

# CEOI 2026



Dan 1  
(Slovenščina)

Day 1  
(Slovenian)



# Opazovalci ptic

*Omejitev časa: 10 s    Omejitev pomnilnika: 512 MiB*

Društvo opazovalcev ptic z otočja San Serriffe ima nenavadno nabuhlo in stalno spreminjajočo se notranjo organizacijo. Društvo je sestavljeno iz  $n$  odsekov, vsak član društva pa pripada natanko enemu odseku. Odseki so oštevilčeni od 1 do  $n$ ;  $i$ -ti odsek ima  $m_i$  članov. Društvo ima tako vsega skupaj  $M = m_1 + m_2 + \dots + m_n$  članov.

Vsak odsek vodi eden od njegovih članov, ki se v tej vlogi imenuje *načelnik* odseka. Načelniki so oštevilčeni enako kot odseki, tako da je (za vsak  $i = 1, \dots, n$ ) načelnik  $i$  tisti, ki vodi odsek  $i$ .

Poleg tega so načelniki organizirani hierarhično preko sistema mentorstev: vsak načelnik razen enega ima *mentorja*, ki je načelnik nekega drugega odseka. Edini načelnik, ki nima mentorja, je predsednik društva. Če je načelnik  $a$  mentor načelnika  $b$ , pravimo tudi, da je načelnik  $b$  *varovanec* načelnika  $a$ . Noben načelnik ni posredno ali neposredno mentor samemu sebi, tako da, če sledimo zaporedju od načelnika do njegovega mentorja, mentorjevega mentorja in tako dalje, sčasoma vedno dosežemo predsednika društva.

*Vpliv* posameznega načelnika je definiran kot vsota števila članov odseka, ki ga ta načelnik vodi, ter vplivov njegovih varovancev (če jih kaj ima). Hitro se vidi, da je načelnik z najvišjim vplivom ravno predsednik društva, čigar vpliv je vedno enak  $M$ . Načelnikom, ki imajo vpliv  $\geq M/2$ , rečemo *višji načelniki*.

Pravilnik društva določa, da tisti višji načelnik, ki ima med vsemi višjimi načelniki najnižji vpliv, opravlja tudi vlogo *blagajnika* društva.

Občasno lahko kak načelnik (tak, ki ni predsednik društva) *zamenja mentorja*, tako da po tem postane varovanec nekega drugega mentorja kot prej (seveda ob pogoju, da novi mentor ni eden od njegovih varovancev, varovancev njegovih varovancev itd.). Zaradi teh sprememb se lahko spremeni tudi vpliv nekaterih načelnikov in mogoče je, da dobi vlogo blagajnika nekdo drug kot prej.

## Naloga

Napiši program, ki prebere opis začetnega stanja društva ter zaporedje zamenjav mentorjev. Tvoj program naj izpiše, kdo je blagajnik v začetnem stanju društva in po vsaki zamenjavi mentorja.

## Vhodni podatki

V prvi vrstici sta dve celi števili,  $n$  in  $q$ , ločeni s presledkom;  $n$  je število odsekov,  $q$  pa število zamenjav mentorjev.

Naslednjih  $n$  vrstic opisuje začetno stanje društva;  $i$ -ta od teh vrstic vsebuje dve celi števili,  $s_i$  in  $m_i$ , ločeni s presledkom;  $s_i$  je mentor načelnika  $i$  (torej načelnika, ki vodi  $i$ -ti odsek),  $m_i$  pa je število članov  $i$ -tega odseka. Vrednost  $s_i = 0$  pomeni, da je načelnik  $i$  predsednik društva in torej nima mentorja.

Preostalih  $q$  vrstic opisuje zamenjave mentorjev;  $j$ -ta od teh vrstic vsebuje dve celi števili,  $\hat{x}_j$  in  $\hat{z}_j$ , ločeni s presledkom. Pomen teh števil je naslednji: označimo s  $t_j$  (za  $j = 0, \dots, q$ ) blagajnika po prvih  $j$  zamenjavah mentorjev (tako je  $t_0$  začetni blagajnik pred prvo zamenjavo mentorja); potem  $j$ -ta zamenjava mentorja sestoji iz tega, da načelnik  $z_j$  postane novi mentor načelnika  $x_j$ , pri čemer je  $x_j = 1 + ((t_{j-1} + \hat{x}_j) \bmod n)$  in  $z_j =$

$1 + ((t_{j-1} + \hat{z}_j) \bmod n)$ . Namen takšne predstavitve vrednosti  $x_j$  in  $z_j$  je prisiliti tvoj program, naj obdeluje zamenjave mentorjev v takem vrstnem redu, v kakršnem se zgodijo.

Vse spremembe mentorjev v vhodnih podatkih bodo veljavne, torej  $z_j$  ne bo niti enak  $x_j$  niti ne bo varovanec načelnika  $x_j$  niti varovanec kakšnega varovanca načelnika  $x_j$  itd. Lahko pa se zgodi, da je  $z_j$  mentor načelnika  $x_j$  že pred  $j$ -to spremembo (torej se v resnici takrat sploh nič ne spremeni).

Opozorilo: če tvoj program nekoč izračuna napačen rezultat  $t_j$ , bo narobe dekodiral tudi kasnejše vhodne podatke  $\hat{x}_{j+1}, \hat{z}_{j+1}$  itd., zato se lahko zgodi, da bodo ti narobe dekodirani vhodni podatki neveljavni (npr. da bo pomotoma dobil kot  $z_{j+1}$  takega načelnika, ki je varovanec načelnika  $x_{j+1}$ ) in da se bo tvoj program zato sesul in od ocenjevalnega strežnika dobil oceno RTE (*runtime error*, napaka med izvajanjem) namesto WA (*wrong answer*, napačen odgovor).

### Omejitve vhodnih podatkov

- $1 \leq n \leq 1\,000\,000$
- $1 \leq q \leq 30\,000$
- $1 \leq m_i$  za vsak  $i = 1, \dots, n$
- $m_1 + m_2 + \dots + m_n \leq 10^9$
- $1 \leq \hat{x}_j \leq n$  in  $1 \leq \hat{z}_j \leq n$  za vsak  $j = 1, \dots, q$ .

### Podnaloge

1. podnaloga (15 točk):  $n \leq 100$
2. podnaloga (10 točk):  $n \leq 1000$
3. podnaloga (50 točk):  $n \leq 300\,000$
4. podnaloga (25 točk): brez dodatnih omejitev.

### Izhodni podatki

Izpiši števila  $t_0, t_1, \dots, t_q$ , vsako v svojo vrstico, pri čemer je  $t_j$  blagajnik po prvih  $j$  zamenjavah mentorjev. Vsak  $t_j$  mora biti seveda celo število z območja  $1 \leq t_j \leq n$ .

### Primer

Vhod	Izhod
7 2	2
0 1	2
1 3	3
1 3	
2 3	
2 1	
5 2	
5 1	
3 7	
2 7	

### Komentar

Na začetku je načelnik 2 blagajnik (torej  $t_0 = 2$ ). Pri prvi menjavi mentorstva preberemo  $\hat{x}_1 = 3$  in  $\hat{z}_1 = 7$  in izračunamo  $x_1 = 1 + ((2+3) \bmod 7) = 6$  in  $z_1 = 1 + ((2+7) \bmod 7) = 3$ . Torej načelnik 3 postane nov mentor načelnika 6, načelnik 2 je še vedno blagajnik (zato  $t_1 = 2$ ). Po drugi menjavi mentorstva preberemo  $\hat{x}_2 = 2$  in  $\hat{z}_2 = 7$  in izračunamo  $x_2 = 1 + ((2+2) \bmod 7) = 5$  in  $z_2 = 1 + ((2+7) \bmod 7) = 3$ . Torej načelnik 3 postane nov mentor načelnika 5 in tako tudi nov blagajnik (zato  $t_2 = 3$ ).



## Iskanje v globino

Omejitev časa: 1 s    Omejitev pomnilnika: 256 MiB

Morda že poznaš slavni algoritem DFS (*depth-first search*, iskanje v globino) za preiskovanje grafov. Pri tej nalogi se bomo ukvarjali le z neusmerjenimi povezanimi preprostimi grafi (torej brez zank in vzporednih povezav). Vozlišča grafa so oštevilčena kot  $0, 1, \dots, n - 1$ . Algoritem DFS bo izpisal vozlišča in njihove globine tako:

```
DFS(d, v):
  izpiši d/v
  označi vozlišče v kot obiskano
  W = seznam v-jevih sosedov, urejenih naraščajoče po številkah
  za vsako w iz seznama W:
    če vozlišče w še ni bilo obiskano:
      DFS(d + 1, w)
```

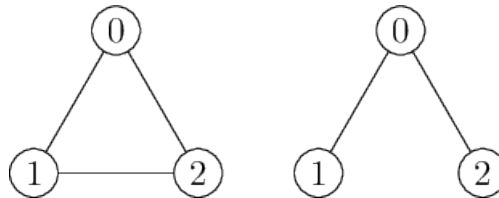
Napiši program, ki izpiše število različnih grafov, pri katerih klic  $\text{DFS}(0, n - 1)$  izpiše enak izpis kot tisti, ki je podan v vhodnih podatkih. Na primer, izpis

0/2

1/0

2/1

nastane, če pokličemo  $\text{DFS}(0, 2)$  na kateremkoli od naslednjih dveh povezanih neusmerjenih preprostih grafov na treh vozliščih:



### Vhod

Na vhodu je izpis, ki nastane pri klicu  $\text{DFS}(0, n - 1)$  na nekem neznanem povezanem neusmerjenem preprostem grafu z  $n$  vozlišči. Vhod je torej sestavljen iz  $n$  vrstic oblike  $d/v$ , pri čemer je v prvi vrstici  $0/n-1$ .

### Omejitve

- $1 \leq n \leq 2 \cdot 10^5$

### Podnaloge

- 1. podnaloga (10 točk):  $n \leq 6$ .
- 2. podnaloga (20 točk):  $n \leq 500$ .
- 3. podnaloga (20 točk):  $n \leq 10^4$ .

- 4. podnaloga (10 točk): za vsak  $i \in \{2, \dots, n\}$  je  $i$ -ta vhodna vrstica  $i-1/i-2$ .
- 5. podnaloga (20 točk): za vsak  $i \in \{2, \dots, n\}$  je  $i$ -ta vhodna vrstica oblike  $i-1/v$  za neki  $v \in \{0, \dots, n-2\}$ .
- 6. podnaloga 6 (20 točk): brez dodatnih omejitev.

## Izhod

Izpiši število različnih grafov z dano lastnostjo. Ker je to število lahko zelo veliko, izpiši le njegov ostanek pri deljenju z 1 000 000 007.

## Primer

Vhod	Izhod
0/2	2
1/0	
2/1	

# Lov na zaklade

Omejitev časa: 8 s      Omejitev pomnilnika: 256 MiB

S pomočjo čarobnega kompasa iščeš davno izgubljene zaklade na veliki mreži  $N \times N$  celic. Na  $K \leq 3$  različnih celicah mreže so skrite skrinje z zakladi. Tvoj cilj je preprost: najti vse zaklade!

Ko položiš kompas na eno od celic, bo našel najkrajšo tako pot do skrinje z zakladom, ki uporablja le premike levo, desno, gor in dol (z drugimi besedami, do zaklada, ki je najbližji po manhattanski razdalji). Kompas bo nato pokazal smer prvega koraka. Če po najkrajši poti do najbližjega zaklada (ali različnih najbližjih zakladov, če jih je več na enaki oddaljenosti) vodi več različnih prvih korakov, bo kompas vrnil vse te korake. Če pa je na celici skrinja z zakladom, bo kompas to tudi pokazal (namesto smeri prvega koraka).

Ko najdeš skrinjo z zakladom, jo izprazniš, skrinje pa ne moreš odstraniti – pretežka je. Tvoj kompas ne ve, ali je skrinja polna ali prazna; vedno bo kazal proti najbližji skrinji, naj bo prazna ali polna.

Poskusi najti položaje vseh skrinj z zakladi in pri tem uporabiti čim manj poizvedb s kompasom.

## Naloga

To je interaktivna naloga. Pri vsakem testnem primeru (torej vsakem izvajanju tvojega programa) bo moral tvoj program rešiti več lovov na zaklade. Z ocenjevalnim sistemom se boš sporazumeval s pomočjo knjižnice, ki so jo pripravili organizatorji tekmovanja. Ta knjižnica vsebuje naslednje deklaracije:

- **void** *NextHunt*(**int** &*N*, **int** &*K*) — to funkcijo pokliči, da začneš naslednji lov na zaklade. V spremenljivki *N* bo vrnila velikost mreže, v *K* pa število zakladov. Če v trenutnem izvajanju ni ostalo nobenega lova na zaklade več, bo funkcija postavila *N* in *K* na  $-1$ ; v tem primeru moraš nato končati svoj program z izhodno kodo 0. Opozorimo na to, da smeš to funkcijo poklicati tudi, če v trenutnem lovu še nisi našel vseh zakladov (npr. če bi rad rešil nalogo za delne točke).
- **enum** { *TREASURE* = 0, *DIR\_RIGHT* = 1, *DIR\_UP* = 2, *DIR\_LEFT* = 4, *DIR\_DOWN* = 8 }; — to so konstante, ki se jih uporablja pri vrednostih, ki jih vrača *Query* (glej spodaj).
- **int** *Query*(**int** *x*, **int** *y*) — če vsebuje celica na koordinatah (*x*, *y*) skrinjo z zakladom, bo ta funkcija vrnila *TREASURE*. Drugače pa vrne vsoto ene ali več izmed konstant *DIR\_RIGHT*, *DIR\_UP*, *DIR\_LEFT*, and *DIR\_DOWN*, ki pove, katere smeri pokaže kompas, če ga postavimo na celico (*x*, *y*). (*DIR\_RIGHT* = desno, *DIR\_UP* = gor, *DIR\_LEFT* = levo, *DIR\_DOWN* = dol.) Koordinati *x* in *y* morata biti celi števili od 0 do  $N - 1$ . (Pozor: pri tej nalogi se *y*-koordinate povečujejo od zgoraj navzdol.)

Po tistem, ko *NextHunt* vrne  $N = K = -1$ , tvoj program ne sme več klicati niti funkcije *NextHunt* niti *Query*. Funkcije *Query* tudi ne sme klicati, preden prvič pokliče

*NextHunt*. Če se tvoj program ne drži teh omejitev ali pa če pri kakšnem lovu na zaklade izvede več kot 1000 poizvedb ali pa če pri klicu funkcije *Query* poda neveljavne vrednosti parametrov  $x$  in/ali  $y$ , bo knjižnica prekinila izvajanje tvojega programa in pri trenutnem testnem primeru boš dobil oceno RTE (*run-time error*, napaka med izvajanjem).

Šteje se, da si zaklad našel, če si vsaj enkrat poizvedoval po celici, v kateri ta zaklad leži. Če tvoj program pokliče *NextHunt*, še preden je našel vse zaklade v trenutnem lovu, se to ne šteje za napako, vendar pa se bo poznalo pri točkah (glej spodaj).

Da boš lahko uporabljal knjižnico, vključi v svoj program datoteko `treasurehuntlib.h`:

```
#include "treasurehuntlib.h"
```

To datoteko si lahko preneseš tukaj: `treasurehuntlib.h`.

Na voljo je tudi preprosta implementacija te knjižnice, ki ti bo morda v pomoč pri razvoju tvoje rešitve: `treasurehuntlib-public.cpp`. Da jo prevedeš skupaj s svojim programom, moraš le dodati njeno ime kot parameter pri klicu prevajalnika, npr.:

```
g++ foo.cpp treasurehuntlib-public.cpp
```

če je `foo.cpp` datoteka z izvorno kodo tvoje rešitve.

Implementacija v datoteki `treasurehuntlib-public.cpp` podpira poleg gornjih deklaracij še eno funkcijo: `void InitFromFile(const char *fileName)`, ki prebere seznam lovov na zaklade iz datoteke in ti omogoči, da igraš igro z njimi, namesto da bi generirala svoje naključne love. Več podrobnosti najdeš v datoteki `treasurehuntlib-public.cpp` sami.

Na ocenjevalnem strežniku bomo uporabljali drugačno implementacijo knjižnice, tako da ne delaj nobenih predpostavk glede tega, kako točno deluje implementacija. Smeš pa predpostaviti, da ne smeti po globalnem imenskem prostoru z nobenimi deklaracijami razen zgoraj naštetih (*NextHunt*, *Query* in petero konstant).

Tvoja koda ne sme brati s standardnega vhoda ali pisati na standardni vhod, kajti njiju uporablja naša implementacija knjižnice na ocenjevalnem strežniku za sporazumevanje s preostankom ocenjevalnega okolja.

## Vhod

### Omejitve

- $2 \leq N \leq 10^6$
- $1 \leq K \leq 3$
- V okviru posameznega izvajanja tvojega programa bo največ 100 000 lovov na zaklade.
- V okviru posameznega lova na zaklade lahko izvedeš največ 1000 poizvedb.

### Podnaloge

- 1. podnaloga (10 točk):  $K = 1$ .
- 2. podnaloga (30 točk):  $K = 2$ .
- 3. podnaloga (60 točk):  $K = 3$ .

## Točkovanje

Podnaloga je lahko sestavljena iz več testnih primerov (več izvajanj tvojega programa), posamezni testni primer pa lahko obsega več lovov na zaklade. Za potrebe točkovanja se vse love pri posamezni podnalogi ocenjuje skupaj, ne glede na to, kako so bili razporejeni med testne primere. Za  $i$ -ti lov na zaklade označimo velikost mreže z  $N_i \times N_i$ , število zakladov s  $K_i$ , število poizvedb, ki jih je izvedel tvoj program, s  $Q_i$ , in število zakladov, ki jih je našel tvoj program, s  $F_i$ . Poleg tega naj bo  $S$  skupno število točk, ki so na voljo pri tej podnalogi. Potem dobi tvoj program pri tej podnalogi naslednje število točk:

- Če je tvoj program vedno našel vse zaklade (torej če je  $F_i = K_i$  pri vseh  $i$ ), je njegova ocena odvisna od  $t_i = \frac{Q_i}{\lceil \log_2 N_i \rceil}$ :

$$\frac{S}{2} + \frac{S}{2} \cdot \min_i f(t_i) \text{ točk, kjer je } f(t_i) = \begin{cases} 1, & t_i \leq 11 \\ 1 - (t_i - 11)/9, & 11 \leq t_i \leq 20 \\ 0, & t \geq 20. \end{cases}$$

- Če pa tvoj program ni vedno našel vseh zakladov (torej če obstaja tak  $i$ , pri katerem je  $F_i < K_i$ ), dobi

$$\frac{S}{2} \cdot \min_i \frac{F_i}{K_i} \text{ točk.}$$

Z drugimi besedami, polovico točk dobiš, če najdeš vse zaklade, drugo polovico pa, če jih najdeš s čim manj poizvedbami. Za vse točke mora tvoja rešitev najti vse zaklade s kvečjemu  $11 \lceil \log_2 N_i \rceil$  poizvedbami. Od  $11 \lceil \log_2 N_i \rceil$  do  $20 \lceil \log_2 N_i \rceil$  poizvedb se število točk zmanjšuje linearno; če pa tvoja rešitev izvede več kot  $20 \lceil \log_2 N_i \rceil$  poizvedb, bo dobila le prvo polovico točk (če najde vse zaklade). (Simbola  $\lceil \cdot \rceil$  pomenita, da se value  $\log_2 N_i$  zaokroži navzgor na najbližje celo število.)

Če gornje formule izračunajo za podnalogo ne-celo število točk, ga bomo zaokrožili na najbližje celo število.

Če se tvoj program sesuje zaradi napake med izvajanjem ali pa se ne drži zgoraj opisanih pravil glede uporabe knjižnice, bo za celotno podnalogo dobil 0 točk. Če torej hočeš dobiti delne točke, ker si našel neko podmnožico vseh zakladov, mora tvoj program lepo končati lov s klicem funkcije *NextHunt*.

## Primer

Klic	Vrnjena vrednost
<code>NextHunt(N, K)</code>	$N = 4, K = 1$
<code>Query(2, 0)</code>	$DIR\_DOWN + DIR\_RIGHT = 9$
<code>Query(3, 1)</code>	$DIR\_DOWN$
<code>Query(3, 2)</code>	$TREASURE$
<code>NextHunt(N, K)</code>	$N = -1, K = -1$