

CEOI 2026



Tag 1
(Deutsch (Ö))

Day 1
(German (A))

Vogelbeobachterinnen

Zeitlimit: 10 s Speicherlimit: 512 MiB

Die Vogelbeobachterinnen-Gesellschaft vom Neusiedler See besitzt eine aufgeblähte, sich stetig wandelnde, interne Struktur. Die Gesellschaft besteht aus n *Ortsgruppen* und jedes Mitglied der Gesellschaft gehört genau einer Ortsgruppe an. Die Ortsgruppen sind von 1 bis n nummeriert und die i -te Ortsgruppe hat m_i Mitglieder. Somit hat die Gesellschaft insgesamt $M = m_1 + m_2 + \dots + m_n$ Mitglieder.

Jede Ortsgruppe wird von einem ihrer Mitglieder geleitet, das in dieser Rolle als *Gruppenleiterin* der Ortsgruppe bezeichnet wird. Die Gruppenleiterinnen sind genauso nummeriert wie die Ortsgruppen, sodass (für jedes $i = 1, \dots, n$) Gruppenleiterin i die Leiterin der Ortsgruppe i ist.

Darüber hinaus sind die Gruppenleiterinnen durch ein Mentorinnen-System hierarchisch organisiert: Jede Gruppenleiterin außer einer hat eine *Mentorin*, welche die Gruppenleiterin einer anderen Ortsgruppe ist. Die einzige Gruppenleiterin ohne Mentorin ist die *Präsidentin* der Gesellschaft. Wenn Gruppenleiterin a die Mentorin von Gruppenleiterin b ist, so sagen wir auch, dass Gruppenleiterin b eine *Schülerin* von Gruppenleiterin a ist.

Keine Gruppenleiterin ist direkt oder indirekt ihre eigene Mentorin; folgt man daher der Kette von einer Gruppenleiterin zu ihrer Mentorin, dann zur Mentorin der Mentorin usw., so erreicht man schließlich immer die Präsidentin.

Wir definieren den *Einfluss* einer Gruppenleiterin als die Summe der Anzahl der Mitglieder ihrer Ortsgruppe und den Einflüssen all ihrer Schülerinnen (falls vorhanden). Es ist leicht ersichtlich, dass die Gruppenleiterin mit dem größten Einfluss die Präsidentin ist. Ihr Einfluss ist stets genau M . Eine Gruppenleiterin wird als *ranghohe Gruppenleiterin* bezeichnet, wenn ihr Einfluss $\geq M/2$ ist.

Die Statuten der Gesellschaft legen fest, dass die ranghohe Gruppenleiterin mit dem geringsten Einfluss (unter allen ranghohen Gruppenleiterinnen) als *Kassier* der Gesellschaft fungiert.

Von Zeit zu Zeit kann eine Gruppenleiterin (außer der Präsidentin) *ihre Mentorin wechseln*, sodass sie fortan Schülerin einer anderen Mentorin als bisher ist (vorausgesetzt, ihre neue Mentorin ist nicht eine ihrer Schülerinnen oder eine Schülerin einer ihrer Schülerinnen usw.). Dadurch kann es vorkommen, dass sich der Einfluss einiger Gruppenleiterinnen ändert und die Rolle des Kassiers auf eine andere Gruppenleiterin übergeht.

Aufgabe

Schreibe ein Programm, das die Beschreibung des Anfangszustands der Gesellschaft sowie eine Folge von Mentorinnenwechseln einliest. Gib zum Anfang des Programmes und nach jedem Mentorinnenwechsel aus, wer aktuell Kassier der Gesellschaft ist.

Eingabe

Die erste Zeile enthält zwei, durch Leerzeichen getrennte, natürliche Zahlen n , die Anzahl der Ortsgruppen, und q , die Anzahl der Mentorinnenwechsel.

Die folgenden n Zeilen beschreiben den Anfangszustand der Gesellschaft. Die i -te dieser Zeilen enthält zwei, durch Leerzeichen getrennte, natürliche Zahlen s_i und m_i . s_i

ist die Mentorin der Gruppenleiterin i (also die Gruppenleiterin, die die Ortsgruppe i leitet). m_i ist die Anzahl der Mitglieder der Ortsgruppe i . Der Wert $s_i = 0$ bedeutet, dass die Gruppenleiterin i die Präsidentin der Gesellschaft ist und daher keine Mentorin hat.

Die verbleibenden q Zeilen beschreiben die Mentorinnenwechsel. Die j -te dieser Zeilen enthält zwei, durch Leerzeichen getrennte, natürliche Zahlen \hat{x}_j und \hat{z}_j . Die Bedeutung dieser Zahlen ist wie folgt:

Bezeichnen wir mit t_j (für $j = 0, \dots, q$) den Kassier nach den ersten j Mentorinnenwechseln (also ist t_0 der anfängliche Kassier vor dem ersten Mentorinnenwechsel). Dann besteht der j -te Mentorinnenwechsel darin, dass Gruppenleiterin z_j die neue Mentorin von Gruppenleiterin x_j wird, wobei

$$x_j = 1 + ((t_{j-1} + \hat{x}_j) \bmod n)$$

und

$$z_j = 1 + ((t_{j-1} + \hat{z}_j) \bmod n).$$

Der Zweck dieser Darstellung der Werte x_j und z_j besteht darin, dich dazu zu zwingen, die Mentorinnenwechsel in Eingabereihenfolge zu verarbeiten.

Die Mentorinnenwechsel in den Eingabedaten sind stets gültig, d. h. z_j und x_j sind verschieden und z_j ist keine Schülerin von x_j , noch eine Schülerin einer Schülerin von x_j usw. Es ist jedoch möglich, dass z_j bereits unmittelbar vor dem j -ten Mentorinnenwechsel die Mentorin von x_j war (sodass sich in diesem Fall tatsächlich nichts ändert).

Beachte, dass dein Programm, falls es an irgendeinem Punkt ein falsches Ergebnis t_j berechnet, auch die nachfolgenden Eingaben $\hat{x}_{j+1}, \hat{z}_{j+1}$ usw. falsch dekodiert. Dadurch könnte es statt einer WA-Wertung (Wrong Answer) sogar zu einer RTE-Wertung (Runtime Error) kommen, weil die falsch dekodierten Eingaben ungültig sein könnten (z. B. könnte fälschlicherweise ein z_{j+1} entstehen, das eine Schülerin von x_{j+1} ist).

Beschränkungen

- $1 \leq n \leq 1000000$
- $1 \leq q \leq 30000$
- $1 \leq m_i$ für jedes $i = 1, \dots, n$
- $m_1 + m_2 + \dots + m_n \leq 10^9$
- $1 \leq \hat{x}_j \leq n$ und $1 \leq \hat{z}_j \leq n$ für jedes $j = 1, \dots, q$.

Teilaufgaben

- Teilaufgabe 1 (15 Punkte): $n \leq 100$
- Teilaufgabe 2 (10 Punkte): $n \leq 1000$
- Teilaufgabe 3 (50 Punkte): $n \leq 300\,000$
- Teilaufgabe 4 (25 Punkte): Keine zusätzlichen Beschränkungen.

Ausgabe

Gib die Zahlen t_0, t_1, \dots, t_q jeweils in einer eigenen Zeile aus, wobei t_j der Kassier nach den ersten j Mentorinnenwechseln ist.

Selbstverständlich muss jedes t_j eine natürliche Zahl aus dem Bereich $1 \leq t_j \leq n$ sein.

Beispiel

Eingabe	Ausgabe
7 2	2
0 1	2
1 3	3
1 3	
2 3	
2 1	
5 2	
5 1	
3 7	
2 7	

Kommentar

Anfänglich ist Gruppenleiterin 2 der Kassier (daher $t_0 = 2$). Beim ersten Mentorinnenwechsel ist $\hat{x}_1 = 3$ und $\hat{z}_1 = 7$ und daraus berechnet $x_1 = 1 + ((2 + 3) \bmod 7) = 6$ und $z_1 = 1 + ((2 + 7) \bmod 7) = 3$. Daher wird Gruppenleiterin 3 die neue Mentorin der Gruppenleiterin 6. Gruppenleiterin 2 bleibt weiterhin Kassier (daher $t_1 = 2$). Beim zweiten Mentorinnenwechsel ist $\hat{x}_2 = 2$ und $\hat{z}_2 = 7$ und daraus berechnet $x_2 = 1 + ((2 + 2) \bmod 7) = 5$ und $z_2 = 1 + ((2 + 7) \bmod 7) = 3$. Daher wird Gruppenleiterin 3 die neue Mentorin von Gruppenleiterin 5 und ebenfalls der neue Kassier (daher $t_2 = 3$).

DFS

Zeitlimit: 1 s Speicherlimit: 256 MiB

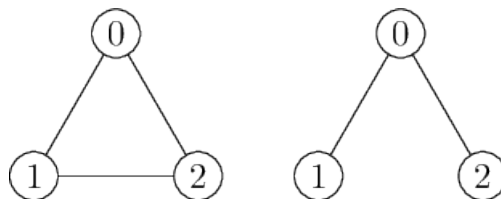
Möglicherweise bist du bereits mit dem bekannten DFS-Algorithmus (Depth-First Search, Tiefensuche) zum Durchlaufen eines Graphen vertraut. In dieser Aufgabe betrachten wir ausschließlich zusammenhängende ungerichtete einfache Graphen, wobei die Knoten mit $0, 1, \dots, n-1$ nummeriert sind. (In einem einfachen Graphen verbindet keine Kante einen Knoten mit sich selber und zwischen jeden zwei Knoten gibt es höchstens eine Kante.) Der DFS-Algorithmus gibt Tiefen und Knoten wie folgt aus:

```
DFS(d, v):
  gib d/v aus
  markiere den Knoten v als besucht
  W = die Liste der Nachbarn von v, aufsteigend nach ihren Nummern sortiert
  für jeden Knoten w in W:
    falls der Knoten w noch nicht besucht wurde:
      DFS(d + 1, w)
```

Schreibe ein Programm, das die Anzahl unterschiedlicher Graphen ausgibt, für die der Aufruf `DFS(0, n - 1)` genau dieselbe Ausgabe erzeugt wie die in der Eingabe angegebene. Zum Beispiel wird die Ausgabe

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

durch den Aufruf `DFS(0, 2)` auf jeden der folgenden beiden zusammenhängenden ungerichteten einfachen Graphen mit 3 Knoten erzeugt:



Eingabe

Die Eingabe ist die Ausgabe des Aufrufs `DFS(0, n - 1)` auf einem unbekanntem zusammenhängenden ungerichteten einfachen Graphen mit n Knoten. Die Eingabe besteht daher aus n Zeilen im Format `d/v`, wobei die erste Zeile `0/n-1` ist.

Beschränkungen

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

Teilaufgaben

- Teilaufgabe 1 (10 Punkte): $n \leq 6$.

- Teilaufgabe 2 (20 Punkte): $n \leq 500$.
- Teilaufgabe 3 (20 Punkte): $n \leq 10^4$.
- Teilaufgabe 4 (10 Punkte): Für jedes $i \in \{2, \dots, n\}$ ist die i -te Eingabezeile $i-1/i-2$.
- Teilaufgabe 5 (20 Punkte): Für jedes $i \in \{2, \dots, n\}$ ist die i -te Eingabezeile $i-1/v$ für ein $v \in \{0, \dots, n-2\}$.
- Teilaufgabe 6 (20 Punkte): Keine zusätzlichen Beschränkungen.

Ausgabe

Gib die Anzahl der Graphen mit der geforderten Eigenschaft aus. Da diese Zahl sehr groß sein kann, gib das Ergebnis modulo 1 000 000 007 aus.

Beispiel

Eingabe

0/2
1/0
2/1

Ausgabe

2

Schatzsuche

Zeitlimit: 8 s Speicherlimit: 256 MiB

Du suchst mit einem magischen Kompass nach einem lange verschollenen Schatz auf einem riesigen Feld aus $N \times N$ Zellen. Insgesamt sind $K \leq 3$ Schatztruhen an verschiedenen Zellen im Feld versteckt. Dein Ziel ist einfach: Finde sie alle!

Wenn du den Kompass auf eine Zelle legst, findet er die kürzesten Wege zu allen Schatztruhen, die am nächsten sind. Dabei werden nur die Bewegungen nach links, oben, rechts und unten verwendet (d. h. die Distanz zu allen nächstgelegenen Schatztruhen gemäß Manhattan-Distanz). Anschließend zeigt er die Richtung des ersten Schritts an. Falls mehrere gültige erste Schritte zu einer nächstgelegenen Schatztruhe (oder zu mehreren von ihnen) führen, gibt der Kompass alle diese Richtungen zurück. Falls die Zelle einen Schatz enthält, zeigt der Kompass dies stattdessen an.

Immer wenn du eine Schatztruhe findest, leerst du ihren Inhalt, aber du kannst die Truhe nicht entfernen – sie ist zu schwer. Dein Kompass weiß nicht, ob eine Truhe voll oder leer ist. Er zeigt den Weg zu einer (oder falls vorhanden mehreren) nächstgelegenen Truhe, egal ob diese leer oder voll ist.

Versuche, die Position aller Schatztruhen mit möglichst wenigen Kompassabfragen zu finden.

Aufgabe

Dies ist eine interaktive Aufgabe. In jedem Testfall (d. h. bei jeder Ausführung deines Programms) muss dein Programm mehrere Schatzsuchen lösen. Du sollst mit dem Grader über eine bereitgestellte Bibliothek interagieren. Diese Bibliothek stellt die folgenden Funktionen bereit:

- **void** *NextHunt*(**int** &*N*, **int** &*K*) — rufe diese Funktion auf, um die nächste Schatzsuche zu starten. Sie setzt den Parameter *N* auf die Größe des Gitters und den Parameter *K* auf die Anzahl der Schätze. Falls in der aktuellen Programmausführung keine weiteren Schatzsuchen mehr zu lösen sind, setzt die Funktion *N* und *K* auf -1 ; in diesem Fall solltest du dein Programm mit dem Exit-Code 0 beenden. Beachte, dass du diese Funktion auch aufrufen darfst, bevor du alle Schätze der aktuellen Schatzsuche gefunden hast, z. B. wenn du nur Teilpunkte anstrebst.
- **enum** {*TREASURE* = 0, *DIR_RIGHT* = 1, *DIR_UP* = 2, *DIR_LEFT* = 4, *DIR_DOWN* = 8 }; — dies sind Konstanten, die in den Rückgabewerten der Funktion *Query* verwendet werden (siehe unten).
- **int** *Query*(**int** *x*, **int** *y*) — falls die Zelle an den Koordinaten (*x*, *y*) einen Schatz enthält, gibt diese Funktion *TREASURE* zurück. Andernfalls gibt sie die Summe einer oder mehrerer der Konstanten *DIR_RIGHT*, *DIR_UP*, *DIR_LEFT* und *DIR_DOWN* zurück und zeigt damit an, welche Bewegungen der Kompass liefert, wenn er auf die Zelle (*x*, *y*) gelegt wird. Die Koordinaten *x* und *y* müssen Ganzzahlen von 0 bis $N - 1$ sein. (Hinweis: In dieser Aufgabe wachsen die *y*-Koordinaten von oben nach unten.)

Nachdem *NextHunt* die Werte $N = K = -1$ gesetzt hat, darf dein Programm weder *NextHunt* noch *Query* erneut aufrufen. Es darf außerdem *Query* nicht vor dem ersten Aufruf von *NextHunt* aufrufen. Falls dein Programm diese Bedingungen nicht einhält, oder falls es mehr als 1000 Abfragen innerhalb einer einzelnen Schatzsuche durchführt, oder falls es Werte für x und/oder y außerhalb des zulässigen Bereichs an *Query* übergibt, beendet die Bibliothek dein Programm und vergibt für den aktuellen Testfall das Urteil Run-Time Error (RTE).

Ein Schatz gilt als gefunden, wenn du die Zelle, die ihn enthält, mindestens einmal abgefragt hast. Falls dein Programm *NextHunt* aufruft, bevor alle Schätze der aktuellen Schatzsuche gefunden wurden, wird dies nicht als Fehler behandelt, beeinflusst jedoch deine Punktzahl (mehr dazu weiter unten).

Um auf die Bibliothek zuzugreifen, sollte dein Programm die Header-Datei `treasurehuntlib.h` einbinden:

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

Du kannst diese Header-Datei hier herunterladen: `treasurehuntlib.h`.

Um dir bei der Entwicklung deiner Lösung zu helfen, steht hier eine einfache Implementierung der Bibliothek zur Verfügung: `treasurehuntlib-public.cpp`. Um sie zusammen mit deinem Programm zu kompilieren, füge ihren Namen einfach als Parameter für den Compiler hinzu, z. B.:

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

wenn `foo.cpp` der Name der Datei ist, die deine Lösung enthält.

Die Implementierung in `treasurehuntlib-public.cpp` unterstützt zusätzlich zu den oben genannten Funktionen eine weitere Funktion namens `void InitFromFile(const char *fileName)`, die eine Liste von Schatzsuchen aus einer Datei einliest und es dir ermöglicht, das Spiel mit diesen zu spielen, anstatt zufällige Schatzsuchen zu generieren. Weitere Details findest du in der Datei `treasurehuntlib-public.cpp`.

Auf dem Grader wird eine andere Implementierung der Bibliothek verwendet, daher solltest du keine Annahmen darüber treffen, wie genau die Implementierung funktioniert. Du darfst jedoch annehmen, dass es im globalen Namensraum nur die oben aufgeführten Deklarationen (*NextHunt*, *Query* und die fünf Konstanten) gibt.

Dein Code darf weder von der Standardeingabe lesen noch auf die Standardausgabe schreiben, da diese von der Implementierung der Bibliothek auf dem Grader zur Kommunikation mit der restlichen Bewertungsumgebung verwendet werden.

Eingabe

Beschränkungen

- $2 \leq N \leq 10^6$
- $1 \leq K \leq 3$
- Innerhalb einer einzelnen Programmausführung wird es höchstens 100 000 Schatzsuchen geben.
- Innerhalb einer einzelnen Schatzsuche dürfen höchstens 1000 Abfragen durchgeführt werden.
- Der Grader ist nicht adaptiv.

Teilaufgaben

- Teilaufgabe 1 (10 Punkte) $K = 1$
- Teilaufgabe 2 (30 Punkte) $K = 2$
- Teilaufgabe 3 (60 Punkte) $K = 3$.

Bewertung

Eine Teilaufgabe kann aus mehreren Testfällen (mehreren Ausführungen deines Programms) bestehen, und jeder Testfall kann mehrere Schatzsuchen enthalten. Für die Bewertung werden alle Schatzsuchen einer gegebenen Teilaufgabe gemeinsam betrachtet, unabhängig davon, wie sie ursprünglich auf die Testfälle verteilt waren. Für die i -te Schatzsuche bezeichnen wir die Gittergröße mit $N_i \times N_i$, die Anzahl der Schätze mit K_i , die Anzahl der von deinem Programm durchgeführten Abfragen mit Q_i und die Anzahl der gefundenen Schätze mit F_i . Weiterhin bezeichnen wir mit S die Gesamtzahl der für diese Teilaufgabe verfügbaren Punkte. Dann ergibt sich die Anzahl der Punkte, die dein Programm für diese Teilaufgabe erhält, wie folgt:

- Falls dein Programm immer alle Schätze gefunden hat (d. h. falls $F_i = K_i$ für alle i gilt), hängt die Punktzahl ab von $t_i = \frac{Q_i}{\lceil \log_2 N_i \rceil}$:

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

- Falls dein Programm nicht immer alle Schätze gefunden hat (d. h. es existiert ein i mit $F_i < K_i$), erhält es

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

Mit anderen Worten: Du erhältst die Hälfte der Punkte dafür, alle Schätze zu finden, und die andere Hälfte dafür, sie mit möglichst wenigen Abfragen zu finden. Für die volle Punktzahl sollte deine Lösung alle Schätze in höchstens $11 \lceil \log_2 N_i \rceil$ Abfragen finden. Zwischen $11 \lceil \log_2 N_i \rceil$ und $20 \lceil \log_2 N_i \rceil$ Abfragen sinkt die Punktzahl linear; und falls deine Lösung mehr als $20 \lceil \log_2 N_i \rceil$ Abfragen benötigt, erhält sie nur die erste Hälfte der Punkte für das Finden aller Schätze. (Die Symbole $\lceil \cdot \rceil$ bedeuten, dass der Wert von $\log_2 N_i$ auf die nächsthöhere ganze Zahl aufgerundet wird.)

Falls die obigen Formeln zu einer nicht ganzzahligen Punktzahl für die Teilaufgabe führen, wird diese auf die nächstgelegene ganze Zahl gerundet.

Falls dein Programm einen Laufzeitfehler verursacht oder sich nicht an das oben beschriebene Protokoll zur Nutzung der Bibliothek hält, erhält es 0 Punkte für die gesamte Teilaufgabe. Um Teilpunkte für das Finden nur eines Teils der Schätze zu erhalten, muss dein Programm die Suche daher ordnungsgemäß durch einen Aufruf von *NextHunt* beenden.

Beispiel

Aufruf	Rückgabewert
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)	$N = -1, K = -1$