

# CEOI 2026



1. deň  
(slovensky)

Day 1  
(Slovak)



## Pozorovatelia vtákov

Časový limit: 10 s      Limit pamäte: 512 MiB

Klub Slávičích Pozorovateľov má pozoruhodne rozbujnenú a stále sa meniacu vnútornú štruktúru. Klub pozostáva z  $n$  oddielov a každý člen klubu patrí presne do jedného oddielu. Oddiely sú očíslované od 1 do  $n$  a  $i$ -ty oddiel má  $m_i$  členov. Klub teda má celkovo  $M = m_1 + m_2 + \dots + m_n$  členov.

Každý oddiel vedie jeden z jeho členov, ktorý sa v tejto úlohe nazýva *vedúci* oddielu. Vedúci sú číslonani rovnako ako oddiely, takže (pre každé  $i = 1, \dots, n$ ) vedúci  $i$  vedie oddiel  $i$ .

Okrem toho sú vedúci hierarchicky organizovaní prostredníctvom systému mentorstva: každý vedúci okrem jedného má *mentora*, ktorým je vedúci nejakého iného oddielu. Jedným vedúcim bez mentora je *Prezident* Klubu. Ak je vedúci  $a$  mentorom vedúceho  $b$ , hovoríme tiež, že vedúci  $b$  je *žiakom* vedúceho  $a$ .

Žiadny vedúci nie je priamo ani nepriamo mentorom samého seba; preto ak sledujeme postupnosť od vedúceho k jeho mentorovi, potom k mentorovi jeho mentora a tak ďalej, vždy nakoniec dospejeme k Prezidentovi.

Definujeme *vplyv* vedúceho ako súčet počtu členov v jeho oddiele a vplyvov všetkých jeho žiakov (ak nejakých má). Ľahko možno vidieť, že vedúcim s najväčším vplyvom je Prezident, ktorého vplyv je vždy rovný  $M$ . Vedúceho voláme *papaláš*, ak je jeho vplyv  $\geq M/2$ .

Stanovy Klubu určujú, že *papaláš* s najmenším vplyvom (spomedzi všetkých *papalášov*) bude vykonávať funkciu *Pokladníka* Klubu.

Z času na čas môže vedúci (iný než Prezident) *zmeniť svoju príslušnosť*, takže sa odteraz stane žiakom iného mentora než predtým (za predpokladu, že jeho nový mentor nie je jedným z jeho žiakov, žiakom niektorého z jeho žiakov a podobne). Z tohto dôvodu sa môže stať, že sa vplyv niektorých vedúcich zmení a úloha *Pokladníka* pripadne inému vedúcemu než predtým.

### Úloha

Napíšte program, ktorý načíta opis počiatočného stavu Klubu a postupnosť zmien príslušnosti. Váš program musí vypísať, kto je *Pokladníkom* v počiatočnom stave Klubu, ako aj po každej zmene príslušnosti.

### Vstup

Prvý riadok obsahuje dve celé čísla  $n$  a  $q$ , oddelené medzerou;  $n$  je počet oddielov a  $q$  je počet zmien príslušnosti.

Nasledujúcich  $n$  riadkov opisuje počiatočný stav Klubu.

$i$ -ty z týchto riadkov obsahuje dve celé čísla  $s_i$  a  $m_i$ , oddelené medzerou;  $s_i$  je mentor vedúceho  $i$  (t. j. vedúceho, ktorý vedie oddiel  $i$ ), zatiaľ čo  $m_i$  je počet členov oddielu  $i$ . Hodnota  $s_i = 0$  znamená, že vedúci  $i$  je *Prezidentom* Klubu, a preto nemá mentora.

Zvyšných  $q$  riadkov opisuje zmeny príslušnosti.

$j$ -ty z týchto riadkov obsahuje dve celé čísla  $\hat{x}_j$  a  $\hat{z}_j$ , oddelené medzerou. Význam týchto čísel je nasledovný.

Označme  $t_j$  (pre  $j = 0, \dots, q$ ) Pokladníka po prvých  $j$  zmenách príslušnosti (teda  $t_0$  je počiatočný Pokladník pred prvou zmenou príslušnosti). Potom  $j$ -ta zmena príslušnosti spočíva v tom, že vedúci  $z_j$  sa stane novým mentorom vedúceho  $x_j$ , kde  $x_j = 1 + ((t_{j-1} + \hat{x}_j) \bmod n)$  a  $z_j = 1 + ((t_{j-1} + \hat{z}_j) \bmod n)$ .

Účelom tejto reprezentácie hodnôt  $x_j$  a  $z_j$  je prinútiť váš program spracovávať zmeny príslušnosti v poradí, v akom sa objavujú.

Zmeny príslušnosti vo vstupných údajoch budú vždy platné, t. j.  $z_j$  nebude rovné  $x_j$ , ani nebude žiakom  $x_j$ , žiakom žiaka  $x_j$  a podobne. Je však možné, že  $z_j$  už bolo mentorom  $x_j$  bezprostredne pred  $j$ -tou zmenou (takže sa v danom okamihu v skutočnosti nič nezmení).

Všimnite si, že ak váš program v nejakom okamihu vypočíta nesprávny výsledok  $t_j$ , bude nesprávne dekodovať aj nasledujúce vstupy  $\hat{x}_{j+1}, \hat{z}_{j+1}$  atď. a môže skončiť s verdiktom RTE (chyba počas vykonávania) namiesto WA (nesprávna odpoveď), pretože nesprávne dekodované vstupy môžu byť neplatné (napríklad môže nesprávne získať  $z_{j+1}$ , ktoré je žiakom  $x_{j+1}$ ).

### Obmedzenia

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

### Podúlohy

- Podúloha 1 (15 bodov):  $n \leq 100$
- Podúloha 2 (10 bodov):  $n \leq 1000$
- Podúloha 3 (50 bodov):  $n \leq 300\,000$
- Podúloha 4 (25 bodov): Bez ďalších obmedzení.

### Výstup

Vypíšte čísla  $t_0, t_1, \dots, t_q$ , každé na samostatný riadok, kde  $t_j$  je Pokladník po prvých  $j$  zmenách príslušnosti.

Prirodzene, každé  $t_j$  musí byť celé číslo z rozsahu  $1 \leq t_j \leq n$ .

### Príklad

Vstup	Výstup
7 2	2
0 1	2
1 3	3
1 3	
2 3	
2 1	
5 2	
5 1	
3 7	
2 7	

### Komentár

Spočiatku je Pokladníkom vedúci 2 ( $t_0 = 2$ ). Pri prvej zmene príslušnosti, dostaneme na vstupe  $\hat{x}_1 = 3$  a  $\hat{z}_1 = 7$ , a vypočítame  $x_1 = 1 + ((2 + 3) \bmod 7) = 6$  a  $z_1 = 1 + ((2 + 7) \bmod 7) = 3$ . Teda vedúci 3 sa stane novým mentorom vedúceho 6; vedúci 2 zostáva Pokladníkom ( $t_1 = 2$ ). Pri druhej zmene príslušnosti dostaneme na vstupe  $\hat{x}_2 = 2$  a  $\hat{z}_2 = 7$  a vypočítame  $x_2 = 1 + ((2 + 2) \bmod 7) = 5$ . Vedúci 3 sa stane novým mentorom vedúceho 5 a zároveň sa stane novým Pokladníkom ( $t_2 = 3$ ).



## DFS

Časový limit: 1 s    Limit pamäte: 256 MiB

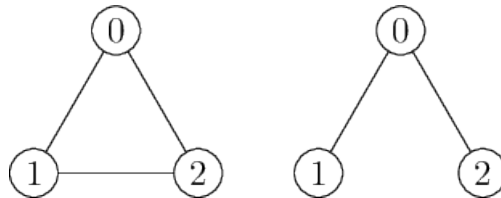
Možno už poznáte známy algoritmus DFS (depth-first search, prehľadávanie do hĺbky) na prechádzanie grafu. V tejto úlohe budeme uvažovať iba súvislé neorientované jednoduché grafy (bez slučiek a násobných hrán) s vrcholmi očíslovanými  $0, 1, \dots, n - 1$ , pričom algoritmus DFS bude vypisovať hĺbky a vrcholy nasledovne:

```
DFS(d, v):
    vypíš d/v
    označ vrchol v ako navštívený
    W = zoznam susedov vrcholu v usporiadaný vzostupne podľa čísiel
    pre každý vrchol w v W:
        ak vrchol w ešte nebol navštívený:
            DFS(d + 1, w)
```

Napíšte program, ktorý vypočíta počet rôznych grafov, pre ktoré volanie  $\text{DFS}(0, n - 1)$  vytvorí rovnaký výpis ako ten, ktorý je zadaný na vstupe. Napríklad výpis

```
0/2
1/0
2/1
```

vznikne volaním  $\text{DFS}(0, 2)$  na ktoromkoľvek z nasledujúcich dvoch súvislých neorientovaných jednoduchých grafov s 3 vrcholmi:



### Vstup

Vstupom je výstup volania  $\text{DFS}(0, n - 1)$  na neznámom súvislom neorientovanom jednoduchom grafe s  $n$  vrcholmi. Vstup teda pozostáva z  $n$  riadkov vo formáte  $d/v$ , pričom prvý riadok je  $0/n-1$ .

### Obmedzenia

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

### Podúlohy

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

- Podúloha 4 (10 bodov): Pre každé  $i \in \{2, \dots, n\}$  je  $i$ -ty vstupný riadok  $i-1/i-2$ .
- Podúloha 5 (20 bodov): Pre každé  $i \in \{2, \dots, n\}$  je  $i$ -ty vstupný riadok  $i-1/v$  pre nejaké  $v \in \{0, \dots, n-2\}$ .
- Podúloha 6 (20 bodov): Bez ďalších obmedzení.

## Výstup

Vypíšte počet rôznych grafov s požadovanou vlastnosťou. Keďže tento počet môže byť veľmi veľký, vypíšte výsledok modulo 1 000 000 007.

## Príklad

### Vstup

0/2  
1/0  
2/1

### Výstup

2

# Lovci pokladov

Časový limit: 8 s      Limit pamäte: 256 MiB

Hľadáš dávno stratený poklad na obrovskom poli buniek s rozmermi  $N \times N$  pomocou čarovného kompasu. Na navzájom rôznych bunkách poľa je ukrytých dokopy  $K \leq 3$  truhlíc s pokladom. Tvoj cieľ je jednoduchý: všetky ich nájdi!

Keď položíš kompas na jednu z buniek, nájde najkratšiu cestu k truhlici s pokladom pomocou pohybov doľava, hore, doprava a dole (t. j. k najbližšej truhlici podľa Manhattanovskej vzdialenosti). Potom ukáže smer prvého kroku. Ak existujú viaceré smery, ktorými môže začínať nejaká najkratšia cesta k niektorej najbližšej truhlici, kompas vráti všetky tieto smery. Ak bunka obsahuje truhlicu, kompas to oznámi namiesto smeru.

Vždy keď nájdeš truhlicu s pokladom, vyrabuješ jej obsah, ale nemôžeš ju odstrániť - je príliš ťažká. Tvoj kompas nevie, či je truhlica plná alebo prázdna. Bude ukazovať smer k najbližšej truhlici, či už je prázdna alebo plná.

Pokús sa nájsť polohu všetkých truhlíc s pokladom na čo najmenší počet použítí kompasu.

## Úloha

Toto je interaktívna úloha. V každom testovacom prípade (t.j. v každom spustení tvojho programu) bude musieť tvoj program vyriešiť niekoľko samostatných dobrodružných výprav. S hodnotiacim systémom by si mal komunikovať pomocou knižnice poskytnutej organizátormi. Táto knižnica obsahuje nasledujúce deklarácie:

- **void** *NextHunt*(**int** &*N*, **int** &*K*) — zavolaním tejto funkcie začneš ďalšiu výpravu za pokladom. Nastaví premennú *N* na veľkosť mriežky a *K* na počet truhlíc. Ak v aktuálnom behu programu už nie je ďalšia výprava, čo riešiť, funkcia nastaví *N* a *K* na  $-1$ ; v takom prípade musíš ukončiť program s návratovým kódom 0. Všimni si, že túto funkciu môžeš zavolať aj bez toho že by si v aktuálnej výprave našiel všetky truhlice, napr. ak sa snažíš získať iba čiastočné body.
- **enum** { *TREASURE* = 0, *DIR\_RIGHT* = 1, *DIR\_UP* = 2, *DIR\_LEFT* = 4, *DIR\_DOWN* = 8 }; — ide o konštanty používané v návratových hodnotách funkcie *Query* (pozri nižšie).
- **int** *Query*(**int** *x*, **int** *y*) — ak bunka na súradniciach (*x*, *y*) obsahuje truhlicu, táto funkcia vráti *TREASURE*. Inak vráti súčet jednej alebo viacerých konštant *DIR\_RIGHT*, *DIR\_UP*, *DIR\_LEFT* a *DIR\_DOWN*, ktoré určujú, aké pohyby kompas ukáže pri umiestnení do bunky (*x*, *y*). Súradnice *x* a *y* musia byť celé čísla od 0 do  $N - 1$ . (Poznámka: v tejto úlohe súradnice *y* stúpajú zhora nadol.)

Po tom, čo *NextHunt* vráti  $N = K = -1$ , tvoj program už nesmie volať ani *NextHunt*, ani *Query*. Taktiež nesmie volať *Query* pred prvým volaním *NextHunt*. Ak tvoj program tieto obmedzenia nedodrží, alebo ak vykoná viac než 1000 dotazov počas jednej výpravy, alebo ak pri volaní *Query* použije hodnoty *x* a/alebo *y* mimo povoleného rozsahu, knižnica ukončí tvoj program a vráti verdikt run-time-error (RTE) pre aktuálny testovací prípad.

Truhlica sa považuje za nájdenú, ak si sa aspoň raz pozrel na bunku, ktorá ju obsahuje. Ak tvoj program zavolá *NextHunt* predtým než nájde všetky truhlice v aktuálnej výprave, nepovažuje sa to za chybu, ale ovplyvní to tvoje skóre (viac o hodnotení nižšie).

Na prístup ku knižnici by mal tvoj program includnúť hlavičkový súbor `treasurehuntlib.h`:

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

Tento hlavičkový súbor si môžeš stiahnuť tu: `treasurehuntlib.h`.

Na pomoc pri vývoji riešenia je dostupná jednoduchá implementácia knižnice tu: `treasurehuntlib-public.cpp`. Ak ju chceš skompilovať spolu so svojím programom, jednoducho pridaj jej názov ako parameter kompilátora, napr.:

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

kde `foo.cpp` je názov súboru obsahujúceho tvoje riešenie.

Implementácia v súbore `treasurehuntlib-public.cpp` podporuje okrem vyššie uvedených deklarácií aj ďalšiu funkciu s názvom `void InitFromFile(const char *fileName)`, ktorá načíta zoznam výprav zo súboru a umožní vám hrať hru s nimi namiesto generovania vlastných náhodných hľadání pokladov. Viac podrobností nájdete v samotnom súbore `treasurehuntlib-public.cpp`.

Na hodnotiacom serveri bude použitá iná implementácia knižnice, preto by si nemal nič predpokladať o tom, ako presne funguje. Môžeš však predpokladať, že nešpiní globálny namespace žiadnymi deklaráciami okrem vyššie uvedených (*NextHunt*, *Query* a tých piatich konštánt).

Tvoj program nesmie čítať zo štandardného vstupu ani zapisovať na štandardný výstup, pretože tie budú použité našou implementáciou knižnice na hodnotiacom serveri na komunikáciu so zvyškom hodnotiaceho prostredia.

## Vstup

### Obmedzenia

- $2 \leq N \leq 10^6$
- $1 \leq K \leq 3$
- Počas jedného spustenia programu bude najviac 100 000 dobrodružných výprav.
- Počas jednej výpravy môžeš vykonať najviac 1000 dotazov.
- Hodnotiaci systém nie je adaptívny.

### Podúlohy

- Podúloha 1 (10 bodov)  $K = 1$
- Podúloha 2 (30 bodov)  $K = 2$
- Podúloha 3 (60 bodov)  $K = 3$ .

## Hodnotenie

Podúloha môže pozostávať z viacerých testovacích prípadov (viacerých spustení tvojho programu) a každý testovací prípad môže obsahovať viacero výprav. Na účely hodnotenia sa všetky výpravy v rámci danej podúlohy vyhodnocujú spoločne bez ohľadu na to, ako boli pôvodne rozdelené medzi testovacie prípady. Pre  $i$ -tu výpravu označme veľkosť mriežky ako  $N_i \times N_i$ , počet truhlíc ako  $K_i$ , počet dotazov vykonaných programom ako  $Q_i$  a počet nájdených truhlíc ako  $F_i$ . Ďalej označme  $S$  maximálny počet bodov získateľných za túto podúlohu. Počet bodov, ktoré dostane program, sa vypočíta takto:

- Ak tvoj program vždy našiel všetky truhlice (t. j. ak  $F_i = K_i$  pre všetky  $i$ ), jeho skóre závisí od  $t_i = \frac{Q_i}{\lceil \log_2 N_i \rceil}$ :

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

- Ak tvoj program nenašiel vždy všetky truhlice (t. j. existuje  $i$  také, že  $F_i < K_i$ ), dostane

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

Inými slovami, polovicu bodov získaš za to že si našiel všetky truhlice, a druhú polovicu za to že ti stačilo čo najmenej dotazov. Na získanie plného počtu bodov by malo tvoje riešenie nájsť všetky truhlice za najviac  $11 \lceil \log_2 N_i \rceil$  dotazov. Medzi  $11 \lceil \log_2 N_i \rceil$  a  $20 \lceil \log_2 N_i \rceil$  dotazmi skóre lineárne klesá; a ak tvoje riešenie použije viac než  $20 \lceil \log_2 N_i \rceil$  dotazov, získa iba prvú polovicu bodov za to že našlo všetky truhlice. (Symboly  $\lceil \cdot \rceil$  znamenajú, že hodnota  $\log_2 N_i$  sa zaokrúhľuje nahor na najbližšie celé číslo.)

Ak vyššie uvedené vzorce vypočítajú neceločíselný počet bodov, výsledok sa zaokrúhli na najbližšie celé číslo.

Ak tvoj program skončí s chybou počas behu alebo nedodrží vyššie uvedený protokol používania knižnice, získa 0 bodov za celú podúlohu. Na získanie čiastočných bodov za nájdenie iba podmnožiny truhlíc musí tvoj program preto korektne ukončiť výpravu zavolaním *NextHunt*.

## Príklad

Volanie	Výsledok
<code>NextHunt(N, K)</code>	$N = 4, K = 1$
<code>Query(2, 0)</code>	<code>DIR_DOWN + DIR_RIGHT = 9</code>
<code>Query(3, 1)</code>	<code>DIR_DOWN</code>
<code>Query(3, 2)</code>	<code>TREASURE</code>
<code>NextHunt(N, K)</code>	$N = -1, K = -1$