14. srednješolsko tekmovanje ACM v znanju računalništva

Šolsko tekmovanje

18. januarja 2019

NASVETI ZA TEKMOVALCE

Naloge na tem šolskem tekmovanju pokrivajo širok razpon težavnosti, tako da ni nič hudega, če ne znaš rešiti vseh.

Nekatere naloge so tipa **napiši program** (ali **napiši podprogram**), nekatere pa tipa **opiši postopek**. Pri slednjih ti ni treba pisati programa ali podprograma v kakšnem konkretnem programskem jeziku, ampak lahko postopek opišeš tudi kako drugače: z besedami (v naravnem jeziku), psevdokodo (glej spodaj), diagramom poteka itd. Glavno je, da je tvoj opis dovolj natančen, jasen in razumljiv, tako da je iz njega razvidno, da si dejansko našel in razumel pot do rešitve naloge.

Psevdokodi pravijo včasih tudi strukturirani naravni jezik. Postopek opišemo v naravnem jeziku, vendar opis strukturiramo na podoben način kot pri programskih jezikih, tako da se jasno vidi strukturo vejitev, zank in drugih programskih elementov.

Primer opisa postopka v psevdokodi: recimo, da imamo zaporedje besed in bi ga radi razbili na več vrstic tako, da ne bo nobena vrstica preširoka.

```
naj bo trenutna vrstica prazen niz;
pregleduj besede po vrsti od prve do zadnje:
    če bi trenutna vrstica z dodano trenutno besedo (in presledkom
    pred njo) postala predolga,
        izpiši trenutno vrstico in jo potem postavi na prazen niz;
    dodaj trenutno besedo na konec trenutne vrstice;
če trenutna vrstica ni prazen niz, jo izpiši;
```

(Opomba: samo zato, ker je tu primer psevdokode, to še ne pomeni, da moraš tudi ti pisati svoje odgovore v psevdokodi.)

Če pa v okviru neke rešitve pišeš izvorno kodo programa ali podprograma, obvezno poleg te izvorne kode v nekaj stavkih opiši, kako deluje (oz. naj bi delovala) tvoja rešitev in na kakšni ideji temelji.

Pri ocenjevanju so vse naloge vredne enako število točk. Svoje odgovore dobro utemelji. Prizadevaj si predvsem, da bi bile tvoje rešitve pravilne, ob tem pa je zaželeno, da so tudi čim bolj učinkovite (take dobijo več točk kot manj učinkovite). Za manjše sintaktične napake se načeloma ne odbije veliko točk. Priporočljivo in zaželeno je, da so tvoje rešitve napisane pregledno in čitljivo. Če je na listih, ki jih oddajaš, več različic rešitve za kakšno nalogo, jasno označi, katera je tista, ki naj jo ocenjevalci upoštevajo.

Če naloga zahteva branje ali obdelavo vhodnih podatkov, lahko tvoja rešitev (če v nalogi ni drugače napisano) predpostavi, da v vhodnih podatkih ni napak (torej da je njihova vsebina in oblika skladna s tem, kar piše v nalogi).

Nekatere naloge zahtevajo branje podatkov s standardnega vhoda in pisanje na standardni izhod. Za pomoč je tu nekaj primerov programov, ki delajo s standardnim vhodom in izhodom:

- Program, ki prebere s standardnega vhoda dve števili in izpiše na standardni izhod njuno vsoto:

```
program BranjeStevil;
var i, j: integer;
begin
  ReadLn(i, j);
  WriteLn(i, ' + ', j, ' = ', i + j);
end. {BranjeStevil}
```

```
#include <stdio.h>
int main() {
  int i, j; scanf("%d %d", &i, &j);
  printf("%d + %d = %d\n", i, j, i + j);
  return 0;
}
```

- Program, ki bere s standardnega vhoda po vrsticah, jih šteje in prepisuje na standardni izhod, na koncu pa izpiše še skupno dolžino:

```
program BranjeVrstic;
var s: string; i, d: integer;
begin
  i := 0; d := 0;
  while not Eof do begin
    ReadLn(s);
    i := i + 1; d := d + Length(s);
    WriteLn(i, '. vrstica: "', s, '"');
  end; {while}
  WriteLn(i, ' vrstic, ', d, ' znakov.');
end. {BranjeVrstic}
```

```
#include <stdio.h>
#include <string.h>
int main() {
  char s[201]; int i = 0, d = 0;
  while (gets(s)) {
    i++; d += strlen(s);
    printf("%d. vrstica: \"%s\"\n", i, s);
  }
  printf("%d vrstic, %d znakov.\n", i, d);
  return 0;
}
```

Opomba: C-jevska različica gornjega programa predpostavlja, da ni nobena vrstica vhodnega besedila daljša od dvesto znakov. Funkciji gets se je v praksi bolje izogibati, ker pri njej nimamo zaščite pred primeri, ko je vrstica daljša od naše tabele s. Namesto gets bi bilo bolje (in varneje) uporabiti fgets ali fscanf; vendar pa za rešitev naših tekmovalnih nalog zadošča tudi gets.

- Program, ki bere s standardnega vhoda po znakih, jih prepisuje na standardni izhod, na koncu pa izpiše še število prebranih znakov (ne vštevši znakov za konec vrstice):

```
program BranjeZnakov;
var i: integer; c: char;
begin
  i := 0;
  while not Eof do begin
    while not Eoln do
      begin Read(c); Write(c); i := i + 1 end;
    if not Eof then begin ReadLn; WriteLn end;
  end; {while}
  WriteLn('Skupaj ', i, ' znakov.');
end. {BranjeZnakov}
```

```
#include <stdio.h>

int main() {
  int i = 0, c;
  while ((c = getchar()) != EOF) {
    putchar(c); if (i != '\n') i++;
  }
  printf("Skupaj %d znakov.\n", i);
  return 0;
}
```

Še isti trije primeri v pythonu:

```
# Branje dveh števil in izpis vsote:
import sys

a, b = sys.stdin.readline().split()
a = int(a); b = int(b)
print "%d + %d = %d" % (a, b, a + b)
```

```
# Branje standardnega vhoda po vrsticah:
import sys

i = d = 0
for s in sys.stdin:
  s = s.rstrip('\n') # odrežemo znak za konec vrstice
  i += 1; d += len(s)
  print "%d. vrstica: \"%s\"" % (i, s)
print "%d vrstic, %d znakov." % (i, d)
```

```
# Branje standardnega vhoda znak po znak:
import sys

i = 0
while True:
  c = sys.stdin.read(1)
  if c == "": break # EOF
  sys.stdout.write(c)
  if c != '\n': i += 1
print "Skupaj %d znakov." % i
```

Še isti trije primeri v javi:

```
// Branje dveh števil in izpis vsote:
import java.io.*;
import java.util.Scanner;

public class Primer1
{
   public static void main(String[] args) throws IOException
   {
      Scanner fi = new Scanner(System.in);
      int i = fi.nextInt(); int j = fi.nextInt();
      System.out.println(i + " + " + j + " = " + (i + j));
   }
}
```

```
// Branje standardnega vhoda po vrsticah:
import java.io.*;

public class Primer2
{
   public static void main(String[] args) throws IOException
   {
      BufferedReader fi = new BufferedReader(new InputStreamReader(System.in));
      int i = 0, d = 0; String s;
      while ((s = fi.readLine()) != null) {
         i++; d += s.length();
         System.out.println(i + ". vrstica: \"" + s + "\""); }
      System.out.println(i + " vrstic, " + d + " znakov.");
   }
}
```

```
// Branje standardnega vhoda znak po znak:
import java.io.*;

public class Primer3
{
   public static void main(String[] args) throws IOException
   {
      InputStreamReader fi = new InputStreamReader(System.in);
      int i = 0, c;
      while ((c = fi.read()) >= 0) {
         System.out.print((char) c); if (c != '\n' && c != '\r') i++; }
      System.out.println("Skupaj " + i + " znakov.");
   }
}
```

14. srednješolsko tekmovanje ACM v znanju računalništva

Šolsko tekmovanje

18. januarja 2019

NALOGE ZA ŠOLSKO TEKMOVANJE

Svoje odgovore dobro utemelji. Če pišeš izvorno kodo programa ali podprograma, **OBVEZNO** tudi v nekaj stavkih z besedami opiši idejo, na kateri temelji tvoja rešitev. Če ni v nalogi drugače napisano, lahko tvoje rešitve predpostavljajo, da so vhodni podatki brez napak (da ustrezajo formatu in omejitvam, kot jih podaja naloga). Zaželeno je, da so tvoje rešitve, poleg tega, da so pravilne, tudi učinkovite (bolj učinkovite rešitve dobijo več točk). Nalog je pet in pri vsaki nalogi lahko dobiš od 0 do 20 točk.

Rešitve bodo objavljene na `http://rtk.ijs.si/`.

1. e-knjiga

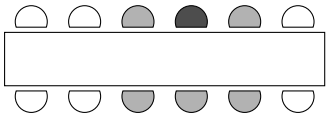
Napiši program, ki prebere besedilo in prešteje, koliko znakov angleške abecede vsebuje besedilo in koliko samoglasnikov. Program lahko bere s standardnega vhoda ali pa iz datoteke `besedilo.txt` (karkoli ti je lažje).

Program naj torej izpiše število znakov med `'a'` in `'z'` ter `'A'` in `'Z'` ter število samoglasnikov (za potrebe te naloge bomo za samoglasnike šteli znake `'A'`, `'a'`, `'E'`, `'e'`, `'I'`, `'i'`, `'O'`, `'o'`, `'U'` in `'u'`). Poleg teh znakov se lahko v vhodnem besedilu pojavljajo še poljubni drugi znaki, ki pa naj jih tvoj program ne šteje.

2. Večerja Franca Jožefa

Na dvoru pri Francu Jožefu so imeli n miz različnih dolžin; te dolžine so podane in jih označimo z $d_1, d_2, \ldots, d_n$. Vsaka miza ima obliko podolgovatega pravokotnika (velikosti $2 \times d_i$), tako da ima vsak obiskovalec sosede levo, desno, nasproti in po diagonalah (razen če sedi pri enem od vogalov — takrat nekaterih sosedov pač nima).

Primer: naslednja slika kaže mizo dolžine $d_i = 6$. Na vsaki strani mize torej sedi šest gostov. Gost, ki je na sliki pobarvan temno sivo, ima pet sosedov, in sicer tiste goste, ki so pobarvani svetlo sivo.



Na pomembni večerji so vse mize polne gostov. Stara natakarica je zelo počasna (in je zgolj ena) in pogosto se zgodi, da so na eni strani omizja hrano že pojedli, na drugi strani pa je sploh še niso dobili. Zato naredi načrt deljenja hrane tako, da bodo ljudje začeli jesti svoj obrok čim kasneje. Vsi gostje se držijo bontona, ki veli, da je nevljudno jesti, dokler nimajo vsi okrog tebe (tudi tisti po diagonalah) svojega krožnika s hrano.

Opiši postopek (ali napiši program ali podprogram, če ti je lažje), ki ugotovi, koliko krožnikov s hrano lahko natakarica razdeli, ne da bi katerikoli od gostov začel jesti, če jih deli optimalno. **Dobro utemelji**, zakaj so rezultati, ki jih vrača tvoj postopek, pravilni. Kot vhodne podatke tvoj postopek dobi števila $n, d_1, d_2, \ldots, d_{n-1}$ in d_n.

3. Robot

V podjetju, ki izdeluje igračke, so se odločili, da bodo pričeli izdelovati novo serijo robotov, ki bodo lahko peli, govorili, jokali, reševali različne naloge ter počeli še vrsto drugih zanimivih reči. Vsekakor najpomembnejša značilnost teh robotov pa je njihovo premikanje. Roboti nove serije se bodo lahko premikali le naprej ter se sukali okrog svoje osi za 90°. Zaradi boljše orientacije v prostoru je za vsakega robota pomembno, da pozna svojo trenutni položaj — koordinate v prostoru. Kot član projektne ekipe v podjetju, ki se ukvarja s pisanjem programske opreme za robote, si dobil nalogo, da **napišeš program** za robota, ki bo lahko na podlagi njegovega premikanja in obračanja ugotovil robotove koordinate v prostoru.

Vhodni podatki: v prvi vrstici vhoda se nahajajo 3 naravna števila n, x in y (vsa med 1 in 1000). Pri tem n predstavlja število ukazov (premikov ali zasukov), ki jih robot izvrši; x in y pa sta začetni koordinati robota. V drugi vrstici vhoda se nahaja n znakov, ki predstavljajo dejanske ukaze, ki jih robot izvrši. Vsak znak je ena od črk `L`, `D` ali `N`, ki imajo naslednji pomen: `L` označuje robotov zasuk za 90° v levo (nasprotna smer urinega kazalca), `D` predstavlja 90° zasuk v desno (smer urinega kazalca), `N` pa premik naprej. Robot se premika po ravni površini, ki jo predstavimo s kartezičnim koordinatnim sistemom z vodoravno osjo x in navpično osjo y ter enoto, ki predstavlja natanko en premik robota. Robot je na svojih začetnih koordinatah x in y vedno obrnjen v smeri x-osi.

Pričakovan izhod programa: vrstica, ki vsebuje dve naravni števili x in y, ki predstavljata končni koordinati robota.

Tvoj program lahko bere s standardnega vhoda in piše na standardni izhod ali pa uporablja datoteke (npr. `vhod.txt` ali `izhod.txt`), karkoli ti je lažje.

Primer vhoda:

```
7 1 3
NNNLNND
```

Pripadajoči izhod:

```
4 5
```

4. Čokolada

V zgodbi o Janku in Metki je Metka ravno potisnila zlobno čarovnico v peč in osvobodila brata Janka, ki je zaklenjen v kletki čakal in se redil, da postane večerja. V silnem navdušenju nad koncem nevarnosti in svežem duhu svobode sta se brat in sestra lotila raziskovanja posesti dobrot. V sladkorni hišici sta našla cekine in dragulje, skrivni prehod pa ju je vodil v podzemno jamo, kjer je bila skrita ogromna čokolada, ki se je raztezala globoko v temno pozemlje. Čokolado sta si ogledovala vrstico po vrstico in kaj kmalu ugotovila, da ne gre za navadno čokolado, temveč za čokolado z lešniki in rozinami!

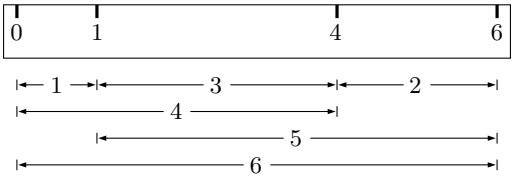
Sestradana Metka pa nad rozinami ni bila najbolj navdušena, zato je z bratom sklenila dogovor, da lahko prva izbere en strnjen odsek čokolade (eno ali več zaporednih vrstic skupaj), ki ji je najbolj všeč, on pa bo vzel preostanek. Pri tem lahko Metka največ dvakrat razlomi čokolado in vzame kos med dvema lomoma oziroma začetni/končni odsek, če je čokolado prelomila enkrat.

Ker je čokolada zelo velika, te prosita za pomoč. Iz jame ti bosta sporočila število vrstic čokolade, recimo mu n, in nato za vsako vrstico še to, koliko rozin in koliko lešnikov je v njej (recimo, da je v i-ti vrstici r_i rozin in ℓ_i lešnikov). Metka dobroto vrstice meri kot razliko v številu lešnikov in rozin ($\ell_i - r_i$), dobroto večjega kosa čokolade (takega, ki obsega več zaporednih vrstic) pa kot vsoto dobrot posameznih vrstic v njem. **Opiši postopek** (ali napiši program ali podprogram, če ti je lažje), ki iz dobljenih podatkov (torej števil $n, r_1, \ldots, r_n, \ell_1, \ldots, \ell_n$) izračuna največjo vrednost dobrote, ki jo lahko doseže strnjen odsek čokolade (torej tak, ki ga sestavlja ena ali več zaporednih vrstic). Zaželeno je, da je tvoj postopek čim bolj učinkovit, tako da bo deloval dovolj hitro tudi za velike n.

5. Golombovo ravnilo

Ravni palici z označenimi in oštevilčenimi centimetri pravimo *ravnilo*. Pri tej nalogi imamo opravka z ravnilom, ki je varčno, a pomanjkljivo: označeni so le centimetri na nakaterih mestih, ne pa nujno na razdaljah vsakega centimetra. Primer takega ravnila je palica z oznakami na centimetrih $\{0, 1, 4, 6\}$:

Kljub pomanjkljivim oznakam lahko s tem ravnilom odmerimo vse celoštevilske razdalje med 0 in 6 cm, če le izberemo ustrezni dve oznaki na ravnilu. Hkrati se pri tem ravnilu nobena razdalja med poljubnima dvema oznakama ne ponovi. Na primer:



Enako lastnost ima na primer tudi ravnilo z oznakami $\{0, 2, 7, 8, 11\}$.

Napiši program, ki bo najprej prebral število oznak na ravnilu, potem pa še toliko celih števil, ki predstavljajo lego vsake oznake. Lege so podane v naraščajočem vrstnem redu (vsaka je strogo večja od prejšnje); prva lega je vedno 0, zadnja lega pa ustreza dolžini ravnila. Dolžina ravnila je največ 100000. Primer podatkov, ki opisujejo gornje ravnilo:

```
4
0
1
4
6
```

Program naj ugotovi in izpiše, ali za ravnilo, ki ga predstavljajo prebrani podatki, veljata naslednji dve lastnosti:

- vsak celoštevilski interval med poljubnima dvema oznakama se ponovi[1] kvečjemu enkrat (lahko pa kakšen interval manjka); (mimogrede: takemu ravnilu pravimo *Golombovo ravnilo*);
- prisotni so vsi celoštevilski intervali od 1 do dolžine ravnila (mimogrede: takemu ravnilu pravimo *popolno* ravnilo).

Tvoj program lahko bere s standardnega vhoda ali pa iz datoteke `ravnilo.txt` (karkoli ti je lažje).

[1] Takšno besedilo smo uporabili na šolskem tekmovanju, vendar je ta formulacija napačna; mišljeno je, da se vsak interval *pojavi* kvečjemu enkrat (ponovi pa se sploh nikoli), npr. tako kot pri ravnilu $\{0, 1, 4, 6\}$ s primera na začetku.

14. srednješolsko tekmovanje ACM v znanju računalništva

Šolsko tekmovanje

18. januarja 2019

REŠITVE NALOG ŠOLSKEGA TEKMOVANJA

1. e-knjiga

Vhodno besedilo lahko beremo znak po znak; pri vsakem branju moramo še preveriti, če smo že prišli do konca (EOF). Med branjem vzdržujemo dve celoštevilski spremenljivki, ki štejeta doslej prebrane črke in samoglasnike. Po branju novega znaka preverimo, če je ta znak črka oz. samoglasnik, in po potrebi povečamo enega ali oba števca. Ko pridemo do konca vhodnih podatkov, moramo le še izpisati vrednosti obeh števcev. Oglejmo si primer takšne rešitve v C/C++:

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

int main()
{
  int c, stCrk = 0, stSamoglasnikov = 0;
  while ((c = fgetc(stdin)) != EOF)
  {
    c = toupper(c);
    if (c < 'A' || c > 'Z') continue;
    stCrk++;
    if (c == 'A' || c == 'E' || c == 'I' || c == 'O' || c == 'U')
      stSamoglasnikov++;
  }
  printf("%d črk, %d samoglasnikov.\n", stCrk, stSamoglasnikov);
  return 0;
}
```

Funkcijo toupper iz C-jeve standardne knjižnice smo uporabili, da nam pretvori male črke v velike, tako da nam kasneje pri preverjanju, ali je znak črka in ali je samoglasnik, ni treba posebej preverjati še malih črk. Šlo pa bi seveda tudi brez tega, le pogoji v stavkih **if** bi bili malo daljši.

Oglejmo si še primer rešitve v pythonu. Ta bo za spremembo brala vhodne podatke po vrsticah:

```
import sys
stCrk = 0; stSamoglasnikov = 0
for s in sys.stdin:
  for c in s:
    if 'A' <= c <= 'Z' or 'a' <= c <= 'z':
      stCrk += 1
      if c in "AEIOUaeiou": stSamoglasnikov += 1
print("%d črk, %d samoglasnikov." % (stCrk, stSamoglasnikov))
```

2. Večerja Franca Jožefa

Dogajanje na eni mizi nič ne vpliva na dogajanje na drugih mizah (ker sosedje posameznega človeka sedijo za isto mizo kot on), zato lahko naš postopek obravnava vsako mizo posebej. Recimo torej, da gledamo neko mizo dolžine d; vprašanje je zdaj, koliko različnim ljudem za to mizo lahko prinesemo večerjo, ne da bi kdo od njih začel jesti.

Mizo si predstavljajmo kot razdeljeno na d stolpcev, oštevilčenih od 1 do d, pri čemer vsak stolpec pokriva dva gosta (enega na vsaki strani mize). Če sta v nekem stolpcu

oba gosta že dobila večerjo, bomo rekli, da je *poln*, sicer pa, da je *prazen*. Vidimo lahko, da ne smemo imeti treh polnih stolpcev skupaj, saj bi tedaj gosta v srednjem od njih že lahko začela jesti. Med dvema zaporednima praznima stolpcema smeta biti torej največ dva polna, ne pa trije ali več.

Levo od prvega (najbolj levega) praznega stolpca pa sme biti največ en poln; če bi bila dva, bi gosta v levem od teh dveh že lahko začela jesti; več kot dva pa tudi ne smeta biti, saj smo že v prejšnjem odstavku videli, da ne smemo imeti treh polnih stolpcev skupaj. Podoben razmislek nam tudi pove, da sme biti desno od zadnjega (najbolj desnega) praznega stolpca največ en poln.

Če imamo torej k praznih stolpcev, je lahko celotna miza dolga največ $3k$ (en poln stolpec levo od prvega praznega; eden desno od zadnjega praznega; po dva med vsakima dvema zaporednima praznima; to je skupaj $1+1+2\cdot(k-1)$ polnih stolpcev; ko prištejemo še k praznih, imamo skupaj $3k$ stolpcev). Pri mizi dolžine d mora torej veljati $d \leq 3k$, kar lahko predelamo v $k \geq d/3$. Ker mora biti k celo število, je najmanjši primerni k enak $\lceil d/3 \rceil$ (to je vrednost, ki jo dobimo, če $d/3$ zaokrožimo navzgor na najbližje celo število).

Imeti moramo torej vsaj $\lceil d/3 \rceil$ praznih stolpcev in v vsakem praznem stolpcu mora biti vsaj en gost brez večerje; večerjo lahko torej prinesemo kvečjemu $2d - \lceil d/3 \rceil$ gostom, to pa je isto kot $\lfloor 5d/3 \rfloor$ (torej vrednost $5d/3$, zaokrožena navzdol na najbližje celo število). To moramo le še sešteti po vseh mizah. Zapišimo dobljeno rešitev v C/C++:

```
int Vecerja(int n, int dolzine[])
{
  int r = 0;
  for (int i = 0; i < n; i++) r += (5 * dolzine[i]) / 3;
  return r;
}
```

Ali v pythonu:

```
def Vecerja(dolzine):
  return sum((5 * d) // 3 for d in dolzine)
```

3. Robot

Niz, ki opisuje gibanje robota, bomo v zanki obdelovali znak po znak in po vsakem koraku ustrezno popravili podatke o položaju robota. Poleg njegove x- in y-koordinate moramo hraniti še smer, v katero je obrnjen, saj je od smeri odvisno, kako se njegove koordinate spremenijo pri premiku naprej.

Smer bi sicer lahko predstavili s kotom v stopinjah, ker pa se robot vedno obrača za 90 stopinj, so za orientacijo robota le štiri možnosti in jih bomo predstavili kar s števili od 0 do 3 (0 = desno, 1 = gor, 2 = levo, 3 = dol). Pri zasuku v levo se torej smer poveča za 1, pri zasuku v desno pa zmanjša za 1; paziti moramo le na to, da če bi pri tem smer padla pod 0 ali narasla nad 3, jo moramo povečati oz. zmanjšati za 4.

Sprememba koordinat pri koraku naprej je odvisna od orientacije robota. Če je obrnjen desno, se x poveča za 1, y pa ostane nespremenjen; če je obrnjen gor, je ravno obratno; itd. Te stvari je najlažje predstaviti kar s parom tabel DX in DY, ki povesta spremembo koordinate x oz. y v odvisnosti od smeri.

Oglejmo si implementacijo takšne rešitve v C++:

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

int main()
{
  const int DX[] = { 1, 0, -1, 0 }, DY[] = { 0, 1, 0, -1 };
  int n, x, y, smer = 0;

  // Preberimo začetni položaj in število korakov.
  cin >> n >> x >> y;

  // Preberimo opis gibanja robota.
  while (n-- > 0)
```

```
  {
    // Preberimo naslednji korak.
    char c; cin >> c;
    // Ustrezno popravimo položaj ali orientacijo robota.
    if (c == 'L') smer = (smer + 1) % 4;
    else if (c == 'D') smer = (smer + 3) % 4;
    else x += DX[smer], y += DY[smer];
  }
  // Izpišimo končni položaj robota.
  cout << x << " " << y << endl;
  return 0;
}
```

In v pythonu:

```
import sys

DX = [1, 0, -1, 0]; DY = [0, 1, 0, -1]
# Preberimo začetni položaj in število korakov.
n, x, y = (int(s) for s in sys.stdin.readline().split())
# Preberimo opis gibanja robota.
s = sys.stdin.readline(); smer = 0
for i in range(n):
  # Obdelajmo naslednji korak robota.
  if s[i] == 'L': smer = (smer + 1) % 4
  elif s[i] == 'D': smer = (smer + 3) % 4
  else: x += DX[smer]; y += DY[smer]
# Izpišimo končni položaj robota.
print("%d %d" % (x, y))
```

4. Čokolada

Označimo dobroto i-te vrstice z d_i; torej $d_i = \ell_i - r_i$. Naloga zdaj pravzaprav sprašuje, kakšna je največja vsota oblike $s_{ij} = d_i + d_{i+1} + \ldots + d_j$ (to je dobrota takega kosa čokolade, ki obsega vrstice od vključno i-te do vključno j-te). Pri tem morata seveda i in j ostati v okvirih $1 \le i \le j \le n$, ker ima naša čokolada n vrstic.

Preprosta rešitev je, da gremo z zanko po vseh i (od 1 do n) in nato pri vsakem i s še eno vgnezdeno zanko po vseh j (od i do n), da izračunamo vse vsote s_{ij} in si zapomnimo največjo. Vsot ni treba računati vsake posebej od začetka; ko pri fiksnem i povečamo j za 1, pridobi vsota s_{ij} na koncu še en seštevanec, vse pred njim pa ostane nespremenjeno. Tako moramo le prišteti novi seštevanec k dosedanji vsoti, pa dobimo novo vsoto za malo večji odsek čokolade. Zapišimo ta postopek s psevdokodo:

```
s* := −∞; (* Najboljši doslej najdeni rezultat. *)
for i := 1 to n:
  s := 0;
  for j := i to n:
    (* Na tem mestu je s = d_i + ... + d_{j−1}.  *)
    s := s + d_j;
    (* Zdaj je s = d_i + ... + d_j. Poglejmo, če je to najboljša rešitev doslej. *)
    if s > s* then s* := s;
return s*; (* Vrnimo najboljšo rešitev. *)
```

Pregledati moramo $O(n^2)$ parov (i, j), pri vsakem pa imamo $O(1)$ dela, da popravimo vsoto in preverimo, če je najboljša doslej (in si jo zapomnimo, če je res najboljša doslej). Časovna zahtevnost te rešitve je torej $O(n^2)$.

Do boljše rešitve pridemo z naslednjim opažanjem: vsoto od i-tega do j-tega člena lahko dobimo tako, da vzamemo vsoto prvih j členov in od nje odštejemo vsoto prvih $i - 1$ členov. Definirajmo torej kumulativne vsote: $c_k = d_1 + d_2 + \ldots + d_k$; potem za dobroto tistega kosa čokolade, ki obsega vrstice od i-te do j-te, velja $s_{ij} = c_j - c_{i-1}$.

Če primerjamo zdaj vrednosti s_{ij} za različne i pri fiksnem j, vidimo, da imajo vse isti c_j, vendar različne c_{i-1}. Da bo s_{ij} čim večja, mora biti c_{i-1} čim manjša. Ko smo pri nekem j, torej pravzaprav ni treba pregledati vseh možnih i, ampak le tistega, ki da najmanjšo c_{i-1} (izmed vseh doslej pregledanih). Zdaj ne potrebujemo več dveh gnezdenih zank, ampak le eno:

$s^\star := -\infty$; (* *Najboljši doslej najdeni rezultat.* *)
$c := 0$; $m := 0$; (* *c je trenutna kumulativna vsota, m je najmanjša doslej.* *)
for $j := 1$ **to** n:
  (* *Na tem mestu je $c = c_{j-1}$ in $m = \min\{c_0, \ldots, c_{j-1}\}$.* *)
  $c := c + d_j$;
  (* *Zdaj je $c = c_j$.* *)
  $s := c - m$;
  (* *Zdaj je $s = \max_{1 \le i \le j} s_{ij}$. Poglejmo, če je to najboljša rešitev doslej.* *)
  if $s > s^\star$ **then** $s^\star := s$;
  (* *Poglejmo, če je c najmanjša kumulativna vsota doslej.* *)
  if $c < m$ **then** $m := c$;
return $s^\star$; (* *Vrnimo najboljšo rešitev.* *)

Pri vsakem j imamo zdaj le konstantno mnogo dela, tako da je časovna zahtevnost naše nove rešitve le $O(n)$.

5. Golombovo ravnilo

Označimo z n število oznak na ravnilu, njihove lege pa (od leve proti desni) z $x_1, \ldots, x_n$. Dolžino ravnila označimo z d; naloga pravi, da je $x_1 = 0$ in $x_n = d$.

Na koliko načinov si lahko izberemo dve oznaki, med katerima bomo merili razdaljo? Če si za levo oznako izberemo x_1, si lahko za desno izberemo katerokoli od ostalih $n-1$ oznak; če si za levo oznako izberemo x_2, si lahko za desno izberemo katerokoli od $n-2$ oznak, ki ležijo desno od nje; in tako naprej. Skupaj imamo torej $(n-1) + (n-2) + \ldots + 2 + 1 = n(n-1)/2$ različnih parov oznak. Vse možne pare lahko pregledamo z dvema gnezdenima zankama, eno za levo in eno za desno oznako, in pri vsakem paru (i, j) izračunamo razdaljo med njima, torej razliko $x_j - x_i$.

Ravnilo je Golombovo, če so vse te razdalje med seboj različne, in popolno, če med njimi nastopi vsaj enkrat vsako število od 1 do d. Koristno je torej, če si nekam zapisujemo, katere razdalje smo že videli; tako lahko sproti preverjamo, če smo trenutno razdaljo videli že kdaj prej (če smo jo, lahko zaključimo, da ravnilo *ni* Golombovo), na koncu pa lahko za vse razdalje od 1 do d preverimo, če smo jih sploh kdaj videli (če kakšne nismo, lahko zaključimo, da ravnilo *ni* popolno).

Vprašanje je še, v kakšni podatkovni strukturi hraniti podatke o tem, katere razdalje smo že videli. Za našo nalogo, kjer so ravnila razmeroma kratka, je primerna kar navadna tabela ali vektor logičnih vrednosti; takšna rešitev porabi $O(d)$ pomnilnika, preverjanje tega, ali smo nek element že videli, pa je preprosto in hitro. Še ena možnost bi bila razpršena tabela oz. množica (npr. `unordered_set` v C++, `set` v pythonu ipd.), kjer bi bila poraba pomnilnika sorazmerna s tem, koliko je *različnih* razdalj, to pa je $O(\min\{n^2, d\})$. To bi bilo lahko koristno, če je število oznak n majhno v primerjavi z dolžino ravnila d. Časovna zahtevnost naše rešitve je v obeh primerih $O(n^2 + d)$.

Oglejmo si primer rešitve v C++:

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

int main()
{
  // Preberimo vhodne podatke.
  int n; cin >> n;
  vector<int> x(n);
  for (int i = 0; i < n; i++) cin >> x[i];

  // Preglejmo vse razdalje med dvema oznakama.
  int d = x[n - 1];
```

```
  vector<bool> zeVideno(d + 1, false);
  bool jeGolombovo = true;
  for (int j = 1; j < n; j++) for (int i = 0; i < j; i++)
  {
    int r = x[j] - x[i];
    // Če razdalje r ne vidimo prvič, ravnilo ni Golombovo.
    if (zeVideno[r]) jeGolombovo = false;
    else zeVideno[r] = true;
  }
  // Preverimo, če je ravnilo popolno.
  bool jePopolno = true;
  for (int r = 1; r <= d; r++)
    if (! zeVideno[r]) { jePopolno = false; break; }
  // Izpišimo rezultate.
  cout << "Ravnilo " << (jeGolombovo ? "je" : "ni") << " Golombovo in " <<
    (jePopolno ? "je" : "ni") << " popolno." << endl;
  return 0;
}
```

Še podobna rešitev v pythonu:

```
import sys

# Preberimo vhodne podatke.
n = int(sys.stdin.readline())
x = [int(sys.stdin.readline()) for i in range(n)]

# Preglejmo vse razdalje med dvema oznakama.
d = x[-1]; zeVideno = [False] * (d + 1)
jeGolombovo = True
for j in range(1, n):
  for i in range(j):
    r = x[j] - x[i]
    if zeVideno[r]: jeGolombovo = False
    else: zeVideno[r] = True

# Preverimo, ce je ravnilo popolno.
jePopolno = all(zeVideno[1:])

# Izpišimo rezultate.
print("Ravnilo %s Golombovo in %s popolno." % ("je" if jeGolombovo else "ni",
  "je" if jePopolno else "ni"))
```

Mimogrede, če se ravnilo izkaže za Golombovo, bi lahko to, ali je tudi popolno, preverili še enostavneje: takrat vemo, da so vse razdalje različne, da jih je $n(n-1)/2$ in da posamezna razdalja ne more biti manjša od 1 ali večja od d; torej, če je $d > n(n-1)/2$, potem gotovo niso prisotne vse možne razdalje od 1 do d (in ravnilo ni popolno), če pa je $d = n(n-1)/2$, potem gotovo so prisotne vse (in ravnilo je popolno).

Razmislimo še o različici naloge, pri kateri ravnilo šteje za Golombovo takrat, ko se vsak interval *ponovi* kvečjemu enkrat (namesto *pojavi* kvečjemu enkrat) — torej se sme pojaviti kvečjemu dvakrat. Vse, kar moramo spremeniti v naši rešitvi, je, da namesto tabele logičnih vrednosti uporabimo tabelo celih števil, ki štejejo, kolikokrat smo posamezni interval doslej že videli. Ko opazimo, da bi se pri nekem intervalu ta števec povečal z 2 na 3, lahko takoj zaključimo, da ravnilo ni Golombovo po tej novi definiciji:

```
  vector<int> zeVideno(d + 1, 0);
  bool jeGolombovo = true;
  for (int j = 1; j < n; j++) for (int i = 0; i < j; i++)
  {
    int r = x[j] - x[i];
    // Če smo razdaljo r prej videli že dvakrat, ravnilo ni Golombovo.
    if (zeVideno[r] >= 2) jeGolombovo = false;
    else zeVideno[r]++;
  }
```

14. srednješolsko tekmovanje ACM v znanju računalništva

Šolsko tekmovanje

18. januarja 2019

NASVETI ZA MENTORJE O IZVEDBI TEKMOVANJA IN OCENJEVANJU

Tekmovalci naj pišejo svoje odgovore na papir ali pa jih natipkajo z računalnikom; ocenjevanje teh odgovorov poteka v vsakem primeru tako, da jih pregleda in oceni mentor (in ne npr. tako, da bi se poskušalo izvorno kodo, ki so jo tekmovalci napisali v svojih odgovorih, prevesti na računalniku in pognati na kakšnih testnih podatkih). Pri reševanju si lahko tekmovalci pomagajo tudi z literaturo in/ali zapiski, ni pa mišljeno, da bi imeli med reševanjem dostop do interneta ali do kakšnih datotek, ki bi si jih pred tekmovanjem pripravili sami. Čas reševanja je omejen na 180 minut.

Nekatere naloge kot odgovor zahtevajo program ali podprogram v kakšnem konkretnem programskem jeziku, nekatere naloge pa so tipa „opiši postopek“. Pri slednjih je načeloma vseeno, v kakšni obliki je postopek opisan (naravni jezik, psevdokoda, diagram poteka, izvorna koda v kakšnem programskem jeziku, ipd.), samo da je ta opis dovolj jasen in podroben in je iz njega razvidno, da tekmovalec razume rešitev problema.

Glede tega, katere programske jezike tekmovalci uporabljajo, naše tekmovanje ne postavlja posebnih omejitev, niti pri nalogah, pri katerih je rešitev v nekaterih jezikih znatno krajša in enostavnejša kot v drugih (npr. uporaba perla ali pythona pri problemih na temo obdelave nizov).

Kjer se v tekmovalčevem odgovoru pojavlja izvorna koda, naj bo pri ocenjevanju poudarek predvsem na vsebinski pravilnosti, ne pa na sintaktični. Pri ocenjevanju na državnem tekmovanju zaradi manjkajočih podpičij in podobnih sintaktičnih napak odbijemo mogoče kvečjemu eno točko od dvajsetih; glavno vprašanje pri izvorni kodi je, ali se v njej skriva pravilen postopek za rešitev problema. Ravno tako ni nič hudega, če npr. tekmovalec v rešitvi v C-ju pozabi na začetku **#include**ati kakšnega od standardnih headerjev, ki bi jih sicer njegov program potreboval; ali pa če podprogram `main()` napiše tako, da vrača **void** namesto **int**.

Pri vsaki nalogi je možno doseči od 0 do 20 točk. Od rešitve pričakujemo predvsem to, da je pravilna (= da predlagani postopek ali podprogram vrača pravilne rezultate), poleg tega pa je zaželeno tudi, da je učinkovita (manj učinkovite rešitve dobijo manj točk).

Če tekmovalec pri neki nalogi ni uspel sestaviti cele rešitve, pač pa je prehodil vsaj del poti do nje in so v njegovem odgovoru razvidne vsaj nekatere od idej, ki jih rešitev tiste naloge potrebuje, naj vendarle dobi delež točk, ki je približno v skladu s tem, kolikšen delež rešitve je našel.

Če v besedilu naloge ni drugače navedeno, lahko tekmovalčeva rešitev vedno predpostavi, da so vhodni podatki, s katerimi dela, podani v takšni obliki in v okviru takšnih omejitev, kot jih zagotavlja naloga. Tekmovalcem torej načeloma ni treba pisati rešitev, ki bi bile odporne na razne napake v vhodnih podatkih.

V nadaljevanju podajamo še nekaj nasvetov za ocenjevanje pri posameznih nalogah.

1. e-knjiga

- Objavili smo dve rešitvi: eno, ki bere vhodno besedilo po znakih, in eno, ki ga bere po vrsticah. Oboje je enako dobro. Prav tako je enako dobro tudi, če rešitev prebere celo vhodno besedilo naenkrat v pomnilnik. Rešitev sme tudi predpostaviti neko zgornjo mejo za dolžino posamezne vrstice (npr. da je vrstica dolga največ 100 znakov ali kaj podobnega).
- Če rešitev vhodnega besedila sploh ne bere, ampak predpostavi, da ga kar dobi v nekem seznamu ali tabeli, naj se ji zaradi tega odšteje največ tri točke.

- Format izpisa pri tej nalogi ni posebej predpisan, pomembno je le, da je iz njega razvidno število črk in število samoglasnikov v prebranem vhodnem besedilu.
- Iz primerov v besedilu naloge je razvidno, da se v vhodnem besedilu lahko pojavljajo tako male kot velike črke. Če rešitev deluje samo za male ali pa samo za velike črke, naj se ji zaradi tega odšteje 6 točk.
- Rešitev si lahko pomaga s funkcijami iz standardne knjižnice svojega programskega jezika in naj se ji tega ne šteje v slabo, če deluje pravilno. Tak primer je npr. klic funkcije `toupper` v naših C-jevski rešitvi da pretvori morebitne male črke v velike; še en primer bi bil, če bi za preverjanje, ali je znak samoglasnik, uporabili `strchr`.
- Rešitev, ki preverja, ali je znak črka, tako, da ima 26 ali celo 52 pogojnih stavkov (ali pa pogojev v enem velikem pogojnem stavku), naj se zaradi tega odšteje 2 točki.
- Pri tej nalogi ni mišljeno, da bi se morala rešitev kaj ukvarjati s tem, kako so znaki v vhodni datoteki zakodirani (npr. ASCII, UTF-8 itd.), oz. se sploh kakorkoli zavedati te problematike.
- Besedilo naloge se dá razumeti tudi tako, kot da je treba šteti male črke posebej in velike črke posebej. Rešitve, ki temeljijo na tej interpretaciji, naj se obravnava kot enako dobre, kot če bi štele oboje črke skupaj.

2. Večerja Franca Jožefa

- Verjetno se bo mnogim reševalcem intuitivno zdelo očitno, da je treba pustiti brez večerje po enega gosta v vsakem tretjem stolpcu, pomembno pa je, da njihov odgovor tudi dovolj dobro utemelji, zakaj naj bi bilo to res. Rešitev brez utemeljitve naj dobi največ polovico možnih točk.
- Pri ocenjevanju utemeljitve je dobro imeti v mislih še tole. Ni težko pokazati, da z določeno razporeditvijo tega, katerim gostom ne prinesemo večerje, uspešno preprečimo, da bi kdorkoli začel jesti. Težje pa je pokazati, da ne obstaja noben drug razpored, pri katerem bi ostalo brez krožnika manj gostov, pa vendarle nihče ne bi začel jesti. Toda ravno to slednje moramo pokazati, če se hočemo prepričati, da je naša rešitev optimalna.
- Naša rešitev se z vsako mizo ukvarja le konstantno mnogo časa. Rešitve, ki bi za mizo dolžine d porabile $O(d)$ časa, naj dobijo največ 13 točk, če so drugače pravilne. Rešitve, ki bi porabile eksponentno veliko časa v odvisnosti od d (npr. ker z rekurzijo preizkusijo vse možnosti glede tega, kateri gostje ne dobijo večerje), naj dobijo največ 7 točk, če so drugače pravilne.

3. Robot

- Pri takšnih nalogah mnogo tekmovalcev piše rešitve, v katerih je skoraj enaka izvorna koda razmnožena v štirih kopijah, po eni za vsako smer. To ni elegantno, ni pa tudi samo po sebi narobe. Takšnim rešitvam naj se odšteje največ tri točke, če so drugače pravilne.
- Rešitev lahko bere opis poti robota po znakih (kot npr. naša rešitev v C++) ali pa célo naenkrat (kot npr. naša rešitev v pythonu); oboje je enako dobro.
- Zaradi morebitnih drobnih napak pri branju vhodnih podatkov naj se rešitvi odšteje največ tri točke.
- V naši rešitvi smo smer popravljali s formulami oblike `(smer + 1) % 4` in podobno, da smo poskrbeli, da `smer` ostane na območju od 0 do 3. Enako dobro je tudi, če tekmovalčeva rešitev za to poskrbi tudi kako drugače, na primer s pogojnim stavkom (`smer += 1; if (smer >= 4) smer -= 4` ipd.). Če pa rešitev nikakor ne upošteva dejstva, da je smer robota cikličen pojem, naj se ji zaradi tega odšteje pet točk.

4. Čokolada

- Naloga pravi, naj bo rešitev čim bolj učinkovita. Rešitve s časovno zahtevnostjo $O(n^3)$ ali slabšo naj dobijo največ 15 točk. Rešitve s časovno zahtevnostjo $O(n^2)$ naj dobijo največ 18 točk.
- Naloga pravi, da si mora Metka izbrati eno ali več zaporednih vrstic. Če bi rešitev pomotoma dovolila tudi, da si Metka izbere 0 vrstic (kar lahko včasih pripelje do boljših rešitev, npr. če je v vsaki vrstici čokolade več rozin kot lešnikov), naj se ji zaradi tega odbije tri točke.
- Če rešitev šteje vrstice od 0 do $n-1$ namesto od 1 do n, naj se ji zaradi tega ne odbija točk (če je drugače pravilna).

5. Golombovo ravnilo

- Ker je dolžina ravnila (s tem pa tudi največje možno število oznak na njem) pri tej nalogi še kar velika, pričakujemo rešitve s časovno zahtevnostjo manj kot $O(n^3)$. Naša rešitev ima časovno zahtevnost $O(n^2)$, vse točke pa lahko dobijo tudi rešitve v času $O(n^2 \log n)$, na primer če rešitev shranjuje že videne razdalje v drevo (npr. razred `map` v C++) ali pa v seznam, ki ga na koncu uredi s quicksortom ali čim podobnim. Rešitve s časovno zahtevnostjo $O(n^3)$ naj dobijo največ 15 točk, take z zahtevnostjo $O(n^4)$ ali slabšo pa največ 10 točk, če so drugače pravilne.
- Pri tej nalogi ni mišljeno, da bi bil poudarek na branju vhodnih podatkov. Za drobne napake pri branju vhodnih podatkov naj se rešitvi odšteje največ 2 točki.
- Format izpisa pri tej nalogi ni posebej določen; glavno je, da program ugotovi, ali je ravnilo Golombovo in ali je popolno, ni pa pomembno, kako to izpiše.
- Lastnosti, po katerih sprašuje naloga (ali je ravnilo Golombovo in ali je popolno), sta neodvisni (ravnilo ima lahko eno, drugo, obe ali nobene od teh dveh lastnosti). Rešitvi, ki preverja le eno od teh dveh lastnosti, ne pa obeh, naj se odbije 7 točk.

Težavnost nalog

Državno tekmovanje ACM v znanju računalništva poteka v treh težavnostnih skupinah (prva je najlažja, tretja pa najtežja); na tem šolskem tekmovanju pa je skupina ena sama, vendar naloge v njej pokrivajo razmeroma širok razpon zahtevnosti. Za občutek povejmo, s katero skupino državnega tekmovanja so po svoji težavnosti primerljive posamezne naloge letošnjega šolskega tekmovanja:

Naloga	Kam bi sodila po težavnosti na državnem tekmovanju ACM
1. e-knjiga	lažja naloga v prvi skupini
2. Večerja	težka naloga v prvi ali lažja v drugi skupini
3. Robot	težja naloga v prvi ali lahka v drugi skupini
4. Čokolada	srednje težka naloga v drugi ali lažja v tretji skupini
5. Golomb	srednje težka naloga v drugi ali lahka v tretji skupini

Če torej na primer nek tekmovalec reši le eno ali dve lažji nalogi, pri ostalih pa ne naredi (skoraj) ničesar, to še ne pomeni, da ni primeren za udeležbo na državnem tekmovanju; pač pa je najbrž pametno, če na državnem tekmovanju ne gre v drugo ali tretjo skupino, pač pa v prvo.

Podobno kot prejšnja leta si tudi letos želimo, da bi čim več tekmovalcev s šolskega tekmovanja prišlo tudi na državno tekmovanje in da bi bilo šolsko tekmovanje predvsem v pomoč tekmovalcem in mentorjem pri razmišljanju o tem, v kateri težavnostni skupini državnega tekmovanja naj kdo tekmuje.

Zadnja leta na državnem tekmovanju opažamo, da je v prvi skupini izrazito veliko tekmovalcev v primerjavi z drugo in tretjo, med njimi pa je tudi veliko takih z zelo dobrimi rezultati, ki bi prav lahko tekmovali tudi v kakšni težji skupini. Mentorjem zato priporočamo, naj tekmovalce, če se jim zdi to primerno, spodbudijo k udeležbi v zahtevnejših skupinah.