Problem A: Years

Rising stars only

Author:

Vuk Vuković

Implementation and analysis:

Drago Šmitran

Filip Kovačević

Statement:

During one of the space missions, humans have found an evidence of previous life at one of the planets. They were lucky enough to find a book with birth and death years of each individual that had been living at this planet. What is interesting is that these years are in the range $(1,10^9)$! Therefore, the planet was named Longlifer.

In order to learn more about Longlifer's previous population, scientists need to determine the year with maximum number of individuals that were alive, as well as the number of alive individuals in that year. Your task is to help scientists solve this problem!

Input:

The first line contains an integer **n** $(1 \le \mathbf{n} \le 10^5)$ - the number of people.

Each of the following n lines contain two integers **b** and **d** $(1 \le \mathbf{b} < \mathbf{d} \le 10^9)$ representing birth and death year (respectively) of each individual.

Output:

Print two integer numbers separated by blank character, **y** - the year with a maximum number of people alive and **k** - the number of people alive in year **y**.

In the case of multiple possible solutions, print the solution with minimum year.

Example input 1:

```
3
1 5
2 4
5 6
```

Example output 1:

```
2 2
```

Example input 2:

```
4
3 4
4 5
4 6
8 10
```

Example output 2:

```
4 2
```

Note:

You can assume that an individual living from **b** to **d** has been born at the beginning of **b** and died at the beginning of **d**, and therefore living for **d** - **b** years.

```
Time and memory limit: 1s / 256 MB
```

Solution and analysis

Let us first solve this problem in $O(MAX\_YEAR)$. To do that, we have to keep the number of births and deaths which occurred at the start of each year. We will denote these counts for the ith year as $births_i$ and $deaths_i$. Also, let us denote the number of people which are alive in ith year as $alive_i$. We can then use these counts to calculate the number of people living in the ith year as $alive_i = \sum_{j=1}^{i} births_j - deaths_j$. The reason this formula works is because if an individual is born in year $b < i$ and dies at the start of year $d > i$, $alive_i$ will be increased by one, but if an individual dies at the start of year $d \leq i$, $alive_i$ will not be increased. Similarly, in case of an individual's birth year being greater than i, they will not be counted. Unfortunately, iterating over all years is too slow in this problem as years can be as high as 10^9. We can observe that the count of living people changes only in years i where $births_i$ or $deaths_i$ are greater than zero. This means we do not have to iterate through all the years, but only through the years in which some person was born or died and only then update the result. To do this, we can create a new list of pairs and for each individual add two pairs to the list. First element of both pairs is year of birth or death, and the second element is a flag denoting whether the first element is a birth year or death year. Afterwards, sort the list and iterate through it to find the result. Time complexity is $O(NlogN)$.

Problem B: Ancient Language

Rising stars only

Author: **Vuk Vuković**

Implementation and analysis: **Drago Šmitran**, **Andrej Jakovljević**

Statement:

While exploring the old caves, researchers found a book, or more precisely, a stash of mixed pages from a book. Luckily, all the original pages are present, and each page contains its number. Therefore, the researchers can reconstruct the book.

After taking a deeper look into the contents of these pages, linguists think that this may be some kind of dictionary. What is interesting is that this ancient civilization used an alphabet which is a subset of the English alphabet, however, the order of these letters in the alphabet is not like the one in the English language.

Given the contents of pages that researchers have found, your task is to reconstruct the alphabet of this ancient civilization using the provided pages from the dictionary.

Input:

The first line contains two integers: **n** and **k** ($1 \le n, k \le 10^3$) - the number of pages that the scientists have found, and the number of the words present at each page. The following **n** groups contain a line with a single integer $\mathbf{p_i}$ ($0 \le \mathbf{p_i} \le 10^3$) - the number of **i**th page, as well as **k** lines, each line containing one of the strings (up to **100** characters) written on the page numbered $\mathbf{p_i}$.

Output:

Output a string representing the reconstructed alphabet of this ancient civilization. If the book found is not a dictionary, output "IMPOSSIBLE'' without quotes. In case there are multiple solutions, output any of them.

Example input 1:

```
3 3
2
b
b
bbac
0
a
aca
```

```
acba
1
ab
c
ccb
```

Example output:

```
acb
```

```
Time and memory limit: 1s / 256 MB
```

Solution and analysis

We can solve this problem by converting words from pages to a directed graph and then doing topological sort on that graph.

First of all, sort the pages by their number and then squash all the pages into a single page. Let n be the total number of words and S_i be the ith word of the resulting squashed page. If there exists an index k such that S_{k+1} is a proper prefix of S_k, then no solution exists. Otherwise let's continue and create a directed graph from the words. We will process all words S_i except the last one and we have the following two cases emerge:

- S_i is a prefix of S_{i+1} in which case we should do nothing,
- S_i is not a prefix of S_{i+1}, then there exists a position j such that j-th character of S_i is different from j-th character of S_{i+1}. Let's also add a constraint that j is the first such position where S_i and S_{i+1} differ. Let c_1 be the j-th character of S_i and c_2 be the j-th character of S_{i+1}. For this case we add a directed edge $(c1, c2)$ in our resulting graph.

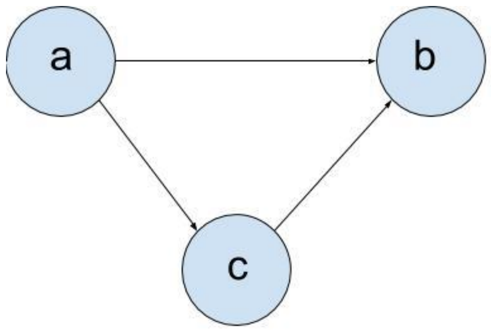
Now that we have the graph, we can run topological sort on that graph and print out the topological ordering if it exists. In case the topological ordering does not exist, print out that it is impossible to find the reconstructed alphabet.

Let's apply this solution to the sample test. Once we order the words and squash them to a single page, we get an array of words $[a, aca, acba, ab, c, ccb, b, b, bbac]$. Let's process the words one by one.

- a is a prefix of aca, do nothing
- aca differs from $acba$ in third character, add edge (a, b)
- $acba$ differs from ab in second character, add edge (c, b)
- ab differs from c in first character, add edge (a, c)
- c is prefix of ccb, do nothing

...

Once we finish this construction, we get the following graph.



Topological ordering of this graph is acb which represents the solution for this input. We can also observe that no other topological ordering exists.

Problem C: Lonely Numbers

Premier League and Rising Stars

Author:

Nikola Aleksić

Implementation and analysis:

Nikola Aleksić

Nikola Pešić

Statement:

In number world, two different numbers are friends if they have a lot in common, but also each one has unique perks.

More precisely, two different numbers a and b are friends if $\gcd(a, b)$, $\frac{a}{\gcd(a,b)}$, $\frac{b}{\gcd(a,b)}$ can form sides of a triangle.

Three numbers a, b and c can form sides of a triangle if $a + b > c$, $b + c > a$ and $c + a > b$.

In a group of numbers, a number is lonely if it does not have any friends in that group.

Given a group of numbers containing all numbers from $1, 2, 3, \dots, N$ how many numbers in that group are lonely?

Input:

The first line contains a single integer t - number of test cases.

On next line there are t numbers, n_i - meaning that in case i you should solve for numbers 1, 2, 3, …, n_i.

Output:

For each test case, print the answer in separate lines: number of lonely numbers in group 1, 2, 3, …, n_i.

Constraints:

- $1 \leq t \leq 10^6$
- $1 \leq n_i \leq 10^6$

Example input 1:

```
3
1 5 10
```

Example output:

```
1
3
3
```

Explanation:

For first test case, 1 is the only number and therefore lonely.

For the second test case where n = 5, numbers 1, 3 and 5 are lonely.

For the third test case where n = 10, numbers 1, 5 and 7 are lonely.

```
Time and memory limit: 1.5s / 256 MB
```

Solution and analysis

Any composite number $a = b * c$, where $b \geq c, b \geq 2, c \geq 2$, is friends with $(b-1) * c$, because $\gcd(b, b-1) = 1 => \gcd(b * c, (b-1) * c) = c$ and $b + b - 1 > c, b + c > b - 1, b - 1 + c > b$. Hence, composite numbers are not lonely.

Any prime number p is not friends with coprime number a, because either $a > p + 1$ or $p > a + 1$.

If $p \mid a$, then $\gcd(p, a) = p, \frac{p}{\gcd(a,p)} = 1$. For them to be friends, $p + 1$ has to be larger than $\frac{a}{p}$ and $\frac{a}{p} + 1$ has to be larger than p. That can only happen if $\frac{a}{p} = p => a = p^2$.

That means that prime numbers have friends in range $1,2,3, \dots, N$ only if $p \leq \sqrt{N}$. To count all lonely number below or equal to N, count prime numbers larger than $\sqrt{N}$, and add 1 (1 is always lonely).

Use the sieve of Eratosthenes to determine all prime numbers below 10^6, preprocess the count of prime numbers below i for each i, and for each testcase return primesBelow[N] – primesBelow[(int)sqrt(N)] + 1.

Problem D: Valuable Paper

Premier League and Rising Stars

Author:

Đorđe Stupar

Implementation and analysis:

Nikola Pešić

Drago Šmitran

Statement:

The pandemic is upon us, and the world is in shortage of the most important resource: toilet paper. As one of the best prepared nations for this crisis, BubbleLand promised to help all other world nations with this valuable resource. To do that, the country will send airplanes to other countries carrying toilet paper.

In BubbleLand, there are N toilet paper factories, and N airports. Because of how much it takes to build a road, and of course legal issues, every factory must send paper to only one airport, and every airport can only take toilet paper from one factory.

Also, a road cannot be built between all airport-factory pairs, again because of legal issues. Every possible road has number d given, number of days it takes to build that road.

Your job is to choose N factory-airport pairs, such that if the country starts building all roads at the same time, it takes the least amount of days to complete them.

Input:

The first line contains two integers N - number of airports/factories, and M - number of available pairs to build a road between.

On next M lines, there are three integers u, v, d - meaning that you can build a road between airport u and factory v in d days.

Output:

If there are no solutions, output -1. If there exists a solution, output the minimal number of days to complete all roads, equal to maximal d among all chosen roads.

Constraints:

- $1 \leq N \leq 10^4$
- $1 \leq M \leq 10^5$
- $1 \leq u, v \leq N$
- $1 \leq d \leq 10^9$

Example input:

```
3 5
1 2 1
2 3 2
3 3 3
2 1 4
2 2 5
```

Example output:

```
4
```

`Time and memory limit: 2s / 256 MB`

Solution and analysis

We can reduce this problem to finding a perfect bipartite matching where the maximum weight of all edges in the matching is minimized. First, we construct a bipartite graph, by having the first disjoint set be the factories, and the second disjoint set be the airports. Weight of each edge connecting an airport and a factory is the number of days it takes to build a road between that airport and factory.

Now that we have the graph setup, we can observe that if there exists a perfect matching M such that no edge of M has weight greater than W, then a perfect matching exists for all weights W_2 which are greater than W. The reason this is true is because we know that there is no edge in M which has weight greater than W, which implies that there cannot be an edge in M which has weight greater than W_2 (because $W_2 > W$), so we can use the same perfect matching M to satisfy W_2.

This observation enables us to apply binary search on weights of edges to find the answer. For each iteration of binary search, we have to check whether a perfect matching exists for some weight W and we can do that by applying the Hopcroft-Karp algorithm for finding the maximum cardinality bipartite matching. Time complexity of the Hopcroft-Karp algorithm is $O(E \cdot \sqrt{V})$ where E is the number of edges and V is the number of nodes in the graph. After multiplying this with the additional logarithm factor from binary search, total time complexity of this solution ends up being $O(E \cdot \sqrt{V} \cdot logD)$ where D is the maximum weight among all edges.

We can further reduce the complexity by first sorting the edges by weight, and then applying binary search on that sorted list of edges. This reduces the logarithm factor from $logD$ to $logE$ for a final complexity of $O(E \cdot \sqrt{V} \cdot logE)$.

Problem E: Coins

Premier League and Rising Stars

Author: *Implementation and analysis:*

Filip Kovačević ***Aleksandar Lukač***

Filip Kovačević

Statement:

A famous gang of pirates, Sea Dogs, has come back to their hideout from one of their extravagant plunders. They want to split their treasure fairly amongst themselves, and that is why You, their trusted financial advisor, devised a game to help them:

All of them take a sit at their round table, some of them with the golden coins they have just stolen. At each iteration of the game, if one of them has equal or more than 2 coins, he is eligible to the splitting and he gives one coin to each pirate sitting next to him. If there are more candidates (pirates with equal or more than 2 coins) then **You** are the one that chooses which **one** of them will do the splitting in that iteration. The game ends when there are no more candidates eligible to do the splitting.

The pirates can call it a day only when the game ends. Since they are beings with a finite amount of time at their disposal, they would prefer if the game that they are playing can end after finite iterations, and if so, they call it a **good game**. On the other hand, if no matter how **You** do the splitting, the game cannot end in finite iterations, they call it a **bad game**. Can You help them figure out before they start playing if the game will be good or bad?

Input:

The first line of input contains two integer numbers n and k ($1 \le n \le 10^9$, $0 \le k \le 2 \cdot 10^5$), where n denotes the total number of the pirates and k is the number of the pirates that have any coins.

The next k lines of input contain integers a_i and b_i ($1 \le a_i \le n$, $1 \le b_i \le 10^9$), where a_i denotes the index of the pirate sitting at the round table (n and 1 are neighbors) and b_i the total number of the coins that pirate a_i has at the start of the game.

Output:

Print 1 if the game is a **good game**: There is a way to do the splitting so the game ends after the finite number of iterations.

Print −1 if the game is a **bad game**: No matter how You do the splitting the game does not end in the finite number of iterations.

Example input 1:

```
4 2
```

```
1 2
2 2
```

Example output 1:

```
1
```

Example input 2:

```
6 2
2 3
4 1
```

Example output 2:

```
1
```

Example input 3:

```
3 2
1 1
2 2
```

Example output 3:

```
-1
```

Note

In the third example the game has no end, because You always have only one candidate and after the splitting you end up in the same position as the starting one.

```
Time and memory limit: 1s / 256 MB
```

Solution and analysis

Firstly, let's denote the sum of all the coins that are in possession of the pirates as **S**. We have 3 separate cases depending on relation **S** to **n** (the number of the pirates):

1. **S > n**
   There are more coins than there are pirates, so no matter how we do the splitting we can never end up in a position where every pirate has at most one coin, the position at which no more splits can be done, therefore the **game can never end**.
2. **S < n**

If pirate A gives a coin to pirate B and then at some later point pirate B will be doing the splitting and will be returning the “same” coin to pirate A. We can therefore denote that coin with $\mathbf{z_{AB}}$. Since we have more pirates than we have coins, that means that there exist 2 neighboring pirates that never exchange any coins. From that we can conclude that there exists a pirate that can do the split the finite amount of times (even 0). We can now conclude inductively that all the splits are done the finite amount of times, since the neighbor of the pirate that does finitely many splits must also do the finite number of splits. This is because if the neighbor does infinitely many splits, then his neighbor that does them finitely many times will have an infinite number of coins, which is a contradiction. Therefore, **the game always ends**.

3. **S = n**

We can take notice of invariance of each split. Let us label each coin by its position, where by position we mean the index (1-n) of the pirate in whose possession it is. We can see that after a pirate does the splitting, the sum of labels mod n of the two coins split is invariant. Let’s denote the label of the two coins as **x**, then after the split we lose 2*x, and gain (x-1) + (x+1) in case of x not equal to n, and (n-1) + (1) in case of x equaling n. Therefore, we can indeed conclude that the sum of all labels mod n will be invariant to any splitting. We can therefore see that the necessary condition for the game to end is that the sum of labels of all coins must be congruent to n*(n+1)/2 mod n, because that is the sum of the labels in the ending position (each pirate has one coin).

To prove that the condition is necessary, we need to firstly look at an example where all but 2 pirates have one coin, one has 2 coins, and one has zero coins and prove that the game cannot end. This can be done by making the sum of all labels and proving that it can never be congruent to n*(n+1)/2. This is because the sum will be n*(n+1)/2 - x+y where x is the index of the pirate with 0 coins, and y is the index of the pirate with 2 coins. For n*(n+1)/2 - x+y to be congruent to n*(n+1)/2 mod n, we must have (y-x) congruent to 0 mod n, which cannot happen because 1≤x,y≤n and x≠y.

Now we can devise an algorithm for the splitting so that we always end up in the aforementioned position if the starting sum was not congruent to n*(n+1)/2. To do this, we firstly take notice of a concept of **island**: an array of pirates where each one has at least one coin, they all form an array of consecutive indexes (e.g. 2,3,4,5,6) and the first and last pirate in the array have neighbors that have 0 coins. There is at least one island (or else the game is already finished) since there are the same number of coins as the pirates. Now we do the splitting in the following manner:

If the island has no pirates with more than one coin, leave it as it is. If it has at least one pirate with more than one coin, not including the case of the edge pirates with exactly two coins, then do the following: Do the splitting for that pirate, and after that do the splitting for his neighbors (because they had at least one coins and they have just gained one more) and then of their neighbors that are closer to the end of the island and so on until the end of the island (on both left and right end). This will allow the island to expand by one pirate to the left, and one pirate to the right (you basically propagate two coins from the first pirate to the 2 pirates with zero coins that are at the end of the island). Now the expanded island might be concatenated with its neighboring islands, forming a bigger one.

Repeat this process until all the islands are in one of the following 3 forms:

- All the pirates have just one coin

- Two edge pirates have two coins, all others have one.
- One edge pirate has two coins, all others have one.

If we do the beforementioned process of splitting on the third kind of island, we will end up with the first kind (simply check this). That is why we end up with only 1. and 2. kind of islands. Also, If we have more than one island of the 2nd kind, meaning that there exist different ending left and right pirates with zero coins, then again by the repeated process we end up with two islands of the first kind. That is why we only have two cases:

I. All the islands are of the first kind, meaning there is only one island of the first kind (because the number of pirates is the same as coins) which means that the **game has an end**.
II. There is exactly one island of the 2nd kind and all others are of the first kind, meaning that there is only one island of the 2nd kind (again because number of coins is the same as pirates) which is the case we have already covered at the beginning, meaning the **game has no end.**

Since the sum of all labels is an invariance and we have just proven that we can always end up in either I or II, and we know that the state in II is not congruent to n*(n+1)/2, we know that we will always end up in I using this algorithm for the splitting if the sum of labels is congruent to n*(n+1)/2. Thus, we have proven that the congruence test is the sufficient condition to check if the game has an end.

Problem F: Light switches

Premier League and Rising Stars

Author:

Nikola Pešić

Implementation and analysis:

Đorđe Stupar

Drago Šmitran

Statement:

Nikola owns a large warehouse which is illuminated by N light bulbs, numbered 1 to N. At the exit of the warehouse, there are S light switches, numbered 1 to S. Each switch swaps the on/off state for some light bulbs, so if a light bulb is off, flipping the switch turns it on, and if the light bulb is on, flipping the switch turns it off.

At the end of the day, Nikola wants to turn all the lights off. To achieve this, he will flip some of the light switches at the exit of the warehouse, but since Nikola is lazy, he wants to flip the minimum number of switches required to turn all the lights off. Since Nikola was not able to calculate the minimum number of switches, he asked you to help him. During a period of D days, Nikola noted which light bulbs were off and which were on at the end of each day. He wants you to tell him the minimum number of switches he needed to flip to turn all the lights off for each of the D days or tell him that it is impossible.

Input:

First line contains three integers, N, S and D -- representing number of light bulbs, the number of light switches, and the number of days, respectively.

The next S lines contain the description of each light switch as follows: The first number in the line, C_i, represents the number of light bulbs for which the on/off state is swapped by light switch i, the next C_i numbers (sorted in increasing order) represent the indices of those light bulbs.

The next D lines contain the description of light bulbs for each day as follows: The first number in the line, T_i, represents the number of light bulbs which are on at the end of day i, the next T_i numbers (sorted in increasing order) represent the indices of those light bulbs.

Output:

Print D lines, one for each day. In the i^th line, print the minimum number of switches that need to be flipped on day i, or −1 if it is impossible to turn all the lights off.

Constraints:

- $1 \le N \le 10^3$
- $1 \le S \le 30$
- $1 \le D \le 10^3$

- $1 \le C_i \le N$
- $1 \le T_i \le N$

Example input:

```
4 3 4
2 1 2
2 2 3
1 2
1 1
2 1 3
3 1 2 3
3 1 2 4
```

Example output:

```
2
2
3
-1
```

`Time and memory limit: 3s / 1024 MB`

Solution and analysis

Let's consider the naive solution in $O(\mathrm{D} * 2^S * \mathrm{N})$. Try all combinations of switches and keep the one with minimum number of flicks needed. We can optimize this solution to $O(\mathrm{D} * 2^S * (\mathrm{N}/64))$ by using bitset or an array of longs for keeping the combination of lit up light bulbs. Still, this optimized naive solution times out.

We can use the meet-in-the-middle technique to reduce the time complexity to $O((\mathrm{N}/64) * (2^{20} + D * 2^{10}))$. Preprocess all the combinations for first 20 switches and keep the resulting light bulb combinations in a hashmap. In case of multiple switch combinations producing the same light bulb combination, keep the one which needs the minimum number of switches to be flicked. Map key will be the light bulb combination, and the map value is minimum number of switches that need to be flicked to get the light bulb combination. After the preprocessing is complete, we can process the days.

For each day, calculate all combinations of the left over 10 switches and process each one. Each switch combination produces some light bulb combination. Since we want all the lightbulbs to be turned off, we can query the hashmap for some combination of first 20 switches which results in all the lightbulbs being turned off. While processing combinations from the left over 10 switches, keep the minimum number of switch flicks needed to turn off all bulbs and after the processing has ended, print this number. Time complexity of the solution is $O((\mathrm{N}/64) * (2^{20} + D * 2^{10}))$.

Problem G: Does anyone else hate the wind?

Premier League and Rising Stars

Author:

Danilo Tonić

Implementation and analysis:

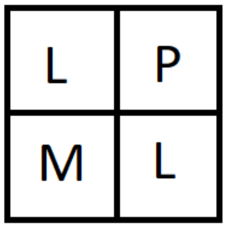
Drago Šmitran

Danilo Tonić

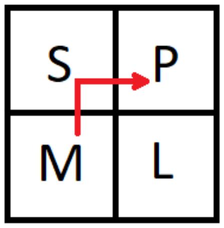
Statement:

Mark and his crew are sailing across the sea of Aeolus (in Greek mythology Aeolus was the keeper of the winds). They have the map which represents the NxM matrix with land and sea fields and they want to get to the port (the port is considered as sea field). They are in a hurry because the wind is very strong and changeable there and they have the food for only K days on the sea (that is the maximum that they can carry on the ship). John, the guy from Mark's crew, knows how to predict the direction of the wind on a daily basis for W days, which is enough time for them to reach the port or to run out of the food. Mark can move the ship in four directions (north, east, south, west) by one field for one day, but he can also stay in the same place. The wind can blow in four directions (north, east, south, west) or just not blow that day. The wind is so strong in the sea of Aeolus that it moves the ship for one whole field in the direction which it blows in. The ship's resulting movement is the sum of the ship's action and the wind from that day. Mark must be careful to keep the ship on the sea, the resulting movement must end on the sea field and there must be a 4-connected path through the sea from the starting field. A 4-connected path is a path where you can go from one cell to another only if they share a side.

For example, in the following image the ship cannot move to the port as there is no 4-connected path through the sea.



In the next image, the ship can move to the port as there is a 4-connected path through the sea as shown with the red arrow. Furthermore, the ship can move to the port in one move if one of the following happens. The wind is blowing east and Mark moves the ship north, or the wind is blowing north, and Mark moves the ship east. In either of these scenarios the ship will end up in the port in one move.



Mark must also keep the ship on the map because he does not know what is outside. Luckily for Mark and his crew, there are T fish shops on the sea where they can replenish their food supplies to the maximum, but each shop is working on only one day of the week. That means that Mark and his crew must be at the shop's position on the exact working day to replenish their food supplies. Help Mark to find the minimum of the days that he and his crew need to reach the port or print −1 if that is impossible with the food supplies they have.

Input:

The first line contains two integer numbers *N* and *M* ($1 \leq N, M \leq 200$) - representing the number of the rows and the number of the columns of the map.

The second line contains three integers, *K* ($0 \leq K \leq 200$) which is the number of days with the available food supplies, *T* ($0 \leq T \leq 20$) which is the number of the fields with additional food supplies and *W* ($0 \leq W \leq 10^6$) which is the number of the days with wind information.

Next is the *NxM* char matrix filled with the values of 'L', 'S', 'P' or 'M'. 'L' is for the land and 'S' is for the sea parts. 'P' is for the port field and 'M' is the starting field for the ship.

The next line contains *W* chars with the wind direction information for every day. The possible inputs are 'N' - north, 'S' - south, 'E' - east, 'W' - west and 'C' - no wind.

In the end there are *T* lines with the food supplies positions. Each line contains three integers, Y_i and X_i ($0 \leq Y_i < N$, $0 \leq X_i < M$) representing the coordinates (*Y* is row number and *X* is column number) of the food supply and F_i ($0 \leq F_i \leq 10^6$) representing the number of days from the starting day on which the food supply is available.

Output:

One integer number representing the minimal days to reach the port or *-1* if that is impossible.

Example input 1:

```
3 3
5 2 15
M S S
S S S
S S P
S W N N N N N N N N N N N N N
2 1 0
1 2 0
```

Example output 1:

```
-1
```

Explanation 1:

The ship is not able to move rightwards.

Example input 2:

```
3 3
5 2 15
M S S
S S S
S S P
S E N N N N N N N N N N N N N
2 1 0
1 2 0
```

Example input 2:

```
2
```

Explanation 1:

The ship can move rightwards.

Example input 3:

```
5 5
4 1 15
M S S S S
S S S S L
S S S L L
S S S S S
S S S S P
C C C C S S E E C C C C C C C
0 1 4
```

Example output 2:

```
8
```

Explanation 3:

The ship must wait for food.

`Time and memory limit: 1.5s / 256 MB`

Solution and analysis

Let’s create a matrix that is the same size as the map from the input and let’s tell that each field contains the minimum number of days from the starting position to reach it or *-1* if that field is unreachable for some number of days. The solution is in the dynamic programming approach where you are updating that matrix for every day using the matrix from the previous day. If the ship can reach some field until the previous day and if wind allows reaching some other field that stands as unreachable until the current day, then we fill that the other field in our matrix with the current day number which from that day stands as minimum days to reach that field. If the value in the matrix on the port field is still *-1* after *K* days of updating then we add to queue all food fields that are reachable for *K* days and that are working within the minimum number of days to reach them and the maximum number that we can be on the sea (in this case that is *K* days). Then we start creating the same matrix again but with food fields as starting points and considering that processing started from the day on which we replenished our supplies using a food field. Every food field processing will maybe “unlock” some other food field or will give solution for the port. We must push unlocked food fields to queue, process them all and find minimum solution for the port. The complexity of this solution is *O(N*M*K*T).*

Problem H: 5G Antenna Towers

Premier League and Rising Stars

Author:

Andrej Jakovljević

Vuk Vuković

Implementation and analysis:

Drago Šmitran

Vuk Vuković

Statement:

After making a strategic plan with carriers for the expansion of the mobile network throughout the country, the government decided to cover rural areas with the last generation of 5G network.

Since 5G antenna towers will be built in the area of mainly private properties, the government needs an easy way to find information about landowners for each **property** partially or fully contained in the planned building area.

The planned building area is represented as a rectangle with **width** and **height** sides.

Every 5G antenna tower occupies a **circle** with a center in **(x,y)** and radius **r**.

There is a database of the Geodetic Institute containing information about each property. Each property is defined with its **identification number** and **polygon** represented as an array of **(x,y)** points in the counter-clockwise direction.

Your task is to build an IT system which can handle queries of type **(x, y, r)** in which **(x,y)** represents a circle center, while r represents its radius. The IT system should return the total area of the properties that need to be acquired for the building of a tower so that the government can estimate the price. Furthermore, the system should return a list of **identification numbers** of these properties (so that the owners can be contacted for land acquisition).

A property needs to be acquired if the circle of the antenna tower is intersecting or touching it.

Input:

The first line contains the size of the building area as double values **width**, **height**, and an integer **n** - the number of properties in the database.

Each of the next n lines contains the description of a single property in the form of an integer number **v** ($3 < v \leq 40$) - the number of points that define a property, as well as 2v double numbers - the coordinates **(x, y)** of each property point. Line i ($0 \leq i < n$) contains the information for the property with id i.

The next line contains an integer **q** ($1 \leq n * q \leq 10^6$) - the number of queries.

Each of the next q lines contains double values **x, y, r** - the coordinates of an antenna circle center (x, y) and its radius r.

Output:

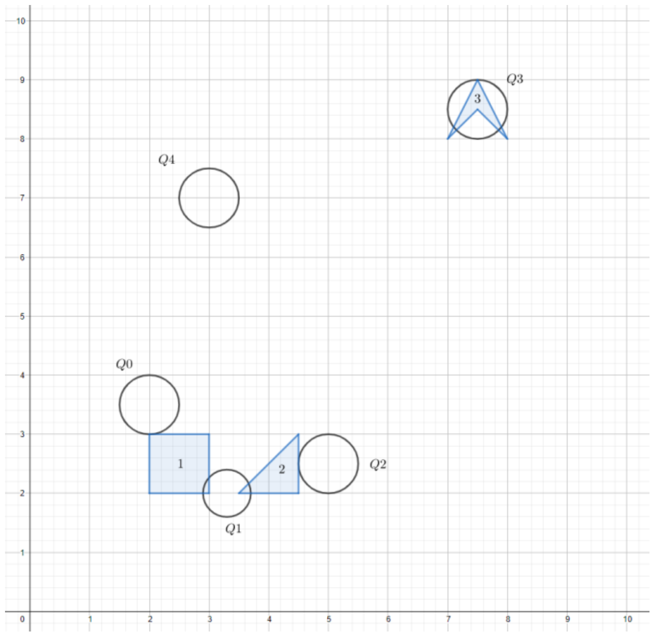
For each of the **q** queries, your program should output a line containing the total area of all the properties that need to be acquired, an integer representing the number of such properties, as well as the list of ids of these properties (separated by blank characters, arbitrary order).

Example input:

```
10 10 3
4 2 2 3 2 3 3 2 3
3 3.5 2 4.5 2 4.5 3
4 7 8 7.5 8.5 8 8 7.5 9
5
2 3.5 0.5
3.3 2 0.4
5 2.5 0.5
7.5 8.5 0.5
3 7 0.5
```

Example output:

```
1.000000 1 0
1.500000 2 0 1
0.500000 1 1
0.250000 1 2
0.000000 0
```



Notes:

You can assume that the land not covered with properties (polygons) is under the government's ownership and therefore does not need to be acquired.

The properties do not intersect with each other. Precision being used for solution checking is **10^{-4}.**

```
Time and memory limit: 3s / 256 MB
```

Solution and analysis

To solve the task, we have to be able to check if a circle intersects or touches a given polygon (property).

A circle is intersecting a polygon if 1) circle center is inside the polygon or 2) circle intersects or touches any of the polygon segments.

1) Checking whether a point is inside the polygon can be checked by using the *Ray casting* algorithm. The idea is to test how many times a ray, starting from the point and going in any fixed direction, intersects the edges of the polygon. If the point is on the outside of the polygon the ray will intersect its edge an even number of times. If the point is on the inside of the polygon then it will intersect the edge an odd number of times. Note that one must specially handle cases when the point is on the edge of the polygon or if the ray intersects a polygon point.
2) Checking whether a circle intersects a segment can be accomplished using basic geometry and solving a quadratic equation.
    1. Calculate a line defined by the segment in the form $ax + by + c = 0$.
    2. Circle with center (x_0, y_0) and radius r can be represented using the following equation: $(x - x_0)^2 + (y - y_0)^2 = r^2$
    3. If the following system of equations has solution(s) and at least one of the solutions is inside the limits of a segment, the circle intersects or touches the polygon.

Intersection of circle and polygon can be checked in linear time, with respect to the number of points of the polygon.

Polygon area can be calculated using the *Shoelace formula*.

$$Area = {}^1/_2 \left| \sum_{i=0}^{n-2} x_i y_{i+1} + x_{n-1} y_0 - \sum_{i=0}^{n-2} x_{i+1} y_i - x_0 y_{n-1} \right|$$

Polygon area can be calculated in linear time, with respect to the number of points of the polygon.

Looking at the constraints, we can conclude that this task can be solved in $O(N * Q * V)$. Therefore, for each query, we can iterate through all the properties and check whether the property's polygon is intersecting the given circle. In the case of intersection, we should calculate the property's area and add it to the total sum of areas of properties that are intersecting the given circle. Also, the id of the property should be temporarily stored for further output. Note that we don't have to check if some two properties intersect when summing these areas, as there is a constraint in the problem which does not allow the intersection of different polygons.

Problem I: BubbleSquare Tokens

Premier League and Rising Stars

Author:

Bojan Vučković

Implementation and analysis:

Nikola Pešić

Bojan Vučković

Statement:

BubbleSquare social network is celebrating its 13th anniversary and rewarding its members with special edition BubbleSquare tokens. Every member receives one personal token. Also, two additional tokens are awarded to each member for every friend they have on the network. Yet, there is a twist - everyone should end up with a different number of tokens from all their friends. Each member may return one received token. Also, each pair of friends may agree to each return one or two tokens they have obtained on behalf of their friendship.

Input:

The first line of input contains two integer numbers n and k – the number of members in the network and the number of friendships.

Next k lines contain two integer numbers a_i and b_i ($1 \le a_i, b_i \le n$, $a_i \ne b_i$) - meaning members a_i and b_i are friends.

Output:

The first line of output should specify the number of members who are keeping their personal token.

The second line should contain the space separated list of members who are keeping their personal token.

Each of the following k lines should contain three space separated numbers, representing friend pairs and the number of tokens each of them gets on behalf of their friendship.

Constraints:

- $2 \le n \le 12500$
- $1 \le k \le 1000000$

Example input 1:

```
2 1
1 2
```

Example output 1:

```
1
1
1 2 0
```

Explanation 1:

In the first test case, only the first member will keep its personal token and no tokens will be awarded for friendship between the first and the second member.

Example input 2:

```
3 3
1 2
1 3
2 3
```

Example output 2:

```
0
1 2 1
1 3 2
2 3 0
```

Explanation 2:

In the second test case, none of the members will keep their personal tokens. The first member will receive two tokens (for friendship with the third member), the second member will receive one token (for friendship with the third member) and the third member will receive three tokens (for friendships with the first and the second member).

```
Time and memory limit: 3s / 256 MB
```

Solution and analysis

Let X_i denote members of the network, where $1 < i \leq n$. At the start, let every member keep their personal token, and let everyone take one token for every friendship in the network. Let $X_1, \ldots, X_n$ be arbitrary order of members. Starting from $i = 2$ till $i = n$, increasing by 1, do the following. Let $Y = \{Y_1, .., Y_m\}$ be friends of X_i preceding it, that is for every j in $[1, \ldots, m]$, Y_j is a friend of X_i from $\{X_1, \ldots, X_{i-1}\}$. Now, let k be the number of members from Y that have taken their personal token and l be number of them that did not take the token; that is $k + l = m$. Let p be current sum of tokens of X_i. Since there are $m + 1$ numbers in interval $[p - l, p + k]$, there is at least one number among them that differs from number of tokens of each member from Y. Let that be $q = p + r, -l \leq r \leq k$. If $r < 0$, choose arbitrary r members of Y that did not take personal token. Now, let each of them take their token and together with X_i return one token they obtained on behalf of their friendship. If $r > 0$, choose arbitrary r members of Y that took personal token. Let each of them return their token and together with X_i each take one more token on behalf of their friendship. After these operations, all members X_j , with $j < i$, will remain with the same number of tokens as before, while X_i will now have different number of tokens from all of its friends from Y. Continuing this way until X_n we obtain distribution of tokens satisfying the problem's condition.

Problem J: Wakanda forever

Premier League and Rising Stars

Author:

Aleksandar Lukač

Implementation and analysis:

Aleksandar Lukač

Dušan Živanović

Nikola Pešić

Statement:

In the Kingdom of Wakanda, the 2020 economic crisis has made a great impact on each city and its surrounding area. The cities have made a plan to build a fast train rail between them to boost the economy, but because of the insufficient funds, each city can only build a rail with one other city, and they want to do it together.

The cities which are paired up in the plan will share the cost of building the rail between them, and one city might need to pay more than the other. Each city knows the estimated cost of building their part of the rail to every other city. One city cannot have the same cost of building the rail with two different cities.
If in the plan there are two cities that are not connected, but the cost to create a rail between them is lower for each of them than the cost of building the rail with their current pairs, then that plan is not acceptable and the collaboration will not go on. Your task is to create a suitable plan for the cities (pairing of the cities) or say that such plan does not exist.

Input:

The first line contains one integer N ($2 \le N \le 10^3$) - the number of cities.

Each of the next N lines contains N−1 integers $A_{i,1}, A_{i,2}, \ldots, A_{i,i-1}, A_{i,i+1}, \ldots, A_{i,N-1}$ ($1 \le A_{i,j} \le 10^9$) - where $A_{i,j}$ represents the cost for city i to build the rail to city j. Note that in each line $A_{i,i}$ is skipped.

Output:

Output should contain N integers $O_1, O_2, \ldots, O_N$, where O_i represents the city with which city i should build the rail with, or -1 if it is not possible to find the stable pairing.

Constraints:

- $1 < N < 1001$
- $1 < A_{i,j} < 1000000001$

Input example 1:

```
4
35 19 20
76 14 75
23 43 78
14 76 98
```

Output example 1:

```
3
4
1
2
```

Input example 2:

```
4
2 5 8
7 1 12
4 6 7
8 4 5
```

Output example 2:

```
-1
```

`Time and memory limit: 1.5s / 256 MB`

Solution and analysis

Let's interpret cities as nodes in a complete digraph (directed graph) and cost for city A to build its half of the railway with city B is the cost of an edge pointed from A to B.
After that, we should sort the lengths of all nodes pointed from each node and store them (lengths are no longer important, just the preference list of favored nodes for each node). Now we can look at this problem as a standard stable matching problem (Stable roommate problem).
An efficient algorithm was proposed by Irving in 1985[1]. The algorithm will determine, for any instance of the problem, whether a stable matching exists, and if so, will find such a matching. Irving's algorithm has O(n^2) complexity, provided suitable data structures are used to implement the necessary manipulation of the preference lists and identification of rotations.

[1] https://www.sciencedirect.com/science/article/abs/pii/0196677485900331?via%3Dihub

Problem K: Lookup tables

Premier League and Rising Stars

Author:

Dušan Živanović

Implementation and analysis:

Danilo Tonić

Dušan Živanović

Statement:

John has Q closed intervals of consecutive 2K-bit numbers $[l_i, r_i]$ and one 16-bit value v_i for each interval $(0 \le i < Q)$.

John wants to implement a function F that maps 2K-bit numbers to 16-bit numbers in such a way that inputs from each interval are mapped to that interval's value. In other words:

$$F(x) = v_i, \text{ for every } 0 \le i < Q, \text{ and every } x \in [l_i, r_i].$$

The output of F for other inputs is unimportant.

John wants to make his implementation of F fast, so he has decided to use lookup tables. A single 2K-bit lookup table would be too large to fit in the memory, so instead John plans to use two K-bit lookup tables, LSBTable and MSBTable. His implementation will look like this:

F(x) = LSBTable[lowKBits(x)] & MSBTable[highKBits(x)]

In other words, it returns the "bitwise and" of results of looking up the K least significant bits in LSBTable and the K most significant bits in MSBTable.

John needs your help. Given K, Q and Q intervals $[l_i, r_i]$ and values v_i, find any two lookup tables which can implement F or report that such tables do not exist.

Input:

The first line contains two integers K and Q ($1 \le K \le 16$, $1 \le Q \le 2 \cdot 10^5$).
Each of the next Q lines contains three integers l_i, r_i and v_i. ($0 \le l_i \le r_i < 2^{2K}$, $0 \le v_i < 2^{16}$).

Output:

On the first line output "possible" (without quotes) if two tables satisfying the conditions exist, or "impossible" (without quotes) if they do not exist.

If a solution exists, in the next $2 \cdot 2^K$ lines your program should output all values of the two lookup tables (LSBTable and MSBTable) it found. When there are multiple pairs of tables satisfying the conditions, your program may output any such pair.

On lines 1 + i output LSBTable[i]. ($0 \le i < 2K$, $0 \le \text{LSBTable}[i] < 2^{16}$).

On lines $1 + 2^K + i$ output MSBTable[i]. ($0 \le i < 2K$, $0 \le \text{MSBTable[i]} < 2^{16}$).

Example input 1:

```
1 2
0 2 1
3 3 3
```

Example output 1:

```
possible
1
3
1
3
```

Explanation 1:

In the first sample, tables LSBTable=[1,3] and MSBTable=[1,3] satisfy the conditions:
F[0]=LSBTable[0]&MSBTable[0]=1&1=1,
F[1]=LSBTable[1]&MSBTable[0]=3&1=1,
F[2]=LSBTable[0]&MSBTable[1]=1&3=1,
F[3]=LSBTable[1]&MSBTable[1]=3&3=3.

Example input 2:

```
2 4
4 5 3
6 7 2
0 3 0
12 13 1
```

Example output 2:

```
possible
3
3
2
2
0
3
0
1
```

Explanation 2:

In the second sample, tables LSBTable = [3,3,2,2] and MSBTable = [0,3,0,1] satisfy all the conditions.

Example input 3:

```
2 3
4 4 3
5 6 2
12 14 1
```

Example output 3:

```
impossible
```

Explanation 3:

There are no two lookup tables which can satisfy the conditions.

```
Time and memory limit: 1s / 256 MB
```

Solution and analysis

We can solve this problem by considering each of the 16-bits of v_is and the lookup tables separately. If we can find a solution for each of the 16 bits, then combining the bits of these 16 pairs of 1-bit lookup tables will produce a pair of 16-bit lookup tables that solves the 16-bit problem. The rest of the explanation will deal with 1-bit v_is and lookup tables.

The constraints of the problem pose a system of equations involving the two lookup tables. Whenever the problem requires that F[x] = v it requires that msb[highKbits(x)] & lsb[lowKbits(x)] = v. Equations where v = 1 are easy to deal with, because we know that both msb[highKbits(x)] and lsb[lowKbits(x)] need to be 1. Equations where v = 0 are a bit tougher because there is a choice of which term is zero. To deal with this we set to zero all terms which are not set to one while satisfying equations with v = 1. If this does not satisfy all equations with v = 0 then the solution does not exist, because setting any additional term to 0 will invalidate some equation with v = 1.

That gives us a simple algorithm to solve the problem. Start with empty msb and lsb. For each query and every bit of v_i set to 1, go through all x in [l_i, r_i] and set the corresponding bits of msb and lsb to 1. After that go through all queries and all bits of v_i which are zero, and for every x in [l_i, r_i] check that either the corresponding bit of msb or lsb is 0. If for one query it is not then the solution doesn't exist, otherwise the msb and lsb you found are a solution.

This algorithm runs in O(16*Q*2^k), which is way too slow. To make our algorithm faster we can do range operations on the msb and lsb tables. Instead of going through numbers in [l,r] one at a time, we will split that interval into three pieces: from l to the largest number with the same highKbits as l, from the smallest number with a larger highKbits than l to the largest number with highKbits less than that of r, and from the first number with the same highKbits as r to r. All these intervals represent all numbers with highKbits in [high1, high2] and lowKbits in [low1, low2] for some intervals [high1, high2], [low1, low2].

If we want to set msb and lsb to 1 for all numbers in such an interval, we need to set msb[high1 to hig2] to 1 and lsb[low1 to low2] to 1.

If we want to check that for each value in such a range either msb or lsb is zero, then we need to check if either msb is zero on the whole range from high1 to high2 or lsb is zero on the whole range from low1 to low2.

The operations of setting a subarray to 1 or checking if there is a 1 in a subarray can be done in O(1) using prefix sums. So, the whole problem can be solved in O(16*Q + 2^K).

Problem L: Dušan's Railway

Premier League and Rising Stars

Author:

Lazar Milenković

Implementation and analysis:

Nikola Aleksić

Statement:

As you may already know, Dušan is keen on playing with railway models. He has a big map with the cities that are connected with railways. His map can be seen as a graph where the vertices are the cities and the railways connecting them are the edges. So far, the graph corresponding to his map is a tree. As you already know, a tree is a connected acyclic undirected graph.
Dušan is curious to find out whether his railway can be optimized somehow. He wants to add so-called *shortcuts*, which are also the railways connecting pairs of the cities. This shortcut will *represent* the railways in the unique path in the tree between the pair of the cities it connects. Since Dušan does not like repeating the railways, he has also defined good paths in his newly obtained network (notice that after adding the shortcuts, his graph is no more a tree). He calls a path *good* if no edge appears more than once, either as a regular railway edge or as an edge represented by some shortcut (Every shortcut in a good path has length 1, but uses up all the edges it represents - they cannot appear again in that path). Having defined good paths, he defines *good distance* between two cities to be the length of the shortest good path between them. Finally, the *shortcutting diameter* of his network is the largest good distance between any two cities.

Now he is curious to find out whether it is possible to achieve shortcutting diameter k, while adding as few shortcuts as possible.

Your solution should add no more than 10 · n shortcuts.

Input:

The first line in the standard input contains an integer n ($1 \le n \le 10^4$), representing the number of the cities in Dušan's railway map, and an integer k ($3 \le k \le n$) representing the shortcutting diameter that he wants to achieve.

Each of the following n - 1 lines will contain two integers u_i and v_i ($1 \le u_i, v_i \le n, u_i \ne v_i$)), meaning that there is a railway between cities u_i and v_i.

Output:

The first line of the output should contain a number t representing the number of the shortcuts that were added.
Each of the following t lines should contain two integers u_i and v_i, signifying that a shortcut is added between cities u_i and v_i.

Example input:

```
10 3
1 2
2 3
3 4
4 5
5 6
6 7
7 8
8 9
9 10
```

Example output:

```
8
3 7
3 5
3 6
3 1
7 9
7 10
7 4
7 5
```

Explanation:

Notice that adding a shortcut between all cities and city 1 will make a graph theoretic diameter become 2. On the other hand, the paths obtained that way might not be good, since some of the edges might get duplicated. In the example, adding a shortcut between all cities and city 1 does not create a valid solution, because for cities 5 and 10 the path that uses shortcuts 5-1 and 1-10 is not valid because it uses edges 1-2, 2-3, 3-4, 4-5 twice.

`Time and memory limit: 1s / 256 MB`

Solution and analysis

Consider the following procedure for constructing the bridges. First, arbitrarily root the tree and find the set of *cut vertices*, so that their respective subtree sizes are at most $\sqrt{n}$. Then, connect with bridges each pair of cut vertices between each other. There will be $O(n)$ bridges added in this step. For each cut vertex v , consider all the subtrees neighboring it. Add a bridge between v and every node in all the neighboring subtrees. It is not hard to see that the number of the bridges here will also be linear. Finally, recursively repeat the procedure for all the subtrees. There will be at most $O(\log\log n)$ recursive calls, each of which contributes $O(n)$ bridges, yielding a total of $O(n \log\log n)$ bridges added. This completes the description of the algorithm.

For the proof of correctness, consider two vertices u and v , and consider the last recursive calls when they were in the same subtree. Let $c(u)$ be the cut vertex neighboring the subtree of u and let $c(v)$ be the cut vertex neighboring the subtree of v . By the construction above, there exist bridges between $c(u)$ and u , as well as between $c(v)$ and v . Finally, since we connected $c(u)$ with $c(v)$ with an edge, it follows that there exists a path with at most 3 bridges between u and v . It is not hard to see that such path is good. A similar idea can be applied to the case when $c(u) = c(v)$.

An interesting thing is to notice that one cannot achieve better than $O(n \log\log n)$ bridges, when $k = \mathbf{3}$. Similarly, there is an algorithm that achieves $O(n \log n)$ bridges when $k = 2,$ and this is also the best that one can achieve. Finally, a generalized solution that achieves $O\big(n\alpha_k(n)\big)$, where α_k is an Ackerman-type function, is given in [1].

[1] Shay Solomon. Sparse Euclidean Spanners with Tiny Diameter**.** ACM Transaction on Algorithms 2013.

Problem M: Milutin's Plums

Premier League and Rising Stars

Author: *Implementation and analysis:*

Lazar Milenković ***Filip Ćosović***

Statement:

As you all know, the plum harvesting season is open! Little Milutin had his plums planted in an orchard that can be represented as an n by m matrix. While he was harvesting, he wrote the heights of all trees in a matrix of dimensions n by m.

At night, when he has spare time, he likes to perform various statistics on his trees. This time, he is curious to find out the height of his lowest tree. So far, he has discovered some interesting properties of his orchard. There is one particular property that he thinks is useful for finding the tree with the smallest height.

Formally, let L(i) be the leftmost tree with the smallest height in the i^{th} row of his orchard. He knows that $L(i) \leq L(i+1)$ for all $1 \leq i \leq n-1$. Moreover, if he takes a submatrix induced by any subset of rows and any subset of columns, $L(i) \leq L(i+1)$ will hold for all $1 \leq i \leq n'-1$, where n' is the number of rows in that submatrix.

Since the season is at its peak and he is short on time, he asks you to help him find the plum tree with the minimal height.

Input:

This problem is interactive.

The first line of input will contain two integers n and m, representing the number of the rows and the number of the columns in Milutin's orchard. It is guaranteed that $1 \leq n, m \leq 10^6$.

The following lines will contain the answers to your queries.

Output:

Once you have found the minimum value r, you should print ! r to the standard output.

Interaction:

Your code is allowed to query for an entry (i, j) of a matrix (i.e. get the height of the tree which is in the i^{th} row and j^{th} column). The query should be formatted as ? i j, so that $1 \leq i \leq n$ and $1 \leq j \leq m$.

You may assume that the entries of the matrix will be integers between 1 and 10^9.

Your solution should use not more than $4 \cdot (n + m)$ queries.

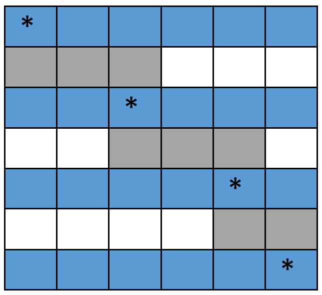
This is an interactive problem. You have to use a flush operation right after printing each line. For example, in C++ you should use the function `fflush(stdout)`*, in Java —* `System.out.flush()`*, in Pascal —* `flush(output)` *and in Python —* `sys.stdout.flush()`*.*

`Time and memory limit: 1s / 1024 MB`

Solution and analysis

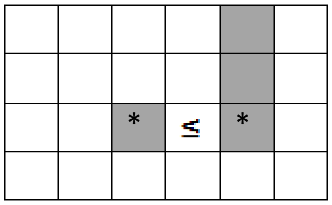
The matrices satisfying L(i) ≤ L(i+1) for all 1 ≤ i ≤ n−1 are known as *monotone matrices*. If the condition holds for all the submatrices, as defined in the statement, then a matrix is called *totally monotone matrix*. We will proceed to explain the O(n + m) procedure for finding the smallest entry in such matrices.

Let's consider the case where the number of the columns m is at most the number of the rows n. In that case, we can recursively find the row minimas for the matrix consisting of every second row, as depicted below. Notice that the matrix induced by blue rows is also totally monotone, simply by the definition of total monotonicity.

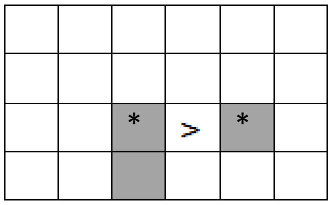


Suppose that the fields denoted by asterisks are the row minimas returned by the recursive call. Using the monotonicity of the matrix, we know that minimum elements in the row i could be found only in the columns between L(i – 1) and L(i + 1), as defined above by the grey cells. In total, there will be at most O(n + m) such cells.

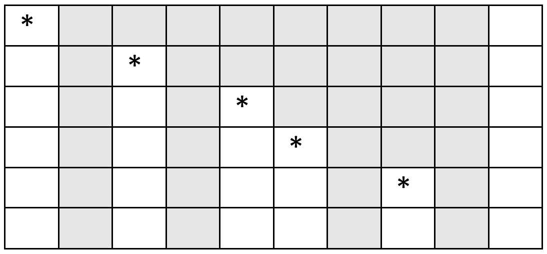
Now, consider the case where the number of the columns is bigger than the number of the rows. Using a clever pruning procedure, we can reduce the number of the columns to at most n. Consider comparing two cells in the same rows, M(i, j) and M(i, k) .The following two cases may arise. First, M(i, j) ≤ M(i, k), as shown below. In this case, all the entries from M(i, k) and above are not the candidates for the leftmost minimum element.



In the other case, when M(i, j) ≥ M(i, k), we can similarly exclude all the entries starting from M(i, j) below, as depicted below.



This way, we can completely eliminate some columns and keep a stack that will contain at most n columns such that their top entries are eliminated. For an illustration, consider the example below, where gray entries denote the eliminated elements and the indices on the stack are denoted by asterisks.



This pruning procedure performs at most O(m) queries and yields a matrix that has number of the columns less than or equal to the number of the rows. The overall algorithm will then take the following form. First, check if m > n and if so, prune it as above. Perform a recursive call to find the row minima in every second row. Finally, find the row minima in the remaining rows in linear time. Overall, it is not hard to see that such recursive algorithm results in O(n + m) running time, and thus the same limit on the number of matrix queries applies.

The explained procedure is known as SMAWK (pronounced as *smoke*) algorithm, after properly permuting authors' initials [1]. For a good and more detailed explanation, see https://jeffe.cs.illinois.edu/teaching/algorithms/notes/D-faster-dynprog.pdf.

[1] Alok Aggarwal, Maria M. Klawe, Shloo Moran, Peter Shor, Robert Wilber. *Geometric applications of a matrix-searching algorithm*. Algorithmica 1987.

Problem N: BubbleCup hypothesis

Premier League and Rising Stars

Author:

Đorđe Stupar

Implementation and analysis:

Nikola Aleksić

Nikola Pešić

Statement:

The BubbleCup hypothesis has stood unsolved for 130 years. Whoever proves the hypothesis will be regarded as one of the greatest mathematicians of our time! Famous mathematician Jerry Mao has managed to reduce the hypothesis to this problem:

Given a number m, how many polynomials P with coefficients in set {0, 1, 2, 3, 4, 5, 6, 7} have: P(2) = m?

Help Jerry Mao solve the long-standing problem!

Input:

The first line contains a single integer t - number of test cases. In the next line, there are t numbers, m_i - meaning that in case i you should solve for number m_i.

Output:

For each test case i, print the answer in separate lines: number of polynomials P as described in statement so that P(2) = m, modulo $10^9 + 7$.

Constraints:

- $1 \le t \le 500\,000$
- $1 \le m_i \le 10^{18}$

Example input:

```
2
2  4
```

Example output:

```
2
4
```

Explanation:

In the first case, for m = 2, polynomials that satisfy the constraint are x and 2.

In the second case, for m = 4, polynomials that satisfy the constraint are x^2, x + 2, 2x and 4.

`Time and memory limit: 1s / 256 MB`

Solution and analysis

Let's prove that the number of polynomials P, as described in the statement, such that $P(2) = m_i$ is the same as the number of solutions of equation $x + 2y + 4z = m_i$.

Let $P(x) = c_0 + c_1x + \dots + c_nx^n$ and $P(x) = m_i$. And let $x_nx_{n-1}\dots x_1x_0$, $y_ny_{n-1}\dots y_1y_0$ and $z_nz_{n-1}\dots z_1z_0$ be x, y and z written in binary, respectively. For each c_i we can determine the i-th binary digit of x, y and z in the following way: let $c_{i2}c_{i1}c_{i0}$ be c_i written in binary, x_i is c_{i0}, y_i is c_{i1} and z_i is c_{i2} (This is a bijection).

Now we need to find the number of solutions of $x + 2y + 4z = m_i$. Number of solutions for equation $x + 2y = m$ is $\left\lfloor \frac{m}{2} \right\rfloor + 1$ (If we fix y, x is predetermined. There are $\left\lfloor \frac{m}{2} \right\rfloor + 1$ ways to fix y). After rewriting $x + 2y + 4z = m_i$ as $x + 2(y + 2z) = m_i$, and fix $y + 2z = i$, x is predetermined. As proved earlier, number of solutions for this is $\left\lfloor \frac{i}{2} \right\rfloor + 1$. We can fix $y + 2z$ to all numbers between 0 and $\left\lfloor \frac{m}{2} \right\rfloor$, so the number of solutions is $\sum_{i=0}^{\left\lfloor \frac{m}{2} \right\rfloor} \left\lfloor \frac{i}{2} \right\rfloor + 1$. Closed form of the sum is given with the following formula: $\left(\left\lfloor \frac{m}{4} \right\rfloor + 1\right)\left(\left\lfloor \frac{m}{2} \right\rfloor - \left\lfloor \frac{m}{4} \right\rfloor + 1\right)$. Complexity of the solution is $O(t)$.

Alternate derivation

Let $P(x) = c_0 + c_1x + \dots + c_nx^n + \cdots$ and $P(x) = m_i$. If we rewrite $P(x)$ as
$(c_0 + 8c_3 + 64c_6 + \cdots 8^kc_{3k} + \cdots) + 2 * (c_1 + 8c_4 + 64c_7 + \cdots) + 4 * (c_2 + 8c_5 + 64c_8 + \cdots)$
we can notice that all 3 parts correspond to 3 numbers written in octal. From this follows that there is a bijection between number of polynomials with $P(2) = m_i$ and solutions to $x + 2y + 4z = m$.

Similar as in first solution, we use the fact that the number of solutions for $x + 2y = m$ is $\left\lfloor \frac{m}{2} \right\rfloor + 1$.

If we move $4z$ to the other side, we get the equation $x + 2y = m_i - 4z$. If we some the number of solutions for different values of z we get:

$$\sum_{z=0}^{\left\lfloor \frac{m}{4} \right\rfloor} \left(\left\lfloor \frac{m-4z}{2} \right\rfloor + 1\right) = \sum_{z=0}^{\left\lfloor \frac{m}{4} \right\rfloor} \left(\left\lfloor \frac{m}{2} \right\rfloor - 2z + 1\right) = \left\lfloor \frac{m}{2} \right\rfloor \left(\left\lfloor \frac{m}{4} \right\rfloor + 1\right) - \left\lfloor \frac{m}{4} \right\rfloor \left(\left\lfloor \frac{m}{4} \right\rfloor + 1\right) + \left(\left\lfloor \frac{m}{4} \right\rfloor + 1\right) =$$

$$\left(\left\lfloor \frac{m}{4} \right\rfloor + 1\right)\left(\left\lfloor \frac{m}{2} \right\rfloor - \left\lfloor \frac{m}{4} \right\rfloor + 1\right)$$

Problem O: Virus

Premier League only

Author:

Svetislav Gajić

Filip Ćosović

Implementation and analysis:

Nikola Pešić

Svetislav Gajić

Filip Ćosović

Statement:

In Bubbleland a group of special programming forces gets a top-secret job to calculate the number of potentially infected people by a new unknown virus. The state has a population of **n** people and every day there is new information about new contacts between people. The job of the special programming forces is to calculate how many contacts a given person had in the last **k** days.

The new virus has an incubation period of **k** days, and after that time people are considered as non-infectious. Because the new virus is extremely dangerous, the government marks as suspicious everybody who had direct or indirect contact in the last **k** days, independently of the order of contacts.

This virus is very strange, and people cannot get durable immunity.

You need to help the special programming forces to calculate the number of suspicious people for a given person (number of people who had contact with a given person).

There are 3 given inputs at the beginning: **n** - population, **q** - number of queries, and **k** - virus incubation time in days. Each query is one of three types:

1. (**x**, **y**) person xx and person **y** met that day (**x** ≠ **y**).
2. (**z**) return the number of people in contact with **z**, counting himself.
3. The end of the current day moves on to the next day.

Input:

The first line of input contains three integers n ($1 \le n \le 10^5$) the number of people in the state, q ($1 \le q \le$ 500 000) number of queries and k ($1 \le k \le$ 100 000) virus incubation time in days.

Each of the next q lines starts with an integer t ($1 \le t \le 3$) the type of the query.

A pair of integers x and y ($1 \le x, y \le n$) follows in the query of the first type.

An integer i ($1 \le i \le n$) follows in the query of the second type.

Query of third type does not have the following number.

Output:

For the queries of the second type print in a separate line the current number of people in contact with a given person.

Example input 1:

```
5 12 1
1 1 2
1 1 3
1 3 4
2 4
2 5
3
2 1
1 1 2
1 3 2
2 1
3
2 1
```

Example output 1:

```
4
1
1
3
1
```

Example input 2:

```
5 12 2
1 1 2
1 1 3
1 3 4
2 4
2 5
3
2 1
1 1 2
1 3 2
2 1
3
2 1
```

Example output 2:

```
4
1
4
4
3
```

Example input 3:

```
10 25 2
1 9 3
2 5
1 1 3
1 3 1
2 2
1 8 3
1 5 6
3
1 9 2
1 8 3
2 9
1 3 1
2 5
1 6 4
3
3
2 4
3
1 10 9
1 1 7
3
2 2
3
1 5 6
1 1 4
```

Example output 3:

```
1
1
5
2
1
1
```

`Time and memory limit: 5s / 256 MB`

Solution and analysis

Since every edge expires after exactly K days, we can solve dynamic connectivity offline.

First, we construct a segment tree over the K time moments. Then, each operation of type “edge e is available from time L to time R” is inserted in the O(log(n)) segment tree nodes which cover the interval [L, R]. This step takes O(n*log(n)) time and memory.

Then we traverse the segment tree by using DFS starting from the root. During the traversal we maintain the connected components of the graph using disjoint sets. When we enter a segment tree node, we perform a Union operation for each edge which is stored in that node. We also store the successful Union operations in a stack of operations so that, when we exit a segment tree node, we can undo all the Union operations performed when entering the node.

Before exiting a leaf node of the segment tree node which corresponds to a query of second type, we have the answer as the size of set that contains node z.

Problem P: Growing flowers

Premier League only

Author:

Nikola Pešić

Implementation and analysis:

Dušan Živanović

Drago Šmitran

Statement:

Sarah has always been a lover of nature, and a couple of years ago she saved up enough money to travel the world and explore all the things built by nature over its lifetime on earth. During this time, she visited some truly special places which were left untouched for centuries, and did amazing things, from watching icebergs in the freezing weather to scuba-diving in the oceans and admiring the sea life, residing unseen. These experiences were enhanced with breathtaking views built by mountains over time and left there for visitors to see for years on end. Over time, all these expeditions took a toll on Sarah and culminated in her decision to settle down in the suburbs and live a quiet life.

However, as Sarah's love for nature never faded, she started growing flowers in her garden in an attempt to stay connected with nature. In the beginning, she planted only blue orchids, but over time she started using different flower types to add variety to her collection of flowers. This collection of flowers can be represented as an array of N flowers and the i^{th} of them has a type associated with it, denoted as A_i. Each resident, passing by her collection and limited by the width of his view, can only see K contiguous flowers at each moment in time. To see the whole collection, the resident will look at the first K contiguous flowers A_1, A_2, …, A_k, then shift his view by one flower and look at the next section of K contiguous flowers A_2, A_3, …, A_{K+1} and so on until they scan the whole collection, ending with section A_{N-K+1}, …, A_{N-1}, A_N.

Each resident determines the beautiness of a section of K flowers as the number of distinct flower types in that section. Furthermore, the **beautiness** of the whole collection is calculated by summing the beautiness values of each contiguous section. Formally, beautiness B_i of a section starting at the i^{th} position is calculated as B_i = distinct(A_i, A_{i+1}, …, A_{i+K-1}) and **beautiness** of the collection B is calculated as B = $B_1 + B_2 + … + B_{N-K+1}$.

In addition, as Sarah wants to keep her collection of flowers have a fresh feel, she can also pick two points L and R, dispose flowers between those two points and plant new flowers, all of them being the same type.

You will be given Q queries and each of those queries will be of the following two types:

- You will be given three integers L, R, X describing that Sarah has planted flowers of type X between positions L and R inclusive. Formally, the collection is changed so that A[i] = X for all I in range [L..R].
- You will be given integer K, the width of the resident's view and you have to determine the beautiness value B resident has associated with the collection

For each query of the second type print the result - beautiness B of the collection.

Input:

First line contains two integers N and Q - number of flowers and the number of queries, respectively.

The second line contains N integers A_1, A_2, ..., A_N --- where A_i represents type of the ith flower.

Each of the next Q lines describe queries and start with integer T.

- If T=1, there will be three more integers in the line L, R, X - L and R describing boundaries and X describing the flower type.
- If T=2, there will be one more integer in the line K - resident's width of view.

Output:

For each query of the second type print the beautiness B of the collection.

Constraints

- $1 \le N, Q \le 10^5$
- $T \in \{1, 2\}$
- $1 \le A_i, X \le 10^9$
- $1 \le L, R, K \le N$

Sample input

```
5 5
1 2 3 4 5
2 3
1 1 2 5
2 4
1 2 4 5
2 2
```

Sample output

```
9
6
4
```

Explanation

Initially the collection is [1, 2, 3, 4, 5]. In the first query K = 3, we consider sections of three flowers with the first being [1, 2, 3]. Since beautiness of the section is the number of distinct flower types in that section, B_1=3. Second section is [2, 3, 4] and B_2=3. Third section is [3, 4, 5] and B_3=3, since the flower types are all distinct. The beautiness value resident has associated with the collection is $B = B_1 + B_2 + B_3 = 3 + 3 + 3 = 9$.

After the second query, the collection becomes [5, 5, 3, 4, 5].

For the third query K = 4, so we consider sections of four flowers with the first being [5, 5, 3, 4]. There are three distinct flower types [5, 3, 4] in this section, so $B_1 = 3$. Second section [5, 3, 4, 5] also has 3 distinct flower types, so $B_2=3$. The beautiness value resident has associated with the collection is $B = B_1 + B_2 = 3 + 3 = 6$

After the fourth query, the collection becomes [5, 5, 5, 5, 5].

For the fifth query K = 2 and in this case all the four sections are same with each of them being [5, 5]. Beautiness of [5, 5] is 1 since there is only one distinct element in this section [5]. Beautiness of the whole collection is $B = B_1 + B_2 + B_3 + B_4 = 1 + 1 + 1 + 1 = 4$.

```
Time and memory limit: 4s / 256 MB
```

Solution and analysis

Assume that all the flower types are unique in all ranges and then subtract the number of the repeated flower types. Let's first solve the problem when no update queries exist. We will keep a segment tree where the sum up to the ith element (from position 1 to i) stores how much we need to subtract from the answer if the resident's width of view is i. Let's look at groups of same numbers, if a group contains numbers at positions $p_1, p_2, p_3, \ldots, p_k$ for every two adjacent positions ($p_1, p_2; p_2, p_3; p_3, p_4; \ldots; p_{k-1}, p_k$), we will subtract 1 for the intervals that contain both of these positions. For example, if we have the same number at index 3 and at index 5 (indices are indexed from 1) and there are 8 elements in the array, we have one subarray of size 3, two subarrays of size 4, three subarrays of size 5, three subarrays of size 6, two subarrays of size 7 and one subarray of size 8 that contain both of these positions. For this, in our segment tree we need to add 1 in the range from 3 to 5, and -1 in the range from 7 to 9.

Now to extend this solution to processing range updates, we can keep ranges of equal numbers and update a range of same numbers at once. If we have some ranges of numbers that are the same from l to r, that is the same as adding r-l pairs of adjacent positions. For example, if we have l=2 and r=4 and n=8, for pair at positions 2 and 3, we will add 1 in the range from 2 to 4 and -1 in the range from 7 to 9. And for positions 3 and 4, we will add 1 in the range from 2 to 5 and -1 in the range from 6 to 9. The updates for adding 1 can be grouped together and the updates for adding -1 can be grouped together. When grouped we can process them with an arithmetic progression segment tree.

When keeping the ranges of equal numbers, we will have an amortized linear number of changes.

The final complexity for the solution is $O((q + n) \log(n))$.