pascal su npr. Free Pascal [7] i GNU Pascal [11].

Zadatak 3.4. *Napišite program "Hello world" u programskom jeziku C :*

```
#include <stdio.h>
main()
{ printf("Hello, world!\n");
}
```

Prevedite ga s pomoću nekog kompilatora za C i pokrenite. Besplatno dostupni kompilatori za C su npr. GCC [8] i Code::Blocks [4].

Zadatak 3.5. *Napišite program "Hello world" u programskom jeziku Python i pokrenite ga s pomoću odgovarajućeg interpretera:*

```
print("Hello, world!")
```

Interpreteri i dokumentacija za Python besplatno su dostupni na web stranici [22].

Zadatak 3.6. *Instalirajte programski jezik Scratch [23] i poigrajte se s njime. Natjerajte macu da govori "Hello, world", mijauče, hoda po ekranu i crta s pomoću kornjačine grafike.*

Poglavlje 4

Pseudojezik

Da bismo algoritam mogli izvoditi na računalu, moramo ga kodirati u nekom programskom jeziku. Kôd mora biti potpuno u skladu sa sintaksom izabranog formalnog jezika i algoritam mora biti do kraja precizno opisan, čak i detalji koji nam se čine jasnima iz konteksta.

Ako je naš algoritam namijenjen za čitanje ljudima, možemo ga zapisati na manje formalan način. Možemo si dozvoliti slobodniju sintaksu i detaljno raspisati samo bitne dijelove algoritma. Takav način zapisivanja algoritama naziva se *pseudojezik* ili *pseudokôd*.

Pseudojezik se koristi u mnogim knjigama i člancima koji se bave algoritmima, ali zapravo ne postoji standardna verzija pseudojezika. Ovisno o kontekstu koristi se sažetiji ili detaljniji način zapisivanja algoritama. Dijelove algoritma možemo zamijeniti pojedinačnim naredbama, opisati ih prirodnim jezikom, ili ih raspisati do u detalje kao u pravom programskom jeziku. U ovom poglavlju upoznat ćemo se sa stilom pisanja pseudokoda koji ćemo koristiti u nastavku skripte. U pravilu ćemo algoritme zapisivati dosta detaljno, tako da ih se lako može prevesti na neki pravi programski jezik.

Algoritam, odnosno računalni program, sastoji se od niza naredbi koje se izvode redom kojim su napisane. U pseudojeziku za naredbe koristimo hrvatske riječi. Zapisujemo ih uspravnim slovima, jednu ispod druge. Ponekad je potrebno grupirati nekoliko uzastopnih naredbi u jednu cjelinu, što činimo uglatim zagradama:

$$\left[\begin{array}{l} \text{prva naredba} \\ \text{druga naredba} \\ \text{treća naredba} \end{array}\right.$$

Ulazni podaci, međurezultati i rezultati proračuna pohranjuju se u varijablama. Varijable ćemo označavati s jednim ili više *kosih slova*. Slično kao u programskom jeziku Python, nećemo deklarirati tip podataka koji spremamo u varijable.

Spomenuli smo da algoritam uzima ulazne podatke i vraća izlazne podatke, koji predstavljaju rješenje. Za ulaz i izlaz podataka u pseudojeziku koristimo naredbe 'učitaj' i 'ispiši'.

Primjer 4.1. *Program "Hello world" u pseudojeziku izgleda ovako:*

ispiši `"Hello, world!"`

Ovaj program doslovno ispisuje tekst u navodnicima. Ako iza naredbe 'ispiši' stavimo ime varijable (bez navodnika), neće se ispisati ime nego sadržaj pohranjen u varijabli. Iza naredbe za ulaz podataka uvijek stoji ime jedne ili više varijabli u koje će se učitani podaci spremiti.

Primjer 4.2. *Program koji učitava dva prirodna broja i ispisuje njihov zbroj:*

učitaj m, n
ispiši $m + n$

Ovdje nismo raspisivali algoritam za zbrajanje prirodnih brojeva opisan u prvom poglavlju, nego smo samo napisali izraz $m + n$. Programski jezici sadrže gotovo rješenje za zbrajanje prirodnih brojeva, a isto ćemo pretpostavljati za pseudojezik.

Jedna od najvažnijih naredbi pseudojezika i pravih programskih jezika je naredba koja nekoj varijabli pridružuje određenu vrijednost. U primjerima algoritama iz prethodnih poglavlja već smo se susreli s pridruživanjem. Označavali smo ga uglavnom znakom jednakosti '='. Od sada ćemo pridruživanje uvijek označavati znakom '$\leftarrow$', da bismo ga razlikovali od uspoređivanja.

Na lijevoj strani naredbe za pridruživanje '$\leftarrow$' stoji ime varijable. Na desnoj strani može biti konstanta, ime varijable ili izraz sastavljen od konstanti i varijabli s pomoću aritmetičkih, logičkih i znakovnih operacija. Izraz na desnoj strani se evaluira: imena varijabli se zamjenjuju sa sadržajem pohranjenim u tim varijablama i izračunavaju se sve operacije. Konačna vrijednost se pohranjuje u varijablu napisanu s lijeva.

Primjer 4.3.

$a \leftarrow 2$
$b \leftarrow a + 3$
$c \leftarrow a \cdot b$
ispiši c

Prva naredba pohranjuje u varijablu a broj 2*. Druga naredba pohranjuje u varijablu b vijednost spremljenu u a uvećanu za* 3*, tj. broj* 5*. Treća naredba pohranjuje u c produkt vrijednosti spremljenih u a i b. Program na kraju ispisuje vrijednost spremljenu u c, tj.* 10*.*

Često je potrebno povećati vrijednost neke varijable. S time smo se već susreli u algoritmu iz dokaza teorema 2.2.

Primjer 4.4.

učitaj x
$x \leftarrow x + 1$
ispiši x

Program učitava broj u varijablu x, povećava vrijednost x za jedan i ispisuje novi x. Naprimjer, ako korisnik upiše broj 17*, program će ispisati* 18*.*

Vidimo da izraz s desne strane naredbe za pridruživanje može sadržati i varijablu koju smo naveli na lijevoj strani. Naprimjer, naredba $x \leftarrow 2x$ broj spremljen u x množi s 2 i rezultat pohranjuje u x. S lijeve strane naredbe za pridruživanje smije stajati isključivo

ime jedne varijable, nikada konstanta ili izraz. Pridruživanja $2 \leftarrow x$, $x+1 \leftarrow x$ i $x+y \leftarrow 2$ nemaju smisla. Osim brojeva, varijablama možemo pridruživati znakove, nizove znakova (stringove) i logičke vrijednosti.

Primjer 4.5. *Ovaj program također ispisuje poruku "Hello, world!".*

poruka ← `"Hello, world!"`
ispiši *poruka*

Primjer 4.6. *Program ispisuje ISTINA ako korisnik unese pozitivan broj, a u suprotnom ispisuje LAŽ.*

učitaj *br*
poz ← *br* > 0
ispiši *poz*

Argumenti operatora uspoređivanja $>$, $<$, $\leq$, $\geq$, $=$ i $\neq$ su brojevi ili izrazi kojima je vrijednost broj, a rezultat tih operacija je logička vrijednost ISTINA ili LAŽ. Logičkim operatorima 'i', 'ili', 'ne' argumenti i rezultat su logičke vrijednosti. Još jednom naglašavamo razliku između naredbe za pridruživanje '←' i operatora uspoređivanja '='.

Primjer 4.7. *Opišite što rade sljedeće naredbe. Koje od njih nemaju smisla?*

$$uvjet \leftarrow (x < -1) \text{ ili } (x > 1)$$

$$i = i + 1$$

$$nula \leftarrow x^2 - x - 6 = 0$$

$$a \leftarrow (x + 2) \text{ i } (y - 7)$$

$$b \leftarrow (x \geq 0) + (y \geq 0)$$

Rješenje. Prva naredba u varijablu *uvjet* sprema logičku vrijednost izraza '$(x < -1)$ ili $(x > 1)$'. Vrijednost je ISTINA ako je u varijabli x pohranjen broj koji je manji od -1 ili veći od 1, a u suprotnom je LAŽ.

U drugom retku naveden je logički izraz, a ne naredba. Sam za sebe taj izraz nema smisla. U sklopu neke naredbe izraz bi uvijek poprimio vrijednost LAŽ, bez obzira na sadržaj varijable i. Naime, i ne može biti jednak $i + 1$. Ako želimo povećati sadržaj varijable i za 1, trebamo napisati naredbu za pridruživanje $i \leftarrow i + 1$.

Treća naredba pohranjuje u varijablu *nula* logičku vrijednost ISTINA ili LAŽ ovisno o tome je li u x pohranjena nultočka polinoma $x^2 - x - 6$ ili nije.

Četvrta i peta naredba nemaju smisla. Na desnoj strani četvrte naredbe je logička operacija 'i' primijenjena na aritmetičke izraze $x+2$ i $y-7$, a u petoj naredbi je aritmetička operacija '+' primijenjena na logičke izraze $x \geq 0$ i $y \geq 0$. Napominjemo da se u nekim programskim jezicima (npr. C i Python) umjesto logičkih vrijednosti ISTINA i LAŽ koriste brojevi 1 i 0, pa bi tamo peta naredba imala smisla. □

Primjer 4.8. *Napišite program koji učitava neke vrijednosti u varijable x i y, zamjenjuje sadržaj tih varijabli i ispisuje ih.*

Rješenje. Ako korisnik unese redom vrijednosti 5, 7, traženi program treba ispisati 7, 5. Sljedeći program ponaša se identično, ali ipak ne predstavlja rješenje zadatka.

učitaj x, y
ispiši y, x

Naime, u varijablama su i dalje pohranjene vrijednosti $x = 5$, $y = 7$. Niti ovaj program ne radi ispravno:

učitaj x, y
$x \leftarrow y$
$y \leftarrow x$
ispiši x, y

Nakon prvog pridruživanja obje varijable sadrže broj 7, a broj 5 je izgubljen. Program će ispisati 7, 7. Rješenje je da prije pridruživanja $x \leftarrow y$ broj koji je bio u x spremimo u pomoćnu varijablu. Na kraju ćemo varijabli y pridružiti broj iz pomoćne varijable.

učitaj x, y
$pom \leftarrow x$
$x \leftarrow y$
$y \leftarrow pom$
ispiši x, y

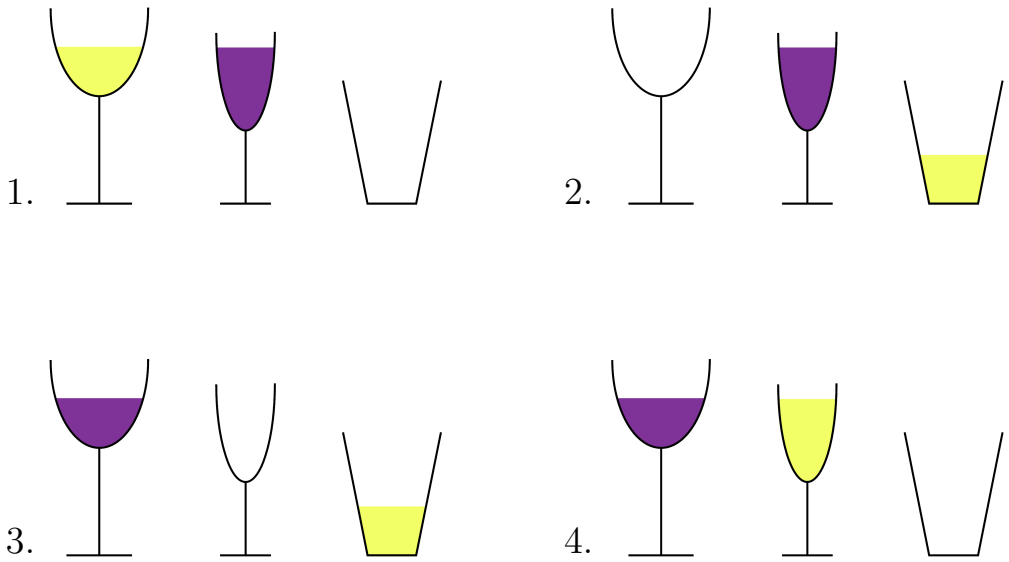
□

Zamjena varijabli česta je i važna operacija. Koristit ćemo je npr. u algoritmima za sortiranje. Možemo si je predočiti preko problema zamjene dviju vrsti pića ulivenih u pogrešne čaše. Zamislimo da smo crno vino ulili u čašu za bijelo vino, a bijelo u čašu za crno. Za zamijenu pića koristimo treću, pomoćnu čašu (vidi sliku 4.1). Analogija ipak nije potpuna – kad u čašu B prelijemo piće iz čaše A, čaša A ostane prazna. S druge strane, nakon pridruživanja $b \leftarrow a$ obje varijable a, b sadrže vrijednost koja je bila u a.

U dosadašnjim primjerima algoritama naredbe se izvode točno onim redom kojim su napisane. Često je potrebno utjecati na tijek izvođenja algoritma. U programskim jezicima i u pseudojeziku za to služi naredba za grananje, koja je oblika

ako je *logički izraz* onda
[*prvi niz naredbi*
inače
[*drugi niz naredbi*

Naredba za grananje evaluira logički izraz. Ako je njegova vrijednost ISTINA, izvodi se prvi niz naredbi, a u suprotnom se izvodi drugi niz naredbi. Cijeli drugi dio ('inače' i odgovarajući niz naredbi) može se izostaviti. Ključne riječi *ako je ... onda ... inače* na engleskom glase $if \ldots then \ldots else$, zato se naredba za grananje ponekad naziva if naredbom.



Slika 4.1: Zamjena pića ulivenih u pogrešne čaše.

Primjer 4.9. *Program za rješavanje kvadratne jednadžbe* $ax^2 + bx + c = 0$.

učitaj a, b, c
ako je $a \neq 0$ onda
  $d \leftarrow b^2 - 4ac$
  ako je $d > 0$ onda
    $x_1 \leftarrow (-b + \sqrt{d})/(2a)$
    $x_2 \leftarrow (-b - \sqrt{d})/(2a)$
    ispiši `"Rješenja su "`, x_1, x_2
  inače
    ako je $d = 0$ onda
      $x_1 \leftarrow -b/(2a)$
      ispiši `"Dvostruko rješenje je "`, x_1
    inače
      ispiši `"Nema realnih rješenja"`
inače
  ako je $b \neq 0$ onda
    $x \leftarrow -c/b$
    ispiši `"Rješenje je "`, x
  inače
    ako je $c = 0$ onda
      ispiši `"Svi realni brojevi su rješenja"`
    inače
      ispiši `"Nema rješenja"`

Ovaj program predstavlja detaljni zapis u pseudojeziku načina na koji izračunavamo

formulu

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Raspisali smo sve moguće slučajeve (diskriminanta pozitivna, nula ili negativna, vodeći koeficijent nula).

Još jednu važnu grupu naredbi koje utječu na tijek izvođenja programa čine naredbe za ponavljanje, takozvane petlje (eng. *loops*). Pretpostavimo da želimo ispisati prvih 100 prirodnih brojeva. Bez naredbe za ponavljanje morali bismo napisati 100 naredbi za ispis:

ispiši 1
ispiši 2
ispiši 3
⋮

Elegantnije rješenje sastoji se od petlje koja nekoj varijabli redom pridružuje brojeve od 1 do 100 i ispisuje vrijednost te varijable:

za $i = 1, \ldots, 100$ ponavljaj
[ispiši i

Ovaj oblik naredbe za ponavljanje naziva se for petljom, prema engleskom zapisu

for $i = 1, \ldots, 100$ do
[⋮

Varijabla koja poprima vrijednosti od 1 do 100 naziva se *kontrolnom varijablom*. Naravno, moguće je staviti neku drugu donju i gornju granicu za kontrolnu varijablu umjesto 1 i 100. Niz naredbi koje se ponavljaju zvat ćemo *tijelom petlje*. U tijelu for petlje nije potrebno povećavati vrijednost kontrolne varijable – pretpostavljamo da se to događa automatski.

Postoji i drugi oblik naredbe za ponavljanje, takozvana while petlja:

dok je *logički izraz* ponavljaj
[⋮

ili, na engleskom:

while *logički izraz* do
[⋮

Tijelo while petlje se ponavlja sve dok je logički izraz istinit. Kad bismo s pomoću while petlje željeli ispisati brojeve od 1 do 100, morali bismo sami inicijalizirati i povećavati kontrolnu varijablu:

$i \leftarrow 1$
dok je $i \leq 100$ ponavljaj
[ispiši i
[$i \leftarrow i + 1$

Za neke probleme praktičnija je for petlja, a za druge while petlja.

Primjer 4.10. *Napišite program koji učitava prirodan broj n i ispisuje kvadrate prvih n prirodnih brojeva.*

Rješenje. Ovaj zadatak lakše je riješiti s for petljom, jer ne moramo brinuti o kontrolnoj varijabli:

učitaj n
za $i = 1, \ldots, n$ ponavljaj
[ispiši i^2

□

Primjer 4.11. *Napišite program koji učitava prirodan broj n i ispisuje prirodne brojeve iz skupa $\{1, \ldots, n\}$ koji su kvadrati nekog prirodnog broja.*

Rješenje. Nespretno rješenje bila bi for petlja s kontrolnom varijablom $i = 1, \ldots, n$. U tijelu petlje provjeravali bismo je li i kvadrat prirodnog broja i ispisivali ga ako jest.

Spretnije je umjesto i ispisivati i^2, ali tada gornja granica za kontrolnu varijablu nije n. Petlja treba ići do najvećeg prirodnog broja i za koji je $i^2 \leq n$. Taj uvjet možemo napisati u while petlji.

učitaj n
$i \leftarrow 1$
dok je $i^2 \leq n$ ponavljaj
[ispiši i^2
[$i \leftarrow i + 1$

□

Primjer 4.12. *Napišite program koji učitava* 100 *brojeva i ispisuje njihov zbroj.*

Rješenje. Koristimo pomoćnu varijablu s kojoj na početku pridružimo 0. Svaki puta kad učitamo broj x, pribrojimo ga pomoćnoj varijabli: $s \leftarrow s + x$ (pridružimo varijabli s staru vrijednost od s uvećanu za x). Na kraju s sadrži zbroj svih učitanih brojeva.

$s \leftarrow 0$
za $i = 1, \ldots 100$ ponavljaj
[učitaj x
[$s \leftarrow s + x$
ispiši s

□

Primjer 4.13. *Napišite program koji učitava prirodne brojeve sve dok se ne učita nula. Program treba ispisati produkt učitanih prirodnih brojeva.*

Rješenje. U ovom zadatku ne znamo unaprijed koliko će se brojeva učitati, pa se koristimo while petljom. Produkt izračunavamo slično kao sumu: pomoćnu varijablu p inicijaliziramo na 1 i množimo je učitanim brojevima.

$p \leftarrow 1$
učitaj x
dok je $x \neq 0$ ponavljaj
[$p \leftarrow p \cdot x$
[učitaj x
ispiši p

Pretpostavimo da korisnik upiše redom brojeve 2, 5, 3, 0. Primijetimo da se prvi broj učitava prije ulaza u petlju. Kada bi prvi učitani broj bio 0, tijelo petlje uopće se ne bi izvodilo, a program bi ispisao "prazan produkt" 1. U našem slučaju program će učitati $x = 2$ i prvi puta izvesti tijelo petlje. Varijabli p se pridružuje $p \cdot x = 1 \cdot 2 = 2$ i učitava se sljedeći broj, $x = 5$. Sada se ponovo provjerava uvjet $x \neq 0$, koji je istinit, i drugi puta izvodi tijelo petlje. Varijabli p se pridružuje $2 \cdot 5 = 10$ i učitava se $x = 3$. U trećem prolazu kroz petlju p postaje 30 i učitava se 0. Sada uvjet $x \neq 0$ više nije istinit, pa se izlazi iz petlje. Na kraju se ispisuje varijablu p, tj. produkt učitanih brojeva 30. □

Primjer 4.14. *Napišite program koji učitava prirodan broj n i nakon toga n brojeva. Program treba ispisati koliko među učitanim brojevima ima negativnih.*

Rješenje. U ovom primjeru pomoćnu varijablu koristimo kao brojač. Na početku je inicijaliziramo na 0 i povećavamo je za 1 svaki puta kad korisnik unese negativan broj.

učitaj n
$br \leftarrow 0$
za $i = 1, \ldots, n$ ponavljaj
⎡ učitaj x
⎢ ako je $x < 0$ onda
⎣ [$br \leftarrow br + 1$
ispiši br

Pokušajte sami detaljno opisati rad programa ako korisnik upiše $n = 5$ i niz brojeva 2, -1, 0, -7, 14. □

Primjer 4.15. *Napišite program koji učitava brojeve sve dok se ne učita nula. Program treba ispisati aritmetičku sredinu učitanih brojeva (osim nule).*

Rješenje. Aritmetička sredina jednaka je sumi učitanih brojeva podijeljenoj s brojem učitanih brojeva:

$$A = \frac{x_1 + x_2 + \ldots + x_n}{n}.$$

Koristit ćemo dvije pomoćne varijable – prva prebrojava učitane brojeve, a druga izračunava sumu.

$n \leftarrow 0$
$s \leftarrow 0$
učitaj x
dok je $x \neq 0$ ponavljaj
⎡ $n \leftarrow n + 1$
⎢ $s \leftarrow s + x$
⎣ učitaj x
ako je $n > 0$ onda
[ispiši s/n
inače
[ispiši `"Nije učitan niti jedan broj"`

Prije ispisa aritmetičke sredine provjeravamo je li učitan barem jedan broj, da bismo izbjegli dijeljenje s nulom. □

Primjer 4.16. *Napišite program koji učitava n brojeva i ispisuje najveći od učitanih brojeva.*

Rješenje. U ovom programu koristit ćemo pomoćnu varijablu max u kojoj pamtimo najveći od do sada učitanih brojeva. Ako je sljedeći učitani broj veći od max, pridružimo ga varijabli max:

za $i = 1 \ldots, n$ ponavljaj
⎡ učitaj x
⎢ ako je $x > max$ onda
⎣ [$max \leftarrow x$

Varijablu max moramo prije ulaza u petlju inicijalizirati. Ako znamo da će korisnik upisati pozitivne brojeve, možemo max postaviti na 0. Međutim, nije isključeno da će svi upisani brojevi biti negativni. Tada bi program ispisao 0, što nije najveći od učitanih brojeva. Rješenje je da prvi broj učitamo direktno u max, a nakon toga idućih $n - 1$ brojeva učitavamo u petlji. Osim toga na početku programa treba učitati n, a na kraju ispisati max.

učitaj n
učitaj max
za $i = 1 \ldots, n - 1$ ponavljaj
⎡ učitaj x
⎢ ako je $x > max$ onda
⎣ [$max \leftarrow x$
ispiši max

□

Naredbe koje smo do sada upoznali dovoljne su za zapisivanje svih algoritama kojima ćemo se baviti u ovoj skripti. Naša verzija pseudojezika ima samo šest naredbi: 'učitaj', 'ispiši', pridruživanje '$\leftarrow$', grananje ('ako...onda...inače'), for petlju i while petlju. U šestom poglavlju upoznat ćemo se pobliže s indeksiranim varijablama (nizovima, odnosno poljima), koje smo već susreli u uvodnim primjerima algoritma za zbrajanje i algoritma iz dokaza teorema 2.2.

U mnogim programskim jezicima osim opisanih naredbi postoji mogućnost definiranja funkcija, odnosno potprograma (vidi primjere na kraju prethodnog poglavlja). Funkcije i potprogrami potrebni su za elegantno zapisivanje rekurzivnih algoritama. Time se u ovoj skripti nećemo baviti, pa ne uvodimo u pseudojezik sintaksu za funkcije i potprograme.

Zadaci

Zadatak 4.1. *Objasnite razliku između pridruživanja '$\leftarrow$' i uspoređivanja '$=$'.*

Zadatak 4.2. *Ako se na ulazu unesu brojevi $a = 15$ i $b = 21$, što će ispisati sljedeći*

program?

učitaj a, b
$a \leftarrow a - b$
$b \leftarrow 2\,a + 1$
$a \leftarrow 3\,b - 7$
ispiši $a \cdot b$

Zadatak 4.3. *Istražite što radi sljedeći program.*

učitaj x, y
$x \leftarrow x - y$
$y \leftarrow x + y$
$x \leftarrow y - x$
ispiši x, y

Zadatak 4.4. *Nadopunite program za rješavanje kvadratne jednadžbe (primjer 4.9) tako da ispisuje kompleksna rješenja ako je diskriminanta negativna.*

Zadatak 4.5. *Napišite program za rješavanje sustava od dvije linearne jednadžbe s dvije nepoznanice*

$$\begin{aligned} a\,x + b\,y &= e \\ c\,x + d\,y &= f \end{aligned} .$$

Program treba raditi ispravno u svim mogućim slučajevima (jedinstveno rješenje, beskonačno mnogo rješenja, bez rješenja).

Zadatak 4.6. *Za zadani n, napišite formulu za najveći prirodan broj i za koji je $i^2 \leq n$. Riješite zadatak iz primjera 4.11 s pomoću for petlje.*

Zadatak 4.7. *Napišite program koji učitava prirodan broj n i ispisuje prvih n parnih prirodnih brojeva.*

Zadatak 4.8. *Napišite program koji učitava prirodan broj n i ispisuje prvih n prirodnih brojeva koji pri dijeljenju s 3 daju ostatak 1.*

Zadatak 4.9. *Napišite program koji učitava prirodan broj n i ispisuje sve brojeve iz skupa $\{1, 2, \ldots, n\}$ koji su djeljivi s 5.*

Zadatak 4.10. *Napišite program koji učitava prirodan broj n i ispisuje sve brojeve iz skupa $\{1, 2, \ldots, n\}$ koji su parni, ali nisu djeljivi s 4.*

Zadatak 4.11. *Napišite program koji učitava n prirodnih brojeva i ispisuje sumu svih učitanih brojeva koji su parni.*

Zadatak 4.12. *Napišite program koji učitava brojeve sve dok se ne učita nula. Program treba ispisati produkt svih negativnih učitanih brojeva.*

Zadatak 4.13. *Napišite program koji učitava cijele brojeve sve dok se ne učita nula. Program treba ispisati koliko među učitanim brojevima ima neparnih.*

Zadatak 4.14. *Napišite program koji učitava brojeve sve dok se ne učita nula. Program treba ispisati geometrijsku sredinu učitanih brojeva (osim nule):*

$$G = \sqrt[n]{x_1 \cdot x_2 \cdots x_n}.$$

Zadatak 4.15. *Riješite prethodni zadatak za kvadratnu sredinu*

$$K = \sqrt{\frac{x_1^2 + x_2^2 + \ldots + x_n^2}{n}}$$

i harmonijsku sredinu

$$H = \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \ldots + \frac{1}{x_n}}.$$

Zadatak 4.16. *Napišite program koji učitava brojeve sve dok se ne učita nula. Program treba ispisati najmanji pozitivan broj od učitanih. Ako nije učitan niti jedan pozitivan broj, program treba ispisati poruku o tome.*

Zadatak 4.17. *Napišite program koji učitava cijele brojeve sve dok se ne učita nula. Program treba ispisati najveći neparni učitani broj. Ako nije učitan niti jedan neparni broj, program treba ispisati poruku o tome.*

Zadatak 4.18. *Niz* Fibonaccijevih brojeva *definiran je rekurzivno. Prva dva člana niza su* $F_1 = F_2 = 1$*, a svaki sljedeći je zbroj dva prethodna:* $F_n = F_{n-1} + F_{n-2}$*, za* $n \geq 2$*. Napišite program koji učitava* n *i ispisuje prvih* n *članova tog niza.*

Zadatak 4.19. *Napišite program koji učitava prirodan broj* n *i ispisuje sve Fibonaccijeve brojeve iz skupa* $\{1, \ldots, n\}$*.*

Zadatak 4.20. *Napišite program koji učitava prirodan broj* n *i ispisuje* n*-ti Fibonaccijev broj* F_n*.*

Zadatak 4.21. *Napišite program koji učitava prirodan broj* n *i ispisuje najveći Fibonaccijev broj koji je manji ili jednak od* n*.*

Zadatak 4.22. *Napišite program koji učitava prirodan broj* n *i ispisuje najmanji Fibonaccijev broj koji je veći ili jednak od* n*.*

Poglavlje 5

Neki algoritmi za prirodne brojeve

U ovom poglavlju upoznat ćemo neke algoritme koji rade s prirodnim brojevima. Počinjemo s problemima o znamenkama zadanog prirodnog broja. Ako želimo ispisati znamenke jednu po jednu, koristimo algoritam koji smo upoznali u drugom poglavlju:

učitaj n
dok je $n \neq 0$ ponavljaj
$\left[\right.$ ispiši n mod 10
$\left.\right[$ $n \leftarrow n$ div 10

Ovaj algoritam ispisuje dekadske znamenke, ali ga jednostavno možemo modificirati tako da radi s bilo kojom bazom b. Primijetimo da se znamenke ispisuju s desna na lijevo, tj. od namanje značajne prema značajnijima (prvo znamenka jedinice, zatim desetice, stotice itd.). Koristili smo while petlju jer ne znamo unaprijed koliko će biti znamenaka.

Osnovnu petlju po znamenkama zadanog prirodnog broja sad ćemo koristiti za probleme kakve smo imali u prethodnom poglavlju – prebrojavanje, izračunavanje sume, traženje maksimuma...

Primjer 5.1. *Napišite program koji učitava prirodan broj n i ispisuje koliko n ima znamenaka.*

Rješenje. Modificiramo prethodni algoritam tako da se umjesto ispisivanja znamenaka povećava brojač.

učitaj n
$br \leftarrow 0$
dok je $n \neq 0$ ponavljaj
$\left[\right.$ $br \leftarrow br + 1$
$\left.\right[$ $n \leftarrow n$ div 10
ispiši br

Ako smijemo koristiti logaritamsku funkciju i funkciju "najveće cijelo", problem možemo riješiti bez petlje.

učitaj n
ispiši $\lfloor \log n \rfloor + 1$

□

Primjer 5.2. *Napišite program koji učitava prirodan broj n i ispisuje sumu njegovih znamenaka.*

Rješenje.

učitaj n
$s \leftarrow 0$
dok je $n \neq 0$ ponavljaj
$\left[\begin{array}{l} s \leftarrow s + (n \bmod 10) \\ n \leftarrow n \text{ div } 10 \end{array}\right.$
ispiši s

□

Primjer 5.3. *Napišite program koji učitava prirodan broj n i ispisuje njegovu najveću znamenku.*

Rješenje.

učitaj n
$max \leftarrow 0$
dok je $n \neq 0$ ponavljaj
$\left[\begin{array}{l} z \leftarrow n \bmod 10 \\ \text{ako je } z > max \text{ onda } max \leftarrow z \\ n \leftarrow n \text{ div } 10 \end{array}\right.$
ispiši max

□

Primjer 5.4. *Napišite program koji učitava prirodan broj n i definira prirodan broj m koji se sastoji od istih znamenaka kao n, ali u obrnutom redoslijedu. Naprimjer, ako korisnik unese $n = 123$, program treba ispisati $m = 321$.*

Rješenje.

učitaj n
$m \leftarrow 0$
dok je $n \neq 0$ ponavljaj
$\left[\begin{array}{l} z \leftarrow n \bmod 10 \\ m \leftarrow m \cdot 10 + z \\ n \leftarrow n \text{ div } 10 \end{array}\right.$
ispiši m

□

Iduća grupa algoritama radi s djeliteljima zadanog prirodnog broja. Ako želimo ispisati sve djelitelje od n, najprirodnije rješenje je for petlja od 1 do n:

učitaj n
za $d = 1, \ldots, n$ ponavljaj
$\left[\right.$ ako je $n \bmod d = 0$ onda ispiši d

Broj d je djelitelj od n ako je ostatak pri dijeljenju n sa d jednak nuli, tj. ako vrijedi $n \bmod d = 0$. Slijedi nekoliko primjera koji se rješavaju s pomoću petlje po svim djeliteljima.

Primjer 5.5. *Napišite program koji učitava prirodan broj n i ispisuje koliko n ima djelitelja.*

Rješenje.

učitaj n
$br \leftarrow 0$
za $d = 1, \ldots, n$ ponavljaj
$\lfloor$ ako je $n \bmod d = 0$ onda $br \leftarrow br + 1$
ispiši br

□

Primjer 5.6. *Napišite program koji učitava prirodan broj n i ispisuje sumu svih djelitelja od n.*

Rješenje.

učitaj n
$s \leftarrow 0$
za $d = 1, \ldots, n$ ponavljaj
$\lfloor$ ako je $n \bmod d = 0$ onda $s \leftarrow s + d$
ispiši s

□

Primjer 5.7. *Napišite program koji učitava prirodan broj n i ispisuje njegov najmanji djelitelj veći od 1 i najveći djelitelj manji od n.*

Rješenje. Najmanji djelitelj svakog prirodnog broja je 1, a najveći on sam. Zato smo ta dva djelitelja isključili iz razmatranja. Za ovaj problem dat ćemo elegantnije rješenje od for petlje po svim djeliteljima s pomoćnim varijablama min i max. Takva petlja pronalazi djelitelje redom od manjih prema većima. Prvi kojeg pronađe nakon 1 je traženi najmanji djeljitelj, pa možemo odmah izaći iz petlje i ispisati ga.

učitaj n
$d \leftarrow 2$
dok je $n \bmod d \neq 0$ ponavljaj $d \leftarrow d + 1$
ispiši d

Primijetimo da je najmanji djelitelj d uvijek prost broj. U suprotnom bismo d mogli napisati kao $d = a \cdot b$ za neke $a, b < d$, pa bi a i b bili još manji djelitelji od n. Za najveći djelitelj koristimo silaznu petlju:

učitaj n
$d \leftarrow n - 1$
dok je $n \bmod d \neq 0$ ponavljaj $d \leftarrow d - 1$
ispiši d

Program možemo učiniti efikasnijim tako da petlja kreće od n div 2 umjesto od $n - 1$:

učitaj n
$d \leftarrow n$ div 2
dok je $n \bmod d \neq 0$ ponavljaj $d \leftarrow d - 1$
ispiši d

Naime, niti jedan broj između n div 2 i n ne može biti djelitelj od n. Ako želimo ispisati najmanji i najveći djelitelj, ne moramo imati obje petlje (uzlaznu i silaznu). Najveći djeljitelj očito je jednak broju n podijeljenim s najmanjim djeliteljem.

učitaj n
$d \leftarrow 2$
dok je $n \bmod d \neq 0$ ponavljaj $d \leftarrow d + 1$
ispiši d
ispiši n div d

□

Primjer 5.8. *Napišite program koji učitava prirodne brojeve m i n te ispisuje sve njihove zajedničke djelitelje.*

Rješenje.

učitaj m, n
za $d = 1, \ldots, m$ ponavljaj
[ako je $(m \bmod d = 0)$ i $(n \bmod d = 0)$ onda ispiši d

□

Primjer 5.9. *Napišite program koji učitava prirodne brojeve m i n te ispisuje njihov najveći zajednički djelitelj.*

Rješenje.

učitaj m, n
$d \leftarrow m$
dok je $(m \bmod d \neq 0)$ ili $(n \bmod d \neq 0)$ ponavljaj $d \leftarrow d - 1$
ispiši d

□

Prethodni primjer rješili smo slično kao primjer 5.7. Ponuđeno rješenje je vrlo jednostavno; korektnost slijedi direktno iz definicije najvećeg zajedničkog djelitelja. Za problem najvećeg zajedničkog djelitelja postoji manje trivijalan, ali efikasniji i ljepši algoritam. To je poznati **Euklidov algoritam**:

učitaj m, n
dok je $n \neq 0$ ponavljaj
[$pom \leftarrow n$
[$n \leftarrow m \bmod n$
[$m \leftarrow pom$
ispiši m

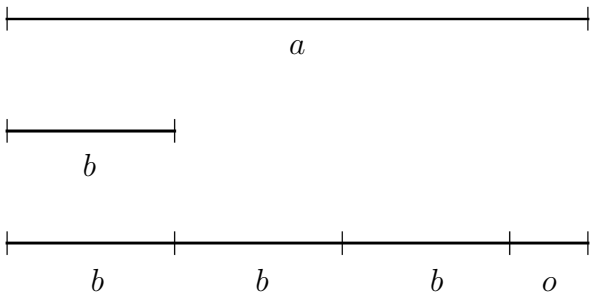
Algoritam učitava prirodne brojeve m i n. Pretpostavljamo da je prvi broj veći ili jednak od drugog, $m \geq n$. Petlja se izvodi dok je manji broj različit od nule. U tijelu petlje varijabli n se pridružuje ostatak pri dijeljenju m sa n, a varijabli m stara vrijednost od n. Primijetimo da je prethodno stara vrijednost od n pohranjena u pomoćnu varijablu pom, slično kao kod zamjene varijabli. Kad varijabla n postane nula, algoritam izlazi iz petlje i ispisuje m.

Primjer 5.10. *Opišimo rad Euklidova algoritma za ulaz* $m = 30$, $n = 21$. *U prvom prolazu kroz petlju u varijablu pom se pohranjuje* 21, *varijabla n postaje* $30 \bmod 21 = 9$, *a u m se prepisuje broj pohranjen u pom. Vrijednosti varijabli su tada* $m = 21$, $n = 9$. *Budući da n nije nula, drugi puta se izvodi tijelo petlje. Vrijednosti varijabli postaju* $m = 9$, $n = 21 \bmod 9 = 3$. *U trećem prolazu kroz petlju varijable postaju* $m = 3$, $n = 9 \bmod 3 = 0$. *Tada algoritam izlazi iz petlje i ispisuje* $m = 3$, *što je najveći zajednički djelitelj od* 30 *i* 21.

Korektnost Euklidova algoritma nije očita. Trebamo se uvjeriti da algoritam završava u konačno mnogo koraka za bilo koje brojeve m, n na ulazu te da je broj kojeg ispisuje na kraju zaista najveći zajednički djelitelj od m i n. Konačnost slijedi iz teorema o dijeljenju s ostatkom (teorem 2.3). U svakom koraku zamjenjujemo n s ostatkom pri dijeljenju m sa n, koji je strogo manji od n. Dobivamo strogo padajući niz nenegativnih cijelih brojeva, koji ima konačno mnogo članova prije nego što dođe do nule. Tada algoritam izlazi iz petlje, ispisuje m i završava s radom.

Najveći zajednički djelitelj označavat ćemo $\mathrm{NZM}(m, n)$ (prema "najveća zajednička mjera"). Ako je m djeljiv s n, očito je $\mathrm{NZM}(m, n) = n$. Ako m nije djeljiv s n, može se pokazati da vrijedi $\mathrm{NZM}(m, n) = \mathrm{NZM}(n, m \bmod n)$. Iz toga slijedi da je broj kojeg Euklidov algoritam ispisuje upravo $\mathrm{NZM}(m, n)$. Algoritam unutar petlje zamjenjuje m i n redom s n i $m \bmod n$. Pritom se ne mijenja najveći zajednički djelitelj brojeva pohranjenih u varijablama m i n. U zadnjem prolazu kroz petlju n postaje nula, a to znači da je prethodno m bio djeljiv s n. Ispisuje se vrijednost varijable m u zadnjem koraku, koja je jednaka vrijednosti n u prethodnom koraku. Tada je m bio djeljiv s n, pa je vrijedilo $n = \mathrm{NZM}(m, n)$. Dakle, Euklidov algoritam ispisuje $\mathrm{NZM}(m, n)$.

Najveći zajednički djelitelj naziva se i najvećom zajedničkom *mjerom*, a Euklidov algoritam duboko je povezan s problemom mjerenja. Pretpostavimo da su zadane dvije dužine duljina a i b. Brojevi a i b su pozitivni realni brojevi za koje pretpostavljamo $a > b$, tj. da je prva dužina veća od druge. "Izmjeriti prvu dužinu drugom" znači ispitati koliko puta druga dužina stane u prvu:



Slika 5.1: Mjerenje dužine a dužinom b.

Ako a nije cjelobrojni višekratnik od b, ostat će nam dio kraći od b ("ostatak"). Postupak možemo opisati sljedećim algoritmom, koji uzastopno oduzima b od a dok a ne

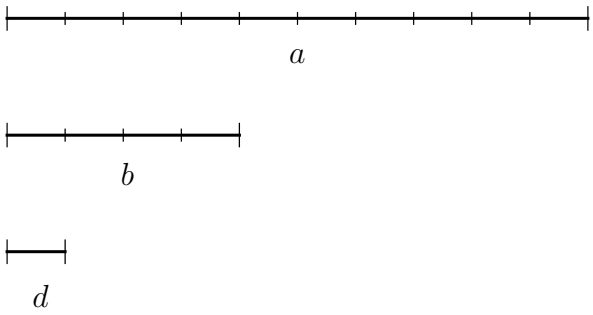
postane manji od b.

učitaj a, b
$k \leftarrow 0$
dok je $a \geq b$ ponavljaj
$\left[\begin{array}{l} a \leftarrow a - b \\ k \leftarrow k + 1 \end{array} \right.$
ispiši k, a

Algoritam ispisuje cijeli broj k, koji nam kaže koliko puta b stane u a, i duljinu ostatka. To je zapravo dokaz tvrdnje o egzistenciji iz sljedeće generalizacije teorema o dijeljenju s ostatkom.

Teorem 5.11. *Neka su $a, b > 0$ pozitivni realni brojevi. Tada postoje jedinstveni brojevi $k \in \mathbb{N}_0$ i $o \in \mathbb{R}$, $0 \leq o < b$ takvi da je $a = k \cdot b + o$.*

Problem mjerenja sastoji se u pronalaženju "zajedničke mjere" kojom možemo izmjeriti obje dužine bez ostatka. To je dužina duljine d takve da su a i b njezini cjelobrojni višekratnici: $a = m \cdot d$ i $b = n \cdot d$ za neke $m, n \in \mathbb{N}$. Tada je razlomak $\frac{m}{n}$ odgovor na pitanje "koliko puta b stane u a". Što je zajednička mjera d veća, to su brojnik i nazivnik manji i mjerenje nam je praktičnije. Zato tražimo najveću zajedničku mjeru.



Slika 5.2: Zajednička mjera d dužina a i b.

Euklidov algoritam je pokušaj da se problem mjerenja sustavno riješi. U prvom koraku mjerimo veću dužinu a manjom dužinom b. Ako ostatak o_1 nije nula, nastavljamo mjeriti manju dužinu b s ostatkom o_1. Ako ni u drugom koraku ostatak o_2 nije nula, mjerimo o_1 s ostatkom o_2 i tako dalje. Kada dobijemo ostatak nula, sve promatrane veličine su cjelobrojni višekratnici zadnjeg pozitivnog ostatka. Tako nalazimo najveću zajedničku mjeru dužina a i b.

Vidjeli smo da se jedno mjerenje svodi na uzastopno oduzimanje manje dužine od veće. Proširenje tog postupka daje nam drugu verziju Euklidova algoritma. Oduzimamo b od a dok a ne postane manji od b. Tada nastavljamo oduzimati a od b, i tako dalje sve dok se a i b ne izjednače.

učitaj a, b
dok je $a \neq b$ ponavljaj
⌈ ako je $a > b$ onda $a \leftarrow a - b$
⌊ inače $b \leftarrow b - a$
ispiši a

Ova verzija Euklidova algoritma zasniva se na oduzimanju. Za prirodne brojeve $a, b \in \mathbb{N}$ ispisuje isti rezultat kao prva verzija Euklidova algoritma, u kojoj smo računali ostatke pri dijeljenju. Algoritam s oduzimanjem obično treba više koraka od prve verzije algoritma, ali ga možemo primijeniti na pozitivne realne brojeve, a ne samo na prirodne brojeve.

Postavlja se pitanje hoće li algoritam završiti u konačno mnogo koraka za bilo koje realne brojeve $a, b > 0$ na ulazu? Ako algoritam pronađe najveću zajedničku mjeru d, onda je $\frac{a}{b} = \frac{m \cdot d}{n \cdot d} = \frac{m}{n}$ racionalan broj. Prema tome, ako je omjer brojeva na ulazu iracionalan, algoritam ne završava.

Definicija 5.12. *Za realne brojeve $a, b \neq 0$ kažemo da su* sumjerljivi *ako je njihov omjer $\frac{a}{b}$ racionalan broj. U suprotnom kažemo da su* nesumjerljivi.

Starogrčki matematičari bili su zapanjeni otkrićem nesumjerljivih veličina, naprimjer stranice i dijagonale kvadrata. Za takve dužine ne postoji zajednička mjera, a Euklidov algoritam izvodio bi se beskonačno. Naravno, svako mjerenje u stvarnom životu nužno uključuje pogrešku i kao rezultat daje aproksimaciju. Međutim, ovo pokazuje da čak ni idealizirano mjerenje ne može biti egzaktno.

Vratimo se algoritmima za prirodne brojeve. U sljedećim primjerima koristimo se Euklidovim algoritmom.

Primjer 5.13. *Napišite program koji učitava dva prirodna broja i ispisuje poruku o tome jesu li relativno prosti.*

Rješenje. Brojevi su relativno prosti ako je njihov najveći zajednički djelitelj 1. Izračunavamo ga Euklidovim algoritmom i uspoređujemo s 1:

učitaj m, n
dok je $n \neq 0$ ponavljaj
⌈ $pom \leftarrow n$
| $n \leftarrow m \bmod n$
⌊ $m \leftarrow pom$
ako je $m = 1$ onda
⌈ ispiši `"Brojevi su relativno prosti"`
inače
⌈ ispiši `"Brojevi nisu relativno prosti"`

□

Primjer 5.14. *Napišite program koji učitava brojnik i nazivnik razlomka i ispisuje ga u do kraja skraćenom obliku.*

Rješenje. Trebamo podijeliti brojnik i nazivnik njihovom najvećom zajedničkom mjerom. Budući da Euklidov algoritam mijenja vrijednosti varijabli m i n, pohranit ćemo početne vrijednosti u pomoćne varijable.

učitaj m, n
$a \leftarrow m$
$b \leftarrow n$
dok je $n \neq 0$ ponavljaj
⎡ $pom \leftarrow n$
⎢ $n \leftarrow m \bmod n$
⎣ $m \leftarrow pom$
ispiši a div m, `" / "`, b div m

□

Primjer 5.15. *Napišite program koji učitava dva prirodna broja i ispisuje njihov najmanji zajednički višekratnik.*

Rješenje. Najmanji zajednički višekratnik dobijemo tako da pomnožimo brojeve i podijelimo produkt s najvećim zajedničkim djeliteljem.

učitaj m, n
$p \leftarrow m \cdot n$
dok je $n \neq 0$ ponavljaj
⎡ $pom \leftarrow n$
⎢ $n \leftarrow m \bmod n$
⎣ $m \leftarrow pom$
ispiši p div m

□

Važan matematički pojam je pojam prostog broja. Za prirodan broj $n > 1$ kažemo da je *prost* ako su mu jedini djelitelji 1 i n. Možemo reći da je n prost ako ima točno dva djelitelja. Brojeve s više od dva djelitelja nazivamo *složenim*, a broj 1 nije niti prost niti složen. Sljedeći algoritam učitava n i provjerava je li prost.

učitaj n
$br \leftarrow 0$
za $d = 1, \ldots, n$ ponavljaj
⎡ ako je $n \bmod d = 0$ onda $br \leftarrow br + 1$
ako je $br = 2$ onda
⎡ ispiši `"Broj je prost"`
inače
⎡ ispiši `"Broj nije prost"`

Algoritam prebrojava djelitelje od n. Ako ima točno dva djelitelja onda je prost, a u suprotnom je $n = 1$ ili je n složen broj. Čim nađemo prvi djelitelj između 1 i n, znamo da je riječ o složenom broju. Ovaj algoritam napisan je s for petljom i nastavlja prebrojavati

djelitelje. S while petljom možemo napisati efikasniji algoritam, koji će prekinuti potragu kad nađe prvi djelitelj. U primjeru 5.7 već smo spomenuli da brojevi između (n div 2) i n ne mogu biti djelitelji, pa ih ne moramo provjeravati.

učitaj n
ako je $n = 1$ onda
⌊ ispiši `"Broj nije ni prost ni složen"`
inače
⌈ $d \leftarrow 2$
| dok je ($n \bmod d \neq 0$) i ($d \leq n$ div 2) ponavljaj $d \leftarrow d + 1$
| ako je $d \leq n$ div 2 onda
| ⌊ ispiši `"Broj je složen"`
| inače
⌊ ⌊ ispiši `"Broj je prost"`

Algoritam možemo učiniti još efikasnijim tako da potragu za djeliteljima ograničimo do $\sqrt{n}$. Naime, može se pokazati da najmanji djelitelj između 1 i n ne može biti veći od $\sqrt{n}$ (ako postoji).

učitaj n
ako je $n = 1$ onda
⌊ ispiši `"Broj nije ni prost ni složen"`
inače
⌈ $d \leftarrow 2$
| dok je ($n \bmod d \neq 0$) i ($d^2 \leq n$) ponavljaj $d \leftarrow d + 1$
| ako je $d^2 \leq n$ onda
| ⌊ ispiši `"Broj je složen"`
| inače
⌊ ⌊ ispiši `"Broj je prost"`

Primjer 5.16. *Ako korisnik unese $n = 17$, prebrojimo za koliko brojeva d prethodna tri algoritma provjeravaju jesu li djelitelji od n. Prvi algoritam provjerava sve brojeve od 1 do n, dakle napravi ukupno 17 provjera. Drugi algoritam traži djelitelje od 2 do 17 div 2 = 8, dakle napravi 7 provjera. Treći algoritam provjerava brojeve $d \geq 2$ koji zadovoljavaju $d^2 \leq 17$, tj. brojeve od 2 do 4. Nakon samo tri provjere ispravno zaključuje da je 17 prost broj.*

Prema osnovnom teoremu aritmetike, svaki prirodan broj može se prikazati kao produkt prostih brojeva. Prikaz je jedinstven do na poredak faktora. Nije teško napisati algoritam koji učitava prirodan broj n i ispisuje njegove proste faktore. Koristit ćemo primjedbu iz primjera 5.7 prema kojoj je najmanji djelitelj veći od 1 nužno prost. Dok je $n > 1$, ispisivat ćemo najmanji djelitelj d i podijeliti n sa d. Tako dobivamo proste faktore od n u rastućem redoslijedu. Pritom ne moramo provjeravati jesu li brojevi koje

ispisujemo prosti.

učitaj n
$d \leftarrow 2$
dok je $n > 1$ ponavljaj
  ako je $n \bmod d = 0$ onda
    ispiši d
    $n \leftarrow n \operatorname{div} d$
  inače $d \leftarrow d + 1$

Primjer 5.17. *Opišimo kako prethodni algoritam nalazi proste faktore broja* $n = 84$. *Pri ulazu u petlju vrijednosti varijabli su* $n = 84$, $d = 2$. *Uvjet* $n \bmod d = 0$ *je ispunjen, pa se ispisuje* 2 *i* n *postaje* 42. *Pri drugom prolazu kroz petlju uvjet je i dalje ispunjen. Algoritam još jednom ispisuje* 2 *i postavlja* n *na* 21. *Taj broj više nije djeljiv s* 2*; pri trećem prolazu kroz petlju algoritam povećava* d *za jedan. Vrijednosti varijabli kod četvrtog ulaza u petlju su* $n = 21$, $d = 3$. *Algoritam ispisuje* 3 *i postavlja* n *na* 7. *Zatim redom povećava* d *od* 4 *do* 6. *Broj* $n = 7$ *nije djeljiv s tim brojevima, pa se ništa ne ispisuje. Tek kad* d *postane* 7*, ispisuje se* 7 *i* n *postavlja na* 1. *Sada više nije ispunjen uvjet* $n > 1$. *Algoritam izlazi iz petlje i završava s radom. Ispisao je redom brojeve* 2, 2, 3, 7*, a to su prosti faktori od* 84.

U ovom poglavlju nastojali smo razviti efikasne algoritme, koji završavaju s radom u što manje koraka. Budući da moderna računala izvode operacije fantastičnim brzinama, postavlja se pitanje je li to uopće bitno? Pretpostavimo da se u jednoj sekundi izvodi milijardu (10^9) koraka algoritma. To je usporedivo s mogućnostima današnjih procesora, iako se u stvarnosti svi koraci ne izvode jednakom brzinom. Razmotrit ćemo koliko vremena treba za tri karakteristična problema: izračunavanje najvećeg zajedničkog djelitelja, provjera je li broj prost i rastavljanje na proste faktore. Na ulazu ćemo pretpostavljati velike prirodne brojeve s oko 50 znamenaka, tj. reda veličine 10^{50}.

U primjeru 5.9 dali smo prvi algoritam za najveći zajednički djelitelj. Ako na ulazu imamo relativno proste brojeve reda veličine 10^{50}, algoritam smanjuje varijablu d od 10^{50} do 1 i tek tada ispisuje najveći zajednički djelitelj 1. Dakle, treba mu oko 10^{50} koraka. To znači da bi izvođenje algoritma trajalo oko 10^{41} sekundi, tj. oko $3.2 \cdot 10^{33}$ godina. Za usporedbu, starost svemira procjenjuje se na $1.4 \cdot 10^{10}$ godina (prema [26]).

Brojevi s 50 znamenaka ipak nisu izvan dosega današnjih računala. Euklidov algoritam vrlo brzo nalazi najveći zajednički djelitelj čak i tako velikih brojeva. Netrivijalan je, ali dugo poznat rezultat [16] da Euklidovom algoritmu treba najviše koraka ako su na ulazu uzastopni Fibonaccijevi brojevi. Točnije, najmanji brojevi za koje Euklidov algoritam n puta prolazi kroz petlju su F_{n+2} i F_{n+1}. Prva dva Fibonaccijeva broja veća od 10^{50} su F_{241} i F_{242}. Euklidov algoritam izračunava njihov najveći zajednički djelitelj nakon 240 prolaza kroz petlju. Ako jedan prolaz brojimo kao tri koraka (zbog tri pridruživanja unutar petlje), algoritam završava za $7.2 \cdot 10^{-7}$ sekundi, tj. u vremenu kraćem od mikrosekunde (milijuntog dijela sekunde). U stvarnosti za operacije s tako velikim brojevima treba više vremena nego što smo pretpostavili, ali cijeli račun zaista završava u djeliću sekunde.

Za algoritam koji provjerava je li prirodan broj prost, najgori slučaj je kad zadamo prost broj. Prva verzija algoritma na str. 46 provjerava sve brojeve iz skupa $\{1, \ldots, n\}$. Dakle, za broj s 50 znamenaka treba oko 10^{50} koraka i vrijeme izvođenja je $3.2 \cdot 10^{33}$

godina. Efikasnija verzija algoritma na str. 47 provjerava brojeve do $\sqrt{n}$. Za prost broj s 50 znamenaka treba oko 10^{25} koraka, ili $3.2 \cdot 10^8$ godina. To je kraće od starosti svemira, ali je također duže nego što bismo željeli čekati.

Postoje efikasni algoritmi za provjeru prostosti, koji brojeve s 50 znamenaka rješavaju brzo. Prvi takav algoritam u smislu kojeg koristimo u ovoj skripti opisan je 2004. godine u članku [1]. Od ranije su poznati randomizirani algoritmi i algoritmi kojima korektnost ovisi o nedokazanim hipotezama o prostim brojevima (vidi [29]). Ti su algoritmi relativno komplicirani i nećemo se njima baviti.

Za problem rastavljanja zadanog prirodnog broja na proste faktore do danas nije poznat niti jedan efikasan algoritam. Naš algoritam na str. 48 za prost broj reda veličine 10^{50} treba vrijeme dulje od starosti svemira. Poznati su znatno efikasniji algoritmi, ali niti jedan na današnjim računalima ne može brzo faktorizirati brojeve s 200 znamenaka. Općenito, najgori slučaj za faktorizaciju su brojevi koji su produkt dvaju velikih prostih brojeva.

Računski problemi s velikim prirodnim brojevima nisu samo od teorijskog značaja. Neki od kriptografskih algoritama koji se koriste u praksi za bankovne transakcije putem interneta i drugo zasnivaju se upravo na problemima koje smo razmatrali. Oni ne bi bili mogući da ne možemo brzo računati najveću zajedničku mjeru i provjeravati prostost, a sigurnost im se zasniva na nemogućnosti faktorizacije velikih prirodnih brojeva.

U sljedećem poglavlju opisat ćemo kako se procjenjuje efikasnost algoritama bez računanja vremena izvođenja za konkretne ulazne vrijednosti.

Zadaci

Zadatak 5.1. *Napišite program koji učitava prirodne brojeve n i $b \geq 2$ te ispisuje znamenke broja n u bazi b.*

Zadatak 5.2. *Napišite program koji učitava prirodne brojeve n i $b \geq 2$ te ispisuje aritmetičku sredinu znamenaka broja n u bazi b.*

Zadatak 5.3. *Napišite program koji učitava prirodne brojeve n i $b \geq 2$ te ispisuje koliko znamenaka različitih od nule ima u zapisu broja n u bazi b.*

Zadatak 5.4. *Napišite program koji učitava prirodan broj n i ispisuje bazu b u kojoj n ima najviše znamenaka različitih od nule.*

Zadatak 5.5. *Napišite program koji učitava prirodne brojeve n i $b \geq 2$ te ispisuje znamenke broja n u bazi b s lijeva na desno, tj. od najznačajnije prema manje značajnima.*

Zadatak 5.6. *Napišite program koji učitava prirodan broj n i ispisuje produkt svih djelitelja od n.*

Zadatak 5.7. *Napišite program koji učitava prirodan broj n i ispisuje aritmetičku sredinu svih djelitelja od n.*

Zadatak 5.8. *Dokažite: za svaki prirodan broj n, najmanji broj $d > 1$ koji dijeli n je prost.*

Zadatak 5.9. *Dokažite: za svaki prirodan broj n, brojevi između* $(n \operatorname{div} 2)$ *i n nisu djelitelji od n.*

Zadatak 5.10. *Istražite što će se dogoditi ako u Euklidov algoritam na str. 42 upišemo brojeve $m < n$.*

Zadatak 5.11. *Dokažite: ako je prirodan broj n djelitelj od m, onda vrijedi* $\mathrm{NZM}(m, n) = n$.

Zadatak 5.12. *Dokažite: ako prirodan broj n nije djelitelj od m, onda vrijedi* $\mathrm{NZM}(m, n) = \mathrm{NZM}(n, m \bmod n)$.

Zadatak 5.13. *Dokažite teorem 5.11.*

Zadatak 5.14. *Opišite rad verzije Euklidova algoritma s oduzimanjem (na str. 45) za ulaz $m = 30$, $n = 21$. Koliko prolaza kroz petlju treba tom algoritmu, a koliko Euklidovu algoritmu na str. 42?*

Zadatak 5.15. *Dokažite da su duljina stranice i duljina dijagonale kvadrata nesumjerljivi brojevi.*

Zadatak 5.16. *Napišite program koji učitava dva razlomka i ispisuje njihov produkt u do kraja skraćenom obliku.*

Zadatak 5.17. *Napišite program koji učitava dva razlomka i ispisuje njihov zbroj u do kraja skraćenom obliku.*

Zadatak 5.18. *Neka je* $\mathrm{NZM}(m, n)$ *najveći zajednički djelitelj, a* $\mathrm{NZV}(m, n)$ *najmanji zajednički višekratnik brojeva m i n. Dokažite da za sve prirodne brojeve m, n vrijedi* $\mathrm{NZM}(m, n) \cdot \mathrm{NZV}(m, n) = m \cdot n$.

Zadatak 5.19. *Napišite program koji učitava prirodne brojeve sve dok se ne učita nula. Program treba ispisati najveći zajednički djelitelj svih učitanih brojeva (osim nule).*

Zadatak 5.20. *Eulerova funkcija $\varphi(n)$ definira se kao broj prirodnih brojeva $k \leq n$ koji su relativno prosti s n. Napišite program koji učitava prirodan broj n i ispisuje $\varphi(n)$.*

Zadatak 5.21. *Neka je n složen broj. Ako je d njegov najmanji djelitelj veći od* 1*, dokažite da je $d \leq \sqrt{n}$.*

Zadatak 5.22. *Napišite program koji učitava prirodan broj n i ispisuje sve proste brojeve iz skupa $\{1, \ldots, n\}$.*

Zadatak 5.23. *Napišite program koji učitava prirodan broj n i ispisuje prvih n prostih brojeva.*

Zadatak 5.24. *Napišite program koji učitava prirodan broj n i ispisuje n-ti po redu prost broj.*

Zadatak 5.25. *Napišite program koji učitava prirodan broj n i ispisuje najmanji prost broj $p \geq n$.*

Zadatak 5.26. *Napišite program koji učitava prirodan broj $n \geq 2$ i ispisuje najveći prost broj $p \leq n$.*

Zadatak 5.27. *Prisjetite se algoritma za računanje najvećeg zajedničkog djelitelja iz osnovne škole: brojeve m i n rastavimo na proste faktore, tražimo zajedničke proste faktore i pomnožimo ih. Pokušajte precizno zapisati taj algoritam! Bi li taj algoritam bio efikasan za velike prirodne brojeve m i n?*

Poglavlje 6

Nizovi i sortiranje

Nizovi ili polja (eng. *arrays*) su indeksirane varijable. Često je u algoritmu potrebno pamtiti veći broj podataka, koji može ovisiti o instanci problema kojeg rješavamo. Tada ih nije praktično pohraniti u varijable "fiksnih imena" a, b, $c, \ldots$, nego ih pohranjujemo u niz varijabli $a_1, a_2, \ldots, a_n$. Duljina niza n također može biti varijabla.

U uvodnom poglavlju već smo se susreli s nizovima varijabli. Algoritam za zbrajanje prirodnih brojeva kao ulaz je uzimao dva niza znamenaka pribrojnika, a izlaz je bio niz znamenaka zbroja. Označavali smo ih $a_1, \ldots, a_n$, $b_1, \ldots, b_n$ i $c_1, \ldots, c_{n+1}$. Ubuduće ćemo indekse nizova pisati u uglatim zagradama: $a[1], \ldots, a[n]$, $b[1], \ldots, b[n]$ i $c[1], \ldots, c[n+1]$. Takva notacija koristi se u mnogim programskim jezicima (vidi primjere na str. 24).

U klasičnim programskim jezicima kao što su FORTRAN, Pascal i C potrebno je deklarirati tip podataka koji se spremaju u niz (npr. cijeli brojevi, realni brojevi ili znakovi) i maksimalnu duljinu niza. Računalo rezervira odgovarajuću količinu memorije za pohranjivanje podataka u nizu, obično na uzastopnim memorijskim lokacijama. U Pythonu je interna reprezentacija nizova drugačija i moguće je tijekom izvođenja programa rezervirati dodatnu memoriju za podatke u nizu. Na razini pseudojezika nećemo se baviti takvim detaljima. Pretpostavljat ćemo da nizovi mogu imati po volji velik (konačan) broj članova i nećemo im deklarirati tip i maksimalnu duljinu.

Još jedan detalj koji ovisi o programskom jeziku je početni indeks niza. U nekim programskim jezicima nizovi su indeksirani od 1, tj. niz duljine n sastoji se od članova $a[1], \ldots, a[n]$. U drugim programskim jezicima početni indeks je 0, a niz duljine n sadrži članove $a[0], \ldots, a[n-1]$. U pseudojeziku ćemo nekad koristiti jednu, a nekad drugu konvenciju.

Susreli smo se s algoritmima koji na ulazu uzimaju niz brojeva (primjeri 4.12–4.16). Svi dosadašnji primjeri probleme su rješavali obrađujući članove niza onim redom kojim se unose. U bilo kojem trenutku izvođenja programa u varijablama je bio pohranjen samo jedan član niza, ili najviše dva uzastopna člana u algoritmima s Fibonaccijevim brojevima (zadaci 4.18–4.22). Ako želimo ispisati podatke u obrnutom redoslijedu od unešenog, ili ih algoritam obrađuje u nekom drugom redoslijedu, potrebno je istovremeno pamtiti sve podatke u nizu varijabli.

Primjer 6.1. *Program koji učitava prirodan broj n i niz od n brojeva te ih ispisuje u obrnutom redoslijedu.*

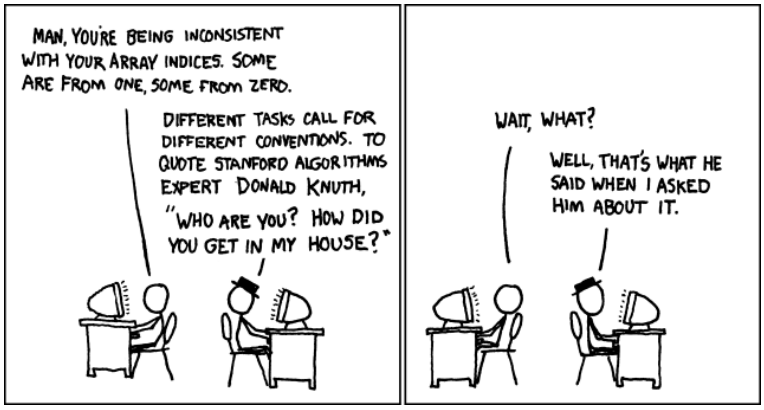
Rješenje. Ako indeksi niza počinju od 1, program izgleda ovako:

učitaj n
za $i = 1, \ldots, n$ ponavljaj učitaj $a[i]$
za $i = 0, \ldots, n-1$ ponavljaj ispiši $a[n-i]$

Ovo je rješenje ako niz počinje s indeksom 0:

učitaj n
za $i = 0, \ldots, n-1$ ponavljaj učitaj $a[i]$
za $i = 1, \ldots, n$ ponavljaj ispiši $a[n-i]$

□



Slika 6.1: Počinju li indeksi nizova od 1 ili od 0?[1]

U sljedećih nekoliko primjera nizove ćemo indeksirati od 1. U idućem primjeru duljina niza se ne učitava na početku, nego se zadaje unošenjem dogovorenog podatka (nule) koji označava kraj niza.

Primjer 6.2. *Program koji učitava brojeve i sprema ih u niz sve dok se ne učita nula. Na kraju ispisuje duljinu niza, prvi i zadnji član niza.*

Rješenje.

$n \leftarrow 0$
učitaj x
dok je $x \neq 0$ ponavljaj
  $n \leftarrow n + 1$
  $a[n] \leftarrow x$
  učitaj x
ispiši n, $a[1]$, $a[n]$

□

[1]Strip je preuzet s web stranice `http://xkcd.com/163/`

Algoritmi iz primjera 5.1–5.4 rade sa znamenkama zadanog prirodnog broja. Redoslijed kojim dolaze do znamenaka i obrađuju ih ide od najmanje značajne prema značajnijima. Ako znamenke želimo ispisati u prirodnom redoslijedu, od najznačajnije prema manje znajajnima, možemo ih pohraniti u niz.

Primjer 6.3. *Program učitava prirodne brojeve n i $b \geq 2$. Sprema znamenke broja n u bazi b u niz i ispisuje ih od najznačajnije prema manje značajnima.*

Rješenje.

učitaj n, b
$k \leftarrow 0$
dok je $n \neq 0$ ponavljaj
$\left[\begin{array}{l} k \leftarrow k + 1 \\ z[k] \leftarrow n \bmod b \\ n \leftarrow n \operatorname{div} b \end{array}\right.$
za $i = 0, \ldots, k-1$ ponavljaj ispiši $z[k-i]$

□

Važno je razlikovati članove niza $a[1], \ldots, a[n]$ od njihovih indeksa $1, \ldots, n$. Iduća tri primjera ilustriraju tu razliku.

Primjer 6.4. *Program učitava niz od n cijelih brojeva i ispisuje sve parne članove niza.*

Rješenje.

učitaj n
za $i = 1, \ldots, n$ ponavljaj učitaj $a[i]$
za $i = 1, \ldots, n$ ponavljaj
$\left[\right.$ ako je $a[i] \bmod 2 = 0$ onda ispiši $a[i]$

□

Ako korisnik unese niz brojeva 4, 7, 2, 6, 9, 3, 10, prethodni program ispisuje 4, 2, 6, 10.

Primjer 6.5. *Program učitava niz od n cijelih brojeva i ispisuje indekse svih parnih članova niza.*

Rješenje.

učitaj n
za $i = 1, \ldots, n$ ponavljaj učitaj $a[i]$
za $i = 1, \ldots, n$ ponavljaj
$\left[\right.$ ako je $a[i] \bmod 2 = 0$ onda ispiši i

□

Za isti ulaz 4, 7, 2, 6, 9, 3, 10, ovaj program ispisuje 1, 3, 4, 7.

Primjer 6.6. *Program učitava niz od n cijelih brojeva i ispisuje sve članove niza s parnim indeksom.*

Rješenje.

učitaj n
za $i = 1, \ldots, n$ ponavljaj učitaj $a[i]$
$i \leftarrow 2$
dok je $i \leq n$ ponavljaj
⌈ ispiši $a[i]$
⌊ $i \leftarrow i + 2$

□

Ovaj program za ulaz 4, 7, 2, 6, 9, 3, 10 ispisuje brojeve 7, 6, 3. U problemima u kojima se traži najveći član niza često je važan ne samo maksimum, nego i mjesto na kojem se on nalazi u nizu.

Primjer 6.7. *Program učitava niz od n brojeva, ispisuje najveći član niza i indeks mjesta na kojem se on nalazi.*

Rješenje.

učitaj n
za $i = 1, \ldots, n$ ponavljaj učitaj $a[i]$
$max \leftarrow a[1]$
$imax \leftarrow 1$
za $i = 2, \ldots, n$ ponavljaj
⌈ ako je $a[i] > max$ onda
⌈ ⌈ $max \leftarrow a[i]$
⌊ ⌊ $imax \leftarrow i$
ispiši `"Najveći član je"`, max
ispiši `"Nalazi se na mjestu"`, $imax$

□

Točnije, ako se najveći broj javlja nekoliko puta u nizu, ovaj program ispisuje prvi po redu od indeksa na kojima se nalazi. Naprimjer, ako je niz 3, 10, 4, 8, 10, 7, program ispisuje broj 10 i indeks 2. Program iz idućeg primjera ispisuje sve indekse na kojima se nalazi najveći broj, tj. u ovom slučaju indekse 2 i 5.

Primjer 6.8. *Program učitava niz od n brojeva, ispisuje najveći član niza i indekse svih mjesta na kojima se on nalazi.*

Rješenje.

učitaj n
za $i = 1, \ldots, n$ ponavljaj učitaj $a[i]$
$max \leftarrow a[1]$
za $i = 2, \ldots, n$ ponavljaj
⌊ ako je $a[i] > max$ onda $max \leftarrow a[i]$
ispiši `"Najveći član je"`, max
ispiši `"Nalazi se na mjestima:"`
za $i = 1, \ldots, n$ ponavljaj
⌊ ako je $a[i] = max$ onda ispiši i

□

Sljedećih nekoliko primjera ilustrira kako možemo promijeniti niz, tj. zamijeniti redoslijed nekih njegovih članova.

Primjer 6.9. *Program učitava niz od n brojeva, zamjenjuje prvi i zadnji član niza i ispisuje promijenjeni niz.*

Rješenje.

učitaj n
za $i = 1, \ldots, n$ ponavljaj učitaj $a[i]$
$pom \leftarrow a[1]$
$a[1] \leftarrow a[n]$
$a[n] \leftarrow pom$
za $i = 1, \ldots, n$ ponavljaj ispiši $a[i]$

□

Ako korisnik unese niz 3, 10, 4, 8, 9, 7, prethodni program ispisuje 7, 10, 4, 8, 9, 3.

Primjer 6.10. *Program učitava niz od n brojeva i "okreće" ga tako da na prvo mjesto dođe zadnji član, na drugo mjesto predzadnji član itd. Program ispisuje promijenjeni niz.*

Rješenje.

učitaj n
za $i = 1, \ldots, n$ ponavljaj učitaj $a[i]$
za $i = 1, \ldots, (n \text{ div } 2)$ ponavljaj
$\left[\begin{array}{l} pom \leftarrow a[i] \\ a[i] \leftarrow a[n+1-i] \\ a[n+1-i] \leftarrow pom \end{array}\right.$
za $i = 1, \ldots, n$ ponavljaj ispiši $a[i]$

□

Za niz 3, 10, 4, 8, 9, 7 na ulazu ovaj program ispisuje 7, 9, 8, 4, 10, 3. Važno je da petlja koja mijenja niz ide samo do n div 2, a ne do n. Petlja do n okrenula bi niz dva puta, pa bi se ispisao isti niz koji je učitan.

Primjer 6.11. *Program učitava niz od n brojeva i "rotira" ga za jedno mjesto ulijevo: na prvo mjesto dolazi drugi član niza, na drugo mjesto treći član itd. Prvi član niza dolazi na zadnje mjesto. Program ispisuje promijenjeni niz.*

Rješenje.

učitaj n
za $i = 1, \ldots, n$ ponavljaj učitaj $a[i]$
$pom \leftarrow a[1]$
za $i = 1, \ldots, n-1$ ponavljaj $a[i] \leftarrow a[i+1]$
$a[n] \leftarrow pom$
za $i = 1, \ldots, n$ ponavljaj ispiši $a[i]$

□

Ako korisnik unese niz 3, 10, 4, 8, 9, 7, ovaj program ispisuje 10, 4, 8, 9, 7, 3.

Primjer 6.12. *Program učitava niz od n brojeva, pronalazi najveći član i zamjenjuje ga s prvim članom niza. Program ispisuje promijenjeni niz.*

Rješenje.

učitaj n
za $i = 1, \ldots, n$ ponavljaj učitaj $a[i]$
$max \leftarrow a[1]$
$imax \leftarrow 1$
za $i = 2, \ldots, n$ ponavljaj
⌈ ako je $a[i] > max$ onda
⌊ ⌈ $max \leftarrow a[i]$
⌊ ⌈ $imax \leftarrow i$
$pom \leftarrow a[1]$
$a[1] \leftarrow a[imax]$
$a[imax] \leftarrow pom$
za $i = 1, \ldots, n$ ponavljaj ispiši $a[i]$

□

Najveći član niza 5, 9, 4, 10, 8, 7 je $max = 10$ i nalazi se na mjestu $imax = 4$. Nakon zamjene s prvim članom program ispisuje niz 10, 9, 4, 5, 8, 7.

Primjer 6.13. *Program učitava niz od n brojeva, pronalazi najveći član niza i prebacuje ga na prvo mjesto. Broj na prvom mjestu prebacuje na drugo mjesto, drugi broj na treće mjesto i tako dalje sve do mjesta na kojem je najveći član. Program ispisuje promijenjeni niz.*

Rješenje.

učitaj n
za $i = 1, \ldots, n$ ponavljaj učitaj $a[i]$
$max \leftarrow a[1]$
$imax \leftarrow 1$
za $i = 2, \ldots, n$ ponavljaj
⌈ ako je $a[i] > max$ onda
⌊ ⌈ $max \leftarrow a[i]$
⌊ ⌈ $imax \leftarrow i$
$pom \leftarrow a[imax]$
za $i = 0, \ldots, imax - 2$ ponavljaj $a[imax - i] \leftarrow a[imax - i - 1]$
$a[1] \leftarrow pom$
za $i = 1, \ldots, n$ ponavljaj ispiši $a[i]$

□

Ako je na ulazu niz 5, 9, 4, 10, 8, 7, ovaj program ispisuje 10, 5, 9, 4, 8, 7.

Za niz brojeva $a[1], \ldots, a[n]$ kažemo da je *rastući* ili *uzlazni* ako za sve parove indeksa $i < j$ vrijedi $a[i] \leq a[j]$. Niz je *padajući* ili *silazni* ako za sve parove ineksa $i < j$ vrijedi $a[i] \geq a[j]$. Da bismo po definiciji provjerili je li zadani niz rastući ili padajući,

uspoređujemo članove $a[i]$ i $a[j]$ za sve parove indeksa $i < j$. Za to nam trebaju dvije petlje smještene jedna unutar druge:

za $i = 1, \ldots, n-1$ ponavljaj
⎡ za $j = i+1, \ldots, n$ ponavljaj
⎣ [ispiši i, j

Takve petlje nazivamo *ugniježdenima*. Naprimjer, ako je $n = 4$, vanjska petlja s kontrolnom varijablom i izvodi se od 1 do 3. Za $i = 1$ unutrašnja petlja s kontrolnom varijablom j izvodi se od 2 do 4. Za $i = 2$ unutrašnja petlja mijenja j od 3 do 4, a za $i = 3$ tijelo unutrašnje petlje izvede se samo jednom, za $j = 4$. Rezultat je da program ispisuje redom parove $(1, 2)$, $(1, 3)$, $(1, 4)$, $(2, 3)$, $(2, 4)$, $(3, 4)$. Ako želimo provjeriti je li niz rastući, prije ulaza u petlje učitavamo niz i postavljamo varijablu *raste* na 1. S dvije ugniježdene petlje generiramo sve parove indeksa $i < j$ i u tijelu unutrašnje petlje provjeravamo je li $a[i] > a[j]$. Ako je to ispunjeno bar za jedan par indeksa, niz nije rastući pa postavljamo varijablu *raste* na 0. S druge strane, ako za sve parove indeksa vrijedi $a[i] \leq a[j]$, vrijednost varijable *raste* ostaje 1. Nakon izlaza iz petlji ispisujemo poruku da je niz rastući ili nije ovisno o tome sadrži li varijabla *raste* 1 ili 0.

učitaj n
za $i = 1, \ldots, n$ ponavljaj učitaj $a[i]$
$raste \leftarrow 1$
za $i = 1, \ldots, n-1$ ponavljaj
⎡ za $j = i+1, \ldots, n$ ponavljaj
⎣ [ako je $a[i] > a[j]$ onda $raste \leftarrow 0$
ako je $raste = 1$ onda ispiši `"Niz je rastući"`
inače ispiši `"Niz nije rastući"`

Možemo efikasnije provjeriti je li niz rastući, tako da uspoređujemo samo susjedne članove niza. Niz je rastući ako za sve indekse $i = 1, \ldots, n-1$ vrijedi $a[i] \leq a[i+1]$. Zbog tranzitivnosti uređaja među brojevima tada za sve parove indeksa $i < j$ vrijedi $a[i] \leq a[j]$.

učitaj n
za $i = 1, \ldots, n$ ponavljaj učitaj $a[i]$
$raste \leftarrow 1$
za $i = 1, \ldots, n-1$ ponavljaj
[ako je $a[i] > a[i+1]$ onda $raste \leftarrow 0$
ako je $raste = 1$ onda ispiši `"Niz je rastući"`
inače ispiši `"Niz nije rastući"`

Još efikasniji program uspoređuje susjedne članove niza unutar *while* petlje i izlazi iz petlje čim nađe prvi par sa svojstvom $a[i] > a[i+1]$.

učitaj n
za $i = 1, \ldots, n$ ponavljaj učitaj $a[i]$
$i \leftarrow 1$
dok je $(i < n)$ i $(a[i] \leq a[i+1])$ ponavljaj $i \leftarrow i+1$
ako je $i = n$ onda ispiši `"Niz je rastući"`
inače ispiši `"Niz nije rastući"`

Problem *sortiranja* sastoji se od toga da od proizvoljnog niza napravimo rastući ili padajući niz premještanjem njegovih članova. Ako je rezultat rastući niz kažemo da smo ga *sortirali uzlazno*, a ako dobivamo padajući niz govorimo o *silaznom sortiranju*. Najjednostavniji algoritam za sortiranje uspoređuje svaka dva člana niza i zamjenjuje ih ako nisu u ispravnom redoslijedu. Taj algoritam zovemo *klasičnim algoritmom za sortiranje*.

Primjer 6.14 (Klasični algoritam za sortiranje)**.** *Program uzlazno sortira niz tako da za sve parove indeksa $i < j$ uspoređuje članove $a[i]$ i $a[j]$ i zamjenjuje ih ako je $a[i] > a[j]$.*

Rješenje.

učitaj n
za $i = 1, \ldots, n$ ponavljaj učitaj $a[i]$
za $i = 1, \ldots, n-1$ ponavljaj
⌈ za $j = i+1, \ldots, n$ ponavljaj
⌈ ako je $a[i] > a[j]$ onda
⌈ $pom \leftarrow a[i]$
$a[i] \leftarrow a[j]$
⌊ $a[j] \leftarrow pom$
za $i = 1, \ldots, n$ ponavljaj ispiši $a[i]$

□

Za $i = 1$ unutrašnja petlja uspoređuje prvi član niza sa svim ostalima i zamjenjuje ga ako nađe manji broj. Nakon prvog prolaza kroz vanjsku petlju na prvom mjestu se nalazi najmanji član niza. U drugom prolazu uspoređuje se drugi član niza s narednim članovima. Na kraju je na drugom mjestu najmanji od preostalih članova niza, tj. drugi po veličini u nizu. Kad vanjska petlja dođe do kraja niz je uzlazno sortiran.

Možemo li niz sortirati uspoređujući samo susjedne članove? Ovako izgleda program koji jednom prolazi kroz niz i zamjenjuje susjedne članove ako nisu u ispravnom redoslijedu.

za $j = 1, \ldots, n-1$ ponavljaj
⌈ ako je $a[j] > a[j+1]$ onda
⌈ $pom \leftarrow a[j]$
$a[j] \leftarrow a[j+1]$
⌊ $a[j+1] \leftarrow pom$

Naprimjer, ako ga primijenimo na niz 5, 10, 2, 7, 3, 8, program će redom raditi ove zamjene.

$j = 1$: 5, 10, 2, 7, 3, 8 (nema zamjene jer su 1. i 2. član u ispravnom redoslijedu)
$j = 2$: 5, 2, 10, 7, 3, 8 (zamjenjuje 2. i 3. član)
$j = 3$: 5, 2, 7, 10, 3, 8 (zamjenjuje 3. i 4. član)
$j = 4$: 5, 2, 7, 3, 10, 8 (zamjenjuje 4. i 5. član)
$j = 5$: 5, 2, 7, 3, 8, 10 (zamjenjuje 5. i 6. član)

Vidimo da niz još nije sortiran, ali je najveći član došao na zadnje mjesto. Ako ponovimo postupak s prvih $n-1$ članova niza, na predzadnje mjesto dolazi najveći od tih članova

i tako dalje. Niz će biti sortiran kad ponovimo postupak $n-1$ puta.

```
za i = 1, ..., n − 1 ponavljaj
⌈ za j = 1, ..., n − i ponavljaj
| ⌈ ako je a[j] > a[j + 1] onda
| | ⌈ pom ← a[j]
| | | a[j] ← a[j + 1]
| | ⌊ a[j + 1] ← pom
```

Ovaj algoritam naziva se *bubble sort* ("mjehuričasto sortiranje"). Unutrašnja petlja pomiče velike elemente prema kraju niza kao što mjehurići zraka izlaze na površinu vode. Algoritam možemo učiniti efikasnijim tako da prekinemo izvršavanje vanjske petlje čim prođemo kroz unutrašnju petlju bez ijedne zamijene. Tada su svi susjedni elementi u ispravnom redoslijedu i niz je rastući.

Primjer 6.15 (Bubble sort)**.** *Program uzlazno sortira niz tako da uspoređuje susjedne članove i zamjenjuje ih ako je prvi veći od drugog. Niz je sortiran kad dođemo do kraja bez zamjene.*

Rješenje.

```
učitaj n
za i = 1, ..., n ponavljaj učitaj a[i]
raste ← 0
i ← 1
dok je (i < n) i (raste = 0) ponavljaj
⌈ raste ← 1
| za j = 1, ..., n − i ponavljaj
| ⌈ ako je a[j] > a[j + 1] onda
| | ⌈ pom ← a[j]
| | | a[j] ← a[j + 1]
| | | a[j + 1] ← pom
| | ⌊ raste ← 0
⌊ i ← i + 1
za i = 1, ..., n ponavljaj ispiši a[i]
```

□

Da bismo mogli uspoređivati algoritme, trebamo ocijeniti njihovu efikasnost. Umjesto preciznog računanja vremena izvođenja kao u prethodnom poglavlju, obično se daje gruba ocjena brzine rasta vremena izvođenja u ovisnosti o veličini ulaznih podataka. Veličinu ulaznih podataka ocjenjujemo prirodnim brojem n. Za problem sortiranja n je duljina niza. Kod algoritama koji na ulazu imaju prirodan broj m, kakve smo imali u prethodnom poglavlju, za veličinu se uzima broj njegovih binarnih znamenaka $n = \lfloor \log_2 m \rfloor + 1$.

Primijetimo da ulaznih podataka zadane veličine n može biti puno. Vrijeme izvođenja algoritma zato može varirati za fiksni n. Obično se procjenjuje najgori mogući slučaj, tj. najdulje vrijeme izvođenja za sve ulazne podatke veličine n. Naprimjer, naši algoritmi za sortiranje iz primjera 6.14 i 6.15 rade najbrže ako je na ulazu rastući niz duljine n.

Takav niz je već sortiran, pa algoritmi niti jednom ne zamjenjuju elemente. Ako je na ulazu padajući niz duljine n potreban je najveći broj zamjena, pa je u tom slučaju vrijeme izvođenja najdulje. Ponekad se umjesto najduljeg vremena ocjenjuje prosječno vrijeme izvođenja za ulazne podatke veličine n.

Umjesto preciznog izračunavanja vremena izvođenja procjenjujemo broj operacija koje algoritam napravi, ili čak samo jedne vrste operacija. Ako algoritam tijekom izvođenja radi operacije zbrajanja i množenja, često se procjenjuje samo broj množenja jer je to sporija operacija. Naši algoritmi za sortiranje izvode operacije uspoređivanja elemenata niza i zamjene elemenata niza. Kao procjenu vremena izvođenja uzet ćemo broj uspoređivanja, a broj zamjena ćemo zanemariti.

Klasični algoritam za sortiranje uspoređuje elemente niza u tijelu unutrašnje petlje. Za $i = 1$ unutrašnja petlja se izvodi $n - 1$ puta, za $i = 2$ izvodi se $n - 2$ puta i tako dalje. Ukupan broj uspoređivanja je

$$(n-1) + (n-2) + \ldots + 2 + 1 = \frac{n(n-1)}{2}.$$

Sličnim razmatranjem vidimo da za padajući niz duljine n algoritam bubble sort radi isti broj uspoređivanja, pa ta dva algoritma smatramo jednako efikasnima. Ipak, bubble sort radi manje uspoređivanja za rastuće nizove i nizove koji su "blizu" rastućih.

Broj uspoređivanja $\frac{n(n-1)}{2}$ je kvadratni polinom u varijabli n. To je često jedina informacija koja nas zanima kao procjena efikasnosti algoritma. Vrijeme izvođenja ovisi o koeficijentima polinoma, ali bitno nam je samo da vrijeme raste proporcionalno kvadratu veličine ulaznih podataka n (za dovoljno veliki n linearni član možemo zanemariti). Kažemo da klasični algoritam za sortiranje i bubble sort algoritam imaju *složenost* $O(n^2)$, ili *kvadratnu složenost*. Točno značenje notacije "velikog O" opisano je sljedećom definicijom.

Definicija 6.16. *Neka su $f, g : \mathbb{N} \to \mathbb{R}$ funkcije kojima je domena skup prirodnih brojeva, a kodomena skup realnih brojeva. Pišemo $f(n) = O(g(n))$ ako postoji realan broj $M > 0$ i prirodan broj n_0 takav da za sve prirodne brojeve $n \geq n_0$ vrijedi $f(n) \leq M \cdot |g(n)|$.*

U našem slučaju funkcija $f(n) = \frac{n(n-1)}{2}$ predstavlja broj uspoređivanja pri sortiranju niza duljine n, a $g(n) = n^2$. Očito za $M = 1$ i $n_0 = 1$ vrijedi $f(n) \leq n^2$, za sve $n \geq 1$, pa je $f(n) = O(n^2)$. Postoje efikasniji algoritmi za sortiranje kojima je složenost $O(n \log n)$, naprimjer *mergesort*. Vrlo popularan algoritam je takozvani *quicksort*, kojem prosječno vrijeme izvođenja raste kao $O(n \log n)$ (ako se promatra najgori slučaj složenost mu je ipak kvadratna). Više o tim i mnogim drugim algoritmima za sortiranje možete pročitati u knjigama [15] i [24].

Često je potrebno pronaći mjesto na kojem se zadani element nalazi u nizu, ili ustanoviti da nije u nizu. Ako niz nije sortiran, uspoređujemo zadani element redom s članovima niza dok ga ne nađemo ili dođemo do kraja niza.

Primjer 6.17. *Program učitava niz od n brojeva i broj x. Ispisuje najmanji indeks i na kojem se u nizu nalazi x ili poruku o tome da x nije član niza.*

Rješenje.

učitaj n
za $i = 1, \ldots, n$ ponavljaj učitaj $a[i]$
učitaj x
$i \leftarrow 1$
dok je $(i \leq n)$ i $(a[i] \neq x)$ ponavljaj $i \leftarrow i + 1$
ako je $i \leq n$ onda
[ispiši i
inače
[ispiši `"Niz ne sadrži x"`

□

Za ovaj algoritam najgori je slučaj kad x nije u nizu. Tada ga uspoređujemo sa svim elementima niza, pa je složenost algoritma $O(n)$ (*linearna složenost*). Za pretraživanje sortiranih nizova postoji efikasniji algoritam logaritamske složenosti $O(\log_2 n)$.

Primjer 6.18 (Binarno pretraživanje)**.** *Program učitava rastući niz od n brojeva i broj x. Uspoređuje x sa srednjim elementom niza. Ako je srednji element veći, svi elementi desno od njega također su veći od x i možemo ih izbaciti iz razmatranja. Ako je srednji element manji od x, izbacujemo iz razmatrama elemente lijevo od njega, tj. nastavljamo potragu samo u desnoj polovici niza. Ponavljanjem ovog postupka pronalazimo mjesto na kojem se nalazi x ili zaključujemo da x nije član niza.*

Rješenje.

učitaj n
za $i = 1, \ldots, n$ ponavljaj učitaj $a[i]$
učitaj x
$lijevi \leftarrow 1$
$desni \leftarrow n$
$srednji \leftarrow (lijevi + desni)$ div 2
dok je $(lijevi < desni)$ i $(x \neq a[srednji])$ ponavljaj
[ako je $x < a[srednji]$ onda
[[$desni \leftarrow srednji - 1$
[inače
[[$lijevi \leftarrow srednji + 1$
[$srednji \leftarrow (lijevi + desni)$ div 2
ako je $x = a[srednji]$ onda
[ispiši $srednji$
inače
[ispiši `"Niz ne sadrži x"`

□

Varijable *lijevi* i *desni* označavaju granice dijela niza u kojem tražimo x, a *srednji* pokazuje na sredinu tog dijela niza. One sadrže indekse članova niza na koje pokazuju; takve varijable nazivamo *kazaljkama*. Ako korisnik unese duljinu $n = 9$, niz 2, 7, 10, 11, 13, 16, 20, 21, 25 i element $x = 10$, prije ulaza u petlju vrijednosti kazaljka su $lijevi = 1$,

$desni = 9$ i $srednji = (1 + 9)$ div $2 = 5$. U petlji uspoređujemo x i $a[srednji] = 13$. Budući da je manji od srednjeg elementa, postavljamo $desni$ na $srednji - 1 = 4$. Izračunamo novu vrijednost kazaljke $srednji = (1 + 4)$ div $2 = 2$ i drugi puta izvodimo petlju. Sada je x veći od $a[2] = 7$, pa postavljamo $lijevi$ na 3 i $srednji$ na (3+4) div $2 = 3$. Budući da je $x = a[3] = 10$, izlazimo iz petlje i ispisujemo indeks na kojem se nalazi x.

Promotrimo kako radi algoritam ako u istom nizu tražimo element $x = 22$. Na početku su vrijednosti kažaljki $lijevi = 1$, $desni = 9$, $srednji = 5$. Varijabla x je veća od srednjeg elementa, pa u prvom prolazu kroz petlju kazaljke poprimaju vrijednosti $lijevi = 6$, $desni = 9$, $srednji = 7$. U drugom prolazu kroz petlju x je i dalje veći od $a[srednji]$, pa kazaljke postaju $lijevi = 8$, $desni = 9$, $srednji = 8$. I u trećem prolazu je x veći od srednjeg elementa, pa će se kazaljke izjednačiti: $lijevi = desni = srednji = 9$. Sad izlazimo iz petlje jer više ne vrijedi $lijevi < desni$ i ispisujemo poruku da niz ne sadrži x.

U prvom primjeru uspoređivali smo $x = 10$ samo s tri člana niza: $a[5] = 13$, $a[2] = 7$ i $a[3] = 10$. Tada smo ga pronašli u nizu koji ima ukupno 9 članova. U drugom primjeru uspoređivali smo $x = 23$ s četiri člana niza: $a[5] = 13$, $a[7] = 20$, $a[8] = 21$ i $a[9] = 25$. Tada smo zaključili da x nije u nizu. Općenito, svaki prolaz kroz petlju smanjuje "aktivni dio" niza (u kojem tražimo x) na pola, pa se tijelo petlje izvodi najviše $\lfloor \log_2 n \rfloor$ puta. Zato binarno pretraživanje ima logaritamsku složenost $O(\log_2 n)$.

U radu s nizovima poželjno je da oni budu sortirani jer nam to omogućuje efikasnije algoritme. Česta operacija je spajanje dvaju sortiranih nizova u niz koji također treba biti sortiran. Naprimjer, nastavnik je prilikom ispravljanja podijelio testove u dvije hrpe prema grupama zadataka koje su učenici rješavali. Složio je te dvije hrpe abecednim redom i želi ih spojiti u jednu, također složenu po abecedi.

Naivni pristup bio bi uzeti redom elemente prvog i drugog niza i sortirati spojeni niz. Ako koristimo neki od naših algoritama za sortiranje, dobili bismo algoritam kvadratne složenosti $O(n^2)$. Čak i s efikasnijim algoritmima za sortiranje složenost bi bila $O(n \log n)$, a moguće je postići linearnu složenost $O(n)$. Nastavnik može staviti pred sebe dvije sortirane hrpe testova i uzimati s vrha onaj od dvaju testova koji dolazi prije po abecedi. Kad potroši jednu hrpu, ostatak druge hrpe stavlja na kraj spojene hrpe testova. Tako dobiva sve testove u abecednom redu. Odgovarajući algoritam za nizove zovemo *merge*.

Primjer 6.19 (Merge)**.** *Program učitava dva rastuća niza* $a[1], \ldots, a[m]$ *i* $b[1], \ldots, b[n]$. *Spaja ih u rastući niz* $c[1], \ldots, c[m + n]$.

Rješenje.

učitaj m, n
za $i = 1, \ldots, m$ ponavljaj učitaj $a[i]$
za $j = 1, \ldots, n$ ponavljaj učitaj $b[j]$
$i \leftarrow 1$
$j \leftarrow 1$
$k \leftarrow 0$

```
dok je (i ≤ m) i (j ≤ n) ponavljaj
⌈ k ← k + 1
| ako je a[i] ≤ b[j] onda
| ⌈ c[k] ← a[i]
| ⌊ i ← i + 1
| inače
| ⌈ c[k] ← b[j]
| ⌊ j ← j + 1
⌊
dok je i ≤ m ponavljaj
⌈ k ← k + 1
| c[k] ← a[i]
⌊ i ← i + 1
dok je j ≤ n ponavljaj
⌈ k ← k + 1
| c[k] ← b[j]
⌊ j ← j + 1
za i = 1, ..., m + n ponavljaj ispiši c[i]
```

□

Promotrimo kako radi algoritam za $m = n = 5$, niz a s članovima 2, 5, 8, 9, 11 i niz b s članovima 3, 4, 6, 7, 8. Varijabla i pokazuje na "vrh" neobrađenog dijela niza a, varijabla j na vrh niza b, a varijabla k sadrži trenutačni broj elemenata u spojenom nizu c. Na početku je $i = j = 1$, $k = 0$. Program se sastoji od tri uzastopne while petlje. Prva petlja prepisuje elemente nizova a i b u c dok ne dođemo do kraja jednog od ulaznih nizova. Druga petlja prepisuje ostatak niza a u c, a treća ostatak niza b u c. Ako prva petlja dođe do kraja niza a izvodi se samo treća petlja, a ako dođe do kraja niza b izvodi se samo druga petlja.

Prva petlja uspoređuje $a[i]$ s $b[j]$ i prepisuje manji broj u niz c. Ako su jednaki, prvo prepisuje element iz a. Za naše ulazne nizove varijable se mijenjaju na sljedeći način:

1. prolaz: $k = 1$, $c[1] = 2$, $i = 2$
2. prolaz: $k = 2$, $c[2] = 3$, $j = 2$
3. prolaz: $k = 3$, $c[3] = 4$, $j = 3$
4. prolaz: $k = 4$, $c[4] = 5$, $i = 3$
5. prolaz: $k = 5$, $c[5] = 6$, $j = 4$
6. prolaz: $k = 6$, $c[6] = 7$, $j = 5$
7. prolaz: $k = 7$, $c[7] = 8$, $i = 4$
8. prolaz: $k = 8$, $c[8] = 8$, $j = 6$

Program izlazi iz prve petlje jer je $j > n$ i u drugoj petlji prepisuje ostatak niza a u c:

1. prolaz: $k = 9$, $c[9] = 9$, $i = 5$
2. prolaz: $k = 10$, $c[10] = 11$, $i = 6$

U treću petlju ne ulazi jer i dalje vrijedi $j > n$. Na kraju program ispisuje elemente spojenog niza c: 2, 3, 4, 5, 6, 7, 8, 8, 9, 11. Kao procjenu veličine ulaznih podataka uzimamo ukupan broj članova oba ulazna niza $m + n$, a složenost algoritma merge je

$O(m+n)$. U idućem poglavlju koristit ćemo ovaj algoritam za efikasnu implementaciju operacija sa skupovima.

Zadaci

Zadatak 6.1. *Napišite program koji učitava prirodan broj n i ispisuje sve proste brojeve iz skupa $\{1,\ldots,n\}$ koristeći algoritam poznat kao* Eratostenovo sito. *Program u niz duljine n sprema jedinice, osim na prvo mjesto na koje sprema nulu. Uzastopno ispisuje prvi po redu član niza različit od nule i sve njegove višekratnike postavlja na nulu.*

Zadatak 6.2. *Napišite dva programa koji provjeravaju je li zadani niz padajući. Prvi program neka uspoređuje svaka dva člana niza, a drugi program samo susjedne članove niza.*

Zadatak 6.3. *Opišite kojim redom se mijenjaju elementi niza* 7, 5, 10, 2, 4 *pri uzlaznom sortiranju klasičnim algoritmom i bubble sort algoritmom.*

Zadatak 6.4. *Napišite program koji silazno sortira niz s pomoću klasičnog algoritma.*

Zadatak 6.5. *Napišite program koji silazno sortira niz s pomoću "bubble sort" algoritma.*

Zadatak 6.6. *Modificirajte klasični algoritam za sortiranje tako da unutarnja petlja ne zamjenjuje elemente niza, nego samo traži minimalni element. Nakon izlaza iz unutarnje petlje minimalni element se zamjenjuje s i-tim elementom niza.*

Zadatak 6.7. *Dokažite: ako postoji konačan limes $\lim_{n\to\infty}\frac{f(n)}{g(n)}$, onda vrijedi $f(n)=O(g(n))$.*

Zadatak 6.8. *Ako je $f(n)$ bilo koji polinom stupnja k, dokažite da je $f(n)=O(n^k)$.*

Zadatak 6.9. *Dokažite da za svake dvije baze $a,b>1$ vrijedi $\log_a n = O(\log_b n)$.*

Zadatak 6.10. *Neka su $a,b>1$ baze eksponencijalnih funkcija. Dokažite da vrijedi $a^n=O(b^n)$ ako i samo ako je $a\le b$.*

Zadatak 6.11. *Napišite program koji učitava niz od n brojeva i broj x. Program ispisuje indekse svih članova niza koji su jednaki x.*

Zadatak 6.12. *Mergesort je primjer rekurzivnog algoritma, tj. algoritma koji poziva samog sebe s drugim ulaznim podacima. Želimo sortirati niz duljine n. Ako je $n=1$, niz je već sortiran. U suprotnom podijelimo ga na dva podniza duljine n div 2 (odnosno, za neparni n, na prvi podniz duljine n div $2+1$ i drugi podniz duljine n div 2). Svaki od ta dva podniza sortiramo algoritmom mergesort i zatim ih spojimo algoritmom merge iz primjera 6.19. Opišite rad ovog algoritma za niz* 6, 10, 2, 7, 5, 4, 8, 3 *i pokušajte izvesti njegovu složenost.*

Poglavlje 7

Još neki matematički algoritmi

S pomoću nizova na računalu možemo implementirati složene matematičke objekte kao što su skupovi, permutacije, polinomi i matrice. U ovom poglavlju proučit ćemo algoritme za operacije s tim objektima.

Prvo ćemo implementirati operacije s konačnim skupovima brojeva. Razlika između niza i skupa je u tome što kod skupa redoslijed i ponavljanje pojedinih elemenata nisu važni. Naprimjer, $\{1, 2, 3\}$, $\{1, 2, 2, 3, 3, 3\}$ i $\{3, 2, 1\}$ je jedan te isti skup. Zato skupove u računalu predstavljamo strogo rastućim nizovima, tj. rastućim nizovima bez ponavljanja. Provjeru pripada li neki element skupu tada možemo napraviti binarnim pretraživanjem u logaritamskom vremenu. Skupovne operacije ćemo realizirati u linearnom vremenu.

Unija skupova A i B je skup koji se sastoji od elemenata koji su u A ili u B:

$$A \cup B = \{x \mid x \in A \text{ ili } x \in B\}.$$

Ako su elementi od A pohranjeni u strogo rastućem nizu $a[1], \ldots, a[m]$, a elementi od B u strogo rastućem nizu $b[1], \ldots, b[n]$, uniju $C = A \cup B$ dobivamo algoritmom vrlo sličnim algoritmu merge. Jedina razlika je u tome što elemente koji su u oba niza treba prepisati u niz c samo jednom.

Primjer 7.1. *Program učitava strogo rastuće nizove a i b koji predstavljaju skupove A i B. Definira i ispisuje niz c koji predstavlja uniju $A \cup B$.*

Rješenje.

učitaj m, n
za $i = 1, \ldots, m$ ponavljaj učitaj $a[i]$
za $j = 1, \ldots, n$ ponavljaj učitaj $b[j]$
$i \leftarrow 1$
$j \leftarrow 1$
$k \leftarrow 0$

```
dok je (i ≤ m) i (j ≤ n) ponavljaj
⌈ k ← k + 1
| ako je a[i] ≤ b[j] onda
| ⌈ c[k] ← a[i]
| | ako je a[i] = b[j] onda j ← j + 1
| ⌊ i ← i + 1
| inače
| ⌈ c[k] ← b[j]
⌊ ⌊ j ← j + 1
dok je i ≤ m ponavljaj
⌈ k ← k + 1
| c[k] ← a[i]
⌊ i ← i + 1
dok je j ≤ n ponavljaj
⌈ k ← k + 1
| c[k] ← b[j]
⌊ j ← j + 1
za i = 1, . . . , k ponavljaj ispiši c[i]
```

□

Na kraju izvođenja ovog algoritma broj elemenata unije pohranjen je u varijabli k. Kod algoritma merge iz primjera 6.19 varijabla k na kraju uvijek ima vrijednost $m + n$, a u ovom algoritmu je manja od $m + n$ za broj elemenata presjeka $A \cap B$.

Presjek je skup koji se sastoji od elemenata koji su u A i u B:

$$A \cap B = \{x \mid x \in A \text{ i } x \in B\}.$$

Algoritam za presjek sličan je algoritmu merge, ali prepisuje element u c samo ako se nalazi u oba niza a i b. Druga i treća while petlja nisu potrebne.

Primjer 7.2. *Program učitava strogo rastuće nizove a i b koji predstavljaju skupove A i B. Definira i ispisuje niz c koji predstavlja presjek $A \cap B$.*

Rješenje.

```
učitaj m, n
za i = 1, . . . , m ponavljaj učitaj a[i]
za j = 1, . . . , n ponavljaj učitaj b[j]
i ← 1
j ← 1
k ← 0
```

```
dok je (i ≤ m) i (j ≤ n) ponavljaj
⌈ ako je a[i] = b[j] onda
|   ⌈ k ← k + 1
|   | c[k] ← a[i]
|   | i ← i + 1
|   ⌊ j ← j + 1
|  inače
|   ⌈ ako je a[i] < b[j] onda
|   [ i ← i + 1
|   | inače
|   ⌊ j ← j + 1
⌊
za i = 1, . . . , k ponavljaj ispiši c[i]
```

□

Nakon izvođenja petlje broj elemenata presjeka pohranjen je u varijabli k. Treća operacija koju ćemo implementirati je razlika skupova. Skup $A \setminus B$ sadrži elemente iz A koji nisu u B:

$$A \setminus B = \{x \mid x \in A \text{ i } x \notin B\}.$$

Algoritam prepisuje element iz a u c, osim ako se podudara s elementom iz b. Kad dođe do kraja niza b nastavlja prepisivati preostale elemente iz a u drugoj while petlji, a treća petlja nije potrebna.

Primjer 7.3. *Program učitava strogo rastuće nizove a i b koji predstavljaju skupove A i B. Definira i ispisuje niz c koji predstavlja skupovnu razliku $A \setminus B$.*

Rješenje.

```
učitaj m, n
za i = 1, . . . , m ponavljaj učitaj a[i]
za j = 1, . . . , n ponavljaj učitaj b[j]
i ← 1
j ← 1
k ← 0
dok je (i ≤ m) i (j ≤ n) ponavljaj
⌈ ako je a[i] < b[j] onda
|   ⌈ k ← k + 1
|   | c[k] ← a[i]
|   ⌊ i ← i + 1
|  inače
|   ⌈ ako je a[i] = b[j] onda i ← i + 1
|   ⌊ j ← j + 1
⌊
dok je i ≤ m ponavljaj
⌈ k ← k + 1
| c[k] ← a[i]
⌊ i ← i + 1
za i = 1, . . . , k ponavljaj ispiši c[i]
```

□

Kao i u prethodna dva algoritma, broj elemenata razlike $A \setminus B$ na kraju je pohranjen u varijabli k. Sva tri algoritma imaju složenost $O(m+n)$, isto kao algoritam merge iz primjera 6.19.

Druga vrsta matematičkih objekata za koje ćemo razviti algoritme su permutacije. Permutacije su bijekcije sa skupa $\{1,\ldots,n\}$ na samog sebe. Općenito, funkciju $f : \{1,\ldots,m\} \to \{1,\ldots,n\}$ možemo reprezentirati nizom brojeva duljine m. Ako je $f(i) = j$, na i-to mjesto niza stavljamo broj j. Da bi funkcija bila bijekcija, mora biti injekcija i surjekcija: različite elemente domene mora preslikavati u različite elemente kodomene i svaki element kodomene mora biti "pogođen" nekim elementom iz domene. To je moguće samo ako domena i kodomena imaju jednako monogo elemenata, tj. ako je $m = n$. Osnovna operacija s permutacijama je kompozicija: $(g \circ f)(i) = g(f(i))$.

Primjer 7.4. *Program učitava prirodan broj n i nizove f i g koji predstavljaju permutacije skupa $\{1,\ldots,n\}$. Program definira i ispisuje niz h koji predstavlja njihovu kompoziciju.*

Rješenje.

učitaj n
za $i = 1,\ldots,n$ ponavljaj učitaj $f[i]$
za $i = 1,\ldots,n$ ponavljaj učitaj $g[i]$
za $i = 1,\ldots,n$ ponavljaj
$\lfloor$ $h[i] \leftarrow g[f[i]]$
za $i = 1,\ldots,n$ ponavljaj ispiši $h[i]$

□

U matematici se permutacije ponekad zapisuju kao dva niza brojeva, od kojih je prvi rastući i predstavlja domenu. Naprimjer,

$$f = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 1 & 3 \end{pmatrix}$$

je permutacija koja preslikava $1 \mapsto 2$, $2 \mapsto 4$, $3 \mapsto 1$ i $4 \mapsto 3$. Naša reprezentacija permutacija je ekonomičnija jer pamtimo samo donji niz brojeva. Poznato je da svaka bijekcija ima inverznu funkciju. Inverznu permutaciju od f dobivamo tako da zamijenimo donji i gornji niz te sortiramo novi gornji niz uzlazno, uz odgovarajuću promjenu redoslijeda donjeg niza:

$$f^{-1} = \begin{pmatrix} 2 & 4 & 1 & 3 \\ 1 & 2 & 3 & 4 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 1 & 4 & 2 \end{pmatrix}.$$

Taj algoritam imao bi složenost $O(n^2)$, ili $O(n \log n)$ ako se koristimo efikasnijim algoritmom za sortiranje. Inverznu permutaciju možemo dobiti elegantnijim algoritmom linearne složenosti $O(n)$.

Primjer 7.5. *Program učitava prirodan broj n i niz f koji predstavlja permutaciju skupa $\{1,\ldots,n\}$. Program definira i ispisuje niz g koji predstavlja inverz učitane permutacije.*

Rješenje.

učitaj n
za $i = 1, \ldots, n$ ponavljaj učitaj $f[i]$
za $i = 1, \ldots, n$ ponavljaj
$\lfloor\ g[f[i]] \leftarrow i$
za $i = 1, \ldots, n$ ponavljaj ispiši $g[i]$

□

Druga važna vrsta funkcija su polinomi. Za funkciju $f : \mathbb{R} \to \mathbb{R}$ kažemo da je *polinom* ako je oblika $f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$. Pritom su $a_0, \ldots, a_n \in \mathbb{R}$ konstante koje zovemo *koeficijentima* polinoma. Koeficijent $a_n \neq 0$ zovemo *vodećim koeficijentom*, a n *stupnjem* polinoma. Polinom očito možemo reprezentirati u računalu tako da u niz spremimo njegove koeficijente. Uobičajeno je koeficijent uz x^i označiti indeksom i, pa ćemo u programima za rad s polinomima nizove indeksirati od nule.

Važnost polinoma je u tome što s njima možemo dobro aproksimirati druge, kompliciranije funkcije. Polinomi su jednostavni jer za dani $x \in \mathbb{R}$ vrijednost $f(x)$ možemo izračunati s konačno mnogo zbrajanja i množenja realnih brojeva. Budući da je izračunavanje vrijednosti polinoma vrlo česta operacija, želimo algoritam koji koristi što manje množenja i zbrajanja. Nespretni algoritam izgleda ovako.

Primjer 7.6. *Program učitava prirodan broj n, niz koeficijenata $a[0], \ldots, a[n]$ i broj x te izračunava i ispisuje vrijednost polinoma $f(x)$. Ovaj program koristi puno više operacija zbrajanja i množenja nego što je potrebno.*

Rješenje.

učitaj n
za $i = 0, \ldots, n$ ponavljaj učitaj $a[i]$
učitaj x
$f \leftarrow 0$
za $i = 0, \ldots, n$ ponavljaj
$\left\lfloor \begin{array}{l} p \leftarrow 1 \\ \text{za } j = 1, \ldots, i \text{ ponavljaj } p \leftarrow p \cdot x \\ f \leftarrow f + a[i] \cdot p \end{array} \right.$
ispiši f

□

Ovo je algoritam složenosti $O(n^2)$ u stupnju polinoma n. U unutrašnjoj petlji izračunavamo vrijednost potencije $p = x^i$ uzastopnim množenjem s x. Potenciju možemo efikasnije izračunati uzastopnim kvadriranjem. Naprimjer, x^4 možemo umjesto kao produkt $x \cdot x \cdot x \cdot x$ izračunati kao $(x^2)^2$. Uzastopnim kvadriranjem dobivamo potencije od x kojima su eksponenti potencije od 2: x, x^2, x^4, x^8, x^{16} ... Ako i nije potencija od dva, x^i treba prikazati kao produkt takvih potencija, naprimjer $x^{11} = x^{1+2+8} = x \cdot x^2 \cdot x^8$. Pojavljuju se eksponenti 2^k koji odgovaraju jedinicama u binarnom zapisu broja i. Općeniti algoritam za potenciranje uzastopnim kvadriranjem izgleda ovako.

Primjer 7.7 (Brzo potenciranje)**.** *Program učitava realan broj $x \in \mathbb{R}$ i prirodan broj $i \in \mathbb{N}$ te uzastopnim kvadriranjem izračunava potenciju $p = x^i$.*

Rješenje.

učitaj x, i
$p \leftarrow 1$
dok je $i \neq 0$ ponavljaj
$\left[\begin{array}{l} \text{ako je } (i \bmod 2) = 1 \text{ onda } p \leftarrow p \cdot x \\ x \leftarrow x \cdot x \\ i \leftarrow i \text{ div } 2 \end{array}\right.$
ispiši p

□

Ako je na ulazu $x = 2$, $i = 11$, program postavlja $p = 1$ i zatim radi ovako:

1. prolaz kroz petlju:	$p = 2$,	$x = 4$,	$i = 5$
2. prolaz kroz petlju:	$p = 8$,	$x = 16$,	$i = 2$
3. prolaz kroz petlju:	$p = 8$,	$x = 256$,	$i = 1$
4. prolaz kroz petlju:	$p = 2048$,	$x = 65536$,	$i = 0$

Nakon izlaza iz petlje ispisuje $p = 2048 = 2^{11}$. Ovo je algoritam logaritamske složenosti $O(\log_2 n)$ (za $n = i$), dok je potenciranje uzastopnim množenjem s x (kao u primjeru 7.6) algoritam linearne složenosti. Kad bismo u primjeru 7.6 potencije računali na ovaj način, dobili bismo algoritam za izračunavanje polinoma složenosti $O(n \log_2 n)$. Međutim, na puno jednostavniji način možemo dobiti algoritam linearne složenosti $O(n)$ – samo iskoristimo potenciju x^{i-1} iz prethodnog koraka da dobijemo $x^i = x \cdot x^{i-1}$.

Primjer 7.8. *Program učitava prirodan broj n, niz koeficijenata $a[0], \ldots, a[n]$ i broj x te izračunava i ispisuje vrijednost polinoma $f(x)$. Ovo je program linearne složenosti, ali postoji još efikasniji algoritam.*

Rješenje.

učitaj n
za $i = 0, \ldots, n$ ponavljaj učitaj $a[i]$
učitaj x
$f \leftarrow 0$
$p \leftarrow 1$
za $i = 0, \ldots, n$ ponavljaj
$\left[\begin{array}{l} f \leftarrow f + a[i] \cdot p \\ p \leftarrow p \cdot x \end{array}\right.$
ispiši f

□

Još efikasniji algoritam dobivamo ako polinom $f(x) = a_n x^n + \ldots + a_1 x + a_0$ zapišemo na drugi način:

$$f(x) = ((\cdots(((a_n x + a_{n-1})x + a_{n-2})x + a_{n-3})x + \cdots + a_2)x + a_1)x + a_0.$$

Primijetimo da izračunavanje počinje "iznutra", od vodećeg koeficijenta prema koeficijentima uz niže potencije. Zato ćemo u programu trebati silaznu petlju. Ovaj način izračunavanja polinoma naziva se *Hornerovim algoritmom.*

Primjer 7.9 (Hornerov algoritam)**.** *Program učitava prirodan broj n, niz koeficijenata $a[0], \ldots, a[n]$ i broj x te izračunava i ispisuje vrijednost polinoma $f(x)$. Ovo je najefikasniji algoritam.*

Rješenje.

učitaj n
za $i = 0, \ldots, n$ ponavljaj učitaj $a[i]$
učitaj x
$f \leftarrow 0$
za $i = n, \ldots, 0$ ponavljaj
$[\ f \leftarrow f \cdot x + a[i]$
ispiši f

□

Složenost Hornerova algoritma također je $O(n)$, ali on koristi upola manje množenja od algoritma iz primjera 7.8. Za izračunavanje vrijednosti polinoma stupnja n algoritam iz primjera 7.8 koristi $2(n+1)$ množenja i $n+1$ zbrajanja, a Hornerov algoritam samo $n+1$ množenja i isto toliko zbrajanja.

Zbroj, produkt i kompozicija polinoma je ponovo polinom. Osim toga polinome možemo dijeliti s ostatkom i izračunavati im najveću zajedničku mjeru Euklidovim algoritmom, slično kao za prirodne brojeve. U idućem primjeru je algoritam za množenje polinoma, a ostale operacije s polinomima ostavljamo za zadatke.

Primjer 7.10. *Program učitava stupnjeve m, n i koeficijente $a[0], \ldots, a[m]$, $b[0], \ldots, b[n]$ dvaju polinoma te izračunava i ispisuje koeficijente njihova produkta.*

Rješenje.

učitaj m, n
za $i = 0, \ldots, m$ ponavljaj učitaj $a[i]$
za $i = m+1, \ldots, m+n$ ponavljaj $a[i] \leftarrow 0$
za $i = 0, \ldots, n$ ponavljaj učitaj $b[i]$
za $i = n+1, \ldots, m+n$ ponavljaj $b[i] \leftarrow 0$
za $i = 0, \ldots, m+n$ ponavljaj
$\left[\begin{array}{l} c[i] \leftarrow 0 \\ \text{za } j = 0, \ldots, i \text{ ponavljaj} \\ [\ c[i] \leftarrow c[i] + a[j] \cdot b[i-j] \end{array}\right.$
za $i = 0, \ldots, m+n$ ponavljaj ispiši $c[i]$

Ovaj algoritam redom izračunava koeficijente produkta $c[0], c[1], \ldots, c[m+n]$. Nizove a i b nadopunili smo vodećim nulama jer indeksi $a[j]$ i $b[i-j]$ u untrašnjoj petlji prelaze m, odnosno n. Efikasniji je algoritam koji koeficijente produkta ne izračunava redom, nego

pribraja produkte $a[i] \cdot b[j]$ odgovarajućem koeficijentu $c[i+j]$:

učitaj m, n
za $i = 0, \ldots, m$ ponavljaj učitaj $a[i]$
za $i = 0, \ldots, n$ ponavljaj učitaj $b[i]$
za $i = 0, \ldots, m+n$ ponavljaj $c[i] \leftarrow 0$
za $i = 0, \ldots, m$ ponavljaj
$\left[\right.$ za $j = 0, \ldots, n$ ponavljaj
$\left[\right.$ $\left[\right.$ $c[i+j] \leftarrow c[i+j] + a[i] \cdot b[j]$
za $i = 0, \ldots, m+n$ ponavljaj ispiši $c[i]$

□

Matrice su pravokutne tablice brojeva. O njima možemo razmišljati kao o dvodimenzionalnim nizovima, tj. nizovima s dva indeksa:

$$\begin{array}{ccccc} a[1][1] & a[1][2] & a[1][3] & \cdots & a[1][n] \\ a[2][1] & a[2][2] & a[2][3] & \cdots & a[2][n] \\ \vdots & \vdots & \vdots & & \vdots \\ a[m][1] & a[m][2] & a[m][3] & \cdots & a[m][n] \end{array}$$

Prvi indeks označava redak, a drugi indeks stupac matrice. Ako matrica ima m redaka i n stupaca, govorimo o matrici *tipa* $m \times n$. Za učitavanje elemenata matrice trebaju nam dvije ugniježdene petlje:

učitaj m, n
za $i = 1, \ldots, m$ ponavljaj
$\left[\right.$ za $j = 1, \ldots, n$ ponavljaj
$\left[\right.$ $\left[\right.$ učitaj $a[i][j]$

Da bismo ispisali elemente matrice u tabličnom obliku, trebamo kontrolirati format ispisa i prijelaz u novi red. To u programskim jezicima FORTRAN, Pascal, C i Python možemo na sljedeći način.

Ispisivanje matrice

FORTRAN

```
DO, i=1,m
  WRITE(*,"100I4") (a(i,j), j=1,n)
ENDDO
```

Pascal

```
for i:=1 to m do
begin
  for j:=1 to n do
    write(a[i][j]:4,' ');
  writeln();
end;
```

C

```
for (i=0; i<m; ++i)
{ for (j=0; j<n; ++j)
    printf("%4d ",a[i][j]);
  printf("\n");
}
```

Python

```
for i in range(m):
  for j in range(n):
    print '{0:4d}'.format(a[i][j]),
  print
```

U pseudojeziku nećemo brinuti o formatu, nego ćemo jednostavno ispisati sve elemente matrice:

$$\begin{array}{l}\text{za } i = 1, \ldots, m \text{ ponavljaj}\\ \left[\begin{array}{l}\text{za } j = 1, \ldots, n \text{ ponavljaj}\\ \left[\ \text{ispiši } a[i][j]\right.\end{array}\right.\end{array}$$

S matricama kao matematičkim objektima definirane su operacije zbrajanja, množenja skalarom (brojem) i međusobnog množenja matrica. Zbajati možemo matrice istog tipa, a zbroj je matrica čiji su elementi zbojevi odgovarajućih elemenata prve i druge matrice.

Primjer 7.11. *Program učitava prirodne brojeve m, n i dvije matrice tipa $m \times n$. Definira i ispisuje zbroj učitanih matrica.*

Rješenje.

$$\begin{array}{l}\text{učitaj } m, n\\ \text{za } i = 1, \ldots, m \text{ ponavljaj}\\ \left[\ \text{za } j = 1, \ldots, n \text{ ponavljaj učitaj } a[i][j]\right.\\ \text{za } i = 1, \ldots, m \text{ ponavljaj}\\ \left[\ \text{za } j = 1, \ldots, n \text{ ponavljaj učitaj } b[i][j]\right.\\ \text{za } i = 1, \ldots, m \text{ ponavljaj}\\ \left[\begin{array}{l}\text{za } j = 1, \ldots, n \text{ ponavljaj}\\ \left[\ c[i][j] \leftarrow a[i][j] + b[i][j]\right.\end{array}\right.\\ \text{za } i = 1, \ldots, m \text{ ponavljaj}\\ \left[\ \text{za } j = 1, \ldots, n \text{ ponavljaj ispiši } c[i][j]\right.\end{array}$$

□

Množenje matrice skalarom definirano je slično, po koordinatama (vidi zadatak 7.17). Da bismo matrice mogli međusobno množiti, one moraju biti ulančane, tj. broj stupaca prve matrice mora biti jednak broju redaka druge matrice. Element $c[i][j]$ produkta dobivamo "skalarnim množenjem" i-tog retka prve matrice i j-tog stupca druge matrice:

$$c[i][j] = \sum_{k=1}^{p} a[i][k] \cdot b[k][j].$$

U linearnoj algebri matrice reprezentiraju linearne operatore, a množenje matrica odgovara kompoziciji linearnih operatora.

Primjer 7.12. *Program učitava prirodne brojeve m, p, n, matricu tipa $m \times p$ i matricu tipa $p \times n$. Definira i ispisuje matricu tipa $m \times n$ koja je umnožak učitanih matrica.*

Rješenje.

učitaj m, p, n
za $i = 1, \ldots, m$ ponavljaj
[za $j = 1, \ldots, p$ ponavljaj učitaj $a[i][j]$
za $i = 1, \ldots, p$ ponavljaj
[za $j = 1, \ldots, n$ ponavljaj učitaj $b[i][j]$
za $i = 1, \ldots, m$ ponavljaj
⌈ za $j = 1, \ldots, n$ ponavljaj
⌈ $c[i][j] \leftarrow 0$
za $k = 1, \ldots, p$ ponavljaj
⌊ [$c[i][j] \leftarrow c[i][j] + a[i][k] \cdot b[k][j]$
za $i = 1, \ldots, m$ ponavljaj
[za $j = 1, \ldots, n$ ponavljaj ispiši $c[i][j]$

□

Matrice tipa $n \times n$ zovemo *kvadratnim matricama reda n*. Kod algoritama za rad s matricama red se često uzima kao veličina ulaznih podataka, iako se matrice zapravo sastoje od n^2 brojeva. U tom slučaju zbrajanje matrica je operacija složenosti $O(n^2)$, a množenje matrica operacija složenosti $O(n^3)$ (jer se algoritam sastoji od tri ugniježđene petlje). Još jedna operacija s matricama je *transponiranje*, tj. zamjena uloge redaka i stupaca.

Primjer 7.13. *Program učitava kvadratnu matricu reda n, transponira je i ispisuje.*

Rješenje.

učitaj n
za $i = 1, \ldots, n$ ponavljaj
[za $j = 1, \ldots, n$ ponavljaj učitaj $a[i][j]$
za $i = 1, \ldots, n-1$ ponavljaj
⌈ za $j = i+1, \ldots, n$ ponavljaj
⌈ $pom \leftarrow a[i][j]$
$a[i][j] \leftarrow a[j][i]$
⌊ [$a[j][i] \leftarrow pom$

za $i = 1, \ldots, n$ ponavljaj
[za $j = 1, \ldots, n$ ponavljaj ispiši $a[i][j]$

□

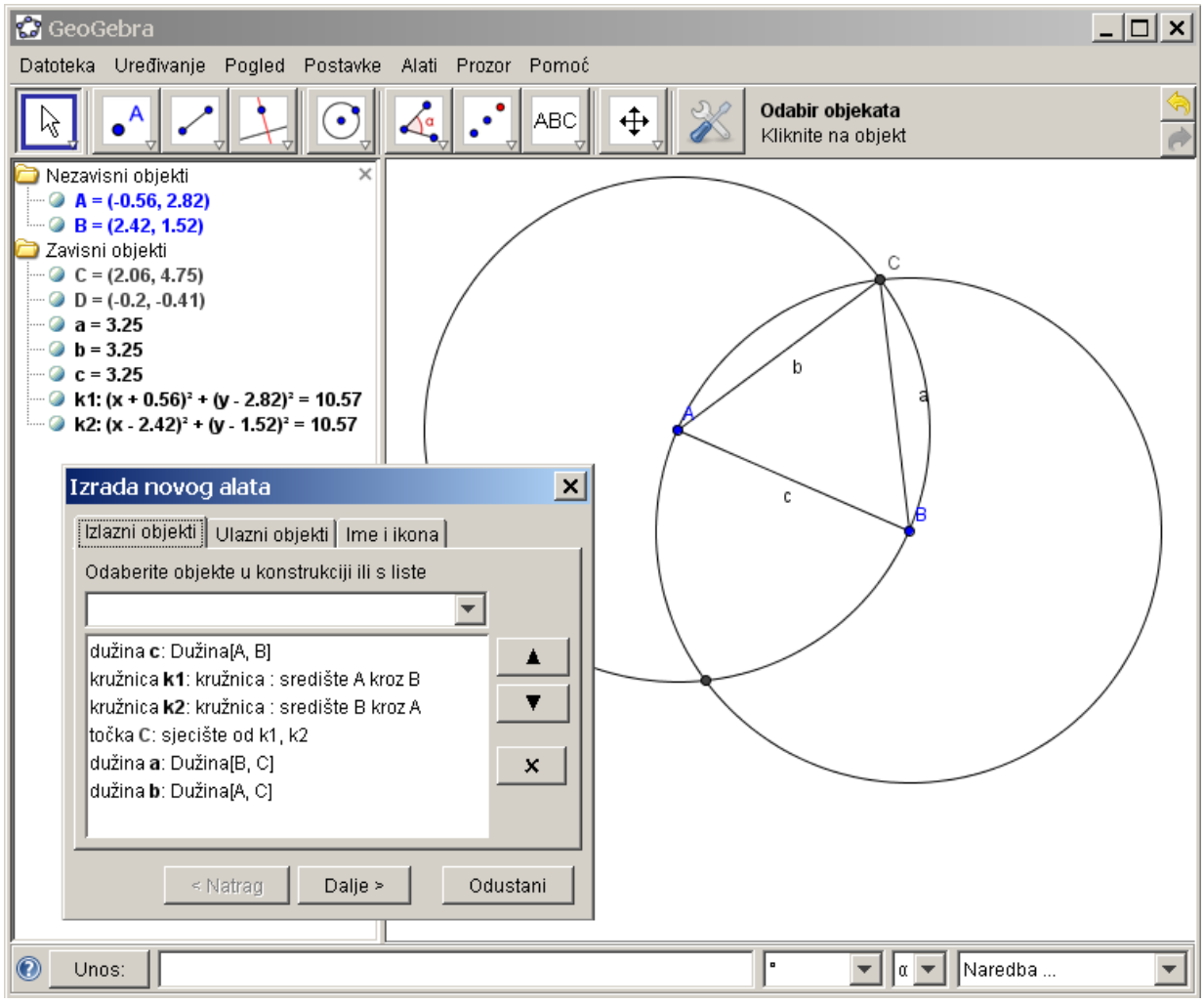
U ovom programu važno je da unutrašnja petlja počinje od $i + 1$. Kad bismo zamjene radili za $j = 1, \ldots, n$, time bismo matricu dva puta transponirali pa bi bila jednaka učitanoj matrici, slično kao u primjeru 6.10. Daljnji primjeri algoritama koji rade s matricama dani su u zadacima 7.20–7.31.

Vidimo da algoritmi imaju važnu ulogu u matematici. U primijenjenoj matematici cilj je efikasno implementirati na računalu matematičke objekte i računske operacije s njima, tako da se izvode što brže. To je važno u matematičkom modeliranju prirodnih pojava, simulacijama, računalnoj grafici i primjenama kao što su digitalne komunikacije, kriptografija, geografski informacijski sustavi, navigacija s pomoću računala, računalna tomografija i drugo. Pitanje postojanja efikasnih algoritama važno je i s teorijskog stanovišta. Jedno od najvećih otvorenih pitanja u matematici je takozvani problem P=NP i tiče se postojanja algoritama polinomijalne složenosti za određene klase problema (vidi [30]). Poznati matematički institut Clay uvrstio je P=NP među milenijske probleme, za koje nudi nagradu od milijun američkih dolara [3].

Algoritamski način razmišljanja prožima matematiku od njezinih početaka. U starogrčkoj matematici računanje nije imalo tako istaknutu ulogu, nego je u središtu interesa bila geometrija. Euklid u svojim "Elementima" opisuje mnoge konstrukcije ravnalom i šestarom (vidi [6] i [13]). S modernog stanovišta možemo reći da su geometrijske konstrukcije zapravo algoritmi za crtanje s pomoću ravnala i šestara. Naprimjer, konstrukciju jednakostraničnog trokuta možemo opisati sljedećim algoritmom. Ova konstrukcija sadržaj je prve propozicije Euklidovih "Elemenata".

⎡ ulaz: točke A, B
⎢ nacrtaj dužinu $\overline{AB}$
⎢ nacrtaj kružnicu k_1 sa središtem u A koja prolazi kroz B
⎢ nacrtaj kružnicu k_2 sa središtem u B koja prolazi kroz A
⎢ označi sjecište kružnica k_1 i k_2 sa C
⎢ nacrtaj dužinu $\overline{BC}$
⎣ nacrtaj dužinu $\overline{AC}$

Geometrijske konstrukcije na računalu možemo implementirati u programima dinamičke geometrije kao što su GeoGebra [9] i The Geometer's Sketchpad [10].



Slika 7.1: Konstrukcija jednakostraničnog trokuta u GeoGebri.

Zadaci

Zadatak 7.1. *Detaljno opišite rad algoritama za uniju, presjek i skupovnu razliku ako su na ulazu skupovi* $A = \{2, 5, 8, 10, 12, 15\}$ *i* $B = \{4, 5, 6, 10, 11, 12, 20\}$.

Zadatak 7.2. *Dokažite da za svaka dva skupa* A *i* B *vrijedi* $(A \setminus B) \cup (B \setminus A) = (A \cup B) \setminus (A \cap B)$. *Taj skup nazivamo* simetričnom razlikom *skupova* A *i* B *i označavamo ga s* $A \Delta B$.

Zadatak 7.3. *Napišite program koji učitava strogo rastuće nizove* a *i* b *koji predstavljaju skupove* A *i* B. *Program definira i ispisuje niz* c *koji predstavlja simetričnu razliku* $A \Delta B$.

Zadatak 7.4. *Napišite program koji učitava prirodne brojeve* m, n *i niz brojeva* $f[1], \ldots,$

$f[m]$ iz skupa $\{1, \ldots, n\}$. Program povjerava i ispisuje je li funkcija $f : \{1, \ldots, m\} \to \{1, \ldots, n\}$ reprezentirana tim nizom injekcija, odnosno surjekcija.

Zadatak 7.5. *Dokažite da je funkcija $f : \{1, \ldots, n\} \to \{1, \ldots, n\}$ injekcija ako i samo ako je surjekcija.*

Zadatak 7.6. *Dokažite da je funkcija bijekcija ako i samo ako ima inverznu funkciju. Na osnovu toga napišite program koji provjerava bijektivnost funkcije $f : \{1, \ldots, n\} \to \{1, \ldots, n\}$ u linearnom vremenu $O(n)$.*

Zadatak 7.7. *Neka je f permutacija skupa $\{1, \ldots, n\}$. Svaki par (i, j) elemenata tog skupa za koji vrijedi $i < j$ i $f(i) > f(j)$ nazivamo* inverzijom *permutacije f. Napišite program koji učitava permutaciju f i ispisuje broj inverzija te permutacije.*

Zadatak 7.8. Ciklička permutacija *ili* ciklus *$(i_1\ i_2 \cdots i_d)$ je permutacija koja preslikava $i_1 \mapsto i_2$, $i_2 \mapsto i_3$, $\ldots$, $i_{d-1} \mapsto i_d$, $i_d \mapsto i_1$, a ostale elemente domene preslikava u same sebe. Napišite program koji učitava proizvoljnu permutaciju f i prikazuje f kao produkt disjunktnih ciklusa.*

Zadatak 7.9. *Napišite program koji učitava prirodan broj n i ispisuje sve permutacije skupa $\{1, \ldots, n\}$.*

Zadatak 7.10. *Detaljno opišite rad algoritma za brzo potenciranje iz primjera 7.7 za ulaz $x = 3$, $i = 10$.*

Zadatak 7.11. *Detaljno opišite rad algoritama iz primjera 7.6, 7.8 i 7.9 za izračunavanje polinoma $f(x) = 2x^3 - x^2 + x + 3$ u točki $x = 2$.*

Zadatak 7.12. *Koliko operacija zbrajanja i množenja brojeva obavlja prvi algoritam iz primjera 7.10, a koliko operacija obavlja drugi algoritam za množenje polinoma?*

Zadatak 7.13. *Napišite program koji učitava stupnjeve m, n i koeficijente $a[0], \ldots, a[m]$, $b[0], \ldots, b[n]$ dvaju polinoma te izračunava i ispisuje koeficijente njihovog zbroja.*

Zadatak 7.14. *Napišite program koji učitava stupnjeve m, n i koeficijente $a[0], \ldots, a[m]$, $b[0], \ldots, b[n]$ dvaju polinoma te izračunava i ispisuje koeficijente njihove kompozicije.*

Zadatak 7.15. *Dokažite teorem o dijeljenju polinoma s ostatkom: za svaki polinom $f(x)$ i svaki polinom $g(x)$ različit od nulpolinoma postoje jedinstveni polinomi $q(x)$ i $r(x)$ takvi da je $f(x) = q(x) \cdot g(x) + r(x)$ i stupanj od $r(x)$ je manji od stupnja od $g(x)$. Polinom $q(x)$ nazivamo* kvocijentom*, a polinom $r(x)$* ostatkom *pri dijeljenju tih polinoma.*

Zadatak 7.16. *Napišite program koji učitava stupnjeve m, n i koeficijente $a[0], \ldots, a[m]$, $b[0], \ldots, b[n]$ dvaju polinoma te izračunava i ispisuje koeficijente kvocijenta i ostatka pri dijeljenju ta dva polinoma.*

Zadatak 7.17. *Napišite program koji učitava matricu tipa $m \times n$ i skalar (broj). Program definira i ispisuje produkt učitane matrice s učitanim skalarom.*

Zadatak 7.18. *Napišite program koji učitava kvadratnu matricu reds n te provjerava i ispisuje je li učitana matrica simetrična. Za matricu kažemo da je simetrična ako je jednaka svojoj transponiranoj matrici, tj. ako za sve indekse vrijedi* $a[i][j] = a[j][i]$.

Zadatak 7.19. *Sjetite se definicije i svojstava determinante kvadratne matrice (vidi [12]). Razmislite o tome koja je složenost algoritama za izračunavanje determinante reda n po definiciji, Laplaceovim razvojem i svođenjem na gornjetrokutasti oblik.*

Zadatak 7.20. *Napišite program koji učitava* $m \times n$ *matricu i ispisuje indekse svih redaka u kojima je zbroj elemenata pozitivan.*

Zadatak 7.21. *Napišite program koji učitava* $m \times n$ *matricu i ispisuje indekse svih stupaca u kojima ima paran broj nula.*

Zadatak 7.22. *Napišite program koji učitava* $m \times n$ *matricu s cjelobrojnim elementima i ispisuje ukupan broj neparnih elemenata matrice na mjestima s parnim indeksom stupca.*

Zadatak 7.23. *Napišite program koji učitava* $m \times n$ *matricu i ispisuje indeks retka u kojem je zbroj elemenata najveći.*

Zadatak 7.24. *Napišite program koji učitava* $m \times n$ *matricu i ispisuje indeks stupca u kojem je produkt elemenata najmanji.*

Zadatak 7.25. *Napišite program koji učitava* $m \times n$ *matricu i ispisuje indeks retka u kojem ima najviše nula te broj nula u tom retku.*

Zadatak 7.26. *Napišite program koji učitava* $m \times n$ *matricu i ispisuje indeks stupca u kojem ima najviše negativnih brojeva te broj negativnih brojeva u tom stupcu.*

Zadatak 7.27. *Napišite program koji učitava* $m \times n$ *matricu s cjelobrojnim elementima i ispisuje indeks retka u kojem ima najviše parnih brojeva.*

Zadatak 7.28. *Napišite program koji učitava* $m \times n$ *matricu i ispisuje indeks stupca u kojem je zbroj pozitivnih elemenata najveći.*

Zadatak 7.29. *Napišite program koji učitava* $m \times n$ *matricu s cjelobrojnim elementima i ispisuje indeks retka u kojem je zbroj parnih elemenata najmanji.*

Zadatak 7.30. *Napišite program koji učitava* $m \times n$ *matricu i pronalazi koji je od svih najvećih elemenata po stupcima najmanji. Program ispisuje taj element i indekse retka i stupca u kojima se nalazi.*

Zadatak 7.31. *Napišite program koji učitava* $m \times n$ *matricu i pronalazi koji je od svih najmanjih elemenata po recima najveći. Program ispisuje taj element i indekse retka i stupca u kojima se nalazi.*

Zadatak 7.32. Binarna realcija *na skupu* $\{1, \ldots, n\}$ *je podskup Kartezijeva produkta* $R \subseteq \{1, \ldots, n\} \times \{1, \ldots, n\}$. *Na računalu je možemo implementirati kao* $n \times n$ *matricu r popunjenu nulama i jedinicama. Ako je* $(i, j) \in R$ *stavljamo* $r[i][j] = 1$, *a ako* $(i, j) \notin R$ *stavljamo* $r[i][j] = 0$. *Napišite program koji učitava* 0-1 *matricu r i provjerava koja od sljedećih svojstava ima odgovarajuća binarna relacija:*

- *refleksivnost: za svaki* $i \in \{1, \ldots, n\}$, $(i, i) \in R$;
- *simetričnost: za sve* $i, j \in \{1, \ldots, n\}$, *ako je* $(i, j) \in R$, *onda je* $(j, i) \in R$;
- *antisimetričnost: za sve* $i, j \in \{1, \ldots, n\}$, *ako je* $(i, j) \in R$ *i* $(j, i) \in R$, *onda je* $i = j$;
- *tranzitivnost: za sve* $i, j, k \in \{1, \ldots, n\}$, *ako je* $(i, j) \in R$ *i* $(j, k) \in R$, *onda je* $(i, k) \in R$.

Zadatak 7.33. *Opišite u obliku algoritma konstrukciju kvadrata i konstrukciju pravilnog šesterokuta sa zadanom stranicom* $\overline{AB}$. *Implementirajte konstrukcije u programu GeoGebra [9] ili u nekom drugom programu za dinamičku geometriju.*

Zadatak 7.34. *Opišite u obliku algoritma i implementirajte na računalu konstrukcije pravilnog peterokuta i pravilnog deseterokuta.*

Poglavlje 8

Prikaz brojeva u računalu

U prethodnim poglavljima pretpostavljali smo da algoritmi egzaktno računaju s cijelim i s realnim brojevima. To u stvarnosti nije moguće zbog ograničenog kapaciteta memorije računala i zbog prirode realnih brojeva. U ovom poglavlju upoznat ćemo se s načinom na koji se brojevi prikazuju u računalu i s ograničenjima algoritama koja iz toga proizlaze.

Memorija računala sastoji se od elemenata koji poprimaju jedno od dva moguća stanja. Radnu memoriju (RAM) čine elektronički sklopovi *bistabili* (eng. flip-flop), realizirani u poluvodičkim integriranim krugovima. Kad ugasimo računalo, radna memorija se briše. Vanjska memorija je trajnog karaktera i za nju postoje različita tehnološka rješenja: tvrdi diskovi (magnetne memorije), CD-i i DVD-i (optički mediji), a u novije vrijeme flash memorije, tzv. "solid state diskovi". Svim tipovima memorije zajedničko je da se podaci pamte kao nizovi binarnih znamenaka, *bitova* – magnetiziranjem površine diska u suprotnim smjerovima, mikroskopskim udubinama i izbočinama na optičkom mediju, ili s dvije razine napona u elektroničkim sklopovima. Zato je binarni brojevni sustav osnova za prikaz brojeva i drugih podataka u računalu.

Memorija je organizirana u nizove od osam bitova koje nazivamo *bajtovima* (eng. byte). Bajt može poprimiti $2^8 = 256$ stanja. Veće jedinice za količinu memorije su kilobajt (kB) – 10^3 bajtova, megabajt (MB) – 10^6 bajtova, gigabajt (GB) – 10^9 bajtova i terabajt (TB) – 10^{12} bajtova. Prefiksi kilo, mega, giga i tera u SI sustavu označavaju potencije od 10. U računarstvu se umjesto toga često koriste bliske potencije od 2: kibibajt (KiB) označava $2^{10} = 1024$ bajtova, a mebibajt (MiB), gibibajt (GiB) i tebibajt (TiB) redom 2^{20}, 2^{30} i 2^{40} bajtova. Osobna računala danas tipično imaju nekoliko gigabajta radne memorije i tvrde diskove od nekoliko stotina gigabajta do nekoliko terabajta.

Procesor računala ne može direktno računati s brojevima u radnoj memoriji, nego ih prethodno mora dohvatiti u registre. Kapacitet procesorskih registara obično iznosi 32 ili 64 bita. Ako za računanje koristimo aritmetiku ugrađenu u procesor, ograničeni smo na 32 ili 64 binarne znamenke. Moguće je napisati vlastite algoritme za računanje s velikim brojevima, ali su oni znatno sporiji od procesorske aritmetike. Objasnit ćemo ograničenja te, strojne aritmetike.

Količina memorije koja stane u registar naziva se *riječ* (eng. word). Označimo s w broj bitova u jednoj riječi (u praksi je najčešće $w = 32$ ili $w = 64$). Niz bitova jedne riječi možemo shvatiti kao prikaz prirodnog broja ili nule u binarnom sustavu. Tome odgovara tip podataka *unsigned integer* u programskom jeziku C i nekim drugim programskim

jezicima. Najmanji prikazivi broj je tada 0, a najveći $2^w - 1$.

Kod prikaza cijelih brojeva vodeći bit označava predznak: 0 za prirodne brojeve i nulu, a 1 za negativne cijele brojeve. Negativni brojevi prikazuju se s pomoću *dvojnog komplementa* (eng. two's complement). Binarne znamenke se komplementiraju, tj. zamijene se znamenke 0 i 1, te se rezultat uveća za 1. Naprimjer, prikaz broja -22 u riječi duljine $w = 8$ dobije se od niza binarnih znamenaka 00010110 komplementiranjem 11101001 i dodavanjem 1:

$$\boxed{11101010}$$

Niz bitova kojim je broj prikazan u računalu označavat ćemo pravokutnim "okvirom". Ovaj način prikazivanja cijelih brojeva odgovara tipu *integer* u klasičnim programskim jezicima. Prednost prikaza s dvojnim komplementom je u tome što nije potreban poseban algoritam za oduzimanje (i odgovarajući aritmetički sklopovi procesora), nego se koristi isti algoritam kao za zbrajanje. Naprimjer, razliku $100-22$ u 8-bitnoj aritmetici računamo tako da zbrojimo prikaze brojeva 100 i -22:

$$\begin{array}{rr} & \boxed{01100100} \\ + & \boxed{11101010} \\ \hline 1 & \boxed{01001110} \end{array}$$

Prijenos u deveti bit zanemarujemo, pa prikaz 01001110 odgovara broju $78 = 100 - 22$.

Analogna metoda oduzimanja funkcionira u pozicionom sustavu s bilo kojom bazom b, a ne samo u binarnom sustavu. Komplementiranje je zamjena znamenaka koje u sumi daju $b - 1$. Naprimjer, u sustavu s bazom 10 zamjenjujemo $0 \leftrightarrow 9$, $1 \leftrightarrow 8$, $2 \leftrightarrow 7$, $3 \leftrightarrow 6$ i $4 \leftrightarrow 5$. Ako želimo izračunati razliku $253219 - 38492$, nadopunimo drugi broj nulom i komplementiramo ga: 961507. Dodamo jedan i zbrojimo 961508 s prvim brojem:

$$\begin{array}{rr} & 253219 \\ + & 961508 \\ \hline & \not{1}214727 \end{array}$$

Zanemarujemo vodeću jedinicu i tako dobijemo razliku 214727. Nije teško razumjeti zašto posupak funkcionira; komplement 961507 je zapravo razlika $999999 - 38492$, a kad ga uvećamo za jedan razlika $1000000 - 38492$. Zbrajanjem s 253219 i zanemarivanjem vodeće "jedinice" 1000000 dobivamo $253219 - 38492$. Objašnjenje je analogno u binarnom i u drugim sustavima.

Najveći prikazivi broj u riječi duljine w na ovaj način je $2^{w-1}-1$ i ima prikaz $\boxed{011\cdots11}$. Broj 0 i broj -1 imaju redom prikaze $\boxed{000\cdots00}$ i $\boxed{111\cdots11}$. Najmanji prikazivi broj je -2^{w-1}, s prikazom $\boxed{100\cdots00}$. Naprimjer, u 32-bitnoj aritmetici najmanji prikazivi cijeli broj je $-2^{31} = -2147483648$, a najveći prikazivi broj je $2^{31} - 1 = 2147483647$. U to se možemo uvjeriti ako prevedemo i pokrenemo sljedeći C program na 32-bitnom računalu.

```
#include <stdio.h>
int main()
{ int n;
  n=1;
  while (n>0) n=n+1;
  printf("Najmanji prikazivi broj: %d\n",n);
  n=n-1;
  printf("Najveci prikazivi broj : %d\n",n);
}
```

Program bi u pseudojeziku glasio ovako:

$n \leftarrow 1$
dok je $n > 0$ ponavljaj $n \leftarrow n + 1$
ispiši `"Najmanji prikazivi broj:"`, n
$n \leftarrow n - 1$
ispiši `"Najveci prikazivi broj :"`, n

Varijabla n inicijalizira se na 1 i povećava sve dok je pozitivna. Kad bismo imali egzaktnu aritmetiku, to bi bila beskonačna petlja. Međutim, zbog ograničenog broja bitova u riječi varijabla n će dosegnuti najveći prikazivi cijeli broj $\boxed{011\cdots11}$. Iduća naredba $n \leftarrow n+1$ pretvara ga u broj s prikazom $\boxed{100\cdots00}$. To je najmanji prikazivi broj i negativan je, pa program izlazi iz petlje i ispisuje ga. Naredba $n \leftarrow n - 1$ ponovo vraća n na najveći prikazivi cijeli broj i zatim ga program ispisuje. Na ekranu dobivamo sljedeći ispis.

```
Najmanji prikazivi broj: -2147483648
Najveci prikazivi broj : 2147483647
```

Odgovarajući program u Pythonu izgleda ovako:

```
n = 1
while n>0:
  n = n + 1
print "Najmanji prikazivi broj:", n
n = n - 1
print "Najveci prikazivi broj:", n
```

Ovaj program zaista se ponaša kao beskonačna petlja jer Python koristi vlastite algoritme za računanje s velikim cijelim brojevima u radnoj memoriji (RAM-u). Za jako velike brojeve ni radna memorija ne bi bila dovoljna, ali varijabla n raste presporo da bismo to dočekali. Za usporedbu, na procesoru od 2 GHz program u C-u dostiže najveći prikazivi broj 2147483647 u vremenu od oko 1.5 sekundi.

Realne brojeve nije moguće egzaktno implementirati na računalu. Problem su iracionalni brojevi, koji imaju beskonačne neperiodične decimalne zapise, naprimjer $\sqrt{2}$ ili π. Za ta dva broja moguće je napisati algoritme koji ispisuju njihove znamenke i to je na neki način njihova konačna egzaktna reprezentacija. Međutim, postoji jednostavan

argument da to nije moguće napraviti za sve iracionalne brojeve: algoritama ima manje nego iracionalnih brojeva.

U praksi nije važno što ne možemo egzaktno reprezentirati realne brojeve jer mjerenjem uvijek dobivamo samo konačno mnogo točnih znamenaka. Svaki realni broj $x \neq 0$ možemo zapisati u obliku $x = \pm 0.z_1 z_2 z_3 \ldots \cdot 10^e$, pri čemu su $z_i \in \{0, \ldots, 9\}$ znamenke i $z_1 \neq 0$, a e je cijeli broj. Naprimjer, 156.2846 zapisujemo kao $0.1562846 \cdot 10^3$, a 0.000123 kao $0.123 \cdot 10^{-3}$. Pamtimo konačno mnogo znamenaka $z_1, \ldots, z_k$ i reprezentiramo cijeli broj e u konačnoj aritmetici. Broj $m = 0.z_1 \ldots z_k$ naziva se *mantisa*, a $e \in \mathbb{Z}$ *eksponent*.

U računalu se koristi binarni sustav umjesto dekadskog; aproksimacija za x prikazuje se u obliku $\pm(0.z_1 \ldots z_k)_2 \cdot 2^e$, $z_i \in \{0, 1\}$, $z_1 \neq 0$. Budući da je z_1 uvijek 1, u memoriji računala pamti se samo preostale binarne znamenke mantise $z_2, \ldots, z_k$ i eksponent e reprezentiran fiksnim brojem bitova. Za predznak je također potreban jedan bit. Ovaj način reprezentacije približnih realnih brojeva naziva se *aritmetika s pomičnim zarezom*[1] (eng. floating-point arithmetic). U "single precision" aritmetici za prikaz se koristi 32 bita, od čega je 23 za mantisu, 8 za eksponent i jedan za predznak. Koristi se i "double precision" aritmetika s 64 bitova i "quadruple precision" aritmetika sa 128 bitova. Detalji implementacije opisani su međunarodnim standardom IEEE 754 [28]. Postoje softverske implementacije artimetike s pomičnim zarezom u kojima korisnik proizvoljno zadaje točnost računanja, npr. u programskom sustavu Mathematica [17]. U tom slučaju ograničenja su dostupna radna memorija računala i brzina izvođenja operacija, koja je znatno sporija nego kad se koriste algoritmi ugrađeni u aritmetičke sklopove procesora.

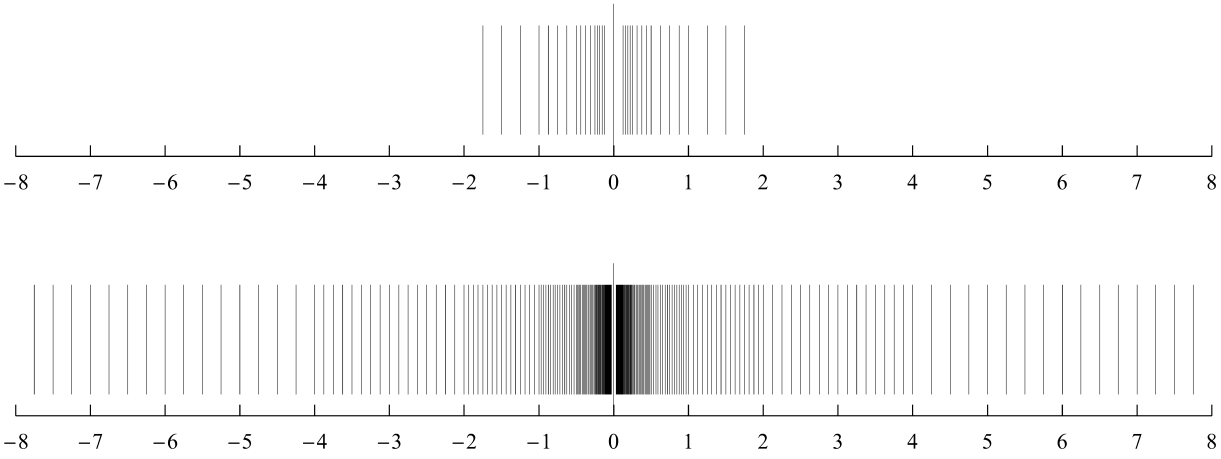
Opisat ćemo neka ograničenja floating point aritmetike, koja se pojavljuju bez obzira na točnost. Promotrimo najprije kako su na brojevnom pravcu raspoređeni prikazivi realni brojevi. Obzirom da koristimo konačan broj bitova, skup prikazivih brojeva također je konačan. Pretpostavimo da imamo samo 2 bita za mantisu, 2 bita za eksponent i 1 bit za predznak. Moguće mantise su $(0.100)_2 = \frac{1}{2}$, $(0.101)_2 = \frac{5}{8}$, $(0.110)_2 = \frac{3}{4}$ i $(0.111)_2 = \frac{7}{8}$. Ako se eksponent sprema u 2-bitnoj cjelobrojnoj aritmetici koju smo opisali,[2] mogući eksponenti su -2, -1, 0 i 1. To znači da binarnu decimalnu točku možemo pomaknuti jedno ili dva mjesta ulijevo i jedno mjesto udesno. Zajedno s predznakom imamo 32 prikazva broja koji su raspoređeni kao na slici 8.1 gore.

Vidimo da su prikazivi brojevi gusto raspoređeni blizu nule, a rjeđe dalje od nule. Ako zaokružimo broj 0.1234 na dvije značajne znamenke (tj. na 0.12) napravili smo malu pogrešku 0.0034, a ako zaokružimo 1234 na dvije značajne znamenke (tj. na 1200) pogreška je 34. Slično je kod binarne aritmetike s pomičnim zarezom. Povećavanjem broja bitova za mantisu raste gustoća prikazivih brojeva, a povećavanjem broja bitova za eksponent proširuje se interval u kojem leže prikazivi brojevi. Na slici 8.1 dolje vidimo prikazive realne brojeve s 4 bita u mantisi i 3 bita u eksponentu. U floating point aritmetici s 32, 64 i 128 bita gustoća prikazivih brojeva je veća i interval u kojem leže je znatno širi, ali raspored izgleda slično.

U aritmetici s pomičnim zarezom neke operacije su "opasne" jer dovode do gubitka

[1] Kao što smo već napomenuli u drugom poglavlju, po pravopisu se piše decimalni zarez umjesto decimalne točke. U ovoj skripti ipak pišemo decimalnu točku zbog čitljivosti nizova i formula u kojima se pojavljuju decimalni brojevi.

[2] Zapravo se eksponenti spremaju drugačije, npr. u single precision aritmetici koristi se 8 bitova, a prikazivi eksponenti su od -126 do 127.



Slika 8.1: Prikazivi realni brojevi uz mantisu od 2 bita i eksponent od 2 bita (gore), odn. mantisu od 4 bita i eksponent od 3 bita (dolje).

točnosti. Jedna takva operacija je oduzimanje dva bliska broja. Recimo da u dekadskom sustavu računamo s četiri značajne znamenke i želimo oduzeti brojeve 0.4697 i 0.4613. Rezultat 0.0084 pamtit će se u obliku $0.8400 \cdot 10^{-2}$, ali zadnje dvije znamenke nisu pouzdane i mogu biti bilo što. Zapravo smo izgubili dvije značajne znamenke. Na to nas računalo neće upozoriti, nego moramo sami voditi računa o pouzdanosti rezultata. Posljedica konačne točnosti jest i gubitak svojstava aritmetičkih operacija na koja smo navikli, npr. asocijativnosti. Recimo da želimo izračunati približnu sumu reda $\sum_{i=1}^{n} a_i$ pozitivnih realnih brojeva u padajućem redoslijedu $a_1 > a_2 > a_3 > \ldots$. Zbrajanjem u navedenom redoslijedu $(((a_1 + a_2) + a_3) + \cdots + a_{n-1}) + a_n$ dobit ćemo veću pogrešku nego zbrajanjem u suprotnom redoslijedu $a_1 + (a_2 + (a_3 + \cdots + (a_{n-1} + a_n)))$. Naime, može se dogoditi da suma velikih članova na početku bude znatno veća od narednih malih članova, pa je oni zbog ograničenog broja znamenaka mantise više ne mogu promijeniti. U floating point aritmetici postoji pozitivan broj $\varepsilon > 0$ sa svojstvom $1+\varepsilon = 1$ (najveći takav ε naziva se *strojnom točnošću*). Ako zbrajamo od manjih članova reda prema većima, parcijalna suma sporije raste i svi zbrojeni članovi utječu na rezultat, pa je pogreška manja.

U numeričkoj matematici mora se voditi računa i o problemima koji proizlaze iz konačne točnosti strojne aritmetike. Osim brzine izvođenja algoritama bitna je njihova *numerička stabilnost*, tj. ocjena pouzdanosti rezultata koje daju. Numerički nestabilni algoritmi dovode do gomilanja pogrešaka i gubitka značajnih znamenaka. Nekoliko primjera vezanih uz to dano je u zadacima.

Zadaci

Zadatak 8.1. *Odredite prikaze brojeva* 9734, -423 *i* -23456 *u* 16*-bitnoj aritmetici.*

Zadatak 8.2. *Koji brojevi u* 8*-bitnoj aritmetici imaju prikaze* $\boxed{01101111}$, $\boxed{11111110}$ *i* $\boxed{10000101}$*?*

Zadatak 8.3. *Koji je najveći, a koji najmanji prikazivi cijeli broj u* 16*-bitnoj aritmetici i koji su njihovi prikazi?*

Zadatak 8.4. *Koji je najveći, a koji najmanji prikazivi cijeli broj u* 64*-bitnoj aritmetici?*

Zadatak 8.5. *U* 8*-bitnoj binarnoj aritmetici izračunajte razliku* $99 - 55$.

Zadatak 8.6. *U* 16*-bitnoj binarnoj aritmetici izračunajte razliku* $4235 - 16777$.

Zadatak 8.7. *Metodom analognom dvojnom komplementu izračunajte razliku* $(212201)_3 - (21102)_3$ *u ternarnom sustavu.*

Zadatak 8.8. *Metodom analognom dvojnom komplementu izračunajte razliku* $(312203)_4 - (21302)_4$ *u sustavu s bazom* 4.

Zadatak 8.9. *Napišite program koji ispisuje najveći i najmanji prikazivi pozitivan realni broj. Inicijalizirajte realnu varijablu na* 1.0 *i množite, odnosno dijelite je s* 2 *dok se ne prestane mijenjati. Ispišite zadnji broj prije nego što se niz stabilizira.*

Zadatak 8.10. *Napišite program koji ispisuje strojnu točnost, tj. najveći pozitivni broj* $\varepsilon > 0$ *sa svojstvom* $1.0 + \varepsilon = 1.0$ *u aritmetici s pomičnim zarezom vašeg računala.*

Zadatak 8.11. *Napišite program za izračunavanje sume reda* $\sum_{n=1}^{\infty} \frac{1}{n^4}$. *Program treba pribrajati članove od većih prema manjima dok se suma ne prestane mijenjati. Zatim treba sumirati* 10 *puta više članova u obrnutom redoslijedu, od manjih prema većima. Ispišite prvu i drugu sumu i usporedite ih s točnim rezultatom,* $\frac{\pi^4}{90}$.

Poglavlje 9

Rješenja nekih zadataka

Zadatak 1.1. Najveći n-znamenkasti prirodni broj je $10^n - 1$. Zbroj bilo koja dva n-znamenkasta broja manji je od dvostrukog najvećeg n-znamenkastog broja, $2 \cdot (10^n - 1)$. Lako se vidi da je to manje od najmanjeg $(n+2)$-znamenkastog broja, 10^{n+1}. Dakle, zbroj bilo koja dva n-znamenkasta broja nema više od $n+1$ znamenaka.

Zadatak 2.3. Broj znamenaka prirodnog broja n u sustavu s bazom b je $\lfloor \log_b n \rfloor + 1$. Pritom je $\lfloor \, . \, \rfloor$ funkcija "najveće cijelo".

Zadatak 2.4. $(100110111)_2 = 311$, $(1201221)_3 = 1267$, $(42310)_5 = 2830$, $(2147)_8 = 1127$, $(\mathrm{DEDA})_{16} = 57050$.

Zadatak 2.5. $5790 = (1011010011110)_2 = (21221110)_3 = (13236)_8 = (169\mathrm{E})_{16}$.

Zadatak 2.6. $(8\mathrm{D6E})_{16} = (106556)_8$.

Zadatak 2.7. $(1111011)_2 + (10110)_2 = (10010001)_2$, $(122)_3 + (211)_3 + (101)_3 + (120)_3 + (222)_3 = (10100)_3$, $(\mathrm{BABA})_{16} + (\mathrm{DEDA})_{16} = (19994)_{16}$.

Zadatak 2.8. $(110011011)_2 - (110110)_2 = (101100101)_2$, $(1302)_4 - (123)_4 = (1113)_4$.

Zadatak 2.9. $(1101)_2 \cdot (101)_2 = (1000001)_2$, $(21201)_3 \cdot (1022)_3 = (100222122)_3$.

Zadatak 2.10. $(110323)_5 : (401)_5 = (123)_5$, $(1101)_2 : (110)_2 = (1.00\overline{01})_2$.

Zadatak 2.12. $(102.21)_3 = 106/9 = 11.\overline{7}$, $(0.4)_7 = 4/7 = 0.\overline{571428}$, $(3.03\overline{2})_5 = 157/50 = 3.14$.

Zadatak 2.13. $17.3125 = (100001.0101)_2 = (122.\overline{0221})_3$.

Zadatak 2.15. $(123.4567)_8 = (1010011.100101110111)_2$, $(10111011.101)_2 = (\mathrm{BB.A})_{16}$, $(443.5274)_8 = (123.\mathrm{ABC})_{16}$.

Zadatak 2.16. $b = 8$.

Zadatak 2.17. $(102342)_5$.

Zadatak 2.18. $b = 7$.

Zadatak 2.19. $(bbb)_{b^2} = (101010)_b$, $(b.b)_{b^2} = (10.1)_b$.

Zadatak 2.21. Vidi 8. poglavlje knjige [5].

Zadatak 3.1. Riječ je o Euklidovu algoritmu, koji izračunava najveću zajedničku mjeru brojeva a i b.

Zadatak 4.2. Ispisat će broj 440.

Zadatak 4.3. Program zamjenjuje brojeve pohranjene u varijablama x i y bez korištenja pomoćne varijable. Ovaj način zamjene funkcionira samo ako varijable sadrže brojeve. Osim toga može doći do pogrešaka zbog netočnog prikaza brojeva u računalu. Zato savjetujemo da se zamjena varijabli radi s pomoćnom varijablom, kao u primjeru 4.8.

Zadatak 4.7.

učitaj n
za $i = 1, \ldots, n$ ponavljaj
[ispiši $2\,i$

Zadatak 4.9. Rješenje s for petljom:

učitaj n
za $i = 1, \ldots, n$ ponavljaj
[ako je $(i \bmod 5) = 0$ onda
[[ispiši i

Rješenje s while petljom:

učitaj n
$i \leftarrow 5$
dok je $i \leq n$ ponavljaj
[ispiši i
[$i \leftarrow i + 5$

Zadatak 4.11.

učitaj n
$s \leftarrow 0$
za $i = 1, \ldots, n$ ponavljaj
[učitaj k
[ako je $(k \bmod 2) = 0$ onda
[[$s \leftarrow s + k$
ispiši s

Zadatak 4.13.

$br \leftarrow 0$
učitaj k
dok je $k \neq 0$ ponavljaj
⎡ ako je $(k \bmod 2) = 1$ onda
⎢ ⎡ $br \leftarrow br + 1$
⎣ učitaj k
ispiši br

Zadatak 4.16.

učitaj x
dok je $x < 0$ ponavljaj
⎡ učitaj x
ako je $x = 0$ onda
⎡ ispiši `"Nije učitan niti jedan pozitivan broj"`
inače
⎡ min $\leftarrow x$
⎢ učitaj x
⎢ dok je $x \neq 0$ ponavljaj
⎢ ⎡ ako je $(x > 0)$ i $(x < min)$ onda
⎢ ⎢ ⎡ $min \leftarrow x$
⎢ ⎣ učitaj x
⎣ ispiši `"Najmanji pozitivan učitani broj je"`, min

Zadatak 4.18.

učitaj n
$prethodni \leftarrow 0$
$sadasnji \leftarrow 1$
za $i = 1, \ldots, n$ ponavljaj
⎡ ispiši $sadasnji$
⎢ $pom \leftarrow sadasnji$
⎢ $sadasnji \leftarrow sadasnji + prethodni$
⎣ $prethodni \leftarrow pom$

Zadatak 4.19.

učitaj n
$prethodni \leftarrow 1$
$sadasnji \leftarrow 1$
dok je $sadasnji \leq n$ ponavljaj
⎡ ispiši $sadasnji$
⎢ $pom \leftarrow sadasnji$
⎢ $sadasnji \leftarrow sadasnji + prethodni$
⎣ $prethodni \leftarrow pom$

Zadatak 5.3.

učitaj n, b
$br \leftarrow 0$
dok je $n \neq 0$ ponavljaj
⌈ ako je $(n \bmod b) \neq 0$ onda $br \leftarrow br + 1$
⌊ $n \leftarrow n \operatorname{div} b$
ispiši br

Zadatak 5.4.

učitaj n
$max \leftarrow 0$
za $b = 2, \ldots, n$ ponavljaj
⌈ $m \leftarrow n$
| $br \leftarrow 0$
| dok je $m \neq 0$ ponavljaj
| ⌈ ako je $(m \bmod b) \neq 0$ onda $br \leftarrow br + 1$
| ⌊ $m \leftarrow m \operatorname{div} b$
| ako je $br > max$ onda
| ⌈ $max \leftarrow br$
⌊ ⌊ $baza \leftarrow b$
ispiši $baza$

Zadatak 5.5.

učitaj n, b
$p \leftarrow 1$
dok je $p \leq n$ ponavljaj $p \leftarrow p \cdot b$
dok je $p > 1$ ponavljaj
⌈ $p \leftarrow p \operatorname{div} b$
| $z \leftarrow n \operatorname{div} p$
| ispiši z
⌊ $n \leftarrow n - z \cdot p$

Zadatak 5.14. Na početku korisnik unese brojeve $a = 30$, $b = 21$. Algoritam ulazi u petlju i redom mijenja vrijednosti varijabli u $a = 9$, $b = 21$; $a = 9$, $b = 12$; $a = 9$, $b = 3$; $a = 6$, $b = 3$; $a = 3$, $b = 3$. Nakon pet prolaza kroz petlju ispisuje najveći zajednički djelitelj 3. Verzija Euklidova algoritma na str. 42 treba tri prolaza kroz petlju.

Zadatak 5.19.

učitaj k
$m \leftarrow k$
dok je $k \neq 0$ ponavljaj
⌈ učitaj k
| ako je $k \neq 0$ ponavljaj
| ⌈ $n \leftarrow k$
| | dok je $n \neq 0$ ponavljaj
| | ⌈ $pom \leftarrow n$
| | | $n \leftarrow m \bmod n$
| | ⌊ $m \leftarrow pom$
⌊
ispiši m

Zadatak 5.20.

učitaj n
$fi \leftarrow 1$
$k \leftarrow 2$
dok je $k < n$ ponavljaj
⌈ $a \leftarrow n$
| $b \leftarrow k$
| dok je $b \neq 0$ ponavljaj
| ⌈ $pom \leftarrow b$
| | $b \leftarrow a \bmod b$
| ⌊ $a \leftarrow pom$
| ako je $a = 1$ onda $fi = fi + 1$
⌊ $k \leftarrow k + 1$
ispiši fi

Zadatak 5.21. Pretpostavimo suprotno, da je $d > \sqrt{n}$. Budući da je d djelitelj od n, postoji $e \in \mathbb{N}$ takav da je $d \cdot e = n$. Broj n je složen, pa je $d < n$ i $e > 1$. Pretpostavili smo da je d najmanji djelitelj veći od 1, pa je $e \geq d > \sqrt{n}$. No tada je $d \cdot e > \sqrt{n} \cdot \sqrt{n} = n$, što je kontradikcija.

Zadatak 5.22.

učitaj n
za $i = 2, \ldots, n$ ponavljaj
⌈ $d \leftarrow 2$
| dok je $(i \bmod d \neq 0)$ i $(d^2 \leq i)$ ponavljaj $d \leftarrow d + 1$
⌊ ako je $d^2 > i$ onda ispiši i

Zadatak 5.23.

učitaj n
$i \leftarrow 2$
$br \leftarrow 0$
dok je $br < n$ ponavljaj
⌈ $d \leftarrow 2$
| dok je $(i \bmod d \neq 0)$ i $(d^2 \leq i)$ ponavljaj $d \leftarrow d + 1$
| ako je $d^2 > i$ onda
| ⌈ ispiši i
| ⌊ $br \leftarrow br + 1$
⌊ $i \leftarrow i + 1$

Zadatak 6.1.

učitaj n
za $i = 1, \ldots, n$ ponavljaj $a[i] \leftarrow 1$
$a[1] \leftarrow 0$
$p \leftarrow 2$
dok je $p \leq n$ ponavljaj
⌈ ako je $a[p] = 0$ onda $p \leftarrow p + 1$
| inače
| ⌈ ispiši p
| | $v \leftarrow p$
| | dok je $v \leq n$ ponavljaj
| | ⌈ $a[v] \leftarrow 0$
⌊ ⌊ ⌊ $v \leftarrow v + p$

Zadatak 6.3. Klasični algoritam sortira niz 7, 5, 10, 2, 4 na sljedeći način:

$i = 1, j = 2$:	5, 7, 10, 2, 4	(uspoređuje i zamjenjuje 1. i 2. član)
$i = 1, j = 3$:	5, 7, 10, 2, 4	(uspoređuje 1. i 3. član, nema zamjene)
$i = 1, j = 4$:	2, 7, 10, 5, 4	(uspoređuje i zamjenjuje 1. i 4. član)
$i = 1, j = 5$:	2, 7, 10, 5, 4	(uspoređuje 1. i 5. član, nema zamjene)
$i = 2, j = 3$:	2, 7, 10, 5, 4	(uspoređuje 2. i 3. član, nema zamjene)
$i = 2, j = 4$:	2, 5, 10, 7, 4	(uspoređuje i zamjenjuje 2. i 4. član)
$i = 2, j = 5$:	2, 4, 10, 7, 5	(uspoređuje i zamjenjuje 2. i 5. član)
$i = 3, j = 4$:	2, 4, 7, 10, 5	(uspoređuje i zamjenjuje 3. i 4. član)
$i = 3, j = 5$:	2, 4, 5, 10, 7	(uspoređuje i zamjenjuje 3. i 5. član)
$i = 4, j = 5$:	2, 4, 5, 7, 10	(uspoređuje i zamjenjuje 4. i 5. član)

Bubble sort algoritam radi zamjene ovim redoslijedom:

$i = 1, j = 1$:	5, 7, 10, 2, 4	(uspoređuje i zamjenjuje 1. i 2. član)
$i = 1, j = 2$:	5, 7, 10, 2, 4	(uspoređuje 2. i 3. član, nema zamjene)
$i = 1, j = 3$:	5, 7, 2, 10, 4	(uspoređuje i zamjenjuje 3. i 4. član)
$i = 1, j = 4$:	5, 7, 2, 4, 10	(uspoređuje i zamjenjuje 4. i 5. član)
$i = 2, j = 1$:	5, 7, 2, 4, 10	(uspoređuje 1. i 2. član, nema zamjene)
$i = 2, j = 2$:	5, 2, 7, 4, 10	(uspoređuje i zamjenjuje 2. i 3. član)
$i = 2, j = 3$:	5, 2, 4, 7, 10	(uspoređuje i zamjenjuje 3. i 4. član)
$i = 3, j = 1$:	2, 5, 4, 7, 10	(uspoređuje i zamjenjuje 1. i 2. član)
$i = 3, j = 2$:	2, 4, 5, 7, 10	(uspoređuje i zamjenjuje 2. i 3. član)
$i = 4, j = 1$:	2, 4, 5, 7, 10	(uspoređuje 1. i 2. član, nema zamjene)

Zadatak 6.6.

učitaj n
za $i = 1, \ldots, n$ ponavljaj učitaj $a[i]$
za $i = 1, \ldots, n-1$ ponavljaj
$\left[\right.$ $min \leftarrow i$
za $j = i+1, \ldots, n$ ponavljaj
$\left[\right.$ ako je $a[j] < a[min]$ onda min $\leftarrow j$
$pom \leftarrow a[i]$
$a[i] \leftarrow a[min]$
$a[min] \leftarrow pom$
za $i = 1, \ldots, n$ ponavljaj ispiši $a[i]$

Zadatak 6.11.

učitaj n
za $i = 1, \ldots, n$ ponavljaj učitaj $a[i]$
učitaj x
za $i = 1, \ldots, n$ ponavljaj
$\left[\right.$ ako je $a[i] = x$ onda ispiši i

Zadatak 6.12. Izvod složenosti algoritma mergesort nalazi se u poglavlju 2.5 skripte [18].

Zadatak 7.6. Ideja je pokušati definirati inverz zadane funkcije kao u primjeru 7.5. Ako neki element niza u kojeg spremamo inverz definiramo dva ili više puta, zadana funkcija nije bijekcija.

učitaj n
za $i = 1, \ldots, n$ ponavljaj učitaj $f[i]$
za $i = 1, \ldots, n$ ponavljaj $g[i] \leftarrow 0$
$bij \leftarrow$ ISTINA
$i \leftarrow 1$
dok je $i \leq n$ i bij ponavljaj
$\left[\right.$ $bij \leftarrow g[f[i]] = 0$
$g[f[i]] \leftarrow i$
$i \leftarrow i + 1$
ako je bij onda ispiši `"Funkcija je bijekcija"`
inače ispiši `"Funkcija nije bijekcija"`

Zadatak 7.8.

```
učitaj n
za i = 1, ..., n ponavljaj učitaj f[i]
i ← 1
dok je i ≤ n ponavljaj
⌈ ako je f[i] ≠ 0 onda
  ⌈ ispiši "(", i
  | j ← f[i]
  | f[i] ← 0
  | dok je j ≠ i ponavljaj
  | ⌈ ispiši j
  | | k ← f[j]
  | | f[j] ← 0
  | ⌊ j ← k
  ⌊ ispiši ")"
⌊ i ← i + 1
```

Zadatak 7.9. Ovaj algoritam ispisuje permutacije u leksikografskom poretku.

```
učitaj n
za i = 1, ..., n ponavljaj f[i] ← i
i ← 1
dok je i > 0 ponavljaj
⌈ za i = 1, ..., n ponavljaj ispiši f[i]
| i ← n − 1
| dok je i > 0 i f[i] > f[i + 1] ponavljaj i ← i − 1
| ako je i > 0 onda
| ⌈ j ← n
| | dok je f[j] < f[i] ponavljaj j ← j − 1
| | pom ← f[i]
| | f[i] ← f[j]
| | f[j] ← pom
| | za j = 1, ..., (n − i) div 2 ponavljaj
| | ⌈ pom ← f[i + j]
| | | f[i + j] ← f[n + 1 − j]
| | ⌊ f[n + 1 − j] ← pom
```

Zadatak 7.17.

učitaj m, n
za $i = 1, \ldots, m$ ponavljaj
$\big[$ za $j = 1, \ldots, n$ ponavljaj učitaj $a[i][j]$
učitaj s
za $i = 1, \ldots, m$ ponavljaj
$\Big[$ za $j = 1, \ldots, n$ ponavljaj
$\big[$ $b[i][j] \leftarrow s \cdot a[i][j]$
za $i = 1, \ldots, m$ ponavljaj
$\big[$ za $j = 1, \ldots, n$ ponavljaj ispiši $b[i][j]$

Zadatak 7.20.

učitaj m, n
za $i = 1, \ldots, m$ ponavljaj
$\big[$ za $j = 1, \ldots, n$ ponavljaj učitaj $a[i][j]$
za $i = 1, \ldots, m$ ponavljaj
$\Big[$ $s \leftarrow 0$
za $j = 1, \ldots, n$ ponavljaj $s \leftarrow s + a[i][j]$
ako je $s > 0$ onda ispiši i

Zadatak 7.21.

učitaj m, n
za $i = 1, \ldots, m$ ponavljaj
$\big[$ za $j = 1, \ldots, n$ ponavljaj učitaj $a[i][j]$
za $j = 1, \ldots, n$ ponavljaj
$\Big[$ $br \leftarrow 0$
za $i = 1, \ldots, m$ ponavljaj
$\big[$ ako je $a[i][j] = 0$ onda $br \leftarrow br + 1$
ako je $(br \bmod 2) = 0$ onda ispiši j

Zadatak 7.22.

učitaj m, n
za $i = 1, \ldots, m$ ponavljaj
$\big[$ za $j = 1, \ldots, n$ ponavljaj učitaj $a[i][j]$
$br \leftarrow 0$
za $i = 1, \ldots, m$ ponavljaj
$\Big[$ $j \leftarrow 2$
dok je $j \leq n$ ponavljaj
$\Big[$ ako je $(a[i][j] \bmod 2) = 1$ onda $br \leftarrow br + 1$
$\big[$ $j \leftarrow j + 2$
ispiši br

Zadatak 7.23.

učitaj m, n
za $i = 1, \ldots, m$ ponavljaj
$\lfloor$ za $j = 1, \ldots, n$ ponavljaj učitaj $a[i][j]$
$s \leftarrow 0$
za $j = 1, \ldots, n$ ponavljaj $s \leftarrow s + a[1][j]$
$max \leftarrow s$
$maxi \leftarrow 1$
za $i = 2, \ldots, m$ ponavljaj
$\lceil$ $s \leftarrow 0$
$\mid$ za $j = 1, \ldots, n$ ponavljaj $s \leftarrow s + a[i][j]$
$\mid$ ako je $s > max$ onda
$\mid$ $\lceil$ $max \leftarrow s$
$\lfloor$ $\lfloor$ $maxi \leftarrow i$
ispiši $maxi$

Zadatak 7.26.

učitaj m, n
za $i = 1, \ldots, m$ ponavljaj
$\lfloor$ za $j = 1, \ldots, n$ ponavljaj učitaj $a[i][j]$
$br \leftarrow 0$
za $i = 1, \ldots, m$ ponavljaj
$\lfloor$ ako je $a[i][1] < 0$ onda $br \leftarrow br + 1$
$max \leftarrow br$
$maxj \leftarrow 1$
za $j = 2, \ldots, n$ ponavljaj
$\lceil$ $br \leftarrow 0$
$\mid$ za $i = 1, \ldots, m$ ponavljaj
$\mid$ $\lfloor$ ako je $a[i][j] < 0$ onda $br \leftarrow br + 1$
$\mid$ ako je $br > max$ onda
$\mid$ $\lceil$ $max \leftarrow br$
$\lfloor$ $\lfloor$ $maxj \leftarrow j$
ispiši $maxj$

Zadatak 7.29.

učitaj m, n
za $i = 1, \ldots, m$ ponavljaj
$\lfloor$ za $j = 1, \ldots, n$ ponavljaj učitaj $a[i][j]$
$s \leftarrow 0$
za $j = 1, \ldots, n$ ponavljaj
$\lfloor$ ako je $(a[1][j] \bmod 2) = 0$ onda $s \leftarrow s + a[1][j]$
$min \leftarrow s$
$mini \leftarrow 1$
za $i = 2, \ldots, m$ ponavljaj
$\lfloor$ $s \leftarrow 0$
za $j = 1, \ldots, n$ ponavljaj
$\lfloor$ ako je $(a[i][j] \bmod 2) = 0$ onda $s \leftarrow s + a[i][j]$
ako je $s < min$ onda
$\lfloor$ $min \leftarrow s$
$mini \leftarrow i$
ispiši $mini$

Zadatak 7.30.

učitaj m, n
za $i = 1, \ldots, m$ ponavljaj
$\lfloor$ za $j = 1, \ldots, n$ ponavljaj učitaj $a[i][j]$
$max \leftarrow a[1][1]$
$maxi \leftarrow 1$
za $i = 2, \ldots, m$ ponavljaj
$\lfloor$ ako je $a[i][1] > max$ onda
$\lfloor$ $max \leftarrow a[i][1]$
$maxi \leftarrow i$
$minmax \leftarrow max$
$mmi \leftarrow maxi$
$mmj \leftarrow 1$
za $j = 2, \ldots, n$ ponavljaj
$\lfloor$ $max \leftarrow a[1][j]$
$maxi \leftarrow 1$
za $i = 2, \ldots, m$ ponavljaj
$\lfloor$ ako je $a[i][j] > max$ onda
$\lfloor$ $max \leftarrow a[i][j]$
$mmi \leftarrow i$
ako je $max < minmax$ onda
$\lfloor$ $minmax \leftarrow max$
$mmi \leftarrow maxi$
$mmj \leftarrow j$
ispiši $minmax, mmi, mmj$

Zadatak 7.34. Konstrukcije pravilnog peterokuta i deseterokuta opisane su u poglavlju 7.4 knjige [21].

Zadatak 8.1. $\boxed{0010011000000110}$, $\boxed{1111111001011001}$ i $\boxed{1010010001100000}$.

Zadatak 8.2. 111, -2 i -123.

Zadatak 8.3. $2^{15} - 1 = 32767$ i $-2^{15} = -32768$ s prikazima

$$\boxed{0111111111111111} \quad \text{i} \quad \boxed{1000000000000000}.$$

Zadatak 8.4. $2^{63} - 1 = 9223372036854775807$ i $-2^{63} = -9223372036854775808$.

Zadatak 8.5. Brojevi 99 i -55 imaju prikaze $\boxed{01100011}$ i $\boxed{11001001}$. Zbrajanjem i zanemarivanjem prijenosa dobivamo $\boxed{00101100}$, a to je prikaz broja 44.

Zadatak 8.6. Brojevi 4235 i -16777 imaju prikaze

$$\boxed{0001000010001011} \quad \text{i} \quad \boxed{1011111001110111}.$$

Zbrajanjem dobivamo $\boxed{1100111100000010}$, a to je prikaz broja -12542.

Zadatak 8.7. Komplement broja $(021102)_3$ je $(201120)_3$, a kad ga uvećamo za jedan $(201121)_3$. Zbrajanjem s $(212201)_3$ uz zanemarivanje vodeće znamenke (prijenosa) dobivamo $(121022)_3$.

Zadatak 8.9.

$x \leftarrow 1.0$
$y \leftarrow 0.0$
dok je $x! = y$ ponavljaj
$\left[\begin{array}{l} z \leftarrow y \\ y \leftarrow x \\ x \leftarrow 2 \cdot x \end{array}\right.$
ispiši `"Najveci prikazivi pozitivan broj je"`, z

$x \leftarrow 1.0$
$y \leftarrow 0.0$
dok je $x! = y$ ponavljaj
$\left[\begin{array}{l} z \leftarrow y \\ y \leftarrow x \\ x \leftarrow x/2 \end{array}\right.$
ispiši `"Najmanji prikazivi pozitivan broj je"`, z

Implementirajte ovaj algoritam na vašem računalu i pogledajte rezultat. Autor je u programskom jeziku Python dobio brojeve $8.98846567431 \cdot 10^{307}$ i $4.94065645841 \cdot 10^{-324}$. U progamskom jeziku C s tipom *float* dobio je $1.70141 \cdot 10^{38}$ i $1.4013 \cdot 10^{-45}$, a s tipom *double* iste vrijednosti kao u Pythonu.

Zadatak 8.10.

$jedan \leftarrow 1.0$
$epsilon \leftarrow 1.0$
dok je $jedan + epsilon \neq jedan$ ponavljaj
$\lfloor$ $epsilon \leftarrow epsilon/2$
ispiši $epsilon$

Implementirajte ovaj algoritam na vašem računalu i pogledajte rezultat. Autor je u programskim jezicima Python i C dobio broj $1.11022302463 \cdot 10^{-16}$.

Zadatak 8.11.

$s \leftarrow 0.0$
$p \leftarrow 1.0$
$n \leftarrow 1$
dok je $s \neq p$ ponavljaj
$\left\lceil \begin{array}{l} p \leftarrow s \\ s \leftarrow s + 1/n^4 \\ n \leftarrow n + 1 \end{array} \right\rfloor$
ispiši s

$n \leftarrow 10 \cdot n$
$s \leftarrow 0.0$
dok je $n > 0$ ponavljaj
$\left\lceil \begin{array}{l} s \leftarrow s + 1/n^4 \\ n \leftarrow n - 1 \end{array} \right\rfloor$
ispiši s

Implementirajte ovaj algoritam na vašem računalu i pogledajte rezultat. Autor je u programskom jeziku Python za prvu sumu dobio 1.082323233710861, a za drugu sumu 1.0823232337111379. Prema programu Mathematica [17] broj $\frac{\pi^4}{90}$ je 1.08232323371113819 na 17 decimala.

Indeks

Bibliografija

[1] M. Agrawal, N. Kayal, N. Saxena, *PRIMES is in P*, Annals of Mathematics, Second Series **160** (2004), 781-793.

[2] L. Budin, *Informatika: udžbenik za 1. razred gimnazije*, Element, Zagreb, 1996.

[3] Clay Mathematics Institute, *The Millennium Prize Problems*. `http://www.claymath.org/millennium/`

[4] *Code::Blocks*, `http://www.codeblocks.org`

[5] A. Dujella, *Fibonaccijevi brojevi*, Hrvatsko matematičko društvo, Zagreb, 2000.

[6] Euklid, *Elementi I–VI*, KruZak, Hrvatski Leskovac, 1999.

[7] *Free Pascal*. `http://www.freepascal.org/`

[8] *GCC, the GNU Compiler Collection*. `http://gcc.gnu.org/`

[9] *GeoGebra*. `http://www.geogebra.org/cms/`

[10] *The Geometer's Sketchpad Resource Center*. `http://www.dynamicgeometry.com/`

[11] *GNU Pascal*. `http://www.gnu-pascal.de/`

[12] K. Horvatić, *Linearna algebra, II. dio*, Sveučilište u Zagrebu, 2001.

[13] D.E. Joyce, *Euclid's Elements*. `http://aleph0.clarku.edu/~djoyce/java/elements/elements.html`

[14] B. Klaić, *Veliki rječnik stranih riječi, izraza i kratica*, Zora, Zagreb, 1968.

[15] D.E. Knuth, *The Art of Computer Programming. Volume 3: Sorting and Searching*, Addison-Wesley, 1975.

[16] G. Lamé, *Note sur la limite du nombre des divisions dans la recherche du plus grand commun diviseur entre deux nombres entiers*, Comptes rendus des séances de l'Académie des Sciences **19** (1844), 867-870.

[17] *Mathematica*. `http://www.wolfram.com/mathematica/`

[18] I. Nakić, *Diskretna matematika*, skripta, PMF-Matematički odsjek, 2009. `http://web.math.hr/nastava/komb/`

[19] J.J. O'Connor, E.F. Robertson, *Abu Ja'far Muhammad ibn Musa Al-Khwarizmi*, The MacTutor History of Mathematics archive, 1999.
http://www-history.mcs.st-andrews.ac.uk/Biographies/Al-Khwarizmi.html

[20] J.J. O'Connor, E.F. Robertson, *Babylonian numerals*, The MacTutor History of Mathematics archive, 2000.
http://www-history.mcs.st-andrews.ac.uk/HistTopics/Babylonian_numerals.html

[21] B. Pavković, D. Veljan, *Elementarna matematika I*, Tehnička knjiga, Zagreb, 1992.

[22] *Python Programming Language*. http://www.python.org/

[23] *Scratch*. http://scratch.mit.edu/

[24] R. Sedgewick, *Algorithms*, Addison-Wesley, 1984.

[25] The University of Chicago School Mathematics Project, *Algorithms in Everiday mathematics*, University of Chicago, 2001.
http://everydaymath.uchicago.edu/about/research/algorithms.pdf

[26] Wikipedia, *Age of the universe*, listopad 2010.
http://en.wikipedia.org/wiki/Age_of_the_universe

[27] Wikipedia, *History of programming languages*, listopad 2009.
http://en.wikipedia.org/wiki/History_of_programming_languages

[28] Wikipedia, *IEEE 754-2008*, prosinac 2010.
http://en.wikipedia.org/wiki/IEEE_754

[29] Wikipedia, *Primality test*, listopad 2010.
http://en.wikipedia.org/wiki/Primality_test

[30] Wikipedia, *P versus NP problem*, prosinac 2010.
http://en.wikipedia.org/wiki/P%3DNP