3. osnovnošolsko tekmovanje ACM v znanju računalništva

Šolsko tekmovanje

23. januar 2026

NASVETI ZA TEKMOVALCE

Naloge na tem šolskem tekmovanju pokrivajo širok razpon težavnosti, tako da ni nič hudega, če ne znaš rešiti vseh. V okviru svoje rešitve obvezno poleg izvorne kode v nekaj stavkih opiši, kako deluje 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('Skupaj ', i, ' vrstic, ', d,
          ' znakov.');
end. {BranjeVrstic}
```

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

int main() {
  char s[101]; int i = 0, d = 0;
  while (scanf("%[^\n]%*c", s) == 1) {
    i++; d += strlen(s);
    printf("%d. vrstica: \"%s\"\n", i, s); }
  printf("Skupaj %d vrstic, %d znakov.\n",
         i, d);
  return 0;
}
```

Opomba: C-jevska različica gornjega programa predpostavlja, da ni nobena vrstica vhodnega besedila daljša od sto znakov. V praksi je bolje uporabljati funkcije oz. načine, ki dovoljujejo omejitev branja tudi glede na velikost tabele, vendar imamo na programerskih tekmovanjih omejitve vhodnih podatkov in zagotovila glede njih, tako da bo tudi tak `scanf` zadoščal.

- 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
  while not EOF do
    begin
      Read(c); Write(c); i := i + 1
    end; {while}
  WriteLn('Skupaj ', i, ' znakov.');
end. {BranjeZnakov}
```

```
#include <stdio.h>
int main() {
  int i = 0, c;
  while ((c = getchar()) != -1){
    i++; putchar(c);
  }
  printf("Skupaj %d znakov.\n", i);
  return 0;
}
```

Še isti trije primeri v pythonu:

```
# Branje dveh števil in izpis vsote:
a, b = input().split()
a = int(a)
b = int(b)
print(a, b, a + b)
```

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

idx_vrstice = st_znakov = 0
for vrstica in sys.stdin:
    vrstica = vrstica.rstrip('\n') # odrežemo znak za konec vrstice
    idx_vrstice += 1
    st_znakov += len(vrstica)
    print(f"{idx_vrstice}. vrstica: '{vrstica}'")
print(f"{idx_vrstice} vrstic, {st_znakov} znakov.")
```

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

st_znakov = 0
while True:
    znak = sys.stdin.read(1)
    if znak == "":
        break # Konec vhoda
    sys.stdout.write(znak) # Izpišemo znak
    if znak != '\n':
        st_znakov += 1
print(f"Skupaj {st_znakov} znakov.")
```

Še isti trije primeri v javi:

```java
// 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));
    }
}
```

```java
// 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.");
    }
}
```

```java
// 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.");
    }
}
```

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). Naloge so štiri in pri vsaki nalogi lahko dobiš od 0 do 25 točk.

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

3. osnovnošolsko tekmovanje ACM v znanju računalništva

Šolsko tekmovanje

23. januar 2026

NALOGE ZA ŠOLSKO TEKMOVANJE

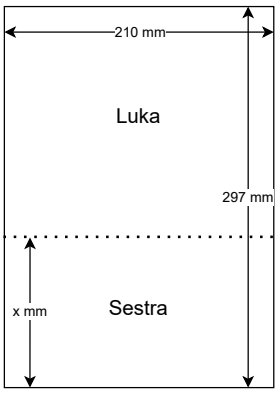
Svoje odgovore dobro utemelji. Če pišeš izvorno kodo programa ali podprograma, **OBVEZNO** tudi v nekaj stavkih z besedami opiši idejo, na kateri temelji tvoja rešitev. Če ni v nalogi drugače napisano, lahko tvoje rešitve predpostavljajo, da so vhodni podatki brez napak (da ustrezajo formatu in omejitvam, kot jih podaja naloga). Zaželeno je, da so tvoje rešitve poleg tega, da so pravilne, tudi učinkovite (bolj učinkovite rešitve dobijo več točk). Naloge so štiri in pri vsaki nalogi lahko dobiš od 0 do 25 točk.

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

1 A4 trak

V dnevno sobo si je Luka prinesel list papirja v formatu A4, da nanj nariše svojo bodočo mojstrovino. Ko je to videla njegova sestra, je želela tudi ona risati, a je papirja zmanjkalo. Luko je prosila, naj ji odreže trak od svojega lista. Luka želi imeti dovolj prostora za svojo risbo, zato te prosi, da napišeš program, ki bo izračunal, kakšna ploščina ostane, ko od lista odreže trak širine x mm.

Dimenzije formata A4 so $210\,\text{mm} \times 297\,\text{mm}$, torej je ploščina celotnega lista $210\,\text{mm} \cdot 297\,\text{mm} = 62370\,\text{mm}^2$. Luka bo rez naredil vzporedno s krajšo stranico lista. To je vodoravno, če imamo list obrnjen kot običajno.



Slika 1: Skica lista A4 z vsemi merami

Vhodni podatki

Na vhodu je celo število x, ki pove širino traku, ki ga bo odrezal Luka, v milimetrih.

Izhodni podatki

Izpiši ploščino lista, ki preostane, ko Luka odreže trak širine x.

Omejitve vhodnih podatkov

- $0 \leq x \leq 297$

Primer

Vhod:	**Izhod:**
95	42420

2 Toplotna regulacija

Sistem za toplotno regulacijo na Fakulteti za matematiko in fiziko v Ljubljani (odslej FMF) je bil grajen še v prejšnjem času in že krepko kaže svoja leta.

Situacija je taka: vodstvo FMF-ja je nastavilo **d**elovno temperaturo stavbe (T_d), pri kateri želijo uporabniki delati. Poleg nje pa za dani trenutek vemo tudi **t**renutno temperaturo stavbe (T_t).

V zgodbi nastopata pa še dva glavna akterja: Agencija Republike Slovenije za okolje (odslej ARSO) in hišnik FMF. ARSO sistemu za toplotno regulacijo sporoči, ali je trenutno poletje ali zima (ostalih letnih časov ob izdelavi sistema še ni bilo); hišnik pa prestavlja ogromno ročico v trebuhu stavbe, ki določa, ali se stavba greje ali hladi.

Delovna in trenutna temperatura se *strinjata*, če lahko uporabniki stavbe trenutno temperaturo približajo delovni s tem, da odprejo kakšno okno. Na primer: poleti je zunaj bolj vroče kot notri, torej se temperaturi *strinjata*, če je trenutna temperatura manjša ali enaka delovni (saj bi pri odprtih oknih trenutna temperatura zrasla in se približala delovni oz. želeni).

Sistem deluje na sledeči način: glede na letni čas preveri, ali se delovna in trenutna temperatura *strinjata* ali ne. Če se strinjata, se stavba toplotno ne regulira, če se ne strinjata, pa se regulira (torej se greje ali hladi, odvisno od tega, kaj je nastavil hišnik).

Sistem bo stavbo pravilno toplotno reguliral, če bo toplotna regulacija pravilno spreminjala trenutno temperaturo. Na primer, če želimo imeti poleti v stavbi 27 stopinj Celzija, je trenutna temperatura previsoka in sistem stavbo še dodatno greje, potem **ne** deluje pravilno (pravilno bi bilo, če bi jo pri previsoki temperaturi hladil). To situacijo opisuje prvi vzorčni primer.

Naloga

Ker je usklajeno delovanje stavbe z njenimi uporabniki pomembno in je cel sistem toplotne regulacije hudo zakompliciran, te vodstvo fakultete prosi, da za dani primer preveriš, ali se sistem obnaša pravilno.

Vhodni podatki

V prvi vrstici vhoda se nahajata T_d in T_t (delovna in trenutna temperatura), ločeni s presledkom.

V drugi vrstici vhoda se nahajata l in r, letni čas ter nastavitev hišnikove ročice. Pozimi velja $l = 0$, poleti pa $l = 1$, ročica na 0 hladi, na 1 pa greje.

Omejitve vhodnih podatkov

- $-273 < T_d, T_t < 1000$,
- $l, r \in \{0, 1\}$.

Izhodni podatki

V eni sami vrstici izpiši pravilnost delovanja sistema. Če se stavba ne regulira, izpiši »SE NE REGULIRA«, če se regulira pravilno, izpiši »SE REGULIRA PRAVILNO«, in, če se ne regulira pravilno, izpiši »SE REGULIRA NAROBE«.

Primer

1. vhod:

```
27 28
1 1
```

1. izhod:

```
SE REGULIRA NAROBE
```

2. vhod:

```
27 40
0 1
```

2. izhod:

```
SE NE REGULIRA
```

Komentar

Naloga je posneta po resničnih dogodkih.

V prvem vzorčnem primeru je poletje in stavba se greje, zato se regulira narobe.

V drugem vzorčnem primeru je zima, trenutna temperatura pa je višja od želene delovne, torej je za zbližanje delovne in trenutne temperature dovolj, če samo odpremo okna.

3 Pretvorba poti

Nekateri programi so ustvarjeni specifično za operacijski sistem Linux. Če jih želimo pretvoriti v program, ki bo deloval na sistemu Windows, moramo običajno spremeniti ali prilagoditi veliko kode. Med drugim moramo popraviti tudi naslove datotek v datotečnem sistemu. Te imajo v Linuxu obliko

```
/home/anze/Documents/sola/predmeti/slovenscina/esej.txt
```

v Windowsih pa

```
C:\Users\anze\Documents\sola\predmeti\slovenscina\esej.txt
```

Poleg zamenjave smeri poševnice (`/` za Linux, `\` za Windows) je torej razlika tudi v začetku poti: uporabnikove datoteke so v Linuxu shranjene pod `/home/uporabnisko_ime`, v Windowsih pa pod `C:\Users\uporabnisko_ime`. V računalniku so tudi druge datoteke, s katerimi pa se v tej nalogi ne bomo ukvarjali.

Naloga

Napiši program, ki bo pretvoril pot iz Linuxovega formata v format Windows.

Vhodni podatki

Na vhodu bo ena vrstica besedila, tj. pot do neke datoteke, ki se začne s `/home`. Vrstica bo dolga največ 200 znakov in sestoji le iz velikih in malih črk angleške abecede, števk in znakov `/`.

Izhodni podatki

Izpiši pot do datoteke, kot bi se prikazala na datotečnem sistemu Windows. Začne naj se s `C:\Users`.

Primer

Vhod:

```
/home/anze/Documents/sola/predmeti/slovenscina/esej.txt
```

Izhod:

```
C:\Users\anze\Documents\sola\predmeti\slovenscina\esej.txt
```

4 Hribarjenje

Sončnega januarskega dne je Jan dobil genialno idejo: privoščil si bo pohod. Primerno se je obul, oblekel ter s seboj vzel vse ostale nujne potrebščine. A takoj ko je zaloputnil vrata, je ugotovil, da je pozabil na najpomembnejše: sploh si še ni določil poti! Na srečo odločitev ne bo težka, saj Jan dobro ve, kakšni sprehodi mu ugajajo in kakšni mu ne.

Edino, kar ga lahko pri sprehodu zmoti, je *monotonost*. Natančneje, nadmorske višine točk njegove poti ne smejo tvoriti monotonega zaporedja. Zaporedje je monotono, če je urejeno bodisi naraščajoče bodisi padajoče. Na primer, med zaporedji $(1, 2, 4, 5)$, $(5, 2, 2, 1, 1)$ in $(3, 1, 4)$ sta monotoni prvi dve, tretje pa ni (ker prvi dve števili padata, zadnji dve pa naraščata).

Jan je s seboj vzel poseben enodimenzionalni zemljevid, na katerem je seznam nadmorskih višin različnih točk hribovja (na primer $(11, 6, 8, 2, 3, 5, 7, 4)$). Vsaka točka pokrajine ima različno nadmorsko višino. Možen *sprehod* je neki odsek tega seznama (na primer $(6, 8, 2, 3)$ v našem primeru). Sprehod je *monoton*, če je zaporedje njegovih nadmorskih višin monotono (na primer $(2, 3, 5)$). Monoton sprehod je *dolgočasen*, če se ga ne da podaljšati na levi ali na desni tako, da ostane monoton. ($(2, 3, 5)$ ni dolgočasen, ker ga lahko podaljšamo na desni v $(2, 3, 5, 7)$, ki je tudi monoton. $(2, 3, 5, 7)$ pa je dolgočasen: če ga podaljšamo, dobimo $(8, 2, 3, 5, 7)$ ali $(2, 3, 5, 7, 4)$, ki oba nista monotona.)

Da se bo lažje odločil, Jana zanima, koliko sprehodov je dolgočasnih in koliko jih je monotonih. Ker je zemljevid prevelik, potrebuje tvojo pomoč pri računanju!

Naloga

Napiši program, ki prebere velikost zemljevida ter zaporedje nadmorskih višin na njem in izpiše števili dolgočasnih ter monotonih sprehodov.

Vhodni podatki

Prva vrstica vsebuje naravno število n — dolžino zemljevida. Druga vsebuje n naravnih števil $a_1, a_2, \ldots, a_n$ — seznam nadmorskih višin.

Omejitve vhodnih podatkov

- $1 \leq n \leq 50000$
- $1 \leq a_i \leq 10^9$ za vsak $1 \leq i \leq n$
- Vsi a_i so si med sabo različni.

Izhodni podatki

V prvi vrstici izpiši skupno število dolgočasnih sprehodov, v drugi pa skupno število monotonih sprehodov. **Če pravilno izračunaš samo skupno število dolgočasnih sprehodov, dobiš 50 % točk.**

Primer

Vhod:	**Izhod:**
`8` `11 6 8 2 3 5 7 4`	`5` `18`

Komentar

V tem primeru so dolgočasni sprehodi $(11, 6)$, $(6, 8)$, $(8, 2)$, $(2, 3, 5, 7)$ in $(7, 4)$; monotoni sprehodi pa (11), (6), (8), (2), (3), (5), (7), (4), $(11, 6)$, $(6, 8)$, $(8, 2)$, $(2, 3)$, $(3, 5)$, $(5, 7)$, $(7, 4)$, $(2, 3, 5)$, $(3, 5, 7)$ ter $(2, 3, 5, 7)$.

Pri tej nalogi je lahko seznam točk dolg — **razmisli tudi o tem, kako učinkovita je tvoja rešitev** (in o tem kaj **zapiši**).

3. osnovnošolsko tekmovanje ACM v znanju računalništva

Šolsko tekmovanje

23. januar 2026

REŠITVE NALOG ŠOLSKEGA TEKMOVANJA

1 A4 trak

V nalogi preberemo pozitivno celo število x in izpišemo $210 \cdot (297 - x)$, ali pa enako $210 \cdot 297 - 297 \cdot x$ in druge oblike računa, ki privede do istega rezultata.

```c
#include <stdio.h>

int main() {
  int x;
  scanf("%d", &x);
  printf("%d\n", 210 * (297 - x));
  return 0;
}
```

Za pravilno rešitev učenec dobi 25 točk. Za kakšno sintaktično napako naj se odbije od 1 do 5 točk. Če je formula za ploščino, ki jo uporabi učenec, napačna, naj dobi največ 10 točk.

2 Toplotna regulacija

Nalogo lahko brez težav ločimo na dva dela: ugotavljanje, ali se FMF toplotno regulira, in če se, ali se regulira pravilno ali narobe.

Stavba se ne regulira, če bi lahko z odpiranjem okna popravili razmak med trenutno in delovno temperaturo. To je tako, ko je poleti trenutna temperatura nižja od delovne, pozimi pa višja.

Nato opazimo, da bomo pozimi vedno želeli greti in poleti hladiti (sicer se stavba regulira narobe). Torej pravilnost regulacije je odvisna samo od tega, kako je hišnik nastavil ročico (torej r) in letnega časa (torej l).

Rešitvi v `C`-ju in v `Python`-u sta ekvivalentni, s tem da različno preverimo, ali se stavba toplotno regulira ali ne. Pri prvi rečemo, da se ne regulira, če sta temperaturi enaki ali če je resničnost izjav »trenutna temperatura je višja od delovne« in »trenutno je poletje« različna (torej ali je trenutna višja od delovne in je zima ali pa je trenutna nižja od delovne in je poletje). V drugi rešitvi pa obe možnosti, v katerih se stavba ne regulira, razpišemo eksplicitno.

Rešitev v `C`-ju:

```c
#include<stdio.h>

int main(){
  int T_d, T_t, l, r;
  scanf("%d %d", &T_d, &T_t);
  scanf("%d %d", &l, &r);
  //  Temperatura se ne regulira, ce sta trenutna in delovna
  //  temperatura enaki ali ce je trenutno pretoplo in je zima
  //  ali je trenutno premrzlo in je poletje.
  if( T_d==T_t || ( (T_d<T_t) != (l==1) ) ) puts("SE NE REGULIRA");
  else{
    //  Ce poleti grejemo ali pozimi hladimo, se regulira narobe.
```

```
    if(l!=r) puts("SE REGULIRA PRAVILNO");
    else puts("SE REGULIRA NAROBE");
  }
  return 0;
}
```

Rešitev v Python-u:

```
T_d, T_t = map(int, input().split())
l, r = map(int, input().split())

# Ce lahko odpremo okno in s tem primerno spremenimo
# temperaturo, se stavba toplotno ne regulira.
if (T_d <= T_t and l == 0) or (T_d >= T_t and l == 1):
    puts("SE NE REGULIRA")
elif l == r:
    print("SE REGULIRA NAROBE")
else:
    print("SE REGULIRA PRAVILNO")
```

Pri tej nalogi je 12 točk vredna ugotovitev in smiselno preverjanje tega, ali se temperatura regulira, 8 točk pa, ali se regulira pravilno ali narobe (torej, da se morata ujemati način regulacije in letni čas). Zadnjih 5 točk pa sledi za pravilno implementacijo. To pomeni, da če tekmovalec pravilno ugotovi, kdaj se kaj dogaja, tega pa ne zna naprogramirati, lahko še vedno dobi 20 točk.

3 Pretvorba poti

Naloga od nas zahteva, da zamenjamo vse poševnice (slash, /) za leve poševnice (backslash, \). Prav tako želimo zamenjati začetni `/home/` s `C:\Users\`. Pri tem moramo paziti, v katerem vrstnem redu to počnemo: če bomo najprej zmenjali vse različne poševnice, potem na začetku ne bomo več imeli `/home/`, temveč `\home\`.

Rešitev v C-ju najprej izpiše del poti, ki bo vedno isti (`C:\Users\`), nato se pa sprehodi čez ostalo originalno pot in poševnice zamenja z levimi poševnicami, ostale pa samo izpiše.

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

int main(){
  char pot[261];
  int i;
  scanf("%s", pot);
  printf("C:\\Users\\");
  for(i=strlen("/home/");pot[i];++i){
    if(pot[i]=='/')
      putchar('\\');
    else
      putchar(pot[i]);
  }
  putchar('\n');
  return 0;
}
```

Rešitev v Python-u pa najprej zamenja začetek poti (domač direktorij) in nato še poševnice ene v druge.

```
pot = input()
pot = 'C:\\Users\\' + pot[len('/home/'):]
pot = pot.replace('/', '\\')
print(pot)
```

Pri tej nalogi bi dali približno 10 točk za zamenjavo poševnic, 10 točk pa za primerno zamenjavo domačega direktorija. Ostalih 5 točk pa za smiseln izpis končne poti. Tu je vseeno, ali pot izpisujemo sproti (kot v C-jevski rešitvi), ali pa najprej izračunamo novo pot in jo na koncu izpišemo (kot v rešitvi v Python). V Python-u, ki nam ponuja »priročne« funkcije za delo z besedilom, moramo paziti, saj bi npr.

```
pot = pot.replace('/home/', 'C:\\Users\\')
```

nadomestil vse[1] `/home/` dele poti, ne pa samo tistega na začetku poti.

Kot vidimo tudi v podanih rešitvah, je desna poševnica \ poseben znak in ga zato v programih pišemo dvojno (kar se pri izvajanju nato obravnava kot en znak). Za napačno uporabo desne poševnice bi odbili kakšno točko do tri, več pa ne.

4 Hribarjenje

Najprej razmislimo, kako najenostavneje prešteti dolgočasne sprehode. To, da se monotonega sprehoda ne da podaljšati na nekem krajišču, pomeni bodisi da tam zaporedje spremeni »smer rasti« bodisi da je to ravno začetek ali konec zemljevida. Torej, če je ta dolgočasen sprehod naraščajoč, bo levo krajišče dolina (točki pred in za krajiščem bosta višji od krajišča) ali začetek zaporedja, desno pa vrh (točki pred in za krajiščem bosta od njega nižji) ali konec zaporedja. Dolinam, vrhovom, začetku in koncu zaporedja skupaj pravimo *ekstremi*.

Torej, da bo neki sprehod dolgočasen, morata biti njegovi krajišči sosednja ekstrema. Poleg tega pa bo vsak sprehod, katerega krajišči sta sosednja ekstrema, tudi dolgočasen sprehod. Sledi, da je za štetje dolgočasnih sprehodov dovolj prešteti ekstreme in od tega števila odšteti ena.

Zdaj se lotimo še štetja monotonih sprehodov. Vidimo, da so to ravno vsi sprehodi, ki so v celoti vsebovani v nekem dolgočasnem sprehodu. Če ima ta dolgočasni sprehod dolžino l, bo vseboval natanko l monotonih sprehodov dolžine 1, $l-1$ monotonih sprehodov dolžine 2, ..., 2 monotona sprehoda dolžine $l-1$ in 1 monoton sprehod dolžine l. Skupno število je torej $1+2+\cdots+l=\frac{l(l+1)}{2}$.

Paziti moramo še, da so nekateri monotoni sprehodi pri tem šteti dvakrat. Vsak sprehod dolžine 1, ki se začne in konča na neki dolini ali nekem vrhu, bo namreč vštet kot »podsprehod« dolgočasnih sprehodov na obeh straneh te točke.

Preprost način, da oboje preštejemo hkrati, je da gremo skozi zemljevid od leve proti desni in v neki spremenljivki beležimo zadnji ekstrem, ki smo ga našli. Ko najdemo naslednjega, k številu dolgočasnih sprehodov prištejemo 1, k številu monotonih pa $\frac{l(l+1)}{2}$, kjer je l število točk med tem in prejšnjim ekstremom (vključno z ekstremoma). Na koncu od števila monotonih sprehodov še odštejemo tiste, ki so bili šteti dvakrat. Teh je toliko, kot je dolin in vrhov, torej ravno število dolgočasnih sprehodov minus 1.

Alternativna metoda je, da k številu monotonih sprehodov sproti prištevamo število monotonih sprehodov, ki se končajo na trenutni točki. Če je med prejšnjim ekstremom in trenutnim položajem natanko k točk (pri čemer štejemo tudi ekstrem in trenutni položaj), bo monotonih poti, ki se končajo na trenutni točki, prav tako k.

Rešitev v C-ju uporablja prvo metodo:

```
#include <stdio.h>

int a[50000];

int main() {
  int n;
  scanf("%d", &n);
  for (int i=0; i<n; i++) scanf("%d", &a[i]);
  int zadnji_ekstrem=0, dolgocasni=1, monotoni=0;
  for (int i=1; i<n-1; i++) {
    // smo na vrhu ali dolini
```

[1] Vsaj vse takšne, ki ne sledijo drugemo `/home/` delu poti.

```
    if ((a[i] > a[i-1] && a[i] > a[i+1]) ||
        (a[i] < a[i-1] && a[i] < a[i+1])) {
      dolgocasni++;
      monotoni += (i-zadnji_ekstrem+1) * (i-zadnji_ekstrem+2) / 2;
      zadnji_ekstrem = i;
    }
  }
  monotoni += (n-1-zadnji_ekstrem+1) * (n-1-zadnji_ekstrem+2) / 2;
  monotoni -= dolgocasni - 1; // pri monotonih odstejemo dvakrat stete
                // odseke (tiste, ki lezijo na prevojih)
  printf("%d\n%d\n", dolgocasni, monotoni);
  return 0;
}
```

rešitev v Python-u pa drugo:

```
n = int(input())
a = list(map(int, input().split()))
if n == 1:                # Ce je samo en vrh, bo predstavljal
    print(1, 1, sep='\n') # edino monotono in tudi edino dolgocasno
    exit(0)               # zaporedje.
# V prvem vrhu se konca eno monotono zaporedje, v drugem pa dve. Na
# zacetku je prvi ekstrem 0-ti vrh.
dolgocasna, monotona, e = 0, 1 + 2, 0
for i in range(2, n):
    # Ce je bilo padanje (oz. usmerjenost) od i-2 do i-1 drugacno, kot
    # je od i-1 do i, potem je i-1 ekstrem. To pomeni, da imamo en
    # nov dolgocasen sprehod in na novo nastavimo zadnji znani ekstrem.
    if (a[i-1] > a[i]) != (a[i-2] > a[i-1]):
        dolgocasna += 1
        e = i - 1
    # V i se koncajo monotona zaporedja, ki se zacnejo na
    # intervalu [e, i], ki pa ima i - e + 1 elementov.
    monotona += i - e + 1
# Na koncu pristejemo se dolgocasno zaporedje,
# ki se konca na koncu seznama.
print(dolgocasna+1, monotona, sep='\n')
```

Pri tej nalogi je 13 točk vredno smiselno preštevanje dolgočasnih sprehodov, 12 točk pa smiselno preštevanje monotonih sprehodov. Ocenjujemo tudi hitrost, glede na to, kako učinkovito tekmovalčeva rešitev računa število sprehodov. Rešitev, ki monotone sprehode prešteje tako, da iz vsakega vrha poišče levi (in ali desni) ekstrem, naj dobi za ta del največ polovico točk. Vse točke naj dobi le rešitev, ki se največ nekajkrat zapelje čez vse vrhove (ne dela pa poizvedb »za vsako točko do ekstrema«) ali česa podobnega (rečemo, da je $\mathcal{O}(n)$; torej taka, ki bi za dvakrat daljši zemljevid porabila le približno dvakrat več časa).

3. osnovnošolsko tekmovanje ACM v znanju računalništva

Šolsko tekmovanje

23. januar 2026

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). Čas reševanja je omejen na 120 minut.

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 25 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.

Če oblika vhodnih podatkov ni natančno določena, si lahko podrobnosti tekmovalec izbere sam. Na primer, če naloga pravi, da dobimo seznam parov, je to lahko v praksi tabela (*array*), vektor, *linked list* ali še kaj drugega, pari pa so lahko bodisi strukture, ki jih je deklarirala tekmovalčeva rešitev, ali pa kaj iz standardne knjižnice (kot je `pair` v C++ ali `tuple` v pythonu).

Težavnost nalog

Državno tekmovanje ACM v znanju računalništva poteka v dveh težavnostnih skupinah (prva je lažja, druga pa težja); 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. A4 trak	lahka v prvi skupini
2. Toplotna regulacija	srednje težka v prvi ali lahka v drugi
3. Pretvorba poti	srednja v prvi ali lahka v drugi skupini
4. Hribarjenje	težja v drugi skupini

Če torej na primer neki 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 se državnega tekmovanja udeleži v prvi skupini.

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.