

CEOI 2026



1. дан
(српски)

Day 1
(Serbian)

Посматрачи птица

Временско ограничење: 10 s Ограничење меморије: 512 MiB

Друштво посматрача птица Сан Серифа има необично надувану и стално променљиву унутрашњу структуру. Друштво се састоји од n *огранака*, и сваки члан Друштва припада тачно једном огранку. Огранци су нумерисани од 1 до n , а i -ти огранак има m_i чланова. Дакле, Друштво укупно има $M = m_1 + m_2 + \dots + m_n$ чланова.

Сваки огранак води један од његових чланова, који се у тој улози назива *старешина* огранка. Старешине су нумерисане исто као и огранци, тако да је (за свако $i = 1, \dots, n$) старешина i задужен за огранак i .

Поред тога, старешине су хијерархијски организоване кроз систем менторства: сваки старешина осим једног има *ментора*, који је старешина неког другог огранка. Једини старешина без ментора је *Председник* Друштва. Ако је старешина a ментор старешини b , такође кажемо да је старешина b *ученик* старешине a . Ниједан старешина није, директно или индиректно, ментор самом себи; дакле, ако пратимо низ од неког старешине до његовог ментора, менторовог ментора, и тако даље, увек на крају стигнемо до Председника.

Утицај старешине дефинишемо као збир броја чланова у његовом огранку и утицаја свих његових ученика (ако их има). Лако се види да је старешина са највећим утицајем Председник, чији је утицај увек једнак M . Старешина се назива *вишим старешином* ако је његов утицај $\geq M/2$.

Статут Друштва прописује да виши старешина са најмањим утицајем (међу свим вишим старешинама) обавља улогу *Благајника* Друштва.

С времена на време, старешина (који није Председник) може *променити припадност*, тако да од тог тренутка постане ученик другог ментора него раније (под условом да његов нови ментор није један од његових ученика, или ученика његових ученика, итд.). Због тога се може десити да се утицај неких старешина промени и да улога Благајника припадне другом старешини него раније.

Задатак

Написати програм који учитава опис почетног стања Друштва и низ промена припадности. Ваш програм мора да испише ко је Благајник у почетном стању Друштва, као и после сваке промене припадности.

Улаз

Први ред садржи два цела броја, n и q , раздвојена размаком; n је број огранака, а q је број промена припадности.

Наредних n редова описује почетно стање Друштва. i -ти од тих редова садржи два цела броја, s_i и m_i , раздвојена размаком; s_i је ментор старешине i (тј. старешине задуженог за огранак i), док је m_i број чланова огранка i . Вредност $s_i = 0$ означава да је старешина i Председник Друштва и да зато нема ментора.

Преосталих q редова описује промене припадности. j -ти од тих редова садржи два цела броја, \hat{x}_j и \hat{z}_j , раздвојена размаком. Значење ових бројева је следеће. Означимо са t_j (за $j = 0, \dots, q$) Благајника после првих j промена припадности (дакле, t_0 је почетни Благајник пре прве промене припадности). Тада се j -та промена

припадности састоји у томе да старешина z_j постаје нови ментор старешине x_j , где је $x_j = 1 + ((t_{j-1} + \hat{x}_j) \bmod n)$ и $z_j = 1 + ((t_{j-1} + \hat{z}_j) \bmod n)$. Сврха оваквог представљања вредности x_j и z_j јесте да примора ваш програм да обрађује промене припадности редом којим се појављују.

Промене припадности у улазним подацима ће увек бити исправне, тј. z_j неће бити једнак x_j , нити ће z_j бити ученик од x_j , ученик ученика од x_j , итд. Ипак, могуће је да је z_j већ био ментор старешине x_j непосредно пре j -те промене (тако да се у том тренутку заправо ништа не мења).

Имајте у виду да ако ваш програм у неком тренутку израчуна погрешан резултат t_j , онда ће и наредне улазе \hat{x}_{j+1} , \hat{z}_{j+1} , итд. декодирати погрешно, па може добити пресуду RTE (runtime error) уместо WA (wrong answer), зато што погрешно декодирани улази могу бити неисправни (нпр. може погрешно добити неко z_{j+1} које је ученик од x_{j+1}).

Ограничења

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

Подзадаци

- Подзадатак 1 (15 поена): $n \leq 100$
- Подзадатак 2 (10 поена): $n \leq 1000$
- Подзадатак 3 (50 поена): $n \leq 300,000$
- Подзадатак 4 (25 поена): Нема додатних ограничења.

Излаз

Исписати бројеве t_0, t_1, \dots, t_q , сваки у посебном реду, где је t_j Благајник после првих j промена припадности. Наравно, свако t_j мора бити цео број из опсега $1 \leq t_j \leq n$.

Пример

Улаз	Излаз
7 2	2
0 1	2
1 3	3
1 3	
2 3	
2 1	
5 2	
5 1	
3 7	
2 7	

Коментар

На почетку је старешина 2 Благајник (дакле $t_0 = 2$). У првој промени припадности читамо $\hat{x}_1 = 3$ и $\hat{z}_1 = 7$ и рачунамо $x_1 = 1 + ((2 + 3) \bmod 7) = 6$ и $z_1 = 1 + ((2 + 7) \bmod 7) = 3$; дакле, старешина 3 постаје нови ментор старешине 6; старешина 2 остаје Благајник (дакле $t_1 = 2$). У другој промени припадности читамо $\hat{x}_2 = 2$ и $\hat{z}_2 = 7$ и рачунамо $x_2 = 1 + ((2 + 2) \bmod 7) = 5$ и $z_2 = 1 + ((2 + 7) \bmod 7) = 3$; дакле, старешина 3 постаје нови ментор старешине 5 и такође постаје нови Благајник (дакле $t_2 = 3$).

DFS

Временско ограничење: 1 s Ограничење меморије: 256 MiB

Можда сте већ упознати са познатим DFS (depth-first search) алгоритмом за обилазак графа. У овом задатку, разматраћемо само повезане неусмерене просте графове (без дуплих грана и без грана из чвора у исти тај чвор) са теменима нумерисаним $0, 1, \dots, n - 1$, а DFS алгоритам ће исписивати дубине и темена на следећи начин:

DFS(d, v):

испиши d/v

означи теме v као посећено

W = листа суседа темена v поређана по растућим бројевима
за свако w у W :

ако теме w још није посећено:

DFS($d + 1, w$)

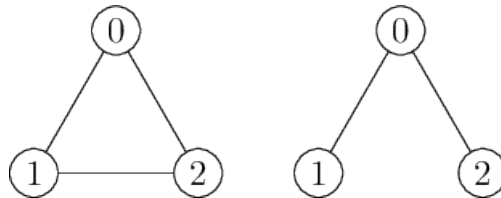
Напишите програм који исписује број различитих графова за које позив DFS($0, n - 1$) производи исти испис као онај дат на улазу. На пример, испис

0/2

1/0

2/1

добија се позивом DFS($0, 2$) на било ком од следећа два повезана неусмерена проста графа са 3 темена:



Улаз

Улаз је испис позива DFS($0, n - 1$) на непознатом повезаном неусмереном простом графу са n темена. Улаз се, дакле, састоји од n линија у формату d/v , при чему је прва линија $0/n-1$.

Ограничења

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

Подзадаци

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

- Подзадатак 4 (10 поена): За свако $i \in \{2, \dots, n\}$, i -та линија улаза је $i-1/i-2$.
- Подзадатак 5 (20 поена): За свако $i \in \{2, \dots, n\}$, i -та линија улаза је $i-1/v$ за неко $v \in \{0, \dots, n-2\}$.
- Подзадатак 6 (20 поена): Нема додатних ограничења.

Излаз

Испишите број различитих графова са траженим својством. Пошто овај број може бити веома велики, испишите резултат по модулу 1 000 000 007.

Пример

Улаз	Излаз
0/2	2
1/0	
2/1	

Потрага за благом

Временско ограничење: 8 s Ограничење меморије: 256 MiB

Тражиш давно изгубљено благо на огромном $N \times N$ пољу хелија помоћу магичног компаса. Укупно постоји $K \leq 3$ сандука са благом сакривених у различитим хелијама поља. Твој циљ је једноставан: пронаћи их све!

Када поставиш компас на једну од хелија, он ће пронаћи најкраће путеве до сандука са благом користећи само потезе лево, горе, десно и доле (тј. до најближег сандука са благом по Менхетн растојању). Затим ће показати смер првог потеза. Ако више исправних првих потеза води до најближег сандука са благом (или до више њих), компас ће вратити све њих. Ако хелија садржи благо, компас ће то назначити.

Кад год пронађеш сандук са благом, испразниш његов садржај, али не можеш да уклониш сандук – превише је тежак. Твој компас не зна да ли је сандук пун или празан. Он ће показивати пут до најближег сандука, празног или пуног.

Покушај да пронађеш локације свих сандука са благом уз што мањи број упита компасу.

Задатак

Ово је интерактиван задатак. У сваком тест примеру (тј. при сваком извршавању твог програма), твој програм ће морати да реши неколико потрага за благом. Са оцењивачем треба да комуницираш користећи библиотеку коју обезбеђују организатори. Ова библиотека садржи следеће декларације:

- **void** *NextHunt*(**int** &*N*, **int** &*K*) — позови ову функцију да започнеш следећу потрагу за благом. Она ће поставити величину мреже у променљиву *N* и број блага у *K*. Ако у текућем извршавању више нема потрага за благом које треба решити, функција ће поставити *N* и *K* на -1 ; у том случају треба да завршиш програм са излазним кодом 0. Имај у виду да ову функцију можеш позвати пре него што пронађеш сва блага у текућој потрази, нпр. ако покушаваш да решиш само за делимичне поене.
- **enum** { *TREASURE* = 0, *DIR_RIGHT* = 1, *DIR_UP* = 2, *DIR_LEFT* = 4, *DIR_DOWN* = 8 }; — ово су константе које се користе у повратним вредностима функције *Query* (види испод).
- **int** *Query*(**int** *x*, **int** *y*) — ако хелија са координатама (*x*, *y*) садржи благо, ова функција враћа *TREASURE*. У супротном враћа збир једне или више константи *DIR_RIGHT*, *DIR_UP*, *DIR_LEFT* и *DIR_DOWN*, означавајући које потезе компас враћа када се постави на хелију (*x*, *y*). Координате *x* и *y* морају бити цели бројеви од 0 до $N - 1$. (Напомена: у овом задатку, *y*-координате расту одозго надоле.)

Након што *NextHunt* постави $N = K = -1$, твој програм не сме поново да позове ни *NextHunt* ни *Query*. Такође не сме да позове *Query* пре првог позива функције *NextHunt*. Ако твој програм не поштује ова ограничења, или ако направи више од

1000 упита у оквиру једне потраге за благом, или ако при позиву функције *Query* проследи вредности x и/или y ван дозвољеног опсега, библиотека ће прекинути твој програм и вратити пресуду `run-time-error` (RTE) за текући тест пример.

Сматра се да је благо пронађено ако си бар једном упитао ћелију која га садржи. Ако твој програм позове *NextHunt* пре него што пронађе сва блага у текућој потрази, то се не сматра грешком, али ће утицати на твој резултат (више о бодовању испод).

Да би приступио библиотеци, твој програм треба да укључи заглавље `treasurehuntlib.h`:

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

Ово заглавље можеш преузети овде: `treasurehuntlib.h`.

Као помоћ при развоју решења, једноставна имплементација библиотеке доступна је овде: `treasurehuntlib-public.cpp`. Да би је компајлирао заједно са својим програмом, само додај њено име као параметар компајлеру, нпр.:

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

ако је `foo.cpp` име фајла који садржи твоје решење.

Имплементација у `treasurehuntlib-public.cpp` подржава, поред горе наведених декларација, још једну функцију звану `void InitFromFile(const char *fileName)`, која чита списак потрага за благом из фајла и омогућава ти да играш игру са њима уместо да генерише сопствене насумичне потраге за благом. Више детаља ћеш наћи у самом фајлу `treasurehuntlib-public.cpp`.

На евалуационом серверу ће се користити друга имплементација библиотеке, тако да не треба да правиш никакве претпоставке о томе како тачно имплементација ради. Можеш, међутим, претпоставити да она не загађује глобални именски простор никаквим декларацијама осим оних које су горе наведене (*NextHunt*, *Query* и пет константи).

Твој код не сме да чита са стандардног улаза нити да пише на стандардни излаз, јер ће их наша имплементација библиотеке на евалуационом серверу користити за комуникацију са остатком евалуационог окружења.

Улаз

Ограничења

- $2 \leq N \leq 10^6$
- $1 \leq K \leq 3$
- У оквиру једног извршавања твог програма биће највише 100 000 потрага за благом.
- У оквиру једне потраге за благом можеш направити највише 1000 упита.
- Систем за оцењивање није адаптиван.

Подзадаци

- Подзadataк 1 (10 поена) $K = 1$
- Подзadataк 2 (30 поена) $K = 2$
- Подзadataк 3 (60 поена) $K = 3$.

Бодовање

Подзадатак се може састојати од више тест примера (више извршавања твог програма), а сваки тест пример се може састојати од више потрага за благом. За потребе бодовања, све потраге за благом датог подзадатка оцењују се заједно, без обзира на то како су првобитно биле распоређене по тест примерима. За i -ту потрагу за благом, означимо величину мреже са $N_i \times N_i$, број блага са K_i , број упита које је направио твој програм са Q_i и број блага које је твој програм пронашао са F_i . Даље, означимо са S укупан број поена доступан за овај подзадатак. Тада је број поена који твој програм добија за овај подзадатак:

- Ако је твој програм увек пронашао сва блага (тј. ако је $F_i = K_i$ за све i), његов резултат зависи од $t_i = \frac{Q_i}{\lceil \log_2 N_i \rceil}$:

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

- Ако твој програм није увек пронашао сва блага (тј. ако постоји i такво да је $F_i < K_i$), добија

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

Другим речима, половину поена добијаш за проналажење свих блага, а другу половину за то да их пронађеш у што мање упита. За максималан број поена, твоје решење треба да пронађе сва блага у највише $11 \lceil \log_2 N_i \rceil$ упита. Између $11 \lceil \log_2 N_i \rceil$ и $20 \lceil \log_2 N_i \rceil$ упита, резултат се линеарно смањује; а ако твоје решење направи више од $20 \lceil \log_2 N_i \rceil$ упита, добиће само прву половину поена за проналажење свих блага. (Симболи $\lceil \cdot \rceil$ значе да се вредност $\log_2 N_i$ заокружује навише на најближи цео број.)

Ако формуле изнад дају нецео број поена за подзадатак, он ће бити заокружен на најближи цео број.

Ако твој програм има грешку у извршавању или се не придржава горе поменутог протокола при коришћењу библиотеке, добиће 0 поена за цео подзадатак. Да би добио делимичне поене за проналажење подскупа блага, твој програм зато мора да заврши претрагу коректно позивом функције *NextHunt*.

Пример

Позив	Повратна вредност
NextHunt(N, K)	$N = 4, K = 1$
Query(2, 0)	$DIR_DOWN + DIR_RIGHT = 9$
Query(3, 1)	DIR_DOWN
Query(3, 2)	$TREASURE$
NextHunt(N, K)	$N = -1, K = -1$