

CEOI 2026



1. den
(česky)

Day 1
(Czech)

Dřemlík tundrový

Časový limit: 10 s Limit paměti: 512 MiB

Spolek pro pozorování vzácného ptačího druhu dřemlíka tundrového má poněkud komplikovanou a neustále se měnící vnitřní strukturu. Spolek se skládá z n oddílů, přičemž každý člen spadá právě do jedné z nich. Oddíly jsou očíslované od 1 do n a i -tý oddíl má m_i členů. Celý spolek má tak $M = m_1 + m_2 + \dots + m_n$ členů.

Každý oddíl je veden jedním ze svých členů, jehož role se nazývá předseda oddílu. Předsedové jsou očíslováni stejně jako oddíly, takže (pro každé $i = 1, \dots, n$) předseda i je ten, kdo vede oddíl i .

Předsedové jsou navíc hierarchicky uspořádáni prostřednictvím systému koučství: každý předseda kromě jednoho má svého *kouče*, kterým je předseda nějaké jiného oddílu. Jedním předsedou bez kouče je *prezident* spolku. Je-li předseda a koučem předsedy b , říkáme také, že předseda b je *následovníkem* předsedy a . Žádný předseda není přímo ani nepřímo svým vlastním koučem; pokud tedy budeme procházet posloupnost od předsedy k jeho kouči, kouči jeho kouče atd., vždy nakonec dojdeme k prezidentovi.

Definujeme *vliv* předsedy jako součet počtu členů v jeho oddílu a vlivů všech jeho následovníků (pokud nějaké má). Je zřejmé, že předsedou s nejvyšším vlivem je prezident, jehož vliv je vždy roven M . Předseda se nazývá *vysoce postavený*, pokud je jeho vliv $\geq M/2$.

Stanovy spolku určují, že vysoce postavený předseda s nejmenším vlivem (ze všech vysoce postavených předsedů) bude vykonávat funkci *hospodáře* spolku.

Jednou za čas může některý předseda (jiný než prezident) *změnit svou oddanost*. Stane se tak následovníkem jiného kouče než dosud (za předpokladu, že jeho nový kouč není jedním z jeho následovníků, následovníků jeho následovníků atd.). V důsledku toho se může stát, že se vliv některých předsedů změní a role hospodáře případně jinému předsedovi než dosud.

Úloha

Napište program, který načte popis počátečního stavu spolku a posloupnost změn oddanosti. Váš program musí vypsát, kdo je hospodářem v počátečním stavu Spolku a také po každé změně oddanosti.

Vstup

První řádek obsahuje dvě celá čísla n a q , oddělená mezerou; n je počet oddílů a q je počet změn oddanosti.

Následujících n řádků popisuje počáteční stav Spolku. Řádek i obsahuje dvě celá čísla s_i a m_i , oddělená mezerou; s_i je kouč předsedy i (tj. kouč předsedy stojícího v čele oddílu i), zatímco m_i je počet členů oddílu i . Hodnota $s_i = 0$ značí, že předseda i je prezidentem spolku a nemá tedy žádného kouče.

Zbývajících q řádků popisuje změny oddanosti. Řádek j obsahuje dvě celá čísla \hat{x}_j a \hat{z}_j , oddělená mezerou. Význam těchto čísel je následující. Označme t_j (pro $j = 0, \dots, q$) hospodáře po prvních j změnách oddanosti (tedy t_0 je počáteční hospodář před první změnou oddanosti). Potom j -tá změna oddanosti se stane tak, že předseda z_j se stane novým koučem předsedy x_j , kde $x_j = 1 + ((t_{j-1} + \hat{x}_j) \bmod n)$ a $z_j = 1 + ((t_{j-1} + \hat{z}_j) \bmod n)$.

Smyslem této reprezentace hodnot x_j a z_j je vás přinutit, abyste museli změny oddanosti zpracovávat v pořadí, ve kterém se stanou.

Změny oddanosti na vstupu budou vždy platné, tj. z_j nebude rovno x_j a ani nebude následovníkem x_j , následovníkem následovníka x_j atd. Je však možné, že z_j již bylo koučem x_j těsně před j -tou změnou (takže se v daném okamžiku ve skutečnosti nic nezmění).

Pozor na to, že pokud váš program v některém okamžiku vypočte chybný výsledek t_j , bude nesprávně dekodovat i následující vstupy $\hat{x}_{j+1}, \hat{z}_{j+1}$, atd. a může skončit verdiktem RTE (runtime error) namísto WA (wrong answer), protože nesprávně dekodované vstupy mohou být neplatné (například můžete chybně získat z_{j+1} , které je následovníkem x_{j+1}).

Omezení

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

Podúlohy

- Podúloha 1 (15 bodů): $n \leq 100$
- Podúloha 2 (10 bodů): $n \leq 1000$
- Podúloha 3 (50 bodů): $n \leq 300\,000$
- Podúloha 4 (25 bodů): Žádná další omezení.

Výstup

Vypište čísla t_0, t_1, \dots, t_q , každé na samostatný řádek, kde t_j je Hospodář po prvních j změnách příslušnosti. Samozřejmě každé t_j musí být celé číslo z rozsahu $1 \leq t_j \leq n$.

Příklad

Vstup

7 2
0 1
1 3
1 3
2 3
2 1
5 2
5 1
3 7
2 7

Výstup

2
2
3

Comment

Na začátku je Hospodářem předseda 2 (takže $t_0 = 2$). Při první změně oddanosti načteme $\hat{x}_1 = 3$ a $\hat{z}_1 = 7$ a spočteme, že $x_1 = 1 + ((2+3) \bmod 7) = 6$ a $z_1 = 1 + ((2+7) \bmod 7) = 3$; takže funkcionář 3 se stane novým koučem předsedy 6; předseda 2 zůstane hospodářem (takže $t_1 = 2$). Při druhé změně oddanosti načteme $\hat{x}_2 = 2$ a $\hat{z}_2 = 7$ a spočítáme $x_2 = 1 + ((2+2) \bmod 7) = 5$ a $z_2 = 1 + ((2+7) \bmod 7) = 3$; takže předseda 3 se stane novým koučem předsedy 5 a také se stane novým hospodářem (takže $t_2 = 3$).

DFS

Časový limit: 1 s Limit paměti: 256 MiB

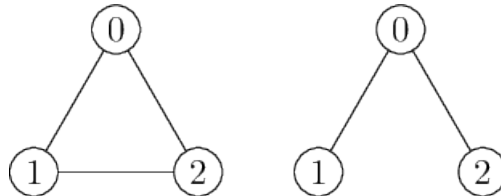
Možná již znáte slavný algoritmus DFS (depth-first search) pro procházení grafu do hloubky. V tomto problému budeme uvažovat pouze souvislé neorientované prosté grafy (bez smyček a násobných hran) s vrcholy očíslovány $0, 1, \dots, n-1$, přičemž algoritmus DFS bude vypisovat hloubky a vrcholy následovně:

```
DFS(d, v):
  vypiš d/v
  označ vrchol v jako navštívený
  W = seznam sousedů vrcholu v seřazený podle rostoucích čísel
  pro každý w z W:
    pokud vrchol w ještě nebyl navštíven:
      DFS(d + 1, w)
```

Napište program, který vypíše počet grafů, pro které volání $\text{DFS}(0, n - 1)$ vytvoří stejný výstup, jaký je zadán na vstupu. Například výstup

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

vznikne voláním $\text{DFS}(0, 2)$ na kterémkoliv z následujících dvou souvislých neorientovaných grafů o 3 vrcholech:



Vstup

Vstup je výstup volání $\text{DFS}(0, n - 1)$ na neznámém souvislém neorientovaném grafu s n vrcholy. Vstup tak tvoří n řádků ve formátu d/v , přičemž první řádek je $0/n-1$.

Omezení

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

Podúlohy

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

- Podúloha 4 (10 bodů): Pro každé $i \in \{2, \dots, n\}$ je i -tý vstupní řádek roven $i-1/i-2$.
- Podúloha 5 (20 bodů): Pro každé $i \in \{2, \dots, n\}$ je i -tý vstupní řádek roven $i-1/v$ pro nějaké $v \in \{0, \dots, n-2\}$.
- Podúloha 6 (20 bodů): Bez dalších omezení.

Výstup

Spočítejte počet různých grafů s požadovanou vlastností. Protože tento počet může být velmi velký, vypište výsledek modulo 1 000 000 007.

Příklad

Vstup

0/2
1/0
2/1

Výstup

2

Lov pokladů

Časový limit: 8 s Limit paměti: 256 MiB

Indiana Jones hledá poklady na dlouho zapomenutém ostrově pomocí magického kompasu. Ostrov si můžeme představit jak obrovskou mřížku $N \times N$ a na $K \leq 3$ jejích políčkách se nachází truhly s pokladem. Váš úkol je jednoduchý: Pomožte Indiana Jonesovi je najít všechny!

Když položíte kompas na jedno z políček, tak nejprve najde nejkratší cesty k truhle pomocí pohybů nahoru, dolů, doleva a doprava (tedy nejbližší truhlici podle Manhattanské vzdálenosti) a poté vám ukáže směr prvního kroku. Pokud existuje více prvních kroků, které vedou k nejbližší truhle (nebo k více nejbližším truhlám), kompas vám ukáže všechny takové kroky. Pokud políčko obsahuje truhlu, tak vám to kompas ukáže místo prvních kroků.

Kdykoliv najdete truhlu, vezmete si z ní poklad, ale truhla zůstane stát tam, kde byla – je příliš těžká. Váš kompas neví, které truhly jsou plné a bude ukazovat k nejbližší truhle, ať už je plná nebo prázdná.

Zkuste najít pozice všech truhel na co nejméně položení kompasu.

Úloha

Tato úloha je interaktivní. Pro každý test case (tedy spuštění vašeho programu) váš program musí provést několik lovů pokladů. S graderem byste měli komunikovat za použití knihovny od organizátorů. Knihovna obsahuje tyto deklarace:

- **void** *NextHunt*(**int** &*N*, **int** &*K*) — Pro začátek dalšího lovu pokladů zavolejte tuto funkci. Do svých argumentů nastaví velikost mřížky *N* a počet pokladů *K*. V případě, že už nezbývají žádné další lovy pokladů, funkce nastaví *N* i *K* na -1 . V tomto případě má váš program hned skončit s návratovým kódem 0. Poznamenejme, že tuto funkci můžete zavolat i před tím, než najdete všechny truhly. (Například pokud se snažíte pouze získat částečné body.)
- **enum** { *TREASURE* = 0, *DIR_RIGHT* = 1, *DIR_UP* = 2, *DIR_LEFT* = 4, *DIR_DOWN* = 8 }; — Jedná se o konstanty používané v návratových hodnotách funkce *Query* (viz níže).
- **int** *Query*(**int** *x*, **int** *y*) — Pokud políčko na pozici (x, y) obsahuje poklad, tato funkce vrátí *TREASURE*. Jinak vrátí součet jedné nebo více konstant *DIR_RIGHT*, *DIR_UP*, *DIR_LEFT*, a *DIR_DOWN* udávající, které kroky ukáže kompas položený na políčku (x, y) . Souřadnice *x* a *y* musí být celá čísla mezi 0 a $N - 1$. (Pozn.: V této úloze se souřadnice *y* zvyšuje zhora dolů.)

Potom, co *NextHunt* nastaví $N = K = -1$, váš program nesmí zavolat *NextHunt* ani *Query*. Dále nesmí zavolat *Query* před prvním zavoláním *NextHunt*. Pokud váš program tato omezení poruší, nebo provede více jak 1000 dotazů během jednoho lovu pokladů, nebo vrátí hodnoty *x* a/nebo *y* mimo rozsah při zavolání *Query*, tak vám knihovna ukončí program a dostanete run-time-error (RTE) verdikt pro tento test case.

Poklad se považuje za nalezený, pokud jste se dotázali na jeho políčko alespoň jednou. Pokud váš program zavolá *NextHunt* před nalezením všech truhel, tak nespadne s běhovou chybou, ale zohlední se to do vašich bodů (o bodování více níže).

Pro použití knihovny by váš program měl includovat hlavičkový soubor `treasurehuntlib.h`:

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

Tento hlavičkový soubor si můžete stáhnout zde: `treasurehuntlib.h`.

Pro zjednodušení psaní vašich řešení si můžete stáhnout jednoduchou implementaci knihovny zde: `treasurehuntlib-public.cpp`. Abyste ji zkompilovali s vaším programem, přidejte její jméno jako parametr překladači, např.:

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

kde `foo.cpp` je jméno souboru obsahujícího vaše řešení.

Implementace v `treasurehuntlib-public.cpp` kromě výše zmíněných deklarací podporuje další funkci `void InitFromFile(const char *fileName)`, která použije seznam lovů pokladů ze souboru místo generování vlastních náhodných lovů pokladů. Více detailů najdete v samotném `treasurehuntlib-public.cpp`.

Na vyhodnocovacím serveru bude použita jiná implementace knihovny, takže byste neměli předpokládat nic o jejím chování. Ale můžete předpokládat, že do globální namespace nepřidává žádné deklarace kromě výše zmíněných (*NextHunt*, *Query* a pět enumových konstant).

Váš kód nesmí číst ze standardního vstupu nebo vypisovat na standardní výstup, protože ty budou použity implementací knihovny na vyhodnocovacím serveru pro komunikaci se zbytkem vyhodnocovacího prostředí.

Vstup

Omezení

- $2 \leq N \leq 10^6$
- $1 \leq K \leq 3$
- Během kteréhokoliv jednoho spuštění vašeho programu bude nejvýše 100 000 lovů pokladů.
- Během kteréhokoliv jednoho lovu pokladů můžete provést nejvýše 1000 dotazů.
- Vyhodnocovací systém není adaptivní.

Podúlohy

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

Bodování

Podúloha se může skládat z několika test casů (spuštění vašeho programu) a každý test case se může skládat z několika lovů pokladů. Pro účely bodování, všechny lovy pokladů v jedné podúloze jsou vyhodnoceny spolu nehledě na to, ve kterých test casech se nacházely. Pro i -tý lov pokladů označme velikost mřížky $N_i \times N_i$, počet pokladů K_i , počet dotazů vašeho programu Q_i a počet vámi nalezených pokladů F_i . Dále označme S jako celkový počet bodů, který lze získat z této podúlohy. Potom počet bodů, které dostanete za tuto podúlohu je:

- Pokud váš program vždy našel všechny truhly (tedy $F_i = K_i$ pro všechna i), váš počet bodů závisí na $t_i = \frac{Q_i}{\lceil \log_2 N_i \rceil}$:

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

- Pokud váš program někdy nenašel všechny truhly (tedy existuje i , takové že $F_i < K_i$), tak dostane

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

Jinak řečeno, jednu polovinu bodů dostanete za nalezení všech truhel a druhou polovinu za jejich nalezení na co nejméně dotazů. Pro plný počet bodů musí vaše řešení najít všechny truhly v nejvýše $11 \lceil \log_2 N_i \rceil$ dotazech. Mezi $11 \lceil \log_2 N_i \rceil$ a $20 \lceil \log_2 N_i \rceil$ dotazy se počet bodů snižuje lineárně. A pokud vaše řešení provede více jak $20 \lceil \log_2 N_i \rceil$ dotazů, dostane pouze první polovinu bodů za nalezení všech truhel. (Symboly $\lceil \cdot \rceil$ znamenají, že hodnota $\log_2 N_i$ se zaokrouhlí nahoru na nejbližší celé číslo.)

Pokud by vám formule výše dala neceločíselný počet bodů za tuto podúlohu, váš počet bodů se zaokrouhlí k nejbližšímu celému číslu.

Pokud váš program dostane runtime error nebo nedodrží výše zmíněný protokol na používání knihovny, dostane 0 bodů za celou podúlohu. Proto, abyste dostali částečné body za nalezení části pokladů, váš program musí skončit zavoláním *NextHunt*.

Příklad

Volání	Návratová hodnota
<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>	$N = -1, K = -1$