PRIRUČNIK ZA LABORATORIJSKE VJEŽBE IZ KOLEGIJA

MATEMATIČKE OSNOVE RAČUNARSTVA

C i C++

```
int main( )
{
int i;

for (i=20; i<=200; i=i+1)
    if (i%5==0) cout <<i;
}
```

Python

```
for i in range (20, 201):
  if i%5 == 0: print(i)
```

FERIT, Osijek

VINKO PETRIČEVIĆ

TOMISLAV RUDEC

Izdavač
Sveučilište Josipa Jurja Strossmayera u Osijeku
Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek

Za izdavača
prof. dr. sc. Tomislav Matić

Autori
doc. dr. sc. Vinko Petričević
doc. dr. sc. Tomislav Rudec

Prijelom i tehnička obrada
Anja Šteko, mag. educ. math. et. inf.

Recenzenti
izv. prof. dr. sc. Alfonzo Baumgartner
Sveučilište Josipa Jurja Strossmayera u Osijeku
Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek
doc. dr. sc. Ivan Vazler
Sveučilište Josipa Jurja Strossmayera u Osijeku
Odjel za fiziku

Lektor
doc. dr. sc. Dragana Božić Lenard
Sveučilište Josipa Jurja Strossmayera u Osijeku
Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek

Prosinac, 2024. godine

ISBN: 978-953-8184-09-3

Sveučilište Josipa Jurja Strossmayera u Osijeku

Suglasnost za izdavanje ovog sveučilišnog udžbenika donio je Senat Sveučilišta Josipa Jurja Strossmayera u Osijeku na 1. sjednici u akademskoj godini 2024./2025. održanoj 30. listopada 2024. godine pod brojem 36/24.

Sveučilište J. J. Strossmayera u Osijeku
Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek

Priručnik za laboratorijske vježbe iz kolegija Matematičke osnove računarstva

Vinko Petričević | **Tomislav Rudec**

Osijek, prosinac 2024.

Sadržaj

UVOD

Matematičke osnove računarstva izvode se na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek s 45 sati predavanja i 15 sati laboratorijskih vježbi.

U 45 sati predavanja obrađuju se matematička logika (s naglaskom na konjunktivnu i disjunktivnu normalnu formu i njihovo pojednostavljivanje i prirodnu dedukciju), relacije, konačni automati i Turingovi strojevi te teorija grafova.

U 15 sati laboratorijskih vježbi na računalima se rješavaju problemi za čije su rješavanje potrebna određena matematička znanja. Stoga, ono što se rješava na laboratorijskim vježbama predstavlja uglavnom primjenu matematičkih rezultata kod rješavanja problema računalom, a manje analizu matematičkih teorija na kojima je nastalo ili se sada nalazi računalstvo (kao što je to slučaj kod 45 sati teorije).

Na svakom od 15 sati (jedan sat tjedno ili češće blok od dvaju školskih sati jednom u dva tjedna) studenti rješavaju jedan lakši zadatak (jedan iz prve skupine, jedan od početnih deset zadataka ovog priručnika) te jedan ili dva teža zadatka, ovisno o vremenu potrebnom za rješavanje.

Budući da studenti dolaze s različitim razinama predznanja iz područja programiranja, potrebno je, zbog onih koji nisu vješti u programiranju, riješiti i lakše zadatke i detaljno objasniti rješenje (što se događa u pojedinom redu programa). Potrebno je i zadati za domaći uradak slične zadatke kako bi studenti s lošijim predznanjem iz programiranja dostigli razinu znanja potrebnu za studiranje na Fakultetu, čija je glavna tema računarstvo i informacijske tehnologije.

Nadalje, kako su neki studenti vještiji u programskom jeziku C ili C++, a neki u Pythonu itd., u priručniku smo pokušali dati rješenja zadataka u različitim programskim jezicima.

Prvih deset zadataka, tj. zadaci iz prvog poglavlja su najjednostavniji i za njihovo rješavanje potrebno je imati osnovna matematička znanja. To su zadaci koji će biti zadani na početku nastave iz laboratorijskih vježbi i za čije će rješavanje student dobiti svega nekoliko minuta.

U drugom dijelu priručnika nalaze se zadaci koji se mogu riješiti samo uz poznavanje i matematičkih rezultata i osnovnih tehnika programiranja.

U trećem dijelu priručnika nalaze se složeniji zadaci.

Pri rješavanju navedenih zadataka na nastavi ili kod kuće pri samostalnom radu, ključno je:

1. nakon što pročitamo zadatak, potrebno ga je prvo pokušati samostalno riješiti. Ako i nakon određenog vremena vidimo da nemamo ideju kako to treba napraviti, poslužimo se predloženom idejom. Nakon toga, čitatelj treba pokušati samostalno programski, u nekom od programskih jezika, riješiti problem.

2. kada je student riješio programskim kodom problem, trebalo bi program detaljno testirati i vidjeti kako se ponaša za određene vrste ulaza. Također, potrebno je pogledati programsko rješenje i pitati se može li se kreirani programski kod ubrzati ili riješiti drugačijom tehnikom, boljom ili lošijom itd. Drugim riječima, kada program završimo, i on radi ispravno, gotov je tek prvi dio posla. Nakon toga, program treba uljepšati, ubrzati, detaljno testirati i njime se poigrati.

1

Zadaci za zagrijavanje

(i za one koji programiranje uče od početka vježbanjem zadataka iz ove zbirke)

Zadatak 1.1.

Napravite program koji će za dva unesena prirodna broja ispisati njihov zbroj, razliku i umnožak.

Primjer 1. Za unesene $a = 4$, $b = 2$ računalo treba ispisati:
Zbroj je: 6
Razlika je: 2
Umnožak je: 8

Primjer 2. Za unesene $a = 4.12$, $b = 0$ računalo treba ispisati:
Zbroj je: 4.12
Razlika je: 4.12
Umnožak je: 0

Rješenje. Očito prvo treba s tipkovnice unijeti dva broja. Programu ćemo reći da ta dva unesena broja spremi u varijable a i b. Bez ta dva slova (tj. bez tih varijabli, koje ne moramo zvati a i b, nego mogu biti i x i y ili *prvibroj* i *drugibroj* itd.) nije moguće unijeti te podatke s tipkovnice.

Zatim ćemo programu s ,,`ispiši a+b`" reći da ispiše zbroj tih dvaju brojeva, s ,,`ispiši a-b`" reći da ispiše razliku tih dvaju brojeva, s ,,`ispiši a·b`" reći da ispiše umnožak tih dvaju brojeva. Dolje navedeni program u programskom jeziku C++ riješen je tako.

Također, program možemo napisati i na drugi način: opet prvo unesemo a i b, a zatim u novu varijablu c spremimo zbroj $a + b$, tj. napišemo $c = a + b$, što računalu znači da

prvo treba zbrojiti unesene brojeve a i b koje smo unijeli s tipkovnice, a onda taj zbroj ne treba ispisivati, nego treba pridružiti varijabli (slovu) c. Isto tako, u novu varijablu d spremimo razliku $a - b$, tj. napišemo $d = a - b$, u novu varijablu e spremimo umnožak $a \cdot b$, tj. napišemo $e = a * b$. Nakon toga, ispišemo c, d i e (rješenje u programskom jeziku C ispod).

Sve te varijable na početku ćemo morati ,,deklarirati", tj. reći programu unaprijed kakve će brojeve (cijele ili decimalne, i na koliko decimala) korisnik unijeti.

Programski kod 1.1: C++ kod za zadatak 1.1

```
#include <iostream>
using namespace std;
int main()
{
    double a, b;

    cout << "Unesi prvi broj: " << endl;
    cout << "a=";
    cin >> a;

    cout << "Unesi drugi broj: " << endl;
    cout << "b=";
    cin >> b;

    cout << "Zbroj je: " << a + b << endl;
    cout << "Razlika je: " << a - b << endl;
    cout << "Umnozak je: " << a * b << endl;
}
```

Programski kod 1.2: C kod za zadatak 1.1

```
#include<stdio.h>
int main()
{
    float a, b, c, d, e;

    printf("Unesi prvi broj: ");
    scanf("%f", &a);
    printf("Unesi drugi broj: ");
    scanf("%f", &b);

    c = a + b;
```

```
12      d = a - b;
13      e = a * b;
14
15      printf("Zbroj je: %f\n", c);
16      printf("Razlika je: %f\n", d);
17      printf("Umnozak je: %f\n", e);
18  }
```

Programski kod 1.3: Python kod za zadatak 1.1

```
1  print("Unesi broj a:")
2  a = float(input('a='))
3
4  print("Unesi broj b:")
5  b = float(input('b='))
6
7  zbroj = a + b
8  razlika = a - b
9  umnozak = a * b
10 print("Zbroj je ", zbroj)
11 print("Razlika je ", razlika)
12 print("Umnozak je ", umnozak)
```

ZADATAK 1.2.

Napravite program koji će za unesene duljine stranica trokuta ispisati njegov opseg i površinu.

Primjer 1. Za unesene $a = 3$, $b = 4$, $c = 5$ računalo treba ispisati
Opseg trokuta: 12
Površina trokuta: 6

Primjer 2. Za unesene $a = 4$, $b = 4.5$, $c = 7$ računalo treba ispisati
Opseg trokuta: 15.5
Površina trokuta: 8.41664

Rješenje. Unosimo tri broja: a, b i c.
Nakon toga, računamo $Opseg = a + b + c$ te površinu po Heronovoj formuli

$$P = \sqrt{s \cdot (s-a) \cdot (s-b) \cdot (s-c)},$$

gdje je s polovina opsega.
Na kraju treba ispisati opseg i površinu.

Programski kod 1.4: C++ kod za zadatak 1.2

```
1  #include <iostream>
2  #include <cmath>
3  using namespace std;
4
5  int main()
6  {
7      double a, b, c;
8
9      cout << "Unesi stranice trokuta. " << endl;
10
11     cout << "a=";
12     cin >> a;
13
14     cout << "b=";
15     cin >> b;
16
17     cout << "c=";
18     cin >> c;
19
20     double Opseg = a + b + c;
```

```
    double s = Opseg / 2;
    double P = sqrt(s * (s - a) * (s - b) * (s - c));

    cout << "Opseg trokuta: " << Opseg << endl;
    cout << "Povrsina trokuta: " << P << endl;
}
```

Programski kod 1.5: C kod za zadatak 1.2

```
#include <stdio.h>
#include <cmath>

int main()
{
    float a, b, c, Op, s, P;

    printf("Unesi stranice trokuta.\n");
    printf("a=");
    scanf("%f", &a);

    printf("b=");
    scanf("%f", &b);

    printf("c=");
    scanf("%f", &c);

    Op = a + b + c;
    s = Op / 2;
    P = sqrt(s * (s - a) * (s - b) * (s - c));

    printf("Opseg trokuta: %f\n", Op);
    printf("Povrsina trokuta: %f\n", P);
}
```

Programski kod 1.6: Python kod za zadatak 1.2

```
import math
print('Unesi stranice trokuta.')

a = float(input('a='))

b = float(input('b='))

```

```
8  c = float(input('c='))
9
10 Op = a + b + c
11 s = Op / 2
12 P = math.sqrt(s * (s - a) * (s - b) * (s - c))
13
14 print('Opseg trokuta: %0.2f' %Op)
15
16 print('Povrsina trokuta: %0.2f' %P)
```

ZADATAK 1.3.

Napišite program koji će ispisati prvu i zadnju znamenku unesenog prirodnog broja.

Primjer 1. Za uneseni broj 2048 računalo treba ispisati:
Prva znamenka je 2.
Zadnja znamenka je 8.

Rješenje. Prvo treba unijeti prirodni broj n.
Zadnja znamenka broja ostatak je pri dijeljenju zadanog broja s 10, tj. $n\%10$.
Prvu ćemo znamenku broja dobiti tako da cjelobrojno neprestano broj dijelimo s deset i tako mu u stvari svakim dijeljenjem s 10 ,,otkidamo" zadnju znamenku, sve dok ne preostane jedna, prva znamenka zadanog broja.
Uz cjelobrojno dijeljenje: $4321 : 10 = 432$, $432 : 10 = 43$, $43 : 10 = 4$.

Programski kod 1.7: C++ kod za zadatak 1.3

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int n;
6
7      cout << "Upisi prirodni broj. ";
8      cout << "n=";
9      cin >> n;
10
11     int zadnja = n % 10;
12     for( ; n >= 10; n = n / 10);
13
14     cout << "Prva znamenka je " << n << "." << endl;
15     cout << "Zadnja znamenka je " << zadnja << "." << endl;
16 }
```

Zadatak se može riješiti i bez petlje. Prva znamenka broja dobit će se tako da zadani broj cjelobrojno podijelimo s najvećom potencijom broja 10 koja je manja od njega. Uz cjelobrojno dijeljenje:

$$365 : 100 = 3, \quad 76\,543\,210 : 10\,000\,000 = 7.$$

Kako pronaći najveću potenciju broja 10 koja je i dalje manja od zadanog broja?
Ako broj ima tri znamenke, kao npr. 365, radi se o 10^2. Ako broj ima 8 znamenki,

kao 76 543 210, ta najveća potencija je 10^7. Dakle, ako broj ima n znamenki, treba ga cjelobrojno podijeliti s 10^{n-1}. Treba, dakle, izračunati koliko uneseni broj n ima znamenki. To je cjelobrojni dio broja $\log n$, gdje je log obični, dekadski logaritam, logaritam po bazi 10.

Dakle:

$$\text{prva} = (\text{int})\,(n/\text{pow}(10, \text{brojznam} - 1)),$$

gdje je $\text{brojznam} = (\text{int})\,\text{log10}(n) + 1$.
Ili, kraće,

$$\text{prva} = (\text{int})\,(n/\text{pow}(10, (\text{int})\,\log 10(n))),$$

gdje je $\text{pow}(x, y)$ zapis za x^y.

Ukupno, u programu:

Programski kod 1.8: Poboljšani C++ kod za zadatak 1.3

```
1  #include <iostream>
2  #include <cmath>
3  using namespace std;
4  int main()
5  {
6      int n;
7
8      cout << "Upisi prirodni broj: " << endl;
9      cout << "n=";
10     cin >> n;
11
12     int prva = (int) (n / pow(10, (int)log10(n)));
13     int zadnja = n % 10;
14
15     cout << "Prva znamenka je " << prva << "." << endl;
16     cout << "Zadnja znamenka je " << zadnja << "." << endl;
17 }
```

Iako je linija

```
12     int prva = (int) (n / pow(10, (int)log10(n)));
```

elegantna i matematički točna, zbog neprecizne pretvorbe iz podatka `long` u podatak `double`, moguće je da će kod velikih brojeva program dati netočan rezultat.

Programski kod 1.9: Python kod za zadatak 1.3

```
1 import math
2
3 n = int(input("Upisi prirodni broj: "))
4 prva = n // 10 ** int(math.log10(n))
5 zadnja = n % 10
6
7 print(f"Prva znamenka je {prva}.")
8 print(f"Zadnja znamenka je {zadnja}.")
```

Umjesto linije

```
3 n = int(input("Upisi prirodni broj: "))
```

moguće je ostaviti i stringovni ulaz, tj. unijeti n kao string (bez naredbe `int` na početku) te onda samo očitati posebnom naredbom prvu (kao $n[0]$) i zadnju (kao $n[-1]$) znamenku.

Na primjer:

```
n = input("Upisi prirodni broj: ")
prva = n[0]
zadnja = n[-1]
print(prva)
print(zadnja)
```

ZADATAK 1.4.

Napišite program koji će za zadani polumjer osnovice stošca i za zadanu visinu stošca u metrima ispisati je li volumen stošca veći od jedne litre (jednog kubnog decimetra) ili ne. Za π uzimamo $\pi = 3.14$.

Primjer 1. Za $r = 3$, $v = 2$ izlaz je
Volumen stošca veći je od jedne litre.

Primjer 2. Za $r = 0.01$, $v = 0.07$ izlaz je
Volumen stošca nije veći od jedne litre.

Rješenje. Unosimo dva podatka, polumjer baze stošca i visinu stošca. Za ta dva podatka treba samo izravno izračunati volumen po formuli

$$V = \frac{r^2 \cdot \pi \cdot v}{3}$$

i onda usporediti to s veličinom od jedne litre, tj. jednog kubičnog decimetra, tj. $1\,\mathrm{dm}^3$. Kako su uneseni podaci u metrima, i kako $1\,\mathrm{m}^3 = 1000\,\mathrm{dm}^3$, ukupni volumen mora biti veći od $0.001\,\mathrm{m}^3$.
Riješimo program u programskom jeziku C++ s $\pi = 3.14$ te u programskom jeziku Python s ugrađenim brojem π koji se nalazi u biblioteci `math`.

Programski kod 1.10: C++ kod za zadatak 1.4

```
1  #include <iostream>
2  #include <cmath>
3  using namespace std;
4
5  int main()
6  {
7      double r, h, c;
8
9      cout << "Unesi polumjer osnovice stosca. " << endl;
10     cout << "r=";
11     cin >> r;
12     cout << "Unesi visinu stosca. " << endl;
13     cout << "h=";
14     cin >> h;
15
16     double V = r * r * 3.14 * h / 3;
17
```

```
18      if (V > 0.001)
19          cout << "Volumen stosca veci je od jedne litre." << endl;
20      if (V <= 0.001)
21          cout << "Volumen stosca nije veci od jedne litre." << endl;
22  }
```

Programski kod 1.11: Python kod za zadatak 1.4

```
1  import math
2
3  print('Unesi polumjer osnovice stosca.')
4  r = float(input('r='))
5
6  print('Unesi visinu stosca.')
7  h = float(input('h='))
8
9  V = r * r * math.pi * h / 3
10
11 if V > 0.001:
12     print('Volumen stosca veci je od jedne litre.' )
13 if V <= 0.001:
14     print('Volumen stosca nije veci od jedne litre.' )
```

ZADATAK 1.5.

Napravite program za računanje indeksa tjelesne mase. Korisnik unosi svoju težinu u kilogramima i visinu u metrima, a računalo ispisuje indeks tjelesne mase te kojoj kategoriji korisnik po tim podacima pripada.

Indeks tjelesne mase računa se tako da se težina u kilogramima podijeli s kvadratom visine u metrima, tj.

$$\text{ITM} = \frac{m}{v^2},$$

gdje je ITM indeks tjelesne mase, m težina ispitanika, a v visina ispitanika.

Tablica po kojoj se odlučuje o stanju s obzirom na težinu dana je s

Tablica 1.1: Tablica indeksa tjelesne mase

ITM	**Stanje**
Ispod 18.5	Pothranjenost
18.5 – 24.9	Idealna težina
25 – 29.9	Umjerena debljina
30 – 39.9	Značajna debljina
Preko 40	Patološka pretilost

Primjer 1. Za unesenu težinu 80 kilograma i visinu 1.9 metara računalo ispisuje: **Idealna težina**.

Primjer 2. Za unesenu težinu 40 kilograma i visinu 1.9 metara računalo ispisuje: **Pothranjenost**.

Rješenje. Prvo ćemo napraviti unos podataka, tj. korisnik će s tipkovnice unijeti svoju težinu m i svoju visinu v. Formulu je lako napisati u programu, a i uvjete, no postavlja se pitanje kako najelegantnije napraviti sva ispitivanja.

Programski kod 1.12: C++ kod za zadatak 1.5

```
1 #include <iostream>
2 #include <cmath>
3 using namespace std;
4
5 int main()
6 {
```

```
7     float m, v;
8
9     cout << "Unesi svoju tezinu u kilogramima." << endl;
10    cout << "m=";
11    cin >> m;
12    cout << "Unesi svoju visinu u metrima (npr. 1.75)." << endl;
13    cout << "v=";
14    cin >> v;
15
16    float ITM = m / (v * v);
17
18    if (ITM < 18.5)
19        cout << "Pothranjeni ste." << endl;
20    if (ITM >= 18.5 and ITM < 25)
21        cout << "Idealne ste tjelesne mase." << endl;
22    if (ITM >= 25 and ITM < 30)
23        cout << "Umjereno ste debeli." << endl;
24    if (ITM >= 30 and ITM < 40)
25        cout << "Znacajno ste debeli." << endl;
26    if (ITM >= 40)
27        cout << "Patoloski ste debeli." << endl;
28
29   // Drugo rjesenje:
30   //  if (ITM < 18.5) cout << "Pothranjeni ste." << endl;
31   //  else if (ITM < 25) cout << "Idealne ste tjelesne mase." << endl;
32   //  else if (ITM < 30) cout << "Umjereno ste debeli." << endl;
33   //  else if (ITM < 40) cout << "Znacajno ste debeli." << endl;
34   //  else cout << "Patoloski ste debeli." << endl;
35 }
```

Programski kod 1.13: Python kod za zadatak 1.5

```
1  print("Unesi svoju tezinu u kilogramima.")
2  m = float(input("m="))
3  print("Unesi svoju visinu u metrima (npr. 1.75).")
4  v = float(input("v="))
5
6  ITM = m / (v * v)
7
8  if ITM < 18.5:
9      print("Pothranjeni ste.")
10 elif 18.5 <= ITM < 25:
11     print("Idealne ste tjelesne mase.")
12 elif 25 <= ITM < 30:
13     print("Umjereno ste debeli.")
```

```
14 elif 30 <= ITM < 40:
15     print("Znacajno ste debeli.")
16 else:
17     print("Patoloski ste debeli.")
```

ZADATAK 1.6.

Napišite program koji će ispisati sve brojeve od 20 do 200 koji su djeljivi s 5 uključujući 20 i 200.

Ispis: 20 25 30 ... 200.

Rješenje. U ovom programu na početku ništa ne unosimo.
U jednoj `for` petlji u kojoj će varijabla i, brojač, početi sa svojom vrijednošću 20 i završiti s 200 i svaki se puta povećavati za jedan, treba ispitati je li promatrani brojač i djeljiv s 5. Ako jest, onda ga ispišemo.

Programski kod 1.14: C++ kod za zadatak 1.6

```
#include <iostream>
using namespace std;
int main( )
{
    for (int i = 20; i <= 200; ++i)
        if (i % 5 == 0)
            cout << i << endl;
}
```

Programski kod 1.15: Python kod za zadatak 1.6

```
for i in range (20, 201):
    if i % 5 == 0: print(i)
```

Postoji i brže rješenje, bez ispitivanja. U petlji treba reći ,,brojaču" i da se svaki puta poveća za pet, a ne za jedan.

Programski kod 1.16: Brži C++ kod za zadatak 1.6

```
#include <iostream>
using namespace std;
int main( )
{
     for (int i = 25; i <= 200; i += 5)
        cout << i << endl;
}
```

Programski kod 1.17: Brži Python kod za zadatak 1.6

```
1 for i in range (20, 201, 5):
2     print(i)
```

ZADATAK 1.7.

Napišite program koji će za zadane prirodne brojeve a, b, c i d ispisati sve prirodne brojeve između a i b (uključujući tu i a i b) koji pri dijeljenju sa c daju ostatak d. Pri tome se podrazumijeva da je $a < b$ i da je $d < c$.

Primjer 1. Za unesene

$$1 \quad 20 \quad 5 \quad 1$$

izlaz je

$$1 \quad 6 \quad 11 \quad 16$$

jer su to brojevi između 1 i 20 koji pri dijeljenju s 5 daju ostatak 1.

Primjer 2. Za unesene

$$20 \quad 40 \quad 7 \quad 1$$

izlaz je

$$22 \quad 29 \quad 36.$$

Rješenje. Prvo moramo s tipkovnice unijeti brojeve a, b, c i d.
Nakon toga moramo se kretati od broja a do broja b i za svaki broj i koji se nalazi u intervalu od a do b, u toj petlji (koristit ćemo `for`), naredbom `if` provjerit ćemo ima li taj promatrani i pri dijeljenju sa c ostatak d. Ako da, treba ga ispisati.

Programski kod 1.18: C++ kod za zadatak 1.7

```
1  #include <iostream>
2  using namespace std;
3  int main( )
4  {
5      int a, b, c, d;
6      cout << "Unesi a, b c i d: ";
7      cin >> a >> b >> c >> d;
8
9      for (int i = a; i <= b; ++i)
10         if (i % c == d)
11             cout << i << endl;
12 }
```

Programski kod 1.19: Python kod za zadatak 1.7

```
a, b, c, d = map(int, input("Unesi a, b, c i d (odvojeni razmakom): ").
    split())

for i in range(a, b + 1):
    if i % c == d:
        print(i)
```

Uočimo da će se u slučaju velikog intervala od a do b petlja dugo izvršavati. U njoj imamo i jedno ispitivanje (`if`). Vrijeme izvršavanja programa može se dodatno smanjiti sljedećom izvedbom programa u kojoj petlja kreće od a i ide do b, ali ako prije nego dođe do b pronađe prvi broj i koji pri dijeljenju sa c ima ostatak d, onda ispisuje taj i te od tog broja sve brojeve koji su redom višekratnici od c zbrojeni s tim i. (Brojevi koji pri dijeljenju s 5 daju ostatak 3 su 3, $3+5$, $3+5+5$, $3+5+5+5$, ..., tj. ako i pri dijeljenju sa c daje rješenje x i ostatak d, onda $i+c$ pri dijeljenju sa c daje rješenje $x+1$ i ostatak d.)

Programski kod 1.20: Brži C++ kod za zadatak 1.7

```
#include <iostream>

using namespace std;
int main( )
{
    int a, b, c, d;
    cout << "Unesi a, b c i d: ";
    cin >> a >> b >> c >> d;

    for (int i = a; i <= b; ++i)
        if (i % c == d) {
            for (int j = i; j <= b; j += c)
                cout << j << " ";
            cout << endl;
            break;
        }
}
```

Programski kod 1.21: Brži Python kod za zadatak 1.7

```
a, b, c, d = map(int, input("Unesi a, b, c i d: ").split())

for i in range(a, b + 1):
    if i % c == d:
```

```
5         for i in range(i, b + 1, c):
6             print(i, end=" ")
7         break
```

Pitamo se može li se ovo rješenje još dodatno ubrzati? Ako su zadani brojevi 10, 30, 5, 2, kako odmah u petlji početi ispisivati brojeve od 12 (12 je prvi broj koji pri dijeljenju s 5 daje ostatak 2) te ispisati brojeve oblika 12, $12+5$, $12+5+5$, ...? Drugim riječima, kako naći prvi broj veći ili jednak broju a koji pri dijeljenju sa c daje ostatak d? Sljedeći korak ubrzanja algoritma ostavljamo čitatelju uz napomenu da je jedno od rješenje iskoristiti da i C i C++ i Python sadrže u svojim matematičkim funkcijama funkciju `ceil` koja daje najmanji cijeli broj koji je i dalje veći ili jednak zadanom, tj. $\text{ceil}(2.8) = 3$, $\text{ceil}(5) = 5$ itd.

ZADATAK 1.8.

Napišite računalni program koji će pronaći najmanji i najveći od n unesenih prirodnih brojeva i ispisati ih.

Primjer 1. Za unos:

3 4 4 15 40 3 2 1

računalo ispisuje:

Najmanji: 1

Najveći: 40

Rješenje. Očito treba prvo unijeti s tipkovnice broj n.
Zatim u petlji od 1 do n treba unijeti n prirodnih brojeva. Unijet ćemo ih u polje, vektor $A[i]$.
Nakon unosa podataka, treba pronaći najmanji i najveći podatak.
Za početak ćemo prvi uneseni broj, $A[1]$, postaviti za najmanji, ali i za najveći. Nakon toga, počevši od drugog podatka, ispitat ćemo je li promatrani podatak manji od do sada najmanjeg i ako jest, promatrani podatak postat će najmanji. Analogno ćemo napraviti i za najveći.

Programski kod 1.22: C kod za zadatak 1.8

```c
#include <stdio.h>
int main()
{
    int n, i, najmanji, najveci;
    printf("Unesi broj elemenata:");
    scanf("%d", &n);
    int A[n+1];

    printf("Unesi %d brojeva:\n", n);
    for (i = 1; i <= n; i = i + 1)
        scanf("%d", &A[i]);

    najmanji = A[1];
    najveci = A[1];

    for (i = 2; i <= n; i = i + 1)
    {
        if (A[i] < najmanji) najmanji = A[i];
        if (A[i] > najveci) najveci = A[i];
```

```
20      }
21
22      printf("Najmanji: %d\n", najmanji);
23      printf("Najveci: %d\n", najveci);
24  }
```

Napomena. Linije 6 i 7

```
6       scanf("%d", &n);
7       int A[n+1];
```

nije moguće izvesti u starijim verzijama programskog jezika C. Također, u linijama 10 i 11, kao i dalje u ovom udžbeniku, namjerno smo, u skladu s matematičkim formulama, indekse postavili od 1 do n, a ne od 0 do $n-1$.

Programski kod 1.23: Python kod za zadatak 1.8

```
1  n = int(input("Unesi broj elemenata: "))
2  A = [int(input()) for i in range(n)]
3
4  najmanji = A[0]
5  najveci = A[0]
6
7  for i in range(1, n):
8      if A[i] < najmanji:
9          najmanji = A[i]
10     if A[i] > najveci:
11         najveci = A[i]
12
13 print(f"Najveci: {najveci}")
14 print(f"Najmanji: {najmanji}")
```

Pogledajmo još dva načina kako se to može napraviti u programskom jeziku Python. U Pythonu, petlja se može kretati po elementima polja A.

Programski kod 1.24: 2. Python kod za zadatak 1.8

```
1  n = int(input("Unesi broj elemenata: "))
2  A = [int(input()) for i in range(n)]
3
4  najmanji = A[0]
5  najveci = A[0]
6
```

```
for n in A:
    if n < najmanji:
        najmanji = n
    if n > najveci:
        najveci = n

print(f"Najveci: {najveci}")
print(f"Najmanji: {najmanji}")
```

Također, možemo koristiti i ugrađenu funkciju `min` koja odmah pronalazi najmanji element, kao i funkciju `max` koja pronalazi najveći element.

Programski kod 1.25: 3. Python kod za zadatak 1.8

```
n = int(input("Unesi broj elemenata: "))
broj = int(input())
najveci = broj
najmanji = broj

for i in range(1, n):
    broj = int(input())
    if broj < najmanji:
        najmanji = broj
    if broj > najveci:
        najveci = broj

print("Najmanji:", najmanji)
print("Najveci:", najveci)
```

No, na zadnji riješeni način unesene brojeve kasnije više ne možemo koristiti jer ih nismo ,,zapamtili" u nekim varijablama ili u polju.

ZADATAK 1.9.

Kreirajte računalnu igru pogađanja broja.
Računalo neka zamisli (slučajnim odabirom odabere) broj od 1 do 100 i neka ispiše poruku ,,Zamislio sam broj od 1 do 100. Pokušaj pogoditi koji je to broj."
Korisnik tada upisuje broj.
Računalo nakon toga uspoređuje korisnikov broj sa zamišljenim brojem. Ako je zamišljeni broj veći, računalo ispisuje ,,Zamišljeni broj je veći.", a ako je manji analogno.
Korisnik dalje redom pogađa broj, a računalo mu daje izvještaj je li zamišljeni broj manji ili veći sve dok korisnik ne pogodi broj. Tada računalo na kraju ispisuje ,,Bravo, pogodio si u n-tom pokušaju.", gdje je n ukupan broj pokušaja koje je korisnik koristio.

Rješenje. Prvo je potrebno u neku varijablu staviti slučajni tajni broj od 1 do 100.

U programskim jezicima C++ i C kod izgleda ovako:

```
srand(time(0));
tajni = rand() % 100 + 1;
```

Dok u programskom jeziku Python, taj kod izgleda ovako:

```
import random
tajni = random.randint(1, 100)
```

U programskom jeziku C na početku je potrebno dodati

```
#include <stdlib.h>
#include <time.h>
```

odnosno u C++

```
#include <cstdlib>
#include <time.h>
```

Zatim u petlji po i radimo sljedeće stvari:

1. od korisnika tražimo da unese s tipkovnice svoj pokušaj i spremamo ga u varijablu `pokusaj`.

2. uspoređujemo broj „`pokusaj`" s brojem „`tajni`" i dajemo izvještaj o tome je li predloženi pokušaj broja veći ili manji od tajnog broja, a ako je pokušaj jednak tajnom broju, izlazimo iz petlje pogađanja i dajemo izvještaj kako je zamišljeni broj pogođen u i-tom pokušaju.

Rješenje u programskom jeziku C++ u sklopu `for` petlje koristi ispitivanje je li pokušaj jednak zamišljenom broju, tj. postupak u petlji izvršava se sve dok je predloženi broj različit od tajnog (`for(i = 1; pokusaj != tajni; ++i)`).

Rješenje u programskom jeziku Python u bezuvjetnoj petlji (`while (True)`) izvršava cijeli postupak unošenja broja i ispitivanja je li uneseni broj veći ili manji od zamišljenog, a iz petlje se izlazi naredbom `break`.

Programski kod 1.26: C++ kod za zadatak 1.9

```
#include <iostream>
#include <cstdlib>
#include <time.h>

using namespace std;
int main()
{
    srand(time(0));
    int tajni = rand() % 100 + 1;
    cout << "Zamislio sam broj od 1 do 100. ";
    cout << "Pokusaj pogoditi koji je to broj." << endl;

    int pokusaj = 0;
    for (int i = 1; pokusaj != tajni; ++i)
    {
        cout << endl << "Pokušaj:";
        cin >> pokusaj;
        if (pokusaj < tajni) cout << "Zamisljeni broj je veci";
        if (pokusaj > tajni) cout << "Zamisljeni broj je manji";
        if (pokusaj == tajni)
            cout << "Bravo, pogodio si u " << i << " - tom pokusaju" << endl;
    }
}
```

Programski kod 1.27: Python kod za zadatak 1.9

```
1  import random
2
3  tajni = random.randint(1, 100)
4  print("Zamislio sam broj od 1 do 100. ")
5  print("Pokusaj pogoditi koji je to broj.")
6  brojpok = 0
7  while(True):
8    pokusaj = int(input("Pokusaj: "))
9    brojpok = brojpok + 1;
10   if tajni < pokusaj:
11       print("Zamisljeni broj je manji.")
12   if tajni > pokusaj:
13       print("Zamisljeni broj je veci.")
14   if tajni == pokusaj:
15       print("Bravo, pogodio si u {}. pokusaju.".format(brojpok))
16       break
```

ZADATAK 1.10.

Napišite računalni program koji će unesenih n prirodnih brojeva poredati po veličini od najmanjeg do najvećeg i ispisati ih tako poredane.

Primjer 1. Za unos:

$$3 \quad 4 \quad 4 \quad 15 \quad 40 \quad 3 \quad 2 \quad 1,$$

računalo ispisuje:

$$1 \quad 2 \quad 3 \quad 3 \quad 4 \quad 4 \quad 15 \quad 40.$$

Rješenje. Brojeve se može poredati po veličini (tj. sortirati) na puno načina. Jedan od najčešćih ,,sortova" je ,,*selection sort*" (sortiranje izborom). Sortiranje izborom može se obavljati također na više načina, a najpoznatiji je način pronaći najmanji element u nizu i postaviti ga za prvo mjesto. Nakon toga, ponovo nalazimo najmanji element u preostalom dijelu niza i postavljamo ga za drugi element i tako dalje.

Postupak:
Ulaz: 3 4 4 15 40 3 2 1
Nakon prvog koraka (*nakon što uočimo da je* 1 *najmanji i zamijenimo ga s prvim elementom*):

$$1 \quad 4 \quad 4 \quad 15 \quad 40 \quad 3 \quad 2 \quad 3$$

Nakon drugog koraka (*nakon što uočimo da je od drugog elementa nadalje 2 najmanji i zamijenimo ga s drugim elementom*):

$$1 \quad 2 \quad 4 \quad 15 \quad 40 \quad 3 \quad 4 \quad 3$$

Nakon trećeg koraka:

$$1 \quad 2 \quad 3 \quad 15 \quad 40 \quad 4 \quad 4 \quad 3$$

I tako dalje.

Potrebno je u petlji proći cijelim poljem i pronaći najmanji element i postaviti ga na njegovo mjesto. To treba ponoviti onoliko puta koliko polje ima elemenata.

Rješenje u programskom jeziku C:

Programski kod 1.28: C kod za zadatak 1.10

```c
#include <stdio.h>
int main()
{
    int n, i, j, mjesto, zamjena;
    printf("Unesi broj elemenata:");
    scanf("%d", &n);

    int a[n+1];
    printf("Unesi %d brojeva:\n", n);
    for (i = 1; i <= n; i = i + 1)
        scanf("%d", &a[i]);

    for (i = 1; i <= n - 1; i = i + 1)
    {
        mjesto = i;
        for (j = i + 1; j <= n; j = j + 1)
            if (a[j] < a[mjesto])
                mjesto = j;

        if (mjesto != i) {
            zamjena = a[i];
            a[i] = a[mjesto];
            a[mjesto] = zamjena;
        }
    }
    printf("Poredano:");
    for(i = 1; i <= n; i = i + 1)
        printf("%d ", a[i]);
    printf("\n");
}
```

Programski kod 1.29: Python kod za zadatak 1.10

```python
n = int(input("Unesi broj elemenata: "))
print ("Unesi brojeve ")
a = [int(input()) for i in range(n)]

for i in range (n - 1):
    mjesto = i
    for j in range(i + 1, n):
```

```
8         if a[j] < a[mjesto]:
9             mjesto = j
10
11    if mjesto != i:
12        a[i], a[mjesto] = a[mjesto], a[i]
13
14 print("Poredano:")
15 for i in range(n):
16    print(a[i], end = ' ')
```

Uočimo u zadnjem kodu liniju

```
a[i], a[mjesto] = a[mjesto], a[i]
```

koja zamjenjuje sadržaje dviju varijabli bez korištenja treće.

2

Kratki tehnički zadaci

Zadatak 2.1.

Napišite program koji će za unesena tri prirodna broja, od kojih su dva od tih triju brojeva sigurno jednaka, a treći je sigurno različit od tih dvaju, ispisati broj koji je različit koristeći što manje naredbi ispitivanja ili petlji.

Primjer 1. Za unesene 2 8 8 treba ispisati 2.

Primjer 2. Za unesene 6 11 6 treba ispisati 11.

Rješenje. Zadatak lako možemo riješiti s tri `if` ispitivanja. Ako su a i b jednaki, ispiši c. Ako su a i c jednaki, ispiši b. Itd.

Programski kod 2.1: C++ kod s tri if-a za zadatak 2.1

```
1  #include <iostream>
2  using namespace std;
3  int main()
4
5  {
6      int a, b, c;
7      cin >> a >> b >> c;
8      if (a == b) cout << c << endl;
9      if (a == c) cout << b << endl;
10     if (b == c) cout << a << endl;
11 }
```

Programski kod 2.2: Python kod s tri if-a za zadatak 2.1

```
a = int(input())
b = int(input())
c = int(input())
if a == b:
    print(c)
elif a == c:
    print(b)
elif b == c:
    print(a)
```

No, znamo da je točno jedan uneseni broj različit od ostalih dvaju, pa ako nisu istinita prva dva `if`-a, treći mora sigurno biti.

Programski kod 2.3: C++ kod s dva if-a za zadatak 2.1

```
#include <iostream>
using namespace std;
int main()
{
    int a, b, c;
    cin >> a >> b >> c;
    if (a == b) cout << c;
    else {
        if (a == c) cout << b;
        else cout << a;
    }
    cout << endl;
}
```

Programski kod 2.4: Python kod s dva if-a za zadatak 2.1

```
a = int(input())
b = int(input())
c = int(input())
if a == b:
    print(c)
elif a == c:
    print(b)
else:
    print(a)
```

Razmislimo kako bismo mogli ovaj zadatak riješiti bez ikakvih ispitivanja (if, ?, switch-case itd.)

Svi su uneseni brojevi prirodni. Ako podijelimo dva prirodna broja (cjelobrojno), dobit ćemo nulu ukoliko podijelimo manji broj većim.

U daljem tekstu $\frac{a}{b}$ označava rezultat cjelobrojnog dijeljenja broja a brojem b. npr. $\frac{2}{7} = 0$. Ako smo uzeli dva unesena broja, na primjer a i b, onda će, ako su a i b različiti, $\frac{a}{b}$ biti nula ili će $\frac{b}{a}$ biti nula.

Dakle,

1. Ako su a i b različiti, bit će
$$\frac{a}{b} \cdot \frac{b}{a} = 0,$$
2. Ako su a i b jednaki, bit će
$$\frac{a}{b} \cdot \frac{b}{a} = 1,$$
3. Ako su a i b jednaki, treba ispisati c.

Stoga je konačni izraz koji treba ispisati:

$$\frac{a}{b} \cdot \frac{b}{a} \cdot c + \frac{a}{c} \cdot \frac{c}{a} \cdot b + \frac{b}{c} \cdot \frac{c}{b} \cdot a.$$

Programski kod 2.5: C++ kod za ,,matematičko" rješenje zadatka 2.1 bez korištenja if-a

```
1 #include<iostream>
2 using namespace std;
3 int main()
4 {
5     int a, b, c;
6     cin >> a >> b >> c;
7     cout << (a/b)*(b/a)*c + (a/c)*(c/a)*b + (b/c)*(c/b)*a << endl;
8 }
```

Ovo je ,,matematičko" rješenje, no postoji i ,,programersko". Za to rješenje trebamo se prisjetiti logičke operacije XOR (,,bitwise" ili bitovni operator).

Operacija XOR označava se s ^ i predstavlja ekskluzivnu disjunkciju dvaju binarnih zapisa, gdje je tablica zadana s

Tablica 2.1: Operacija XOR

A	B	A xor B (ili $A \hat{} B$)
0	0	0
0	1	1
1	0	1
1	1	0

Ako su dva broja jednaka, na primjer $a = 6$, $b = 6$, onda će a xor b biti 110 xor 110, a to je 0, tj. a xor $a = 0$.

Dalje, 0 xor $c = c$, za svaki broj c.

$$0 \quad \text{xor} \quad 0 = 0$$
$$0 \quad \text{xor} \quad 1 = 1.$$

Nula je neutralni element za operaciju xor baš kao što je

$$\frac{a}{b} \cdot \frac{b}{a} = 1$$

neutralni element za množenje u gornjem rješenju. Isto tako, lako se vidi da je xor komutativna i asocijativna operacija.

Stoga je sljedeće rješenje elegantnije i jednostavnije s programerskog stajališta. (Rezultat ,,bitwise" operatora pri ispisu postavljamo u zagradu.)

Programski kod 2.6: C++ kod za ,,programersko" rješenje zadatka 2.1 bez korištenja if-a

```
1 #include <iostream>
2 using namespace std;
3 int main()
4 {
5     int a, b, c;
6     cin >> a >> b >> c;
7     cout << (a ^ b ^ c) << endl;
8 }
```

Programski kod 2.7: Python kod za ,,programersko" rješenje zadatka 2.1 bez korištenja if-a

```
1 a = int(input())
2 b = int(input())
```

```
3 c = int(input())
4
5 print(a ^ b ^ c)
```

ZADATAK 2.2.

Napišite program koji će zamijeniti vrijednosti dviju varijabli bez korištenja treće.

Primjer 1. Za $a = 6$, $b = 4$, na kraju programa mora biti $a = 4$, $b = 6$.

Rješenje. Uz upotrebu treće varijable (od početka programiranja označava se s t ili *temp* – temporary ili privremena varijabla) rješenje je jednostavno:

Programski kod 2.8: C kod za rješenje zadatka 2.2 korištenjem privremene varijable

```c
#include <stdio.h>

int main()
{
    int a, b, temp;
    scanf("%d %d", &a, &b);

    temp = a;
    a = b;
    b = temp;

    printf("%d %d\n", a, b);
}
```

Programski kod 2.9: Python kod za rješenje zadatka 2.2 korištenjem privremene varijable

```python
a=int(input())
b=int(input())

temp = a
a = b
b = temp

print(f"{a} {b}")
```

Bez treće, privremene varijable moramo očito neku vrijednost smjestiti u jednu od dviju postojećih varijabli. Težina je zadatka u tome što se na početku čini kako će, čim smjestimo neku novu vrijednost u neku od varijabli, stara vrijednost biti izgubljena. No, odmah nakon pokušaja, vidi se da nije tako. Pokušajmo ono što je najprirodnije i

smjestimo u prvu varijablu zbroj dviju varijabli, tj.

$$a = a + b.$$

Sada je jasno da podaci nisu izgubljeni jer oduzmemo li $a - b$, i dalje su podaci o početnim vrijednostima a i b sačuvani.

Ukupno:

Programski kod 2.10: C++ kod za rješenje zadatka 2.2 bez korištenja privremene varijable

```cpp
#include <iostream>
using namespace std;
int main()
{
    int a, b;
    cin >> a >> b;

    a = a + b; // a je zbroj ulaznih vrijednosti
    b = a - b; // b je sada ono sto je bio a na pocetku
    a = a - b; // Od zbroja koji je u a oduzimamo u prethodnom redu
       izracunati b koji je u stvari pocetni a

    cout << a << " " << b << endl;
}
```

Napomena. Prethodni kod ne bi dobro radio da smo trebali zamijeniti vrijednost dviju varijabli, npr. tipa `double` jer zbog zaokruživanja brojeva, zbrajanje realnih brojeva nije komutativno.

Programski kod 2.11: Python kod za rješenje zadatka 2.2 bez korištenja privremene varijable

```python
a = float(input())
b = float(input())
a = a + b
b = a - b
a = a - b

print(f"{a} {b}")
```

Slično možemo napraviti s množenjem i dijeljenjem ako među ulaznim podacima nemamo nulu. Ako se ograničimo na cijele brojeve, možemo koristiti i

Programski kod 2.12: C++ kod za rješenje zadatka 2.2

```
#include <iostream>
using namespace std;
int main()
{
    int a, b;
    cin >> a >> b;

    a = a xor b;
    b = a xor b;
    a = a xor b;

    cout << a << " " << b << endl;
}
```

Python dozvoljava i sljedeći oblik rješavanja tog zadatka:

Programski kod 2.13: Python kod za rješenje zadatka 2.2

```
a = int(input())
b = int(input())
a,b = b,a
print(a, b)
```

C++ također ima naredbu `swap` koja zamjenjuje vrijednosti dviju varijabli.

ZADATAK 2.3.

Napišite program koji će za unesena tri broja ispisati udaljenost najudaljenijih dvaju od spomenutih triju, gdje pod udaljenost podrazumijevamo udaljenost njihovih točaka na brojevnom pravcu, ali uz što manje ispitivanja. (Drugim riječima, za unesene a, b i c, ispiši dijametar skupa $\{a, b, c\}$, tj. $\mathrm{diam}(\{a, b, c\})$).

Primjer 1. Za $a = 3$, $b = 7$, $c = 2$ rješenje je 5 jer je najveća udaljenost nekih od tih brojeva udaljenost od 2 do 7, a to je 5.

Primjer 2. Za $a = -3$, $b = 7$, $c = 2$ rješenje je 10 jer je najveća udaljenost nekih od tih brojeva udaljenost od -3 do 7, a to je 10.

Rješenje. Udaljenost dvaju brojeva jednaka je apsolutnoj vrijednosti njihove razlike:

$$d(a, b) = |a - b|.$$

No, kao ćemo naći najudaljenija dva?
Smjestimo zadana tri broja na brojevni pravac. Uočimo koliki je iznos broja udaljenost od a do b plus udaljenost od a do c plus udaljenost od b do c, tj. broja

$$|a - b| + |a - c| + |b - c|.$$

Kako god bili smješteni ti brojevi na pravcu, tj. bez obzira na njihov međusobni položaj, ako su, bez smanjenja općenitosti a i b najudaljeniji, onda je gornji zbroj dvostruka udaljenost od a do b. Navedeni zbroj triju apsolutnih vrijednosti dvostruka je udaljenost od najmanjeg do najvećeg broja.

Razmislimo li, uočavamo da je konačno rješenje polovina gore danog izraza, tj. konačno rješenje dano je sljedećim programom:

Programski kod 2.14: C++ kod za rješenje zadatka 2.3

```
1 #include <iostream>
2 #include <cmath>
3
4 using namespace std;
5 int main()
6 {
7     int a, b, c;
8     cin >> a >> b >> c;
9
```

```
10      cout << (abs(a - b) + abs(a - c) + abs(b - c)) / 2 << endl;
11  }
```

Napomena. Umjesto množenja ili dijeljenja cijelih brojeva potencijama broja 2, računalo će puno brže izvršiti operator << ili >>. Npr. linija 10 mogla bi biti:

```
10      cout << (abs(a - b) + abs(a - c) + abs(b - c) >> 1) << endl;
```

Treba još obratiti pozornost da, za razliku od množenja potencijom broja 2, operatori << i >> imaju manji prioritet od zbrajanja.

Programski kod 2.15: Python kod za rješenje zadatka 2.3

```
1  a = int(input())
2  b = int(input())
3  c = int(input())
4
5  print((abs(a - b) + abs(a - c) + abs(b - c)) // 2)
```

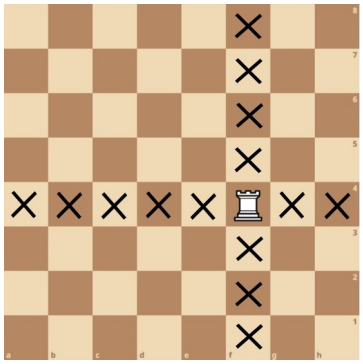
Zadatak 2.4.

Napišite program koji će za unesene koordinate topa i pješaka ispisati može li u sljedećem potezu top uzeti (pojesti) pješaka. (*Za koordinate uzimamo dva broja od* 1 *do* 8*, a pretpostavka je da su sve koordinate prirodni brojevi od* 1 *do* 8 *i da top i pješak ne stoje na istom mjestu te da je, osim tih dviju figura, šahovska ploča prazna, tj. da top ima čist put do pješaka*). Napišite programe i za slučajeve u kojima su na šahovskoj ploči umjesto topa kralj, skakač ili lovac. Koordinate gornjeg lijevog polja su $(1, 1)$, a desnog donjeg $(8, 8)$.

Primjer 1. Za 1 2 1 7 treba ispisati „**Da.**”

Primjer 2. Za 2 3 4 5 treba ispisati „**Ne.**”

Rješenje. Ako se top nalazi na koordinati (a, b) očito je on u a-tom redu i b-tom stupcu. Top može pojesti svaku figuru koja je u tom istom redu ili stupcu.



Slika 2.1: Kretanje topa.

Dakle, dovoljno je da pješak i top imaju prvu koordinatu jednaku i top će ga moći pojesti. Isto tako, ako top i pješak imaju drugu koordinatu jednaku opet, će ga moći pojesti.

Dakle, top koji je na (a, b) napada pješaka koji je na (c, d) ako i samo ako je $a = c$ ili $b = d$.

Programski kod 2.16: C++ kod za rješenje zadatka 2.4

```
1 #include <iostream>
2
3 using namespace std;
4 int main()
5 {
```

```
6     int a, b, c, d;
7     cin >> a >> b >> c >> d;
8
9     if (a == c or b == d)
10        cout << "Da." << endl;
11    else cout << "Ne." << endl;
12 }
```

Gornji kod možemo kraće napisati koristeći uvjetni (kondicionalni, ternarni) operator ,,?".

Programski kod 2.17: C++ kod za rješenje zadatka 2.4 koristeći uvjetni operator

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5     int a, b, c, d;
6     cin >> a >> b >> c >> d;
7
8        cout << (a == c || b == d ? "Da." : "Ne.") << endl;
9     //ili: a == c or b == d ? cout << "Da." << endl : cout << "Ne." << endl;
10  }
```

Programski kod 2.18: Python kod za rješenje zadatka 2.4

```
1 a = int(input())
2 b = int(input())
3 c = int(input())
4 d = int(input())
5
6 print("Da." if a == c or b == d else "Ne.")
```

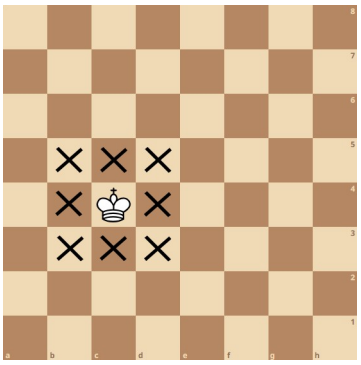
Promotrimo i preostale slučajeve:

1.) Slučaj u kojemu su na šahovskoj ploči postavljeni kralj i pješak:

Kralj se može pomaknuti samo za jedno polje u svih osam smjerova.
Ako je kralj na koordinati (a, b), pitamo se koje sve koordinate on može napasti.
Očito se radi o koordinatama

- $(a-1, b)$ (to je polje lijevo od kralja),
- $(a+1, b)$ (to je polje desno od kralja),

- $(a, b+1)$, $(a, b-1)$, $(a-1, b-1)$, $(a-1, b+1)$, $(a+1, b-1)$ i $(a+1, b+1)$.



Slika 2.2: Kretanje kralja.

Sada uvjet da kralj koji se nalazi na (a, b) uzme pješaka koji se nalazi na (c, d), u sljedećem potezu možemo napisati jednim `if` ispitivanjem u kojemu je 8 uvjeta. `if (a - 1 == c and b == d or ...)`, što je velik i kompliciran uvjet.

Dalje uočavamo kako će kralj moći uzeti pješaka ako se nalazi jedno mjesto od njega, tj. ako je njihova ,,udaljenost" 1 (u bilo kojemu smjeru). Udaljenost od a do c mora biti 0 ili 1 i udaljenost od b do d mora biti 0 ili 1, odnosno $\text{abs}(a-c) \leq 1$ i $\text{abs}(b-d) \leq 1$.

Ukupno:

Programski kod 2.19: C++ kod za slučaj u kojemu su na šahovskoj ploči postavljeni kralj i pješak.

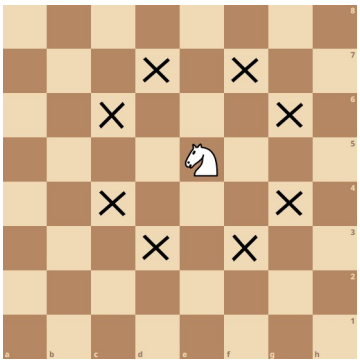
```
#include <iostream>
#include <cmath>

using namespace std;
int main()
{
    int a, b, c, d;
    cin >> a >> b >> c >> d;

    if (abs(a - c) <= 1 and abs(b - d) <= 1) cout << "Da." << endl;
    else cout << "Ne." << endl;
}
```

Programski kod 2.20: Python kod za slučaj u kojemu su na šahovskoj ploči postavljeni kralj i pješak.

```
a = int(input())
b = int(input())
c = int(input())
```

```
4 d = int(input())
5
6 print("Da." if abs(a - c) <= 1 and abs(b - d) <= 1 else "Ne.")
```

2.) Slučaj u kojemu su na šahovskoj ploči skakač i pješak:



Slika 2.3: Kretanje skakača.

Skakač se kreće za dva mjesta u jednom smjeru i onda za jedno mjesto u drugom smjeru, okomitom na prvi smjer.
Uzmimo, radi određenosti, da je skakač na mjestu $(3, 5)$. Koja sve polja on na tom mjestu napada?
To su polja: $(1, 4)$, $(1, 6)$, $(2, 3)$, $(2, 7)$, $(4, 3)$, $(4, 7)$, $(5, 4)$, $(5, 6)$.
Ako je skakač na koordinatama (a, b), moći će ,,skočiti" na bilo koje mjesto koje je oblika ,,$2 + 1$" od početne pozicije. Radi se, dakle, o poziciji $(a \pm 2, b \pm 1)$ ili $(a \pm 1, b \pm 2)$. Dakle, razlika x koordinata je 2, a y koordinata je 1 ili obrnuto. Uvjet uzimanja bi trebao, dakle, izgledati ovako:

```
if ((abs(a-c)==1 and abs(b-d)==2) or (abs(a-c)==2 and abs(b-d)==1))
    cout << "Da.";
```

Uočimo kako ovdje nismo morali izraz

```
abs(a - c) == 2 and abs(b - d) == 1
```

staviti u zagradu jer operator ,,and" u programskom jeziku C++ (a tako je i u programskim jezicima C i Python) ima veći prioritet od operatora (tj. veznika) ,,or".

I sada, uz još samo malo razmišljanja prema optimiziranju, dobivamo konačno rješenje:
`if (abs(a - c) * abs(b - d) == 2)` ispiši ,,**Da.**".

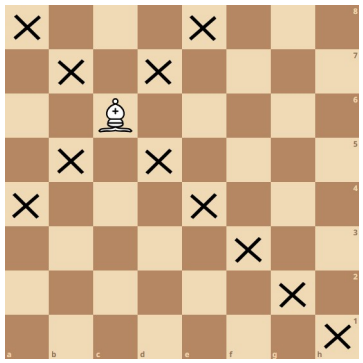
Programski kod 2.21: C++ kod za slučaj u kojemu na šahovskoj ploči stoje skakač i pješak

```cpp
#include <iostream>
#include <cmath>

using namespace std;
int main()
{
    int a, b, c, d;
    cin >> a >> b >> c >> d;

    if (abs(a - c) * abs(b - d) == 2) cout << "Da." << endl;
        else cout << "Ne." << endl;
}
```

Programski kod 2.22: Python kod za slučaj u kojemu na šahovskoj ploči stoje skakač i pješak

```python
a = int(input())
b = int(input())
c = int(input())
d = int(input())

print("Da." if abs(a - c) * abs(b - d) == 2 else "Ne.")
```

3.) Slučaj u kojemu su na šahovskoj ploči lovac i pješak:



Slika 2.4: Kretanje lovca.

Ako je lovac na polju (a, b), on može uzeti svaku onu figuru koja je na nekoj od dviju dijagonala po kojima se lovac kreće.

Ako je lovac na polju s koordinatama (a, b), polja koja su desno dolje su oblika $(a+1, b+1)$, $(a+2, b+2)$, $(a+3, b+3)$, itd., a polja koja su lijevo gore su oblika $(a-1, b-1)$, $(a-2, b-2)$, $(a-3, b-3)$, itd.

Pogledamo li dijagonale šahovske ploče (kose linije koje idu od lijevo gore prema desno dolje), uočavamo da svako polje određene dijagonale ima fiksnu razliku desne i

lijeve koordinate. Stoga, zaključujemo da će lovac s koordinate (a, b) moći dohvatiti sva ona polja čija je razlika koordinata $b - a$.

Stoga će uvjet za polja ove lovčeve dijagonale glasiti ovako:

```
if (d - c == b - a) cout << "Da.";
```

Odnosno, drugačije zapisano,

```
if (a - c == b - d) cout << "Da.";
```

Za drugu dijagonalu razmišljanje je slično, a uvjet je:

```
if (a - c == d - b) cout << "Da.";
```

Ukupno imamo:

Programski kod 2.23: C++ kod za slučaj u kojemu na šahovskoj ploči stoje lovac i pješak

```
#include <iostream>
#include <cmath>

using namespace std;
int main()
{
    int a, b, c, d;
    cin >> a >> b >> c >> d;

    if (abs(a - c) == abs(b - d)) cout << "Da." << endl;
        else cout << "Ne." << endl;
}
```

Programski kod 2.24: Python kod za slučaj u kojemu na šahovskoj ploči stoje lovac i pješak

```
a = int(input())
b = int(input())
c = int(input())
d = int(input())

print("Da." if abs(a - c) == abs(b - d) else "Ne.")
```

ZADATAK 2.5.

Napišite program koji će za uneseni broj a ispisati je li prost ili ne.

Primjer 1. Za uneseni broj 91 računalo ispisuje **„Nije prost"**.

Primjer 2. Za uneseni broj 101 računalo ispisuje **„Prost je"**.

Rješenje. Prirodni je broj prost ako ima točno dva djelitelja, tj. djeljiv je samo s brojem jedan i sa samim sobom. Po dogovoru, također, broj 1 nije prost broj.

Ostali brojevi, kao na primjer navedeni broj 91, koji je djeljiv s 1 i sa samim sobom, kao i svi ostali prirodni brojevi, ali je djeljiv i sa 7 i s 13, zovu se složeni brojevi.

Razmislimo li, uočit ćemo da ne možemo (niti ćemo ikada moći, jer se to kosi sa zakonima logike) računalu reći da „ako je zadani broj a djeljiv samo s jedan i sa samim sobom, ispiši da je prost.". Tako bismo mogli postaviti upit programu kada bismo prethodno zadani prirodni broj rastavili na faktore, npr. $36 = 2 \cdot 2 \cdot 3 \cdot 3$ (svaki se broj po Osnovnom teoremu aritmetike može rastaviti na umnožak prostih faktora).

Ideja 1. Provjerimo je li a djeljiv s nekim od brojeva od 2 do $a-1$ (ne treba provjeravati brojeve koji su veći od a jer a može biti djeljiv samo s brojevima od 1 do a – po definiciji, a je djeljiv s b ako postoji treći prirodni broj, c, takav da je $a = b \cdot c$).

Programski kod 2.25: C++ kod za 1. ideju zadatka 2.5

```
1  #include <iostream>
2
3  using namespace std;
4  int main()
5  {
6      long a;
7      cin >> a;
8
9      int djeljiv = 0;
10
11     for (long i = 2; i < a; ++i) if (a % i == 0) djeljiv = 1;
12
13     if (djeljiv == 0 && a != 1) cout << "Prost je." << endl;
14     else cout << "Nije prost." << endl;
15 }
```

Programski kod 2.26: Python kod za 1. ideju zadatka 2.5

```
a = int(input())
djeljiv = False

for i in range(2, a):
    if a % i == 0:
        djeljiv = True

print("Nije prost." if djeljiv or a == 1 else "Prost je.")
```

Pozitivna strana ideje 1 je njena jednostavnost. Negativno je sljedeće: iz gornjeg programa vidljivo je da ćemo za djeljivost u petlji isprobati sve brojeve do $a - 1$. Ako unesemo, npr. $a = 999\,999\,999$, petlja će raditi dugo i na kraju ispisati da broj nije prost, a to smo shvatili već kod $i = 3$ jer je zadani broj djeljiv s 3.

Ideja 2. Iz petlje treba izaći čim je a djeljiv s nekim brojem iz for petlje, tj. puno je brže sljedeće rješenje (jer naredba `break` spriječi nepotrebne provjere):

Programski kod 2.27: C++ kod za 2. ideju zadatka 2.5

```
#include <iostream>

using namespace std;
int main()
{
    long a;
    cin >> a;

    bool prost = a != 1;

    for (long i = 2; i < a; ++i) if (a % i == 0) {prost = false; break;}

    if (prost) cout << "Prost je." << endl;
    else cout << "Nije prost." << endl;
}
```

Programski kod 2.28: Python kod za 2. ideju zadatka 2.5

```
a = int(input())
djeljiv = False

for i in range(2, a):
```

```
5     if a % i == 0:
6         djeljiv = True
7         break
8
9 print("Nije prost." if djeljiv or a == 1 else "Prost je.")
```

Pozitivna strana ideje 2 je što je programski kod i dalje jednostavan, tj. samo smo dodali jedan „break" koji će nas izvesti iz petlje ako naiđemo na djeljivost.
Negativna strana ideje 1 i ideje 2 je što i dalje provjerava (ako je a prost) sve brojeve od 2 do $a-1$ pa će u slučaju velikog broja a, koji je prost, biti izvršene mnoge nepotrebne provjere.
Pogledajmo situaciju s brojem 50:

$$50 = 2 \cdot 25 = 5 \cdot 10.$$

Ako je broj djeljiv s 2, bit će i s $a/2$. Ako je djeljiv s 3, bit će i s $a/3$. I tako dalje.
Drugim riječima, ako broj a nije djeljiv ni s jednim brojem manjim od $a/2$, neće biti djeljiv niti s jednim većim od $a/2$.

Ideja 3. U petlji po i treba provjeravati samo brojeve od 2 do $a/2$.

Programski kod 2.29: C++ kod za 3. ideju zadatka 2.5

```
1  #include <iostream>
2
3  using namespace std;
4  int main()
5  {
6      long a;
7      cin >> a;
8
9      int djeljiv = 0;
10     long b = a / 2;
11
12     for (long i = 2; i <= b; ++i) if (a % i == 0) {djeljiv = 1; break;}
13     if (a == 1) djeljiv = 1;
14
15     if (djeljiv == 0) cout << "Prost je." << endl;
16     if (djeljiv == 1) cout << "Nije prost." << endl;
17 }
```

Programski kod 2.30: Python kod za 3. ideju zadatka 2.5

```python
a = int(input())
djeljiv = False

for i in range(2, a//2):
    if a % i == 0:
        djeljiv = True
        break

print("Nije prost." if djeljiv or a == 1 else "Prost je.")
```

Sada je kod i dalje jednostavan, a skratili smo posao za pola ako je uneseni broj zaista prost.
Pokušajmo napraviti korak dalje.

$$144 = 1 \cdot 144 = 2 \cdot 72 = 3 \cdot 48 = 4 \cdot 36 = 6 \cdot 24 = 8 \cdot 18 = 9 \cdot 16 = 12 \cdot 12.$$

Pogledamo li rastav broja 144 na faktore, uočit ćemo da je uvijek jedan od faktora manji ili jednak od korijena od 144. To je logično jer kad bi oba čimbenika bila veća od korijena od 144, onda bi njihov umnožak bio veći od 144.

Ideja 4. Za zadani n, petlja po kojoj se isprobava djeljivost broja a s brojevima manjim od njega ne treba provjeravati brojeve veće od korijena od a.

Programski kod 2.31: C++ kod za 4. ideju zadatka 2.5

```cpp
#include <iostream>
#include <cmath>

using namespace std;
int main()
{
    long a;
    cin >> a;

    bool djeljiv = false;
    int korijen = sqrt(a);

    for (int i = 2; i <= korijen; ++i)
        if (a % i == 0) {djeljiv = true; break;}
    if (a == 1) djeljiv = true;

    if (!djeljiv) cout << "Prost je." << endl;
    else cout << "Nije prost." << endl;
```

```
19 }
```

Programski kod 2.32: Python kod za 4. ideju zadatka 2.5

```python
from math import sqrt

a = int(input())
djeljiv = False

for i in range(2, int(sqrt(a))+1):
    if a % i == 0:
        djeljiv = True
        break

print("Nije prost." if djeljiv else "Prost je.")
```

Ideja 4 i dalje je jednostavna; za $a = 1\,000\,000$ petlja ide samo do 1000, ali i dalje imamo velik broj bespotrebnih ispitivanja. Ako smo, na primjer, za $i = 2$ vidjeli da a nije djeljiv s 2, naravno da onda neće biti djeljiv niti s 4, 6, 8, ..., dakle, niti s jednim parnim brojem, a mi ispitujemo sve te djeljivosti.

Ideja 5. Iskoristimo činjenicu da ako broj nije djeljiv s 2, onda djeljiv nije niti s bilo kojim višekratnikom od 2.

Programski kod 2.33: C++ kod za 5. ideju zadatka 2.5

```cpp
#include <iostream>
#include <cmath>

using namespace std;
int main()
{
    long a;
    cin >> a;

    bool djeljiv;
    if (a == 1) djeljiv = true;
    else if (a == 2) djeljiv = false;
    else if (a % 2 == 0) djeljiv = true;
    else {
        djeljiv = false;
        int korijen = sqrt(a);

        for (int i = 3; i <= korijen; i += 2)
```

```
19              if (a % i == 0) {djeljiv = true; break;}
20      }
21
22      if (djeljiv) cout << "Nije prost." << endl;
23      else cout << "Prost je." << endl;
24  }
```

Liniju 13 koja provjerava je li broj djeljiv s 2 možemo zamijeniti linijom

```
13      else if (!(a & 1)) djeljiv = true; //ili: else if (1-a&1) djeljiv=true;
```

Bitwise operator & znatno je brži od računanja ostatka pri dijeljenju.

Programski kod 2.34: Python kod za 5. ideju zadatka 2.5

```
1  from math import sqrt
2
3  a = int(input())
4  djeljiv = False
5
6  if a == 1:
7      djeljiv = True
8  elif a != 2 and a % 2 == 0:
9      djeljiv = True
10 else:
11     for i in range(3, int(sqrt(a))+1, 2):
12         if a % i == 0:
13             djeljiv = True
14             break
15
16 print("Nije prost." if djeljiv else "Prost je.")
```

Ideja 6. Kao što smo napravili s brojem 2 u ideji 5, tako možemo sada nastaviti i s ostalim prostim brojevima.

Sada dolazimo do najbrže ideje za provjeru je li zadani broj prost. Treba ispitati je li on djeljiv s prostim brojevima 2, 3, 5, 7, 11, 13, ... Ostale brojeve ne treba provjeravati jer su oni umnošci prostih brojeva. Ako zadani broj nije djeljiv s 11, neće biti djeljiv ni s jednim njegovim višekratnikom (strogo govoreći, to je vidljivo na sljedeći način: ako je broj a djeljiv s $k \cdot x$, tj. $a = n \cdot kx$, onda je djeljiv i s x, tj. $a = nk \cdot x$, a „kontrapozicija" te izjave je ako broj nije djeljiv s x, nije ni s kx, gdje su a, k, n i x prirodni brojevi). Stoga, u petlji provjeravamo samo proste brojeve.

Program koji provjerava je li zadani broj djeljiv s bilo kojim prostim brojem manjim od 100:

Programski kod 2.35: C++ kod za 6. ideju zadatka 2.5

```cpp
#include <iostream>

using namespace std;
int main()
{
    int p[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53,
       59, 61, 67, 71, 73, 79, 83, 89, 97};

    long a;
    cin >> a;

    bool djeljiv = false;
    for (int i = 0; i <= 24; ++i)
        if (a != p[i] && a % p[i] == 0) {djeljiv = true; break;}

    if (djeljiv || a == 1) cout << "Nije prost." << endl;
    else cout << "Prost je." << endl;
}
```

Programski kod 2.36: Python kod za 6. ideju zadatka 2.5

```python
a = int(input())
djeljiv = False
p = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61,
   67, 71, 73, 79, 83, 89, 97]

for n in p:
    if a % n == 0 and a != n:
        djeljiv = True
        break

print("Nije prost." if djeljiv else "Prost je.")
```

Naravno, gornji program može ispravno ispitati samo brojeve manje od $100 \cdot 100 = 10000$. Za veće brojeve treba povećati polje $p[i]$.

ZADATAK 2.6.

Napišite program koji će za uneseni broj n i unesenih n dužina na brojevnom pravcu ispisati koliko od tih dužina nema dodir niti s jednom od preostalih dužina.
Dužine se unose kao n parova brojeva (p_i, k_i) gdje su p_i i k_i početak i kraj i-te dužine, $p_i < k_i$, $i = 1, 2, \ldots, n$ na brojevnom pravcu. n je najviše 100.

Primjer 1. Za ulaz

```
4
1 2
1 3
2 5
6 8
```

izlaz je 1.

Primjer 2. Za ulaz

```
7
2 7
3 6
1 2
3 4
8 11
4 5
13 14
```

izlaz je 2.
Komentar: Dvije dužine, dužina $(8, 11)$ i dužina $(13, 14)$, su samostalne, tj. ne dodiruju niti jednu od preostalih.

Rješenje. Prvo treba unijeti broj n i nakon toga n početaka i krajeva dužina.

Za svaku pojedinu dužinu (druga petlja po i u donjem programu), pretpostavimo da ona nema presjeka s ostalima i postavimo varijablu sjece na nulu.
Nakon toga, za svaku od ostalih dužina (unutar petlje po i imamo petlju po j) provjerimo siječe li ju promatrana, aktualna dužina.

Globalno, dužina od a do b neće sjeći dužinu od c do d ako je promatrana dužina od a do b ili desno od dužine od c do d, tj. ako je $a > d$, ili ako je dužina od a do b lijevo od

dužine koja ide od c do d, tj. ako je $b < c$. Dakle, uvjet za disjunktnost (ne sijeku se) bit će `if (a>d or b<c)`.

No, mi moramo provjeriti suprotno, tj. u ispitivanju svih ostalih dužina, čim promatrana dužina sječe neku od preostalih, treba postaviti `sjece` na 1 kako promatranu dužinu ne bismo brojali u samostalne. Suprotni uvjet od ,,ne siječe" je ,,siječe", a suprotno od `a>d or b<c` je `a<=d and b>=c`.

Programski kod 2.37: C++ kod za zadatak 2.6

```cpp
#include <iostream>

using namespace std;
int main()
{
    int n, p[101], k[101];

    cout << "Unesi broj duzina. " << endl;
    cout << "n=";
    cin >> n;
    for (int i = 1; i <= n; ++i) {
        cout << "Unesi krajeve duzine " << i << ": ";
        cin >> p[i] >> k[i];
    }

    int brojac = 0;
    for (int i = 1; i <= n; ++i) {
        bool sjece = false;
        for (int j = 1; j <= n; ++j)
        if (i != j && p[j] <= k[i] && k[j] >= p[i])
        sjece = true;
        if (!sjece) ++brojac;
    }

    cout << "Nezavisnih duzina ima " << brojac << "." << endl;
}
```

Programski kod 2.38: Python kod za zadatak 2.6

```python
n=int(input("n="))
p = []
k = []
for i in range(n):
```

```
5      print(f"Unesite pocetak i kraj {i+1}. duzine: ")
6      p.append(int(input()))
7      k.append(int(input()))
8
9  brojac = 0
10 sjece = False
11 for i in range(n):
12     sjece = False
13     for j in range(n):
14         if i != j and p[j] <= k[i] and k[j] >= p[i]:
15             sjece = True
16     if not sjece:
17             brojac += 1
18
19 print(f"Broj nezavisnih duzina je {brojac}.")
```

Očito za svaku od n dužina provjeravamo njen odnos sa svakom od preostalih $n-1$ dužina. Drugim riječima, u petlji po i nalazi se još jedna petlja po j pa je složenost ovog algoritma $O(n^2)$.

Postoji li brže rješenje? Razmislimo bi li nam pomoglo da su dužine na početku sortirane i kolika bi onda bila složenost algoritma. Bismo li morali uzeti u obzir svih n^2 provjera?

Kada bi dužine bile poredane s obzirom na početak dužine ili kada bi npr. ulaz bio

6
2 7
3 6
1 2
3 4
8 11
4 5
13 14

prvo bi sortirali i dobili

6
1 2
2 7
3 6

3 4

4 5

8 11

13 14

onda ne bismo morali napraviti svih n^2 ispitivanja. Sve bi pametnom idejom mogli riješiti u jednoj petlji po n (nakon što elemente poredamo po veličini nekim, od $O(n^2)$, bržim programom za sortiranje).

Promotrimo dužinu $(8, 11)$. Ona započinje u broju 8, tj. desno od kraja svih prije nje navedenih dužina, pa je sa svim tim dužinama navedenima prije nje disjunktna (nema s njima dodirnih točaka). Dalje, kraj te dužine, broj 11, nalazi se lijevo od početka sljedeće dužine, pa slijedi da ni te dvije dužine nemaju niti jedne točke presjeka. Zaključujmo da je dužina $(8, 11)$ disjunktna s ostalim dužinama.

Tako nam se nameće sljedeći algoritam: za svaku dužinu treba provjeriti počinje li ta dužina poslije završetka do tog trenutka najdesnijeg kraja prethodnih dužina. Ako da, ta je dužina desno od svih do sada promatranih dužina, tj. s njima nema dodirnih točaka.

I drugo, ako je desni kraj te dužine broj koji je manji od početka sljedeće dužine, onda je to broj koji je manji i od početka svih sljedećih dužina jer su poredane po veličini s obzirom na početnu točku pa je ta dužina disjunktna s ostalima. Uz pretpostavku da su dužine već poredane po prvoj koordinati program i izgledaju kao dolje.

Programski kod 2.39: Optimalniji C++ kod za zadatak 2.6

```
1  #include <iostream>
2
3  using namespace std;
4  int main( )
5  {
6      int n, p[101], k[101];
7
8      cout << "Unesi broj duzina. ";
9      cout << "n=";
10     cin >> n;
11
12     int brojac = 0;
13     for (int i=1; i<=n; ++i) {
14         cout << "Unesi krajeve duzine " << i << ": ";
15         cin >> p[i] >> k[i];
16     }
17
```

```
18      if (k[1] < p[2]) brojac=1;
19      int najdesniji = k[1];
20
21      for (int i = 2; i <= n - 1; ++i)
22          {
23          if (p[i] > najdesniji && k[i] < p[i + 1])
24              ++brojac;
25          if (k[i] > najdesniji)
26              najdesniji = k[i];
27      }
28      if (p[n] > k[n - 1]) ++brojac;
29
30      cout << "Nezavisnih duzina ima " << brojac << "." << endl;
31  }
```

Programski kod 2.40: Optimalniji Python kod za zadatak 2.6

```
1  n=int(input("n="))
2  p = []
3  k = []
4  for i in range(n):
5      print(f"Unesite koordinate {i+1}. tocke: ")
6      p.append(int(input()))
7      k.append(int(input()))
8
9  brojac = 0
10 sjece = False
11 najdesniji = k[0]
12 for i in range(1, n):
13     if p[i] > najdesniji and k[i] < p [i+1]:
14         brojac += 1
15     if k[i] > najdesniji:
16         najdesniji = k[i]
17
18 if p[n] > k[n-1]:
19     brojac += 1
20
21 print(f"Broj nezavisnih duzina je {brojac}.")
```

Zaključujemo da nam je (npr. u slučaju da zadanih dužina ima jako puno) brže prvo sortirati navedene dužine (sortirati ih po početku dužine), a onda pronaći one koje nemaju dodir ni s jednom od ostalih. Ukupno, ako elemente poslažemo po veličini nekim od algoritama složenosti $O(n \cdot \log n)$, uz naknadno brojanje samostalnih dužina koje je

složenosti $O(n)$ ukupna složenost algoritma je $O(n \cdot \log n)$.

ZADATAK 2.7.

Napišite program koji će za unesene pozicije n šetača koji se nalaze na nekim točkama pravca ispisati mjesto gdje se trebaju sresti ako žele da im ukupan broj koraka (svaki je korak dužine jedan) do mjesta susreta bude što manji. (Zbog jednostavnosti, svi su šetači u točkama čije su vrijednosti prirodni brojevi.)

Primjer 1. Ako su šetači u točkama 12, 14 i 7, trebali bi se naći, na primjer, u točki 12 jer će prvi šetač do te točke napraviti nula koraka, drugi dva koraka, a treći pet koraka, što je ukupno sedam koraka i to je minimum potrebnih koraka.

Primjer 2. Ako su šetači u točkama 2, 8, 16 i 17, mogu se susresti u točki 9 jer će prvi šetač do te točke napraviti 7 koraka, drugi 1 korak, treći 7 koraka i četvrti 8 koraka, što je ukupno prijeđenih 23 koraka i to je minimum potrebnih koraka za susret.

Rješenje. Prva je ideja izračunati za svaku točku na pravcu koliko koraka treba ukupno šetačima da dođu do nje i onda ispisati točku s ukupno najmanjim brojem koraka. Naravno, to je nemoguće jer je svih točaka na pravcu beskonačno.

Druga je ideja uočiti da šetače ne treba poslati na susret u neku daleku točku, nego će mjesto susreta biti negdje između pozicije šetača koji je lijevo od svih ostalih i onoga koji je desno od svih ostalih (jer ako je točka susreta desno od najdesnijeg šetača, onda je ukupan broj koraka do te desne točke veći nego ukupan broj koraka do lokacije najdesnijeg šetača jer je doći do te točke koja je desno od najdesnijeg šetača jednako kao i doći do najdesnijeg šetača i onda bespotrebno pješačiti desno). U prvoj petlji, koja se kreće po točkama koje se nalaze između točke u kojoj je krajnji lijevi šetač i točke u kojoj je krajnji desni šetač, nalazi se druga petlja koja uzima u obzir svakog šetača te se onda računa zbroj udaljenosti svih šetača do promatrane točke. Složenost ovog algoritma je $O(m \cdot n)$, gdje je m razlika pozicije krajnjeg desnog i krajnjeg lijevog šetača, a n je broj šetača.

Treća je ideja sljedeća:
ako se radi o samo jednom šetaču, on je već na mjestu gdje će se sresti sam sa sobom i rješenje je nula koraka.
Ako se radi o dvama šetačima, od kojih se jedan nalazi u točki a, a drugi u točki b, gdje je npr. $a < b$, onda se oni mogu susresti u bilo kojoj točki između a i b i prijeći će $b - a$ koraka ukupno. Od točke a do točke x put je dug $x - a$ koraka, a od točke x do b nalazi se $b - x$ koraka pa je to ukupno $x - a + b - x = b - a$ koraka. Ovo je zadnje razmišljanje ključno.
Ako sada dalje imamo tri šetača od kojih su dvojica opet, bez smanjenja općenitosti, u točkama a i b, a treći stoji između njih, onda je za šetače a i b svejedno gdje će se sresti

s obzirom na točke između a i b, pa treba odabrati točku koja je najbliža šetaču koji se nalazi na poziciji c.
Sada je jasno da za pronalaženje konačnog rješenja zadane točke treba, prvo poredati po veličini.
Ako je zadan neparni broj točaka (kao npr. $n = 5$), onda je vanjskim šetačima, koji su prvi i zadnji na sortiranoj listi (prvi i peti šetač) svejedno gdje će se između njihovih lokacija susresti, pa gledamo na sljedeća dva vanjska šetača (npr. drugog i četvrtog) kojima je također svejedno gdje će se oni susresti itd. pa će konačno mjesto susreta biti lokacija srednjeg šetača. (Takvo se mjesto u sortiranoj listi brojeva zove medijan.)
U slučaju parnog broja šetača, rješenje za mjesto susreta bit će bilo koji broj koji je između unutrašnja dva šetača.

Ukupno, treba naći lokaciju bilo kojeg srednjeg šetača i onda izbrojati koliko koraka ukupno do njega trebaju prijeći svi ostali šetači. Sortirajmo lokacije „*bubble*" sortom (n puta prođimo cijelu listu lokacija i u svakom prolasku usporedimo svaka dva susjeda te im zamijenimo pozicije ako treba).

Programski kod 2.41: C++ kod za zadatak 2.7

```
#include <iostream>
#include <cmath>
#include <vector>
#include <algorithm>

using namespace std;
int main()
{
    int n;
    cout << "Unesi broj setaca: ";
    cin >> n;

    vector<int> L(n + 1);
    for (int i = 1; i <= n; ++i) {
        cout << "Unesi lokaciju setaca " << i << ": ";
        cin >> L[i];
    }

    for (int i = 2; i <= n; ++i)
        for (int j = 1; j <= i - 1; ++j)
            if (L[j] > L[j + 1])
                swap(L[j], L[j + 1]);

    int zbroj = 0;
```

```
25      for (int i = 1; i <= n; ++i)
26          zbroj += abs(L[i] - L[n / 2 + 1]);
27          // ili: zbroj += abs(L[i] - L[(n >> 1) + 1]);
28
29      cout << "Minimalni ukupan broj koraka do susreta je: " << zbroj << "."
            << endl;
30  }
```

Programski kod 2.42: Python kod za zadatak 2.7

```
1  n = int(input())
2  L = [int(input()) for i in range(n)]
3
4  for i in range(n):
5      for j in range(n-1):
6          if L[j] > L[j+1]:
7              L[j], L[j+1] = L[j+1], L[j]
8
9  zbroj = 0
10 for i in range(n):
11     zbroj+= abs(L[i] - L[int(n/2)])
12
13 print(f"Minimalni ukupan broj koraka do susreta je: {zbroj}.")
```

Sada imamo algoritam čija je složenost $O(n^2)$. Pitamo se možemo li dalje pojednostavniti program ili ga ubrzati?

Očito će u sklopu „*bubble*" sorta velik broj ispitivanja biti nepotreban pa će naša optimizacija za početak ubrzati redove u kojima se izvršava sortiranje elemenata. Ako u i-tom krugu nismo zamijenili mjesta susjednim elementima, tj. ako su svi već složeni po veličini, ne treba ići u $i+1$-vi krug, nego prekinuti postupak algoritmom.

Algoritam 2.1: Algoritam za brže soritranje

```
for (int i = 1; i <= n; ++i) {
    bool zamijenjeno = false;
    for (int j = 1; j <= n - 1; ++j)
            if (L[j] > L[j + 1]) {
            int temp = L[j]; L[j] = L[j + 1]; L[j + 1] = temp;
            // ili: swap (L[j], L[j + 1])
            zamijenjeno = true;
        }
        if (!zamijenjeno) break;
}
```

Sljedeća je ideja ubrzavanja algoritma da se umjesto *bubble* sorta koristi C++ naredba

```
sort(&L[1], &L[n + 1]);
```

ili odabrati neki brži algoritam sortiranja (npr. *merge* sort) koji je složenosti $O(n \cdot \log n)$ u najgorem slučaju.

Ovdje ćemo stati, premda nama za računanje treba samo medijan, a za njegovo nalaženje postoje i sofisticiraniji i brži algoritmi.

ZADATAK 2.8.

Napišite program koji će za zadani prirodni broj n ispisati koji je najveći neparni djelitelj od n, a zatim će ispisati i zbroj svih najvećih neparnih djelitelja svih brojeva od 1 do n.

Primjer 1. Za $n = 4$ najveći neparni djelitelj od 4 je 1, a zbroj svih najvećih neparnih djelitelja svih brojeva od 1 do 4 je

$$1 + 1 + 3 + 1 = 6.$$

Primjer 2. Za $n = 19$ najveći neparni djelitelj od 19 je 19, a zbroj svih neparnih djelitelja brojeva od 1 do 19 je

$$1 + 1 + 3 + 1 + 5 + 3 + 7 + 1 + 9 + 5 + 11 + 3 + 13 + 7 + 15 + 1 + 17 + 9 + 19 = 131.$$

Rješenje.

Ideja 1. Za zadani broj n, jednostavno u petlji od 1 do n, možemo provjeriti koji je najveći neparni djelitelj od n.

Programski kod 2.42: C++ kod za 1. ideju zadatka 2.8

```
#include <iostream>
using namespace std;

int main( )
{
    int n;
    cout << "Unesi prirodni broj: ";
    cin >> n;

    int najveci = 1;
    for (int i = 1; i <= n; ++i)
        if ((i & 1) && n % i == 0)
            najveci = i;

    cout << "Najveci neparni djelitelj zadanog broja je: " << najveci <<
        endl;
}
```

Programski kod 2.43: Python kod za 1. ideju zadatka 2.8

```
n = int(input("Unesi prirodni broj: "))
najveci = 1
for i in range(1, n+1):
    if n % i == 0 and i % 2 == 1:
        najveci = i
print(f"Najveci neparni djelitelj zadanog broja je: {najveci}")
```

Na analogan način u dvjema petljama možemo zbrojiti sve te brojeve. No, za veće n ovaj algoritam bit će vrlo spor.

Ideja 2. Kako tražimo samo neparne djelitelje, u petlji ne treba provjeravati sve prirodne brojeve, nego samo neparne. Također, ako je zadani broj neparan, rješenje je sami taj broj jer je najveći neparni djelitelj neparnog broja taj broj sam.

Programski kod 2.44: C++ kod za 2. ideju zadatka 2.8

```
#include <iostream>
using namespace std;

int main( )
{
    int n;
    cout << "Unesi prirodni broj: ";
    cin >> n;

    int najveci = 1;
    if (n & 1) najveci = n;
    else for (int i = 1; i <= n; i += 2)
           if (n % i == 0) najveci = i;

    cout << "Najveci neparni djelitelj zadanog broja je: " << najveci <<
       endl;
}
```

Programski kod 2.45: Python kod za 2. ideju zadatka 2.8

```
n = int(input("Unesi prirodni broj: "))
najveci = 1

if n % 2 == 1:
       najveci = n
else:
```

```
7     for i in range(1, n+1, 2):
8         if n % i == 0:
9             najveci = i
10
11 print(f"Najveci neparni djelitelj zadanog broja je: {najveci}")
```

Ideja 3. Rastavimo li broj na proste faktore, uočavamo da je svaki prirodni broj neka potencija broja 2 pomnožena s neparnim brojem jer je samo broj 2 paran od svih prostih brojeva. Najveći neparni djelitelj broja n dobit ćemo tako da broj n dijelimo brojem 2 sve dok ne dobijemo neparni broj i taj će dobiveni neparni broj biti najveći neparni djelitelj koji tražimo. Za npr. $n = 100$, najveći neparni djelitelj bit će

$$100 : 2 : 2 = 25.$$

Ako je, dakle, broj neparan, ne treba raditi ništa, a ako je paran, ,,prepolavljamo" ga dok ne dobijemo neparan broj.

Programski kod 2.46: Početni C++ kod za 3. ideju zadatka 2.8

```
1  #include <iostream>
2  using namespace std;
3
4  int main( )
5  {
6      int n;
7      cout << "Unesi prirodni broj: ";
8      cin >> n;
9
10     while (n % 2 == 0) n /= 2;
11     // ili: while (!(n & 1)) n >>= 1;
12
13     cout << "Najveci neparni djelitelj zadanog broja je: " << n << endl;
14 }
```

Programski kod 2.47: Početni Python kod za 3. ideju zadatka 2.8

```
1 n = int(input("Unesi prirodni broj: "))
2 while n % 2 == 0:
3     n = n / 2
4 print(f"Najveci neparni djelitelj zadanog broja je: {n}")
```

Sada smo s prvim dijelom gotovi.

Kako zbrojiti sve najveće neparne djelitelje svih brojeva do n, zaključno s n? Ovdje će nam pomoći formula

$$1+3+5+7+9+\ldots+(2n-1)=n^2.$$

Dakle, za uneseni $n=19$, kao u primjeru gore, zbroj svih najvećih neparnih djelitelja neparnih brojeva u stvari je zbroj svih neparnih brojeva do 19, a to je $\left(\frac{19+1}{2}\right)^2$ (a ako je n paran, zbroj svih takvih neparnih djelitelja je $\left(\frac{n}{2}\right)^2$.

No, što je s najvećim neparnim djeliteljima parnih brojeva, koliki je njihov zbroj? Označimo li s d najveći neparni djelitelj broja, imamo npr.

$$\begin{aligned}
&d(1)+d(2)+d(3)+d(4)+d(5)+d(6)+d(7)+d(8)+d(9)+d(10)=\\
=&1+d(2)+3+d(4)+5+d(6)+7+d(8)+9+d(10)=\\
=&1+3+5+7+9+d(2)+d(4)+d(6)+d(8)+d(10)=\\
&\textit{(sjetimo se gornjeg argumenta o dijeljenju s 2)}\\
=&1+3+5+7+9+d(1)+d(2)+d(3)+d(4)+d(5)=\\
=&1+3+5+7+9+1+d(2)+3+d(4)+5=\\
=&1+3+5+7+9+1+3+5+d(2)+d(4)=\\
=&1+3+5+7+9+1+3+5+d(1)+d(2)=\\
=&1+3+5+7+9+1+3+5+1+1=\\
=&5^2+3^2+1^2+1=\\
=&25+9+1+1=\\
=&36.
\end{aligned}$$

Ukupno, za zadani n, sve neparne brojeve prije njega zbrojimo i koristimo gore navedenu formulu, a parne prepolovimo pa ćemo opet dobiti sve uzastopne brojeve od 1 do polovine od n.

Programski kod 2.48: Potpuni C++ kod za 3. ideju zadatka 2.8

```
1  #include <iostream>
2  using namespace std;
3
4  int main( )
5  {
6      int n;
7      cout << "Unesi prirodni broj: ";
8      cin >> n;
9
10     int m = n;
```

```
11      while (!(m & 1))
12          m >>= 1;
13
14      cout << "Najveci neparni djelitelj zadanog broja je: " << m << endl;
15
16      int zbroj = 0;
17      do {
18          if (n & 1) {
19              zbroj += ((n + 1) >> 1) * ((n + 1) >> 1);
20              n = (n - 1) >> 1; // moze i: n >>= 1;
21          }
22          else {
23              zbroj += (n >> 1) * (n >> 1);
24              n >>= 1;
25          }
26     } while (n >= 1);
27
28      cout << " Zbroj=" << zbroj << endl;
29  }
```

Napomena. Uočite da bi umjesto linija 18–25 sasvim korektno radio i sljedeći programski odsječak:

```
18          zbroj += ((n + 1) >> 1) * ((n + 1) >> 1);
19          n >>= 1;
```

Programski kod 2.49: Potpuni Python kod za 3. ideju zadatka 2.8

```
1  n = int(input("Unesi prirodni broj: "))
2  m = n
3
4  while m % 2 == 0:
5      m = m // 2
6
7  print("Najveci neparni djelitelj zadanog broja je:", m)
8
9  zbroj = 0
10 while n >= 1:
11     if n % 2 == 1:
12         zbroj = zbroj + ((n + 1) // 2) * ((n + 1) // 2)
13         n = (n - 1) // 2
14     else:
15         zbroj = zbroj + (n // 2) * (n // 2)
16         n = n // 2
```

```
17
18 print(f"{zbroj=}")
```

ZADATAK 2.9.

(Problem ruksaka ili "knapsack" problem)

Napišite program koji će za uneseni broj n, koji predstavlja broj predmeta, i za unesenih n težina i vrijednosti svakog od n predmeta koje želimo ponijeti na piknik te za uneseni broj K, koji predstavlja kapacitet ruksaka, ispisati najveću moguću vrijednost u ruksaku.

Primjer 1. Za

$$\begin{aligned} n &= 5, \\ K &= 20, \\ t_1 &= 5, v_1 = 16, \\ t_2 &= 4, v_2 = 12, \\ t_3 &= 8, v_3 = 21, \\ t_4 &= 7, v_4 = 9, \\ t_5 &= 4, v_5 = 11, \end{aligned}$$

u ruksak ćemo staviti prva tri predmeta ako želimo imati maksimalnu vrijednost u ruksaku jer je njihova težina 17, što ne prelazi kapacitet ruksaka, a njihova je vrijednost $16 + 12 + 21 = 49$, što ne možemo povećati odabirom nekih drugih predmeta koji također mogu biti ubačeni u ruksak.

Primjer 2. Za

$$\begin{aligned} n &= 6, \\ K &= 22, \\ t_1 &= 5, v_1 = 16, \\ t_2 &= 4, v_2 = 12, \\ t_3 &= 8, v_3 = 21, \\ t_4 &= 7, v_4 = 9, \\ t_5 &= 4, v_5 = 11, \\ t_6 &= 9, v_6 = 18, \end{aligned}$$

u ruksak ćemo staviti prvi, drugi, treći i peti predmet jer je njihova ukupna težina 21, a vrijednost je 60.

Rješenje. Problem ruksaka jedan je od najpoznatijih problema optimizacije (grana matematike u kojoj tražimo najmanju ili najveću vrijednost neke funkcije) (u problemu ruksaka tražimo najveću vrijednost koju možemo staviti u ruksak).

Taj je problem i jedan od najpoznatijih NP teških problema (za zadani problem ruksaka i pitanje postoji li takav odabir predmeta koje ćemo staviti u ruksak, a čija je vrijednost bar V, gdje je V neki unaprijed zadani broj, za sada ne možemo naći polinomijalni algoritmom, ali ako nam netko ponudi rješenje, onda možemo u polinomijalnom vremenu provjeriti je li to prihvatljivo rješenje).

Za rješenje problema ruksaka treba provjeriti sve mogućnosti ubacivanja predmeta.
Možda treba, za optimalno rješenje, ubaciti samo prvi predmet.
Možda samo drugi.
Možda samo treći.
Možda prvi i drugi.
I tako dalje.

Jedan predmet od njih n možemo izabrati na n načina, dva na $\binom{n}{2}$, tri na $\binom{n}{3}$ načina i tako dalje. Za isprobavanje svih mogućnosti, moramo provjeriti zbroj svih tih binomnih koeficijenata, a po binomnom teoremu znamo da je zbroj svih tih brojeva 2^n.

Želimo li naći neki pametniji algoritam, možemo odmah uočiti da ako neki predmeti teže više od K, njih sigurno ne treba uzeti u obzir.

Dalje, ako smo počeli slagati predmete u ruksak, a njihova je ukupna težina u, onda ne treba gledati za dalje ubacivanje predmeta koji su teži od $K - u$ (jer oni više ne stanu unutra).

Ukupno dobivamo sljedeći rekurzivni algoritam koji ubacuje n-ti predmet u ruksak ili ne ubacuje u odnosu na dosadašnju težinu ubačenu u ruksak:

Programski kod 2.50: C kod za zadatak 2.9

```
1  #include <stdio.h>
2
3  double max(double a, double b) {if (a > b) return a; else return b; }
4
5  double knapSack(double K, double tt[], int vri[], int n)
6  {
7      if (n == 0 || K == 0) return 0;
8      if (tt[n - 1] > K) return knapSack(K, tt, vri, n - 1);
9      else return max(vri[n - 1] + knapSack(K- tt[n - 1], tt, vri, n - 1),
           knapSack(K, tt, vri, n - 1));
10 }
11
12 int main()
13 {
```

```
14      int n; double K;
15      K = 20;
16      n = 5;
17      int vrijednost[] = {16, 12, 21, 9, 11};
18      double tezina[] = {5, 4, 8, 7, 4};
19
20      printf("%d\n", knapSack(K, tezina, vrijednost, n));
21  }
```

Programski kod 2.51: Python kod za zadatak 2.9

```
1   def knapSack(K, wt, val, n):
2
3       if n == 0 or K == 0:
4           return 0
5
6       if (wt[n-1] > K):
7           return knapSack(K, wt, val, n-1)
8
9       else:
10          return max(val[n-1] + knapSack( K-wt[n-1], wt, val, n-1), knapSack(K
                , wt, val, n-1))
11
12  if name_== '__main__':
13      K= 50
14      n = 3
15      vrijednost = [60, 100, 120]
16      tezina = [10, 20, 30]
17
18      print (knapSack(K, tezina, vrijednost, n) )
```

U ovim smo algoritmima po prvi puta koristili izravno upisivanje ulaznih podataka u program. Naravno, i u ovom smo programu mogli, kao i u do sada napisanima naredbama `scanf`, `cin >>`, odnosno `input` podatke, u program upisati i s tipkovnice.

Na sličan način može se realizirati drugi NP-težak problem, a to je problem zbroja podskupa kod kojega je zadan skup S čiji su elementi realni brojevi i u kojemu je zadan realan broj x, a pitanje je postoje li elementi u S koji zbrojeni daju baš x.

Napomena. Ako su težine predmeta decimalni brojevi prethodni problem je NP-težak, a ako su težine prirodni brojevi, prethodni se problem metodom dinamičkog programiranja može riješiti algoritmom složenosti $O(K \cdot n)$. Sljedeći algoritam rješava taj problem (a rješenje koje se nalazi unutar komentara dopušta i veći broj istih predmeta u ruksaku,

npr. na put možemo ponijeti i desetak četkica za zube).

Programski kod 2.52: C kod za zadatak 2.9

```c
int maks(int a, int b) { if (a > b) return a; else return b; }

int main()
{
    int n; int K, i, j;
    K = 20;
    n = 5;
    int vrijednost[] = { 16, 12, 21, 9, 11 };
    int tezina[] = { 5, 4, 8, 7, 4 };

    int ruksak[K + 1];

    /*
    ruksak[0] = 0;
    for (i = 0; i < n; i = i + 1)
        for (j = 0; j < K; j = j + 1)
            if (j + tezina[i] > K) break;
            else ruksak[j + tezina[i]] = maks(ruksak[j + tezina[i]],
                                              ruksak[j] + vrijednost[i]);
    */

    for (i = 0; i <= K; i = i + 1)
        ruksak[0] = 0;

    for (i = 0; i < n; i = i + 1)
        for (int j = K; j >= 0; j = j - 1)
            if (j - tezina[i] < 0) break;
                else ruksak[j] = maks(ruksak[j],
                                      ruksak[j - tezina[i]] + vrijednost[i]);

    printf("%d\n", ruksak[K]);
}
```

ZADATAK 2.10.

Napišite program koji će za zadani prirodni broj n ispisati n-ti član Solomon Golombova (Solomon Golomb, 1932. – 2016., američki matematičar) samoopisujućeg niza. To je niz koji je monotono rastući (svaki sljedeći član niza jednak je prethodnom ili je od njega veći), čiji je prvi element niza broj 1, a niz je definiran tako što n-ti element niza govori koliko se puta broj n kao element niza pojavljuje u tom nizu.

Primjer 1. Za ulaz 1, izlaz je 1.

Primjer 2. Za ulaz 18, izlaz je 7.

Rješenje. Prvo moramo dokučiti kako taj samoopisujući niz uopće izgleda.

$$a(1) = 1.$$

Koliko je $a(2)$?

$a(2)$ ne može biti nula niti bilo koji manji broj jer niz je rastući (točnije: monotono rastući) pa sljedeći element može biti samo neki prirodan broj. Dalje, $a(2)$ ne može biti 3 jer bi to značilo da će se u nizu pojaviti 3 broja 2 na nekim mjestima. No, te tri dvojke ne smiju se više pojaviti jer smo kod $a(2)$ već upisali 3 pa bi negdje niz morao biti padajući, a to ne smije. Iz istog razloga, $a(2)$ ne može biti ni neki broj veći od 3. Slijedi $a(2) = 2$, što nam govori da i $a(3)$ mora biti 2 jer smo s $a(2) = 2$ upravo definirali da se u nizu moraju pojaviti dvije dvojke. I tako dalje.

Niz, dakle, izgleda ovako:

$$1, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6, 6, 7, 7, 7, 7, 8, 8, 8, 8, 9, 9, 9, 9, 9,$$
$$10, 10, 10, 10, 10, 11, 11, 11, 11, 11, 12, 12, 12, 12, 12, 12, \ldots$$

(Prva šestica na 12. mjestu govori nam kako je $a(12) = 6$, što znači da će se broj 12 pojaviti 6 puta i zaista se na kraju navedenog dijela niza pojavljuje šest dvanaestica. Broj 6 iza toga govori nam da dalje mora slijedit šest brojeva 13.)

Iskoristimo, dakle, za ispis članova niza činjenicu da je niz ,,samoopisujući". Očito će se brojevi u nizu tako pojavljivati da niti jedan prirodan broj neće biti preskočen (kada bi, na primjer, situacija bila takva da nema u nizu niti jednog broja 50 i neka je to prvi broj koji se ne pojavljuje u nizu, to bi značilo da je $a(50) = 0$, što bi bio pad s $a(49)$ na $a(50)$, a to nije moguće jer je niz monotono rastući). Drugo, kako je niz rastući, a niz je i samoopisujući, jasno je da broj pojavljivanja broja dva biti manji ili jednak od broja

pojavljivanja broja 3 itd.
Drugim riječima, svaki sljedeći broj ima sve veći broj pojavljivanja.

Ideja koje se ovdje moramo dosjetiti je sljedeća:
broj $a(1)$ ili $a(2)$ nećemo računati, nego, ćemo ih na početku zadati (možemo računalu izravno u liniji pridruživanja dati i $a(3)$ itd, a onda ćemo od sljedećeg mjesta sve računati. Ako smo uspjeli upisati npr. sve četvorke, na redu su petice. Koliko ima brojeva 5 koji slijede? To nije problem naći – ima ih $a(5)$).

Stoga ćemo svaki prirodni broj n ispisati u petlji koja ide od 1 do $a(n)$. Ukupno, kod će izgledati ovako:

Programski kod 2.53: C++ kod za zadatak 2.10

```cpp
#include <iostream>
#include <vector>

using namespace std;

int main()
{
    int n, i;
    cin >> n;
    vector<int> a(n + 1);
    a[1] = 1; a[2] = 2; a[3] = 2;
    int sljedeci = 4, trenutnoupisujem = 3;

    do {
        for(i = sljedeci; i <= sljedeci + a[trenutnoupisujem] - 1; ++i) {
            a[i] = trenutnoupisujem;
            if (i == n) break;
        }
        sljedeci = i;
        trenutnoupisujem += 1;
    } while (i < n);

    cout << a[n] << endl;
}
```

Programski kod 2.54: Python kod za zadatak 2.10

```python
def niz(n):
    if n <= 2: return n
```

```
3
4       golomb = [None, 1, 2]
5
6       for i in range(3, n + 1):
7           golomb.append(1 + golomb[i - golomb[golomb[i - 1]]])
8
9       return golomb[n]
10
11  n = int(input("n="))
12  print(f"{niz(n)=}")
```

Rekurzivno rješenje

$$a(i) = 1 + a(i - a(a(i - 1)))$$

u programskom jeziku Python kratko je i jednostavno, ali oslanja se na više teoretskih matematičkih rezultata o ovom nizu koje čitatelj može naći na internetskim stranicama posvećenima ovom matematičaru.

Petlju `do-while` u gore navedenom C++ programu možemo i zaobići, tj. brojeve možemo kreirati u jednoj `for` petlji od 1 do n, s tim da u petlji onda moramo imati i jedno `if` ispitivanje koje će postaviti sljedeći broj za upis.

3

Složeniji zadaci

Zadatak 3.1.

Napišite program koji će za zadani broj n i za zadanih n prirodnih brojeva naći najveći prirodni broj koji se može dobiti konkatenacijom (spajanjem, ljepljenjem) tih zadanih n brojeva.

Primjer 1. Za zadani $n = 3$ i tri prirodna broja 117, 2, 44 najveći broj koji se može naći spajanjem tih brojeva je 442 117.

Primjer 2. Za zadani $n = 3$ i tri prirodna broja 132, 232, 1338 najveći broj koji se može pronaći spajanjem tih brojeva je 2 321 338 132.

Primjer 3. Za uneseni $n = 4$ i za tri prirodna broja 45, 4, 4554 i 45454 najveći prirodni broj koji možemo dobiti spajanjem tih brojeva je 455 445 454 544 (tj. treba upotrijebiti prvo 4554 pa 45 pa 45454 pa 4).

Rješenje. Globalno gledajući, odmah dolazimo do sljedeće ideje:

za početak ćemo unijeti broj n i nakon toga n prirodnih brojeva $a(1)$, $a(2)$, ..., $a(n)$. Zatim ćemo ih poredati s obzirom na to kako želimo da se pojave u novom broju.
Na kraju ćemo elemente promijenjenog polja a spojiti u veliki broj.

Ako brojevi imaju različite prve znamenke, u sklapanju novog broja treba uvijek prvo uzeti onaj s većom prvom znamenkom. Na primjer, za 38, 65 i 22 biramo prvi broj i naša je dilema hoće li nam prva znamenka novog broja biti 3, 6 ili 2.

Ako su prve znamenke brojeva jednake, prioritet u sklapanju novog broja odredit će druge znamenke (za 81, 82 i 83 očito ćemo prvo upotrijebiti 83) itd.

Najveći problem pojavit će nam se ako su početne znamenke ili grupe početnih zna-

menki jednake, a neki od promatranih brojeva više nemaju dalje znamenki za usporedbu kao kod gornjeg primjera s 45, 4, 4554 i 45454. U ovom primjeru, usporedivi su nam jedino 4554 i 45454 ($4 = 4$, $45 = 45$ pa će presuditi treća znamenka te znamo da u novi broj prvo treba staviti 4554 pa onda 45454).

„*Selection*" sort ili sortiranje izborom uvijek uspoređuje dva broja i zamjenjuje im mjesta ako je potrebno. Razmišljajući na sličan način, traženje pravog poretka brojeva svest će nam se na lijepljenje ne svih odjednom, nego na dva broja ili konkatenaciju dvaju brojeva jedan uz drugi. Kako to pametno izvesti? Ako su nam zadani 4545 i 45454, kako ćemo odlučiti koji od tih dvaju brojeva prvi staviti u novi broj? Metoda je jednostavna: spojimo ih i pogledajmo dobijemo li veći broj ako stavimo prvi pa drugi ili dobijemo veći broj ako postavimo drugi pa prvi.

Za prirodne brojeva a i b njihov „spoj" najlakše ćemo dobiti tako da broju a dodamo toliko nula koliko ima broj b znamenki i onda da takvom proširenom broju a samo zbrojimo b.

Na primjer: $a = 34$, $b = 34567$.

$$34 \cdot 100\,000 + 34\,567 = 3\,400\,000 + 34\,567 = 3\,434\,567.$$

Pronalaženje broja znamenaka zadanog broja analizirali smo u zadatku 1.3. Broj a treba, dakle, pomnožiti s 10^x, gdje je x broj znamenaka broja b, a zatim isto učiniti za obrnuti redoslijed.

$$\begin{aligned} \text{novi broj } 1 &= a * \text{pow}(10, (\text{int}) \log 10(b) + 1) + b; \\ \text{novi broj } 2 &= b * \text{pow}(10, (\text{int}) \log 10(a) + 1) + a; \end{aligned}$$

Ukupno:

Programski kod 3.1: C++ kod za zadatak 3.1

```
1  #include <iostream>
2  #include <cmath>
3  #include <vector>
4
5  using namespace std;
6  int main( )
7  {
8      int n;
9      cout << "Unesi broj brojeva: ";
10     cin >> n;
11
12     vector<long long> a(n + 1);
```

```
13      for (int i = 1; i <= n; ++i) {
14          cout << "Unesi" << i << ". broj: ";
15          cin >> a[i];
16      }
17
18      for (int i = 1; i <= n - 1; ++i)
19      {
20          int mjesto = i;
21          for (int j = i + 1; j <= n; ++j)
22              if (a[j] * pow(10, (int)log10(a[mjesto]) + 1) + a[mjesto] >=
23                      a[mjesto] * pow(10, (int)log10(a[j]) + 1) + a[j])
24                  mjesto=j;
25
26          if (mjesto != i) {
27              long long zamjena = a[i]; a[i] = a[mjesto]; a[mjesto] = zamjena;
28          }
29      }
30
31      for (int i = 1; i <= n; ++i)
32          cout << a[i];
33      cout << endl;
34  }
```

Programski kod 3.2: Python kod za zadatak 3.1

```
1   from functools import cmp_to_key
2
3   def usporedi(a, b):
4       ab = a+b
5       ba = b+a
6       return (int(ba) > int(ab)) - (int(ba) < int(ab))
7
8   def broj(n, numbers):
9       if n == 0:
10          return "0"
11
12      sortirani = sorted(numbers, key=cmp_to_key(usporedi), reverse=False)
13
14      najveci = ' '.join(map(str, sortirani))
15
16      return najveci
17
18  n = int(input("Unesi broj brojeva"))
19
20  print("Unesi brojeve")
```

```
21 brojevi = [input() for i in range(n)]
22
23 najveci = broj(n, brojevi)
24 print(f"Najveci broj je: {najveci}")
```

Navedeni program u programskom jeziku Python koristi gotov alat za uspoređivanje po parovima i nakon toga za kreiranje cijelog niza brojeva, odnosno za kreiranje konačnog broja. Iz biblioteke `functools` uzimamo alat `cmp_to_key` koji koristimo u funkciji `broj(n, numbers)` u kojoj kreiramo konačni broj, znamenku po znamenku odvojenu razmakom.

Slično možemo i u C++-u funkciji `sort` poslati kao dodatni parametar funkciju za uspoređivanje dvaju elemenata.

ZADATAK 3.2.

Napišite program koji će, za uneseni broj skijaša i za unesena mjesta koja su zauzeli odmah nakon svog nastupa, ispisati ukupnu ljestvicu, tj. konačni poredak nakon nastupa zadnjeg skijaša.
I obrnuto, napišite i program koji će, za zadani konačni poredak skijaša, ispisati na kojemu je mjestu pojedini skijaš bio odmah nakon svog nastupa (a prije nego što su nastupili skijaši koji su skijali poslije njega).

Primjer 1. Za 6 skijaša čija su mjesta na ljestvici odmah nakon nastupa bila 1, 2, 1, 3, 4, 5 program treba ispisati 3, 1, 4, 5, 6, 2.
Objašnjenje: Prvo skija skijaš broj 1. On je, naravno, prvi nakon svog nastupa jer je i jedini za sada nastupio. Nakon toga skija skijaš broj 2. On je drugi, pa je nakon prvih dvaju skijaša poredak 1, 2. Nakon toga skija treći skijaš i on je prvi pa je ukupni poredak 3, 1, 2. Nakon toga četvrti skijaš je nakon svog nastupa treći pa je poredak 3, 1, 4, 2. Nakon toga skija 5. skijaš i on je nakon svog nastupa četvrti te je ukupni poredak 3, 1, 4, 5, 2. Nakon zadnjeg, šestog skijaša poredak je 3, 1, 4, 5, 6, 2.
Obrnuti problem: Za konačni poredak 6, 4, 5, 3, 1, 2 skijaši su nakon svojih nastupa zauzeli ukupno mjesta 1, 2, 1, 1, 2, 1.

Rješenje. Razmotrimo prvo slučaj s dvama skijašima, tj. slučaj u kojem je $n = 2$.
Prvi skijaš sigurno je nakon svog nastupa prvi, a drugi može biti lošiji od njega, tj. za ulaz 1, 2 treba i izlaz biti 1, 2, te drugi može biti i bolji od prvoga te će za ulaz 1, 1 izlaz biti 2, 1. Za tri skijaša postoji 6 različitih mogućnosti ulaza:

- 1, 1, 1 (pripadno konačno rješenje je 3, 2, 1),
- 1, 1, 2 (pripadno konačno rješenje je 2, 3, 1),
- 1, 1, 3 (pripadno konačno rješenje je 2, 1, 3),
- 1, 2, 1 (pripadno konačno rješenje je 3, 1, 2),
- 1, 2, 2 (pripadno konačno rješenje je 1, 3, 2),
- 1, 2, 3 (pripadno konačno rješenje je 1, 2, 3).

Ukupno, od n skijaša, prvi skijaš može biti samo prvi. Drugi može biti ili 1. ili 2. Treći može biti 1. ili 2. ili 3. Po teoremu o uzastopnom prebrojavanju imat ćemo ukupno $1 \cdot 2 \cdot 3 \cdot \ldots \cdot n = n!$ različitih mogućnosti koje onda daju $n!$ različitih permutacija brojeva od 1 do n.

Za unesene pozicije skijaša odmah nakon nastupa, konačni poredak tražimo na sljedeći način: ako nakon nekog i-tog skijaša imamo ukupni poredak a_1, a_2, ..., a_i, i sada skija $i+1$-vi skijaš i postigne x-to prolazno vrijeme, on automatski biva x-ti u poretku, a svi oni koji su lošiji od njega pomiču se za jedno mjesto prema kraju liste. Na primjer, ako su nakon deset skijaša skijaši na pozicijama 6, 4, 2, 3, 8, 5, 1, 7, 10, 9, a onda 11. skijaš nakon svog nastupa završi na ukupno 6. mjestu, onda on postaje, naravno, šesti, a dotadašnji šesti postaje sedmi, sedmi postaje osmi itd. Treba, dakle, u tom primjeru sve skijaše koji su imali mjesto od 6. do 10. prebaciti za jedno mjesto prema kraju liste, tj. dodati njihovoj poziciji broj 1. To, naravno, radimo tako da krenemo od 10. skijaša koji postaje 11., 9. postaje 10. itd. sve do 6. koji postaje sedmi i time se otvara 6. mjesto na koje postavljamo broj 11.

U polje $a[i]$ unijet ćemo pozicije skijaša nakon njihovih nastupa, a u polju $b[i]$ računat ćemo konačne poretke nakon svakog od nastupa. Uočimo da ako je prvi skijaš prvi, drugi skijaš drugi itd., ne treba ništa raditi i izlazni vektor b izgleda kao i ulazni a, ali ako je $a[i] < i$, tj. ako je skijaš koji skija po redu i-ti bolji od i-tog mjesta, onda će se on ,,ugurati" između nekih već u konačnu listu postavljenih skijaša i povećat će indekse lošijih od njega za jedan, a on će doći na upražnjeno mjesto, tj. bit će $b[a[i]] = i$.

Programski kod 3.3: C++ kod za zadatak 3.2

```
#include <iostream>
#include <vector>

using namespace std;
int main( )
{
    int n;
    cout << "Unesi broj brojeva: ";
    cin >> n;

    vector<int> a(n + 1), b(n + 1);

    cout << "Unesi pozicije nakon prolaza: ";
    for (int i = 1; i <= n; ++i) {
        cin >> a[i];
        if (a[i] < i)
            for (int j = i; j > a[i]; --j)
                b[j] = b[j - 1];
        b[a[i]] = i;
    }

    cout << "Konacni poredak: ";
```

```
23      for (int i = 1; i <= n; ++i)
24          cout << b[i] << endl;
25  }
```

Programski kod 3.4: Python kod za zadatak 3.2

```
1  n = int(input("Unesi broj skijasa"))
2
3  ljestvica = []
4  print ("Unesi pozicije skijasa")
5  for i in range(1, n+1):
6      ljestvica.insert(int(input())-1, i)
7
8  for skijas in ljestvica:
9      print(skijas, end=" ")
```

Za suprotni problem (za zadani konačni poredak skijaša treba pronaći poziciju svakog od skijaša nakon njegovog nastupa) također je teško pronaći linearni pa ovdje ćemo navesti kvadratni algoritam (algoritam vremenske složenosti $O(n^2)$).

Ako je zadan konačni poredak skijaša 6, 1, 4, 3, 2, 7, 5, 8 (podsjetimo se, najbolji je skijaš koji je startao 6., zatim onaj koji je startao 1., itd., a najlošije vrijeme imao je skijaš koji je startao 8.), potrebno je pronaći na kojem je mjestu, na primjer, bio skijaš broj 5 nakon svoga nastupa. Prije skijaša koji je označen brojem 5 nastupili su samo skijaši s brojevima 1, 2, 3 i 4, pa u brojanju i razmišljanju o poziciji petoga nije potrebno uzeti u obzir rezultate skijaša s većim startnim brojem.

Stoga, za određivanje mjesta kojeg je nakon svog nastupa, zauzeo broj 5, treba samo prebrojati od koliko je on, nakon svog nastupa skijaša bio lošiji s obzirom na skijaše koji su krenuli sa starta prije njega. Skijaš sa startnim brojem 5 ukupno je lošiji od skijaša s rednim brojevima 1, 4, 3 i 2, tj. lošiji je od svih koji su nastupili prije njega, stoga će njegovo prolazno vrijeme nakon njegovog nastupa biti peto.

Skijaš s rednim brojem tri ukupno je četvrti. Od njega je bolji skijaš s rednim brojem 1, dakle, on je nakon svog nastupa bio pozicioniran nakon rednog broja 1, tj. bio je drugi.

Analogno, za svakog od skijaša dovoljno je pregledati koliko postoji skijaša koji su ukupno, na kraju utrke, bolje pozicionirani od njega, a imaju manji startni broj.

Ako je skijaš u konačnom poretku završio na mjestu za koje vrijedi da su svi brojevi koji su manji od njega desno od njega, tj. od njega su ukupno lošije pozicionirani, to znači da je on nakon svog nastupa bio bolji od svih njih te je nakon svog nastupa bio prvi.

Ako je skijaš u konačnom poretku završio na mjestu za koje vrijedi da je samo jedan broj koji je manji od njega lijevo od njega, tj. od njega je samo jedan skijaš bolje pozicioniran, to znači da je on nakon svog nastupa bio bolji od svih koji su nastupili prije njega osim od jednoga pa je nakon svog nastupa bio drugi.

Ukupno, kako bismo odredili plasman svakog od skijaša nakon njegovog nastupa, dovoljno je prebrojati koliko je skijaša s brojem manjim od njegovog u ukupnom poretku prije njega i tom broju zbrojiti jedan i dobit ćemo traženi podatak.

U obliku pseudokoda:

```
1: unesi prirodni broj n
2: postavi polje konačnih pozicija b[i] na nulu
3: za svaki i od 1 do n
4:       unesi a[i]
5:       za svaki j od 1 do a[i]
6:             ako je b[j] > 0 onda postavi b[a[i]] = b[a[i]] + 1
7:       b[a[i]] = b[a[i]] + 1
8: za svaki i od 1 do n ispiši b[i]
```

Gore navedene probleme moguće je dodatno ubrzati koristeći strukturu podataka balansirano binarno stablo, koje nam daje rješenje vremenske složenosti $O(n \cdot \log n)$.

ZADATAK 3.3.

Napišite program koji će za zadani broj crvenih, plavih i žutih kockica ispisati koliko se od tih kockica može napraviti igračaka robota, aviona i cvjetova ako vrijedi:

- za jednu igračku robota trebamo 3 crvene, 4 plave i 5 žutih kockica,
- za jednu igračku aviona potrebno nam je 1 crvena, 2 plave i 3 žute kockice i
- za jednu igračku cvijeta potrebno nam je 4 crvene kockice, 5 plavih kockica i 4 žute kockice.

Sve kockice moraju biti potrošene za izgradnju navedenih igračaka.
Ako se od zadanih kockica ne može napraviti određeni broj cijelih zadanih igračaka tako da potrošimo sve zadane kockice, program treba ispisati -1.

Primjer 1. Za unesene 23 30 29, izlaz je 2 1 4 jer se od navedenih kockica može napraviti točno 2 robota, 1 avion i 4 cvijeta.

Primjer 2. Za unesene 80 110 120, izlaz je 10 10 10 jer se od navedenih kockica može napraviti točno 10 robota, 10 aviona i 10 cvjetova.

Primjer 3. Za unesene 14 14 14, potrebno je ispisati -1 jer se od 14 crvenih, 14 plavih i 14 žutih kockica ne može složiti cijeli broj navedenih igračaka.

Rješenje. Želimo li na kraju slaganja imati cijeli broj sastavljenih figura, bez ostataka, potrebno je potrošiti sve zadane crvene, sve zadane plave i sve zadane žute kockice.

Pretpostvimo da nam je korisnik unio kao ulaz u algoritam x crvenih, y plavih i z žutih kockica.

Za jednog robota potrebne su nam 3 crvene kockice, za jedan avion jedna, a za cvijet 4 crvene kockice.

Za a robota, b aviona i c cvjetova potrebno nam je $3a + b + 4c$ crvenih kockica.
Ako želimo da sve zadane kockice budu potrošene, mora vrijediti

$$3a + b + 4c = x.$$

Za jednog robota potrebne su nam 4 plave kockice, za jedan avion dvije, a za cvijet 5 plavih kockica.

Za a robota, b aviona i c cvjetova potrebno nam je $4a + 2b + 5c$ plavih kockica.
Ako želimo da sve zadane kockice budu potrošene, mora vrijediti

$$4a + 2b + 5c = y.$$

Za jednog robota potrebno nam je 5 žutih kockica, za jedan avion 3 žute kockice, a za jedan cvijet 4 žute kockice.
Za a robota, b aviona i c cvjetova potrebno nam je $5a + 3b + 4c$ žutih kockica.
Ako želimo da sve zadane kockice budu potrošene, mora vrijediti

$$5a + 3b + 4c = z.$$

Riješimo li sustav

$$\begin{aligned} 3a + b + 4c &= x \\ 4a + 2b + 5c &= y \\ 5a + 3b + 4c &= z \end{aligned}$$

dobivamo sljedeće rješenje:

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} \frac{1}{4}(7x - 8y + 3z) \\ \frac{1}{4}(-9x + 8y - z) \\ \frac{1}{2}(-x + 2y - z) \end{bmatrix}.$$

Stoga, ako su gore izračunati a, b i c prirodni brojevi, rješenje treba samo ispisati.
Ako ti brojevi nisu prirodni, treba ispisati -1.

Programski kod 3.5: C++ kod za zadatak 3.3

```
1  #include <iostream>
2  #include<cmath>
3
4  using namespace std;
5  int main()
6  {
7
8      float x, y, z;
9      cout << "Unesi broj crvenih, plavih i zutih kockica ";
10     cin >> x >> y >> z;
11
12     float a = (7 * x - 8 * y + 3 * z) / 4,
13         b = (-9 * x + 8 * y - 1 * z) / 4,
14         c = (-1 * x + 2 * y - 1 * z) / 2;
15
```

```
16      if (a == abs(floor(a)) && b == abs(floor(b)) && c == abs(floor(c)))
17          cout << a << " " << b << " " << c << endl;
18      else cout << -1 << endl;
19  }
```

Programski kod 3.6: Python kod za zadatak 3.3

```
1  import math
2
3  x, y, z = map(float, input("Unesi broj crvenih, plavih i zutih kockica ").
       split())
4
5  a = (7 * x - 8 * y + 3 * z) / 4
6  b = (-9 * x + 8 * y - 1 * z) / 4
7  c = (-1 * x + 2 * y - 1 * z) / 2
8
9  if a == abs(math.floor(a)) and b == abs(math.floor(b)) and c == abs(math.
       floor(c)):
10     print(f"{a} {b} {c}")
11 else:
12     print(-1)
```

Funkciju `abs` moramo dodati u provjeru rješenja jer ne želimo za rješenja negativne brojeve.

ZADATAK 3.4.

Selo se sastoji od 50 kuća te su brojevi na njima također postavljeni od 1 do 50.
U kućama broj 1, broj 2 i broj 3 nalazi se po jedna kosilica – robot. Kosilica nepogrešivo sama održava travu (kosi travu, kupi lišće ili grančice s travnjaka, tjera životinje s travnjaka i slično). Seljani su zajedničkom akcijom skupili novce za iznajmiti sva ta tri robota za jednu sezonu, od travnja do listopada, a onda će se tijekom sezone košenja i održavanja travnjaka ti roboti premještati od kuće do kuće.
Pojavila se potreba poslati jednog od tih triju robota na kućni broj 21 gdje će u sljedećih nekoliko dana robot održavati travnjak.
Nakon zahtjeva za odlazak u kuću s brojem 21, do kraja sezone dogodit će se još stotine zahtjeva za premještanjima robota u neka od dvorišta. Trošak premještanja kosilice – robota iz dvorišta x u dvorište y uvijek je isti i iznosi 100 eura, bez obzira koliko su kuće udaljene i bez obzira koja je po redu kuća u nizu zahtjeva za posluživanje kosilicom – robotom. Ako se pojavio zahtjev za odlaskom robota u neku kuću koja već sadržava robota, trošak je, naravno, nula. Postavlja se pitanje kojeg od već postavljenih robota premjestiti u dvorište koje je iskazalo potrebu za sređivanjem. Naravno, seljani tog sela žele da nakon cijele sezone košenja i održavanja travnjaka njihov trošak bude što manji.
Napišite program koji će, na osnovu unaprijed danog niza kućnih brojeva, tj. unaprijed zadanih zahtjeva za robotom odrediti raspored posluživanja i putovanja po kućama uz koji će ukupni trošak putovanja robota biti najmanji.
Program neka ispiše samo iznos konačnog troška putovanja robota.

Primjer 1. Za zadani niz kuća 5, 2, 5, 1, 5, 6 najbolji raspored slanja robota je sljedeći:

1. Robot iz kuće broj 3 ide u kuću 5 i pravi trošak 100 eura.
2. Robot je već u kući broj 2 i tu nema troška premještaja.
3. Robot je već u kući broj 5 i tu nema troška premještaja.
4. Robot je već u kući broj 1 i tu nema troška premještaja.
5. Robot je već u kući broj 5 i tu nema troška premještaja.
6. Robot iz kuće broj 5 ide u 6 i trošak je 100 eura, a ukupni je trošak na kraju 200 eura.

U tablici je zadano:

	1	2	3
5			
2			
5			
1			
5			
6			

Rješenje je:

	1	2	3
5	1	2	$\underline{5}$
2	1	$\underline{2}$	5
5	1	2	$\underline{5}$
1	$\underline{1}$	2	5
5	1	2	$\underline{5}$
6	1	2	$\underline{6}$

Podvučeni broj označava robota koji je poslužio zahtjev.
Za zadani niz kuća 4, 5, 7, 4, 8, 6, 2, 5, 3, 1 najbolji raspored slanja robota je dan u tablici.
Zadano je:

	1	2	3
4			
5			
7			
4			
8			
6			
2			
5			
3			
1			

Rješenje je:

	1	2	3
4	$\underline{4}$	2	3
5	4	2	$\underline{5}$
7	4	2	$\underline{7}$
4	$\underline{4}$	2	7
8	$\underline{8}$	2	7
6	$\underline{6}$	2	7
2	6	$\underline{2}$	7
5	$\underline{5}$	2	7
3	$\underline{3}$	2	7
1	$\underline{1}$	2	7

Ukupni trošak zadovoljavanja svih zahtjeva je 800 eura.

Rješenje. Roboti se u svakom koraku rješavanja nalaze na nekim lokacijama. Naš je zadatak odlučiti kojeg robota poslati na sljedeću lokaciju. Odmah je jasno da možemo isprobati sve mogućnosti. Jasno je isto tako da u svakom koraku imamo tri mogućnosti. Ako niz kuća sadrži dva elementa, tj. ako se tijekom sezone iz pozicije 1, 2 i 3 moramo samo na dvije lokacije seliti, ukupan broj mogućnosti zadovoljavanja tih zahtjeva je 9.

1-1 je oznaka da ćemo na prvi zahtjev poslati prvog robota te na drugi zahtjev također prvog robota. Sada su 1-1, 1-2, 1-3, 2-1, 2-2, 2-3, 3-1, 3-2, 3-3 sve moguće kombinacije i njih je 3^2. Analogno, kod n zahtjeva svih mogućih načina posluge zahtjeva bit će 3^n, što već za $n = 40$ prelazi uobičajene granice računanja u C-u ili C++-u.

Stoga, ovaj eksponencijalno složeni prvi algoritam treba zamijeniti nekim praktičnijim kako bismo mogli rješavati i veće primjerke problema.

Ako smo na početku u lokacijama 1, 2 i 3 te se iza toga pojavljuje veliki niz zahtjeva, među kojima je prvi zahtjev 4, kojeg robota treba poslati na lokaciju 4?

Razmislimo li malo, nije teško vidjeti da ćemo tamo poslati onog robota koji je na lokaciji koju više nećemo trebati posjećivati ili koja će se možda i pojaviti u nadolazećem nizu zahtjeva, ali najkasnije od svih onih kuća u kojima se roboti trenutno nalaze. Na primjer, ako su roboti trenutno u kućama 5, 8 i 1, a sljedeći je zahtjev 6, nakon kojega dolaze zahtjevi 8, 6, 1 i 5, onda ćemo kući s brojem 6 poslati robota koji je u kući broj 5

jer će nam roboti koji su u kućama s brojevima 8 i 1 trebati prije toga jer se brojevi 8 i 1 pojavljuju u nizu zahtjeva prije broja 5.
Tako dolazimo do sljedeće ideje za brzi algoritam za rješavanje posluživanja zahtjeva kosilicama – robotima: ako je zahtjev koji je sljedeći u nizu zahtjeva takav da robot – kosilica već radi na tom kućnom broju, ne treba ništa pomicati; robot koji je već tamo obavit će traženi posao, a ako je novi zahtjev takav da nekog od robota moramo pomicati na novu lokaciju, poslat ćemo onog robota koji je u kući čiji se kućni broj više ne pojavljuje u nadolazećem nizu brojeva kuća koje treba posjetiti ili onog čija se lokacija, ako se u nadolazećem nizu brojeva pojavljuju svi oni brojevi kod kojih su trenutno roboti, u nizu zahtjeva najkasnije pojavljuje od svih triju lokacija u kojima su roboti trenutno. Drugim riječima, zadržat ćemo na svojim lokacijama one robote koji će nam uskoro trebati (jer uskoro možda slijedi u nizu zahtjeva potreba da neki od robota upravo na toj lokaciji sredi travnjak).

Upravo navedeni algoritam kvadratne je vremenske složenosti ($O(n^2)$, gdje je n broj zadanih zahtjeva), dakle znatno je brži od gore navedenog eksponencijalnog algoritma.

Na kraju još razmislimo na koji bi se način trebalo implementirati, tj. konkretno napisati u zadanom programskom jeziku, ideju traženja zahtjeva koji se najkasnije nalazi u nizu zahtjeva.

Ako zadani niz ,,nije dugačak", tj. ako ima 100 ili 1000 članova ili eventualno 10000, možemo ga cijelog pregledati od kraja do početka i zapamtiti zadnje pojavljivanje svakog od zahtjeva, tj. svake od lokacija u kojoj su trenutno kosilice – roboti u svakom koraku algoritma. Zadnje pojavljivanje, gledajući od kraja niza zahtjeva, prvo je pojavljivanje pojedinog kućnog broja gledajući od početka niza.

Ako zadani niz ima jako puno članova, ne isplati ga se cijelog pregledavati u svakom koraku jer će se u slučaju, na primjer, više milijardi zahtjeva, kuće najvjerojatnije ponavljati puno puta u zadanom nizu pa je bolje krenuti od sljedećeg zahtjeva i ići prema kraju niza te zapamtiti sljedeće pojavljivanje trenutno zauzetih kuća za svaki od zahtjeva.

Dolje su dana oba algoritma, tj. prvo je naveden algoritam koji je jednostavniji za manje nizove zahtjeva, a nakon toga naveden je algoritam koji je brži kod velikog niza zahtjeva jer neće u svakom koraku ispitati cijeli niz s obzirom na pozicije kuća koje su trenutno u posjedu kosilice – robota.

Programski kod 3.7: Jednostavniji C++ kod za zadatak 3.4

```cpp
#include <iostream>
#include <vector>

using namespace std;
int main()
{
    int n;

    cout << "Unesi broj zahtjeva: ";
    cin >> n;
    vector<int> z(n + 1);
    cout << "Unesi niz zahtjeva:";
    for(int i = 1; i <= n; ++i)
        cin >> z[i];

    int l1 = 1, l2 = 2, l3 = 3, trosak = 0;

    for(int i = 1; i <= n; ++i) {
        if (l1 == z[i] || l2 == z[i] || l3 == z[i]) continue;

        int maksu = 1, u1 = n + 1, u2 = n + 1, u3 = n + 1;
        for (int j = n; j >= i; --j) {
            if (l1 == z[j]) u1 = j;
            if (l2 == z[j]) u2 = j;
            if (l3 == z[j]) u3 = j;
        }
        if (u2 > u1 && u2 >= u3) maksu = 2;
        if (u3 > u1 && u3 >= u2) maksu = 3;

        ++trosak;
        if (maksu == 1) l1 = z[i];
        if (maksu == 2) l2 = z[i];
        if (maksu == 3) l3 = z[i];
    }
    cout << "Trosak je " << trosak * 100 << endl;
}
```

Programski kod 3.8: Složeniji C++ kod za zadatak 3.4

```cpp
#include <iostream>
#include <vector>

using namespace std;
```

```
5  int main()
6  {
7      int n;
8
9      cout << "Unesi broj zahtjeva: ";
10     cin >> n;
11     vector<int> z(n + 1);
12     cout << "Unesi niz zahtjeva:";
13     for(int i = 1; i <= n; ++i)
14     cin >> z[i];
15
16     int l1 = 1, l2 = 2, l3 = 3, trosak = 0;
17
18     for(int i = 1; i <= n; ++i) {
19         if (l1 == z[i] || l2 == z[i] || l3 == z[i]) continue;
20
21         int maksu = 1, u1 = n + 1, u2 = n + 1, u3 = n + 1;
22         for (int j = i; j <= n; ++j) {
23             if (l1 == z[j] && u1 == n + 1) u1 = j;
24             if (l2 == z[j] && u2 == n + 1) u2 = j;
25             if (l3 == z[j] && u3 == n + 1) u3 = j;
26         }
27         if (u2 > u1 && u2 >= u3) maksu = 2;
28         if (u3 > u1 && u3 >= u2) maksu = 3;
29
30         ++trosak;
31         if (maksu == 1) l1 = z[i];
32         if (maksu == 2) l2 = z[i];
33         if (maksu == 3) l3 = z[i];
34     }
35     cout << "Trosak je " << trosak*100 << endl;
36 }
```

Programski kod 3.9: Python kod za zadatak 3.4

```
1  n = int(input("Unesi broj zahtjeva: "))
2  print("Unesi niz zahtjeva")
3  z= [int(input()) for i in range(n)
4
5  trosak = 0
6  l1 = 1
7  l2 = 2
8  l3 = 3
9
10 for i in range(n):
```

```
11      if l1 == z[i] or l2 == z[i] or l3 == z[i]:
12          continue
13      else:
14          maksu = 1
15          u1 = n + 1
16          u2 = n + 1
17          u3 = n + 1
18          for j in range(i, n):
19              if l1 == z[j] and u1 == n + 1:
20                  u1 = j
21              if l2 == z[j] and u2 == n + 1:
22                  u2 = j
23              if l3 == z[j] and u3 == n + 1:
24                  u3 = j
25          if u2 > u1 and u2 >= u3:
26              maksu = 2
27          if u3 > u1 and u3 >= u2:
28              maksu = 3
29          trosak += 1
30          if maksu == 1:
31              l1 = z[i]
32          if maksu == 2:
33              l2 = z[i]
34          if maksu == 3:
35              l3 = z[i]
36
37  print("Trosak je", trosak*100)
```

ZADATAK 3.5.

Napišite računalni program koji će, za uneseni n i za unesenih n različitih znakova, ispisati sve njihove permutacije.

Primjer 1. Za $n = 3$ i znakove ABC rješenje je

ABC
ACB
BAC
BCA
CAB
CBA.

Primjer 2. Za uneseni $n = 4$ i brojeve 1357 rješenje je

$$1357, 1375, 1537, 1573, 1735, 1753,$$
$$3157, 3175, 3517, 3571, 3715, 3751,$$
$$5137, 5173, 5317, 5371, 5713, 5731,$$
$$7135, 7153, 7315, 7351, 7513, 7531.$$

Rješenje. Ovaj se problem može riješiti na više načina, no najpoznatije je rješenje ono u kojemu se rekurzivnom funkcijom izračuna, za uneseni n, svih $n!$ (n faktorijela) permutacija.
Mi ćemo zadatak pokušati riješiti jednostavnijim i intuitivnijim razmišljanjem.

Ako je korisnik unio $n = 3$ i znakove a_1, a_2 i a_3, potrebno je ispisati sljedeće permutacije:

$a_1a_2a_3$
$a_1a_3a_2$
$a_2a_1a_3$
$a_2a_3a_1$
$a_3a_1a_2$
$a_3a_2a_1$.

Čitatelja sada svakako pozivamo da ne čita odmah dolje navedeno razmišljanje, nego da sam pokuša naći sve permutacije dvočlanog, tročlanog, četveročlanog itd. skupa, a onda da iz indeksa elemenata, pokuša sam zaključiti na koje sve načine možemo naći sve razmještaje elemenata tih skupova.

Što god bili uneseni znakovi, slova ili brojevi, stvar je u tome da kada unesemo n znakova, a_1, a_2, ..., a_n, mi moramo naći permutacije brojeva od 1 do n i onda ispisati simbole koji se odnose na dobivenu permutaciju. Za gornji primjer a_1, a_2 i a_3, moramo pronaći sve permutacije brojeva 1, 2 i 3 i onda ispisati elemente koji se odnose na te indekse.

Pogledajmo sve permutacije prvih četiriju prirodnih brojeva.

1234
1243
1324
1342
1423
1432
2134
2143
...
4321.

Želimo li slijedno poredati permutacije po veličini, tj. ispisati ih po redu, po veličini počevši od ,,najmanje", izgled permutacije moramo analizirati s desne strane.

Pitamo se kako od permutacije 1234 dobiti sljedeću permutaciju 1243, tj. razmještaj tih četiriju znamenki koji je sljedeći po veličini ako te permutacije gledamo kao prirodne četveroznamenkaste brojeve. Očito je potrebno zamijeniti samo zadnja dva broja.

Pogledamo li sve permutacije od četiriju ili pet elemenata, vidjet ćemo da je sljedeću permutaciju najlakše dobiti kada je predzadnji broj manji od zadnjeg jer tada je sljedeća permutacija jednaka prethodnoj, osim što su zadnja dva broja zamijenjena. Na primjer, nakon 3214 dolazi 3241.

Sada smo riješili polovinu broja permutacija (svaku drugu). Nešto je teže pitanje kako naći sljedeću permutaciju ako su zadnja dva broja padajuća.
Koji je sljedeći prirodni broj na redu nakon 3241 (a sastoji se od 1, 2, 3 i 4)?
To je 3412.
Koji je sljedeći prirodni broj na redu nakon 1432 (a sastoji se od 1, 2, 3 i 4)?
To je 2134.

Algoritam je sljedeći: ako polazeći od kraja prema početku permutacije, imamo skupinu od n rastućih znamenki, onda treba otići do prve znamenke koja više nije veća od prethodne (to je $n + 1$ od kraja) i tu znamenku povećati na prvu sljedeću po veličini

koja se nalazi u skupu znamenaka desno od nje u promatranoj permutaciji, a sve ostale znamenke treba složiti po veličini, zajedno s navedenom znamenkom na mjestu $n+1$.
Na primjer: Ako je promatrana permutacija 1432, na kraju je niz od triju rastućih znamenki: 2, 3, 4 nakon kojih, gledajući od kraja niza, dolazi pad na znamenku 1. Sada 1 treba povećati na prvu sljedeću vrijednost desno od nje, a to je dva, a preostale znamenke, 1, 3 i 4 treba složiti po veličini kako bismo dobili sljedeću permutaciju: 2134. Ovo slaganje po veličini bit će složenosti $O(n)$ jer su znamenke već složene po veličini od zadnje znamenke prema onoj kod koje je ,,pad". Nakon 1354762 na redu je 1356247.

Na kraju, uočimo i kako će u prvoj permutaciji zadnji indeks biti veći od predzadnjeg, a zatim obrnuto itd. Taj postupak zajedno s ispisom svih dobivenih permutacija treba ponavljati sve dok prvu znamenku ne budemo mogli više povećati nekom od desnih znamenaka. Tada je postupak gotov.

Postoje i mnogi drugi načini ispisa svih $n!$ permutacija n-članog slupa i svaka je ideja jednako vrijedna (iako i ne jednako vremenski i prostorno učinkovita, što je u ovom problemu važno jer za samo npr. $n = 10$, treba ispisati $10! = 3628800$ permutacija.

Programski kod 3.10: C kod za zadatak 3.5

```
1  int main()
2  {
3      int temp, n, i, j, kraj, mjesto;
4      long long int nf;
5      printf("Unesi broj elemenata ");
6      scanf("%d", &n);
7      printf("Unesi elemente ");
8      char z[n + 1], a[n + 1];
9
10     // uvicatamo od 0 da ucitamo i prelazak u novi red
11     for (i = 0; i <= n; i = i + 1) scanf("%c", &z[i]);
12     for (i = 1; i <= n; i = i + 1) a[i] = i;
13
14     printf("\n");
15     for (i = 1; i <= n; i = i + 1) printf("%c", z[a[i]]);
16
17     nf = 1;
18     for (i = 1; i <= n; i = i + 1) nf = nf * i;
19
20     for (j = 1; j <= nf / 2 - 1; j = j + 1) {
21         temp = a[n]; a[n] = a[n - 1]; a[n - 1] = temp;
22
23         printf("\n");
```

```
24          for (i = 1; i <= n; i = i + 1) printf("%c", z[a[i]]);
25
26          for (i = n; i >= 1; i = i - 1)
27          if (a[i] > a[i - 1]) { mjesto = i - 1; i = 0; }
28
29          for (i = n; i > mjesto; i = i - 1)
30          if (a[i] > a[mjesto]) {
31              temp = a[i];
32              a[i] = a[mjesto];
33              a[mjesto] = temp;
34              i = 0;
35          }
36
37          kraj = (mjesto + 1 + n) / 2;
38          for (i = mjesto + 1; i <= kraj; i = i + 1) {
39              temp = a[i];
40              a[i] = a[n - i + mjesto + 1];
41              a[n - i + mjesto + 1] = temp;
42          }
43
44          printf("\n");
45          for (i = 1; i <= n; i = i + 1) printf("%c", z[a[i]]);
46      }
47
48      printf("\n");
49      for (i = n; i >= 1; i = i - 1) printf("%c", z[i]);
50      printf("\n");
51  }
```

Napomenimo kako programski jezici koje koristimo imaju gotove alate za prikaz ,,sljedeće" permutacije. Za ilustraciju slijedi program u programskom jeziku Python.

Programski kod 3.11: Python kod za zadatak 3.5

```
1  import itertools
2  znakovi = input("Unesite string sa znakovima koje zelite permutirati: ")
3  for permutacija in itertools.permutations(znakovi):
4      print(*permutacija, sep="")
```

Napomenimo na kraju i da C++ ima gotovu funkciju za izračun ,,sljedeće" permutacije, koji dozvoljava i ponavljanje elemenata. Za ilustraciju slijedi program u programskom jeziku C++.

Programski kod 3.12: C++ kod za zadatak 3.5

```cpp
#include <iostream>
#include <string>
#include <algorithm>

using namespace std;

int main() {
    cout << "Unesite string sa znakovima koje zelite permutirati: ";
    string s; cin >> s;

    sort(s.begin(), s.end());

    cout << s << endl;
    while (next_permutation(s.begin(), s.end()))
        cout << s << endl;
}
```

ZADATAK 3.6.

Napišite program koji će za uneseni broj elemenata skupa i za unesene sve elemente skupa koji su ili slovo ili broj, te za uneseni prirodni broj k koji je manji od n, ispisati sve k-člane podskupove tog skupa, odnosno, drugim riječima rečeno, napišite program koji će ispisati sve kombinacije bez ponavljanja r-tog razreda zadanog n-članog skupa.
Ili: Napišite program koji će za uneseni n i k ispisati sve moguće kombinacije koje mogu izaći u „lotu k od n".

Primjer 1. Za $n = 6$ i za skup čiji su elementi 1, 2, 3, 4, 5 i 6 te za $k = 2$ potrebno je ispisati sve dvočlane podskupove skupa $\{1, 2, 3, 4, 5, 6\}$. To su:

$$\{1,2\},\{1,3\},\{1,4\},\{1,5\},\{1,6\},$$
$$\{2,3\},\{2,4\},\{2,5\},\{2,6\},$$
$$\{3,4\},\{3,5\},\{3,6\},$$
$$\{4,5\},\{4,6\},$$
$$\{5,6\}.$$

Primjer 2. Za $n = 4$ i za skup čiji su elementi A, B, C i D te za $k = 3$ potrebno je ispisati sve tročlane podskupove skupa $\{A, B, C, D\}$. To su:

$$\{A,B,C\},\{A,B,D\},\{A,C,D\},\{B,C,D\}.$$

Za prvih 45 brojeva ukupan broj svih šesteročlanih podskupova u stvari je broj mogućih kombinacija koje mogu izaći kao izvučeni brojevi lota 6 od 45.

Rješenje. Od n elemenata, njih k možemo odabrati na ukupno

$$\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$$

načina.
Korisnik treba prvo unijeti broj n te n elemenata skupa.
Nakon toga unosi broj k.
Neka je još jednom $n = 6$ te neka se taj šesteročlani skup sastoji od brojeva 1, 2, 3, 4, 5 i 6 te za $k = 3$, pogledajmo kako te trojke izgledaju. Radi razumijevanja uzorka, navedimo nekoliko početnih podskupova zadanog skupa koji čine tražene kombinacije. To su:

$$\{1,2,3\},\{1,2,4\},\{1,2,5\},\{1,2,6\},\{1,3,4\},\{1,3,5\},\ldots$$

Očito će biti potrebno, za općenitih n članova skupa a_1, a_2, a_3, …, a_n, ispisati k-torke

$$\{a_1,a_2,a_3,\ldots,a_k\},\{a_1,a_2,a_3,\ldots,a_{k+1}\},\ldots$$

Kao u prethodnom zadatku s permutacijama, važne će nam biti samo kombinacije prvih k prirodnih brojeva u n-članom skupu jer će to biti indeksi elemenata skupa za koje nam nije važno jesu li brojevi ili slova itd.

Očito će za zadani k biti potrebna petlja koja će ispisati, tj. odabrati, za početak, prvih k prirodnih brojeva. Zatim ćemo zadnji od tih indeksa povećati za jedan. Zatim ćemo zadnji indeks ponovo povećati za jedan i tako dalje, sve dok taj zadnji indeks ne bude jednak n.

Kada zadnji indeks bude n, povećat ćemo predzadnji indeks za jedan, a zadnji ćemo indeks postaviti na broj za jedan veći od predzadnjeg i tako dalje.

Kad zadnji indeks opet dođe do n, a predzadnji indeks je $n-1$, tada ćemo treći po redu indeks, gledajući od kraja niza, povećati za jedan, a sve indekse desno od njega poredati odmah iza njega.

Postavlja se pitanje kako to napraviti? Jedan od najjednostavnijih i najintuitivnijih načina je sljedeći: u petlji treba krenuti od zadnjeg elementa i povećati ga za jedan. Ako to nije moguće, onda treba pregledati redom od kraja sve indekse i pronaći onaj koji se može povećati za jedan, tj. koji na mjestu desno od njega nema indeks veći za jedan, i zatim tog slobodnog povećati za jedan, a sve one koji su desno od njega treba složiti sortirane na sva sljedeća mjesta nakon njega.

Kratko, algoritmu ćemo reći neka radi po sljedećem principu: kreni od kraja niza i prvi indeks koji na prvom mjestu desno ima broj koji je veći za dva ili više, povećaj za jedan, a sve one indekse koji su veći od njega složi odmah desno uz njega, po veličini.

Provjerimo na primjeru gore predloženi algoritam: za $n = 6$, $k = 3$ i stanje $\{1, 4, 5\}$ redom promatrajmo od kraja trenutni niz i povećavajmo za jedan prvi element koji se može povećati za jedan.

$$\{1,4,5\} \to \{1,4,6\} \to \{1,5,6\} \to \{2,3,4\} \to \{2,3,5\} \to \dots$$

Programski kod 3.13: C kod za zadatak 3.6

```c
#include <stdio.h>

int main()
{
    int temp, n, k, i, j, r;

    printf("Unesi broj elemenata: ");
```

```
    scanf("%d", &n);
    printf("Unesi broj elemenata podskupa (k): ");
    scanf("%d", &k);
    char z[n + 1];
    int a[n + 1];

    printf("Unesi elemente skupa: ");
    for (i = 0; i <= n; i = i + 1)
        scanf("%c", z + i);

    for (i = 1; i <= k; i = i + 1)
        a[i] = i;

    printf("\n");
    for (i = 1; i <= k; i = i + 1)
        printf("%c", z[a[i]]);

    do {
        if (a[k] < n) {
            a[k] = a[k] + 1;
            printf("\n");
            for (i = 1; i <= k; i = i + 1)
                printf("%c", z[a[i]]);
        } else
            for (i = k - 1; i >= 1; i = i - 1)
                if (a[i + 1] - a[i] > 1) {
                    a[i] = a[i] + 1;
                    for (j = i + 1; j <= k; j = j + 1)
                        a[j] = a[i] + j - i;
                    printf("\n");
                    for (r = 1; r <= k; r = r + 1)
                        printf("%c", z[a[r]]);
                    i = 0;
                }
    } while (a[1] <= n - k);
    printf("\n");
}
```

Programski kod 3.14: Python kod za zadatak 3.6

```
import itertools
n = int(input("Unesite broj elemenata skupa: "))
k = int(input("Unesite broj elemenata podskupa: "))

skup = set()
```

```
6  for _ in range(n):
7      skup.add(input())
8
9
10 vidjeno = set()
11 for permutacija in itertools.permutations(skup, k):
12     trenutni = frozenset(permutacija)
13     if trenutni in vidjeno:
14         continue
15     print("{", end="")
16     print(*trenutni, sep=", ", end="")
17     print("}")
18     vidjeno.add(trenutni)
```

Navedeni kod u programskom jeziku Python koristi se gotovom bibliotekom `itertools`.

Nakon unošenja parametara n i k, te nakon unošenja svih elemenata u strukturu skup koju smo i nazvali „skup”, program u for petlji prolazi kroz sve permutacije svih elemenata iz zadanog skupa (te je time navedeni program nešto sporiji od gore navedenog rješenja u programskom jeziku C) i odabire one permutacije koje nisu već ispisane.

ZADATAK 3.7.

Napišite program koji će (u što kraćem vremenu i uz što manji utrošak memorije) za zadani prirodni broj n i za zadanih n prirodnih brojeva pronaći najmanji prirodni broj koji nije među njima.

Primjer 1. Za uneseni $n = 7$ i za brojeve 2, 1, 5, 11, 555, 4, 6 računalo treba ispisati broj 3 jer je to najmanji prirodni broj koji nije član zadanog skupa brojeva.

Primjer 2. Za uneseni $n = 4$ i za brojeve 1, 2, 1, 2 računalo treba ispisati broj 3 jer je to najmanji prirodni broj koji nije član zadanih brojeva.

Rješenje.
Ideja 1. U petlji od broja jedan do najvećeg broja od zadanih n brojeva uvećanog za 1, za sve brojeve isprobat ćemo jesu li među unesenima. Najmanji koji nije u unesenim brojevima je traženi izlaz.

U gornjem prvom primjeru u petlji treba za svaki broj od 1 do 556 isprobati je li u zadanom nizu. Prvi prirodni broj koji nije u tom nizu je rješenje.

U prvoj petlji brojač u njoj postavljen je od 1 do najvećeg broja u zadanom nizu uvećanog za 1. U petlji koja je u njoj (a koja provjerava sve unesene brojeve) brojač ide od prvog do n-tog elementa zadanog niza.

Ukupna je složenost, dakle, $O(M \cdot n)$, gdje je M najveći element niza uvećan za 1 (takva se složenost naziva pseudopolinomijalna složenost).
Dodatnu memoriju nije potrebno koristiti. Ukupno, vremenska je složenost loša (za npr. ulaz 1, 2, 3, ..., 100 000 program provjerava $(10^5 + 1) \cdot 10^5$ mogućnosti), a prostorna je složenost najbolja moguća, tj. osim $O(n)$ za unos podataka, ostatak je $O(1)$.

Programski kod 3.15: Ideja 1. C kod za zadatak 3.7

```c
#include <stdio.h>

int main()
{
    int n, i, j, imaga, M;
    printf("Unesi broj elemenata: ");
    scanf("%d", &n);
    int a[n + 1];

```

```
10      for (i = 1; i <= n; i = i + 1)
11          a[i] = 0;
12
13      printf("Unesi elemente skupa: ");
14      for (i = 1; i <= n; i = i + 1)
15          scanf("%d", &a[i]);
16
17      M = 1;
18      for (i = 1; i <= n; i = i + 1)
19          if (a[i] > M) M = a[i];
20
21      for (i = 1; i <= M + 1; i = i + 1) {
22          imaga = 0;
23          for (j = 1; j <= n; j = j + 1)
24              if (i == a[j])
25          imaga = 1;
26          if (imaga == 0) break;
27      }
28
29      printf("%d\n", i);
30  }
```

Ideja 2. Kad korisnik unese broj n i n prirodnih brojeva, u petlji od 1 do n naći ćemo najveći od tih unesenih brojeva. Označimo taj broj kao gore sa M. Nakon toga, kreirat ćemo novo polje, npr. $b[M]$ koje će biti tipa `bool` (odnosno `char` jer C nema `bool`) i sve elemente tog polja postavit ćemo na 0.
Zatim ćemo u petlji po svim elementima ulaznog niza postaviti sve pripadne članove polja b koji se odnose na element polja $a[\,]$ na 1. Drugim riječima, ako je ulazni niz bio 2, 3, 7, 1, postavit ćemo $b[2] = 1$, $b[3] = 1$, $b[7] = 1$, $b[1] = 1$. Na kraju ćemo samo u jednostrukoj petlji pronaći prvi b koji je i dalje nula, koji nije 1 i njegov će indeks biti rješenje.

Vremenska složenost ovog algoritma je linearna, $O(n)$, no i prostorna je sada također $O(n)$, tj. za ulazni skup brojeva 10, 100 000, iako su zadana samo dva broja i odmah se vidi da je rješenje broj 1, u memoriji za ideju 2 moramo zauzeti 100 000 `char` lokacija za polje b.

Programski kod 3.16: Ideja 2. C kod za zadatak 3.7

```
1   #include <stdio.h>
2
3   int main()
4   {
5       int n, i, M;
```

```
6     printf("Unesi broj elemenata: ");
7     scanf("%d", &n);
8     int a[n + 1];
9
10    printf("Unesi elemente skupa: ");
11    for (i = 1; i <= n; i = i + 1)
12        scanf("%d", &a[i]);
13
14    M = 1;
15    for (i = 1; i <= n; i = i + 1)
16        if (a[i] > M)
17            M = a[i];
18
19    char b[M + 1];
20    for (i = 1; i <= M; i = i + 1)
21        b[i] = 0;
22
23    for (i = 1; i <= n; i = i + 1) b[a[i]] = 1;
24
25    for (i = 1; i <= M; i = i + 1)
26        if (b[i] == 0) break;
27
28    printf("%d\n", i);
29 }
```

Ideja 3. Problem kod ideje 2 je što ako u ulaznom nizu postoji ,,veliki" broj, onda moramo kreirati neko veliko polje b. U ideji 3 pokušat ćemo riješiti problem pojave velikog broja u ulaznom nizu. Također, koristit ćemo tip podatka `vector<bool>` kod kojeg svaki element zauzima samo jedan bit.

Ako unosimo pet brojeva, onda će najmanji prirodni broj, koji nije među tih pet brojeva, sigurno biti između broja 1 i broja 5, ili ako smo u tih pet brojeva unijeli i 1 i 2 i 3 i 4 i 5, onda će traženi broj biti 6.

Dakle, ne treba pretraživati brojeve od 1 do najvećeg unesenog broja uvećanog za 1 kako bismo doznali koji od tih brojeva nije među unesenima, nego samo brojeve od 1 do n i tu pronaći onaj koji nije među unesenim brojevima. Ako ga nema, ispisat ćemo $n+1$. Prvo ćemo unijeti prirodni broj n, a zatim i svaki od n brojeva, ali nećemo te brojeve ,,pamtiti" u vektoru a, nego ćemo samo, kao u ideji 2, polje $a[\,]$ proglasiti bool poljem, a onda pri unosu broja x, proglasiti $a[x] = 1$, i to samo za one brojeve x koji su manji ili jednaki n zbog gore navedenog razloga. Na kraju ćemo samo pregledati polja od $a[1]$ do $a[n]$. Čim je neki od tih članova, npr. $a[y]$, jednak nuli, y će biti rješenje. Ako su svi $a[i]$, gdje i ide od 1 do n jednaki 1, rješenje će biti $n + 1$.

Programski kod 3.17: Ideja 3. C++ kod za zadatak 3.7

```cpp
#include <iostream>
#include <vector>

using namespace std;
int main()
{
    int n, i;
    cout << "Unesi broj elemenata: ";
    cin >> n;
    vector<bool> a(n + 1);

    cout << "Unesi elemente skupa: ";
    for (i = 1; i <= n; ++i) {
        int k; cin >> k;
        if (k <= n) a[k] = true;
    }

    for (i = 1; i <= n; ++i)
        if (!a[i]) break;

    cout << i << endl;
}
```

U Phytonu je potpuno drugačije riješen problem. Nakon unosa broja n i kreiranja skupa brojeva „`brojevi`" te unošenja brojeva, provjerava se samo, u `while` petlji, prisutnost broja u skupu.

Programski kod 3.18: Python kod za zadatak 3.7

```python
n = int(input())
brojevi = set()
for _ in range(n):
    brojevi.add(int(input()))

najmanji = 1
while najmanji in brojevi:
    najmanji += 1

print(f"{najmanji=}")
```

ZADATAK 3.8.

Napišite program koji će za zadani string, koji se sastoji od otvorenih i zatvorenih oblih, uglatih i vitičastih zagrada, ispisati radi li se o pravilno uparenim zagradama ili ne.

Primjer 1. Za zadani string (( )) odgovor je **da**.

Primjer 2. Za zadani string ( )[ ] odgovor je **da**.

Primjer 3. Za zadani string (( )))) odgovor je **ne**.

Primjer 4. Za [ ] { } { } odgovor je **da**.

Primjer 5. Za [( ]) odgovor je **ne**.

Rješenje. Kao prvo, uočavamo da otvorenih zagrada mora biti onoliko koliko i zatvorenih zagrada. Štoviše, otvorenih uglatih zagrada mora biti koliko i zatvorenih uglatih zagrada, otvorenih vitičastih kao i zatvorenih vitičastih itd. Također, niti jedan pripadajući par zagrada ne smije imati prvo zatvorenu pa otvorenu zagradu. No, promotrimo li treći primjer, [( ]) , vidimo da to nisu dovoljni uvjeti za regularnost zadanog stringa.

Promotrimo string ([( )] { }). Očito se radi o dobro uparenim zagradama. U njemu se nalazi jedan (bar jedan) međusobno pripadajući par zagrada koje su jedna uz drugu, otvorena i odmah nakon nje zatvorena zagrada i taj par možemo kompletirati i izbrisati iz stringa. Sada je novi string regularan ako i samo ako je regularan početni zadani string.

Zato bi prva ideja ispitivanja regularnosti bila sljedeća: ići ćemo redom po svim zagradama dok ne nađemo mjesto na kojemu se nalazi neka od triju vrsta zagrada koja je otvorena, i odmah iza nje ta ista vrsta zagrada, ali zatvorena. To je sigurno pripadajući par zagrada i njih ćemo izbrisati iz stringa i onda promatrati manji string. Ponovit ćemo isti postupak sve dok ima još zagrada u stringu. Izbacimo li iz stringa zadnji par zagrada koji se može spariti i izbaciti, a u stringu više ne ostane ništa, string je bio regularan. Ako više ne možemo izbacivati zagrade, a u stringu je još nešto preostalo, string nije regularan.

Za ovu ideju koristit ćemo strukturu podataka stog. Otvorit ćemo stog, na njega naredbom `push` postavljati zagrade, a ako želimo postaviti zatvorenu zagradu na stog odmah iza otvorene, onda ćemo tu otvorenu naredbom `pop` podići sa stoga. Ako nakon zadnje unesene zagrade stog bude prazan, string je bio regularan. Ne zaboravimo da stog može biti prazan u toku postavljanja zagrada na stog i skidanja zagrada sa stoga dok još nismo sve zagrade uzeli u obzir za postavljanje na stog pa petlju moramo početi s `if` naredbom: ako je stog prazan, postavi aktualni znak na stog.

Programski kod 3.19: C++ kod za zadatak 3.8

```cpp
#include <iostream>
#include <string>
#include <stack>

using namespace std;

int main()
{
    string izraz;
    cout << "Unesi izraz";
    cin >> izraz;
    stack <char> stog;

    for (int i = 0; i < izraz.length(); ++i) {
        if (stog.empty())
            stog.push(izraz[i]);
        else if ( (stog.top() == '(' && izraz[i] == ')')
                || (stog.top() == '{' && izraz[i] == '}')
                || (stog.top() == '[' && izraz[i] == ']'))
                   stog.pop();
        else stog.push(izraz[i]);
    }

    if (stog.empty()) cout << "Regularan je.";
        else cout << "Nije regularan";
}
```

Programski kod 3.20: Python kod za zadatak 3.8

```python
ulaz=input()

parentheses = {')': '(', '}': '{', ']': '['}

def provjeri(ulaz):
    stack = []
    for char in ulaz:
        if char in parentheses.values():
            stack.append(char)
        elif char in parentheses.keys():
            if not stack or stack.pop() != parentheses[char]:
                return False

    return not stack
```

```
15
16 print("da" if provjeri(ulaz) else "ne")
```

ZADATAK 3.9.

Napišite računalni program koji će, za zadane prirodne brojeve m i n i za zadanu matricu veličine $m \times n$ koja se sastoji samo od nula i jedinica, a pojedini red predstavlja binarni broj, ispisati najveći zbroj binarnih brojeva matrice koji možemo dobiti tako što ćemo od matrice koja je zadana invertirati proizvoljan red i proizvoljan stupac proizvoljan broj puta. U dolje navedenom tekstu matrice poradi jednostavnosti navodimo u obliku tablice.

Primjer 1. Zadana je matrica

0	1	1
1	1	0
0	0	1
1	0	0

.

1. Invertirajmo treći stupac. Dobivamo:

0	1	0
1	1	1
0	0	0
1	0	1

.

2. Invertirajmo prvi i treći red. Dobivamo:

1	0	1
1	1	1
1	1	1
1	0	1

.

Sada je ukupan zbroj

$$(4+1)+(4+2+1)+(4+2+1)+(4+1)=24.$$

Primjer 2. Zadana je matrica

0	1
1	1
0	0
1	0
1	1

.

1. Invertirajmo prvi i treći red. Dobivamo:

1	0
1	1
1	1
1	0
1	1

.

Sada je ukupan zbroj

$$2 + 3 + 3 + 2 + 3 = 13.$$

Rješenje. Za početak treba uočiti važno svojstvo invertiranja redova i stupaca: u postupku invertiranja redova i stupaca, radi dobivanja matrice s najvećim brojevima, redoslijed invertiranja nije važan, tj. svejedno je hoćemo li prvo invertirati red x pa red y ili prvo red y pa red x. Isto vrijedi i za stupce. No, svejedno je i hoćemo li prvo invertirati stupac x pa red y ili stupac y pa red x. Razlog je sljedeći: kod invertiranja stupca x u tom stupcu, promijenit će se sve znamenke, pa i ona koja je u redu y, tj. koja je u presjeku x-tog stupca i y-tog reda. Nakon toga, kod invertiranja reda y, u redu y promijenit će se sve znamenke, ali i ona jedna znamenka koja je bila u stupcu x. Ukupno će se promijeniti svi brojevi u promatranom redu i stupcu, osim onog koji je u presjeku. Isto će se dogoditi ako prvo promijenimo red y pa nakon toga stupac x.

Druga stvar koju je potrebno uočiti je da najveću vrijednost binarnog broja dobijemo ako taj broj počinje jedinicom. Stoga nam je cilj sve prve znamenke pretvoriti u jedinice. To možemo učiniti na dva načina: ili ćemo sve one redove koji počinju s nulom invertirati kako bi počeli s jedinicom ili ćemo invertirati sve one redove koji počinju jedinicom tako da se prvi stupac sastoji od samih nula, a onda ćemo invertirati taj stupac kako bi dobili opet sve jedinice na počecima binarnih brojeva predstavljenih recima matrice.

Treća stvar koju možemo uočiti (ali i ne moramo jer s dosadašnjim napomenama već možemo kreirati dosta brzi algoritam) je da nakon što smo na prva mjesta svakog broja

namjestili broj jedan (tj. prvi stupac nam se sastoji od samih jedinica), treba invertirati samo i točno one stupce koji imaju više nula nego jedinica jer što više imamo jedinica u drugom stupcu, to će ukupan zbroj binarnih brojeva biti veći. Treba, dakle, za svaki stupac prebrojiti broj jedinica i ako ih je manje od pola, taj stupac treba invertirati.

Zadnje je što možemo uočiti je da, dakle, imamo dva načina za pronalaženje najvećeg zbroja binarnih vrijednosti: ili ćemo što više brojeva u matrici gore navedenim idejama pretvoriti u jedinice, a s naglaskom na prvi stupac, ili ćemo napraviti suprotnu stvar, tj. prvo ćemo što više brojeva pretvoriti u nule, a onda invertirati cijelu matricu red po red (i stupac po stupac). Ako sada razmislimo korak dalje, uočit ćemo da nije potrebno napraviti obje te konstrukcije i odabrati bolju zato što će nam one dati isti rezultat. Evo i razloga za tu jednakost dvaju rezultata: ako prvim načinom invertiramo sve redove kako bi nam sve početne znamenke bile 1, a drugim načinom invertiramo sve redove kako bi nam sve početne znamenke bile nula, očito ćemo nakon tih dvaju postupaka dobiti dvije ,,suprotne" matrice, u smislu da na mjestima gdje prva matrica ima znamenku nula, druga matrica ima znamenku 1 i obrnuto. U drugom koraku prvog načina invertiramo one stupce koji imaju više nula nego jedinica, a u drugom slučaju (u suprotnom slučaju) invertiramo te iste stupce jer su to sada stupci koji sadrže više jedinica nego nula. Tako da ćemo i nakon drugog dijela invertiranja dobiti opet suprotne matrice. Na kraju, kada drugu matricu, onu koja ima nule na početku, invertiramo cijelu, dobit ćemo prvu matricu, tj. istu vrijednost zbroja binarnih vrijednosti redova.

Ukupno, pokušajmo, dakle, što više znamenaka matrice pretvoriti u jedinice.
Prvo ćemo unijeti m, n i $m \cdot n$ elemenata matrice.

Zatim ćemo u petlji pronaći sve one redove koji počinju s nulom i onda u drugoj petlji, koja je u sklopu te prve, sve elemente tog reda invertirati. Invertiranje ćemo napraviti jednostavnom formulom

$$a[i][j] = 1 - a[i][j].$$

Zatim ćemo izaći iz obiju petlji i pronaći stupce koji imaju više nula nego jedinica i njih invertirati, također u dvostrukoj petlji gdje će vanjska petlja ići po stupcima matrice, od drugog do zadnjeg, a prva unutrašnja petlja prebrojat će nule i jedinice, dok će druga, koja će nastupiti nakon prve kada se prva zatvori, promijeniti sve znamenke.

Na kraju ćemo zbrojiti brojeve koje čine redovi matrice tako što ćemo prebrojati jedinice u svakom od stupaca i pomnožiti njihov broj s pripadnom potencijom broja 2.

Nakon ovoga dobivamo i konačnu ideju za algoritam: nije potrebno naknadno prebrojati jedinice jer smo već jednom proveli taj postupak prije druge faze invertiranja, tj. prije invertiranja svakoga od stupaca.

Ukupna je suma, dakle, za unesenu matricu veličine $m \times n$ jednaka $m \cdot 2^{n-1}$ (to je vrijednost prvog stupca koji se sastoji od samih jedinica) $+x_2 \cdot 2^{n-2}$, gdje je x_2 broj onih znamenki kojih je više u drugom stupcu (nakon invertiranja redova i to je vrijednost znamenki u drugom stupcu) $+x_3 \cdot 2^{n-3}$, gdje je x_3 broj onih znamenki kojih je više u trećem stupcu (nakon invertiranja redova i to je vrijednost znamenki u trećem stupcu) i tako dalje.

Ukupno, kod izgleda ovako:

Programski kod 3.21: C kod za zadatak 3.9

```c
#include <stdio.h>

int main() {
    int m, n, i, j, zbroj, jedinice, nule;
    printf("Unesi dimenzije matrice: ");
        scanf("%d %d", &m, &n);

    char a[m + 1][n + 1];
    for (i = 1; i <= m; i = i + 1)
        for (j = 1; j <= n; j = j + 1)
            scanf("%d", &a[i][j]);

    for (i = 1; i <= m; i = i + 1)
        if (a[i][1] == 0)
            for (j = 1; j <= n; j = j + 1)
                a[i][j] = 1 - a[i][j];

    zbroj = m << n - 1;

    for (i = 2; i <= n; i = i + 1) {
        jedinice = 0; nule = 0;
        for (j = 1; j <= m; j = j + 1) {
            if (a[j][i] == 0) nule = nule + 1;
            if (a[j][i] == 1) jedinice = jedinice + 1;
        }
        if (jedinice > nule)
            zbroj = zbroj + (jedinice << n - i);
        else
            zbroj = zbroj + (nule << n - i);
    }

    printf("Zbroj je %d\n", zbroj);
}
```

Uočavamo kako smo na kraju koda ispitivali dva broja jer nam je trebao veći od tih dvaju.

Ta nam se situacija svodi na zadatak u kojemu moramo za zadana dva broja ispisati veći od njih bez korištenja ispitivanja. To možemo napraviti na više načina. Dvije su najzanimljivije ideje

$$\text{veći}(a,b) = 0.5 * (a + b + \text{abs}(a - b))$$
$$\text{veći}(a,b) = a * (\text{bool})(a/b) + b * (\text{bool})(b/a)$$

pa program dalje možemo optimizirati. I zadnji korak, uočimo da je broj jedinica jednak m–(broj nula). Također, zapišimo program radi manjeg broja linija u drugačijoj formi.

Program u programskom jeziku Python rješava navedeni problem na gotovo isti način, a umjesto nula koristi konstantu „`False`" te funkcijom „`not`" pretvara nule u jedinice i obrnuto.

Programski kod 3.22: Python kod za zadatak 3.9

```
1  import math
2
3  m, n, i, j, zbroj, nule = 0, 0, 0, 0, 0, 0
4  print("Unesi dimenzije matrice: ")
5  m, n = map(int, input().split())
6  a = [[False] * (n + 1) for _ in range(m + 1)]
7
8  for i in range(1, m + 1):
9      for j in range(1, n + 1):
10         a[i][j] = bool(int(input()))
11
12 for i in range(1, m + 1):
13     if a[i][1] == False:
14         for j in range(1, n + 1):
15             a[i][j] = not a[i][j]
16
17 zbroj = m * math.pow(2, n - 1)
18
19 for i in range(2, n + 1):
20     nule = 0
21     for j in range(1, m + 1):
22         if a[j][i] == False:
23             nule = nule + 1
24     zbroj = zbroj + (m + abs(m - nule - nule)) // 2 * math.pow(2, n - i)
25
26 print("Zbroj je", zbroj)
```

ZADATAK 3.10.

Napišite program koji će, za uneseni broj n i unesenih n brojeva, od kojih je svaki broj prirodan broj između 1 i n, ispisati bar jedan od brojeva koji se u nizu pojavljuje bar dva puta. Ako takvih brojeva nema, računalo ne treba ispisati ništa.
Također, program mora biti linearne vremenske složenosti, tj. $O(n)$, i konstantne prostorne složenosti, tj. točnije rečeno, osim vektora potrebnog za unos brojeva, više nije dozvoljeno koristiti niti jedan vektor. Također, jednom kada smo unijeli polje prirodnih brojeva, njegov sadržaj mora nam ostati do kraja programa, tj. polje ulaznih podataka ne smijemo izgubiti tijekom rada programa.

Primjer 1. Za uneseni broj 5 i brojeve 3, 2, 4, 2, 5 izlaz je 2.

Primjer 2. Za uneseni broj 9 i brojeve 94, 8, 7, 6, 5, 4, 3, 8, 8, izlaz je 8.

Primjer 3. Za uneseni broj 4 i brojeve 4, 3, 4, 3 izlaz je broj 3 (dakle, treba ispisati bar jedan broj koji se pojavljuje više puta).

Rješenje. Bez dodatnih ograničenja nije teško riješiti zadani problem.

Evo jednostavnog primjera rješenja koje ne zadovoljava uvjet linearne vremenske složenosti. Prvo unosimo broj brojeva n i nakon toga unosimo n prirodnih brojeva u polje $a[\,]$, a nakon toga u dvostrukoj se petlji uspoređuju različiti brojevi unesenog polja te ako se nađu dva jednaka, ispisujemo ih te prekidamo petlje postavljanjem $i = n + 1$. Uočimo kako ovaj program radi dobro i za općenita polja, tj. ne samo za polja koja imaju ograničenja na vrstu i veličinu brojeva.

Programski kod 3.23: C kod bez dodatnih ograničenja za zadatak 3.10

```
1  #include <stdio.h>
2
3  int main() {
4      int n, i, j;
5
6      printf("Unesi broj brojeva: ");
7      scanf("%d", &n);
8      int a[n + 1];
9
10     for (i = 1; i <= n; i = i + 1) scanf("%d", &a[i]);
11
12     for (i = 1; i <= n; i = i + 1)
13         for (j = 1; j <= n; ++j)
```

```
14            if (i != j and a[i] == a[j]) {
15                printf("%d\n", a[i]); i = n + 1; break;
16            }
17  }
```

Evo i rješenja koje je linearne vremenske složenosti gdje koristimo dodatni vektor b tipa `bool`.

Ponovo ćemo, kao gore, unijeti polje prirodnih brojeva. Zatim ćemo, nakon toga, deklarirati novo polje b u kojemu ćemo pamtiti broj pojavljivanja svakog od brojeva između najmanjeg i najvećeg unesenog prirodnog broja.

Čim za neki uneseni broj brojač pojavljivanja $b[\ ]$ bude 2, ispisat ćemo broj koji se prvi pojavio dva puta i prekinut ćemo ispitivanje.

Programski kod 3.24: C++ kod linearne vremenske složenosti za zadatak 3.10

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4  int main()
5  {
6      int n, i, k;
7      cout << "Unesi broj elemenata: ";
8      cin >> n;
9      vector<int> a(n + 1);
10     vector<bool> b(n + 1);
11
12     cout << "Unesi elemente: ";
13     for (i = 1; i <= n; ++i) cin >> a[i];
14
15     for (i = 1; i <= n; ++i) {
16         if (b[a[i]]) { cout << a[i] << endl; break; }
17         b[a[i]] = true;
18     }
19 }
```

Na kraju, pokušajmo zadovoljiti sve zadane uvjete. Pomoći će nam činjenica da u zadanom polju od n elemenata, imamo samo brojeve od 1 do n.

Pretpostavimo da nam je $n = 6$ i da su ulazni podaci sljedeći:

$$a[1] = 4,$$

$$a[2] = 5,$$
$$a[3] = 4,$$
$$a[4] = 1,$$
$$a[5] = 2,$$
$$a[6] = 3.$$

Očito se broj 4 ponavlja dva puta. Moramo nekako zabilježiti, odmah nakon pojavljivanja prvog broja 4, da se taj broj već jednom pojavio. No, za to ne smijemo koristiti neko novo polje, po uvjetima zadatka, nego moramo u polju $a[\,]$ uvesti dodatan podatak koji će zabilježiti da je broj 4, a tako i svaki nadolazeći broj, već imao svoje pojavljivanje. Moramo uvesti neki novi pokazatelj. To ćemo napraviti tako što ćemo $a[4]$, a dalje i a od svakog broja koji se pojavio, pomnožiti s -1. (Druga je taktika da mu dodamo neki broj veći od n ili slično.)

Kada dalje tijekom procesa ispitivanja sljedećih brojeva koji su u zadanom polju ponovo dođe broj 4, morat ćemo $a[4]$ ponovo pomnožiti s -1, no prije toga ćemo ga provjeriti, tj. ako je $a[4]$ već negativan, ne treba više ništa ispitivati, nego samo treba ispisati 4 kao rješenje. Ili, još jednostavnije, pomnožit ćemo ga s -1 pa ako onda bude pozitivan, to će značiti da je prije množenja s -1 bio negativan, tj. da smo ga već vidjeli u do tog trenutka prijeđenom dijela niza.

Kod ispitivanja treba paziti na činjenicu da su neki brojevi već postavljeni na negativnu vrijednost pa ih moramo pregledavati u apsolutnom obliku. To nećemo pregledavati $a[x]$, nego $|a[x]|$.

Na kraju, kako smo neke elemente zadanog polja učinili negativnima, moramo još jednom proći kroz čitavo polje i vrijednosti svih elemenata učiniti pozitivnima.

Programski kod 3.25: C kod koji zadovoljava sve uvjete zadatka 3.10

```c
#include <stdio.h>
#include <math.h>

int main() {
    int n, i, j;

    printf("Unesi broj brojeva: ");
    scanf("%d", &n);
    int a[n + 1];

    for (i = 1; i <= n; i = i + 1) scanf("%d", &a[i]);

```

```
13      for (i = 1; i <= n; i = i + 1) {
14          a[abs(a[i])] = -a[abs(a[i])];
15          if (a[abs(a[i])] > 0) { printf("%d\n", a[i]); break; }
16      }
17      for (i = 1; i <= n; i = i + 1)
18          if (a[i] < 0) a[i] = -a[i];
19  }
```

Literatura

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms*, Second Edition. MIT Press, McGraw-Hill, 2001.

[2] D. Knuth, *The Art of Computer Programming*, Volumes 1-4B Boxed Set. (Reading, Massachusetts: Addison-Wesley, 2023), str. 3904.

[3] Stranica Agencije za odgoj i obrazovanje namijenjena natjecanjima iz informatike. https://informatika.azoo.hr/kategorija/1/Algoritmi (zadnji pristup travanj, 2024.)

[4] Stranica Hrvatskog saveza informatičara. https://hsin.hr/natjecanja.html (zadnji pristup travanj, 2024.)

[5] R. Sedgewick, *Algorithms in C*, Addison-Wesley, 1990.

[6] J. Šribar, B. Motik, *Demistificirani C++*, 3. izdanje, Element, Zagreb, 2010.

[7] D. Kusalić, *Napredno programiranje i algoritmi u C-u i C++-u.*, 5. izdanje, Element, Zagreb, 2014.