NOI 2024

16 March 2024

Tasks	Time Limit	Memory Limit
Task 1: Problem Setter	1 s	1024 MB
Task 2: Shops	2 s	1024 MB
Task 3: Toxic Gene 2	1 s	1024 MB
Task 4: Coin	3 s	1024 MB
Task 5: Field	2 s	2048 MB

Have Fun!

Task 1: `Problem Setter`

Over the past few years, Stuart has set several problems. Now he is considering whether to submit them to competitive programming contests (including this one!) for the honour of having those problems solved by the best competitive programmers of the country, or even the world.

There are c contests that Stuart can submit his problems to. Submitting any problem to the i-th contest will increase his satisfaction by $s[i]$. However, due to the structure of the contest as well as competition from fellow problem setters, Stuart thinks that the i-th contest will only accept problems that have a quality of at least $m[i]$. Note that Stuart can submit any number of problems to any contest, and each problem he submits will increase his satisfaction by $s[i]$.

Stuart has set p problems. He thinks that the j-th problem has a quality of $q[j]$. However, due to the difficulty of the problem preparation process, submitting the j-th problem to any contest will decrease his satisfaction by $d[j]$. Obviously, each problem can be submitted to at most one contest, or not submitted at all if he chooses not to.

Find the maximum amount of satisfaction that Stuart can gain by submitting the right problems to each contest. Note that if all ways to submit problems to contests will give Stuart negative satisfaction, then he can choose not to submit any problems for a final satisfaction of 0.

Input format

Your program must read from standard input.

The first line of input contains 2 space-separated integers, c and p, denoting the number of contests and number of problems respectively.

Then, c lines will follow. The i-th ($1 \leq i \leq c$) of these lines contains 2 space-separated integers $m[i]$ and $s[i]$, denoting the minimum problem quality and satisfaction gained from this contest respectively.

Another p lines follow. The j-th ($1 \leq j \leq p$) of these lines contains 2 spaced-separated integers $q[j]$ and $d[j]$, denoting the problem quality and satisfaction loss from this problem respectively.

Output format

Your program must print to standard output.

The output should contain one integer, the maximum satisfaction Stuart can gain.

The output should contain only a single integer. Do not print any additional text such as `'Enter a number'` or `'The answer is'`.

Subtasks

For all testcases, the input will satisfy the following bounds:

- $1 \le c, p \le 200\,000$
- $0 \le m[i], s[i], q[j], d[j] \le 1\,000\,000$ ($1 \le i \le c, 1 \le j \le p$)

Your program will be tested on input instances that satisfy the following restrictions:

Subtask	Marks	Additional Constraints
0	0	Sample Testcases
1	18	$1 \le c \le 1000, p = 1$
2	16	$1 \le c, p \le 1000$
3	26	$d[j] = 0$
4	40	No additional restrictions

Sample Testcase 1

This testcase is valid for subtasks 2 and 4.

Input	Output

3 3
2 5
1 1
8 3
3 2
9 4
1 3
	4

Sample Testcase 1 Explanation

There are 3 contests and 3 problems.

Stuart can submit problem 1 with quality 3 to contest 1 since its quality is higher than the contest's minimum quality, 2. He gains 5 - 2 = 3 satisfaction from submitting this problem. He can submit problem 2 with quality 9 to contest 1, and gains 5 - 4 = 1 satisfaction from this problem. In total, he gains 4 satisfaction.

Note that he chooses not to submit problem 3, and does not submit any problem to contests 2 and 3.

Sample Testcase 2

This testcase is valid for subtasks 2, 3 and 4.

Input	Output
`3 4`	`11`
`2 3`	
`1 1`	
`8 4`	
`2 0`	
`7 0`	
`1 0`	
`8 0`	

Sample Testcase 2 Explanation

There are 3 contests and 4 problems.

Stuart can submit problems 1 and 2 to contest 1, problem 3 to contest 2 and problem 4 to contest 3. He gains 3 $\times$ 2 + 1 + 4 = 11 satisfaction in total.

Sample Testcase 3

This testcase is valid for subtasks 1, 2 and 4.

Input	Output

`5 1`
`3 4`
`1 2`
`5 6`
`2 8`
`4 0`
`3 6`
	`2`

Sample Testcase 3 Explanation

There are 5 contests and 1 problem.

Stuart submits the only problem to contest 4, and gains $8 - 6 = 2$ satisfaction.

Task 2: `Shops`

James is the mayor of Yuland, which consists of n cities connected by m bidirectional roads of varying distances. It is possible to travel from any city to any other city using only the roads. Note that there might be multiple roads between the same pair of cities.

Each city can have either a bunny or duck shop but not both. Residents of each city want to collect **both** animals. The inconvenience of a city is defined as the maximum between the distance to the nearest bunny shop and the distance to the nearest duck shop.

James has not built the shops yet and needs your help to choose which cities to build which shops to **minimize** the maximum of inconveniences across all cities.

Input format

Your program must read from standard input.

The first line contains two integers n, m.

The next m lines contains 3 integers $u[i], v[i], w[i]$ representing a road connecting cities $u[i]$ and $v[i]$ of distance $w[i]$.

Output format

Your program must print to standard output.

The first line should be the minimal possible maximum of inconveniences across all cities.

The next line should be a string of n characters that are either `B` or `D` where the i^{th} character represents whether you choose to build a bunny or duck shop in the i^{th} city respectively. If there are multiple ways to build the shops such that the maximum inconvenience is as stated in the first line any will be accepted.

Subtasks

For all test cases, the input will satisfy the following bounds:

- $2 \leq n, m \leq 500\,000$

- $1 \leq u[i], v[i] \leq n$
- $1 \leq w[i] \leq 10^9$
- It is possible to travel from any city to any other city using only the roads

Your program will be tested on input instances that satisfy the following restrictions:

Subtask	**Marks**	**Additional Constraints**
0	0	Sample Testcases
1	7	$n \leq 16$
2	13	$m = n - 1, u[i] = i, v[i] = i + 1$
3	18	$m = n - 1$
4	24	$w[i] = 1$
5	38	No additional constraints

Sample Testcase 1

This testcase is valid for subtask 1,5.

Input	Output

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

Sample Testcase 1 Explanation

In this assignment, cities 1 and 2 have a bunny shop while city 3 has a duck shop. Hence, the inconvenience for each city would be $[2, 1, 1]$, the maximum being 2 from city 1.

Sample Testcase 2

This testcase is valid for subtask 1,5.

Input	Output
`5 6`	`9`
`3 2 3`	`DBDDB`
`4 2 1`	
`5 3 9`	
`1 3 5`	
`1 4 2`	
`2 3 1`	

Sample Testcase 2 Explanation

In this assignment, the inconvenience of each city would be $[3, 1, 1, 1, 9]$, the maximum being 9 from city 5.

Task 3: `Toxic Gene 2`

This is an interactive task. Note that only C++ is allowed for this task.

After a very successful career studying toxic bacteria, Benson the Rabbit has retired! James the Alien has taken over his lab and has recently discovered a new type of bacteria which he wants to investigate!

James has n species of bacteria, numbered 0 to $n-1$. Each of them falls into exactly one of 2 types: Regular or Toxic. t of them are Toxic. It is guaranteed that there is at least 1 Toxic and 1 Regular bacteria, i.e. $1 \leq t < n$. Note that James does not know the value of t.

James wants to identify the type of each bacteria. To analyze the bacteria, he bought a new machine to replace Benson's old one! This new machine allows James to place up to 1000 samples of bacteria in a row. Each sample consists of a single species, and multiple samples are allowed to be the same species.

After James has chosen the samples, the machine will run for a number of iterations. In each iteration, any Regular bacteria samples adjacent to a Toxic bacteria sample will die. The machine then shifts all surviving bacteria samples to fill the gaps such that the remaining bacteria form a single row.

The machine will run until the point where the next iteration would have the same number of surviving bacteria as the current iteration. This means that the machine will always run at least 1 iteration, even if no bacteria die in the first iteration.

The machine will then report to James the number of surviving bacteria samples after every iteration. This forms a single query. Each use of the machine takes time, and James does not have a lot of time. He may only use the machine up to 1000 times. Help James determine for each bacteria species, whether it is Regular or Toxic in as few queries as possible.

Implementation Details

This is an interactive task. Do not read from standard input or write to standard output.

You are to implement the following function.

- `void determine_type(int n)`

This function will be called exactly once per testcase. The function will be passed, n, representing the number of bacteria species James has.

You are allowed to call the following grader functions to complete the task:

- `std::vector<int> query_machine(std::vector<int> samples)`
- `void answer_type(std::vector<char> v)`

The function `query_machine` is called with a 1-dimensional array `samples`, describing the species of bacteria you place in the machine, in the order that you place them. The size of `samples` cannot be more than 1000, and must contain integers from 0 to $n - 1$. Additionally, the vector passed to `query_machine` **will not be modified**. In other words, you can expect the vector `samples` to remain the same after calling `query_machine`.

The function will return a vector representing the number of surviving bacteria samples after each iteration until the machine stops running.

The function `answer_type` must be called with a vector of exactly n characters, with the i-th character representing the type of i-th species of bacteria. Each character in the vector may be equal to 'R' or 'T', representing Regular and Toxic bacteria respectively.

The following situations will cause you to receive a *Wrong Answer* verdict and cause the program to terminate immediately:

- If `query_machine` is called more than 1000 times or with more than 1000 samples
- If `answer_type` is called more than once
- If the vector passed into `answer_type` does not have exactly n elements, or has incorrect bacteria types
- If `query_machine` is called after a call to `answer_type`
- If `answer_type` has not been called by the time `determine_type` terminates

Do note that the grader for this task is **not adaptive**. This means that the answer for each testcase is fixed and will not change during the execution of your program.

Sample Interaction

In this example only, $n = 3$. In all other test cases $n = 1000$. Your program does not need to solve this sample to be considered correct.

Suppose that, bacteria species 0 is Toxic, while bacteria species 1 and 2 are Regular. Your function will be called with the following parameters:

```
determine_type(3)
```

A possible interaction could be as follows:

- `query_machine([1, 0, 2, 1]) = [2, 1]`

  This call represents putting 1 sample each of species 0 and 2, and 2 samples of species 1, in the order `[1, 0, 2, 1]`.

  During the first iteration, the first sample, of species 1, and the third sample, of species 2, are killed since they are Regular and are adjacent to a Toxic species, 0. After this, there are 2 surviving bacteria samples, `[0, 1]`.

  During the second iteration, the second sample, of species 1, is killed. After this, there is 1 surviving bacteria sample, `[0]`. At this point, any future iterations will also have 1 surviving bacteria sample, so the machine terminates and returns the number of surviving samples after each iteration: `[2, 1]`.

- `query_machine([1, 0, 1, 0, 0]) = [3]`

  During the first iteration, the first and third sample, of species 1, will both die. After this, there are 3 surviving bacteria samples, `[0, 0, 0]`. At this point, any future iterations will also have 3 surviving bacteria sample, so the machine terminates and returns: `[3]`.

- `query_machine([2, 1]) = [2]`

  During the first iteration, none of the bacteria die. At this point, any future iterations will also have 2 surviving bacteria sample, so the machine terminates and returns: `[2]`

At this point, the program decides that it has enough information to determine the types of bacteria and makes the following call:

- `answer_type(['T', 'R', 'R'])`

This call does not return any value. As the program has correctly identified all $n = 3$ bacteria species types, used no more than 1000 queries, and not made any invalid calls, it will be considered correct for this test case.

Subtasks

Your program will be tested on input instances that satisfy the following restrictions:

- $n = 1000$
- $1 \leq t < 1000$

Subtask	Marks	Additional Constraints
0	0	Sample Testcases
1	10	Bacteria 0 is toxic
2	90	No additional constraints

Scoring

Subtasks 2 is a partial scoring subtask. Your score depends on the number of queries you make across all testcases in that subtask, which we will denote here as q.

- If $q > 1000$, your score is 0.
- If $21 < q \leq 1000$, your score is $90 \times \sqrt{\frac{21}{q}}$
- If $q \leq 21$, your score is 90.

Testing

You may download the grader file, the header file and a solution template under *Attachments*. Two input files are provided for your testing, `sample1.txt` corresponds to the testcase in the sample interaction, and `sample2.txt` contains a testcase with $n = 1000$. Please use the script `compile.sh` to compile and run your solution for testing.

Grader Input Format for Testing

The first line contains one integer n, the number of bacteria species.

The next line contains a string of n charaters (‘R’ or ‘T’), describing the types of all species of bacteria in order.

Grader Output Format for Testing

The result of the call to `determine_type` is printed on a single line.

If your program enters any of the situations that triggers a *Wrong Answer* verdict, the grader prints the corresponding *Wrong Answer* message and terminates immediately.

Otherwise, the grader outputs the number of calls made to `query_machine`. Note that unlike the actual grader, the sample grader does not terminate immediately if more than 1000 calls to `query_machine` are made.

Task 4: `Coins`

Benson has n coins of different weights, and a weighing balance. Whenever he puts two coins x and y on the weighing balance, the relative weights between the coins is revealed, so that he knows which of x and y is heavier.

The rank of coin x is equal to the number of coins not heavier than coin x (including itself), so that the lightest coin has rank 1, second lightest coin has rank 2, etc. and the heaviest coin has rank n.

A coin has a determined rank, based on the existing weighings, if there is only one possible rank for the coin.

For each of the n coins, help Benson determine the first weighing which makes the coin's rank determined, or decide that the coin's rank is never determined.

Input format

Your program must read from standard input.

The first line of input will contain 2 spaced integers n and m.

The next m lines of input will contain 2 integers, x and y, indicating that coin x is lighter than coin y.

Output format

Output n integers. If coin i does not have a determined rank after all m measurements, the i-th integer should be -1. Else, there exists some k in which coin i has a determined rank after k weighings, but does not have a determined rank after $k-1$ weighings. Output this value of k.

Subtasks

For all subtasks, it is guaranteed that:

- $2 \leq n \leq 200\,000$
- $1 \leq m \leq 800\,000$

- $1 \leq x[i], y[i] \leq n$ for all $1 \leq i \leq m$
- There exists a set of weights such that coin $x[i]$ is lighter than coin $y[i]$ (for all $1 \leq i \leq m$).

Your program will be tested on input instances that satisfy the following restrictions:

Subtask	**Marks**	**Additional Constraints**
0	0	Sample Testcases
1	6	$1 \leq n \leq 7, 1 \leq m \leq 20$
2	16	$1 \leq n \leq 100, 1 \leq m \leq 400$
3	10	$1 \leq n \leq 1000, 1 \leq m \leq 4000$
4	68	No additional constraints

For each subtask, you will get 50% of the points if your program correctly determines whether each coin is determined at the end of all m weighings.

Specifically, if you output n integers such that

- If the i-th coin's rank is not determined at the end of m measurements, the i-th integer is -1
- If the i-th coin's rank is determined at the end of m measurements, the i-th integer is any integer from 1 to m inclusive,

then you will score 50% of the points for that subtask.

Sample Testcase 1

Input	Output

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

Explanation for sample testcase 1

We can determine after the first 3 measurements that coin 1 is the heaviest coin, but we are unable to do so by looking at only the first 2 measurements. Therefore, the first integer in the correct output is 3.

Similarly, we can determine that coin 2 is the lightest coin after 4 measurements, but not after 3 measurements. So, the second integer in the correct output is 4.

Observe that both orderings 2,4,3,1 and 2,3,4,1 are both valid orderings of the coin weights. Thus the coin 3 can have rank 2 or 3, both consistent with all weighings, and thus coin 3 never has a determined rank. Similarly, coin 4 never has a determined rank.

If your output was `1 1 -1 -1`, you would score 50% of the points for this subtask.

Sample Testcase 2

Input	Output

`6 8`
`1 5`
`5 4`
`6 2`
`2 5`
`4 3`
`6 1`
`6 5`
`2 1`
	`8 8 5 5 5 6`

Task 5: `Field`

Stuart the Snail lives on a field. The field can be described as an infinite plane (two-dimensional space). There are edible plants on every point of the field with integer coordinates. Stuart's house is located on the origin.

It is raining, so Stuart is able to move around easily. In one hour, he can choose one of the four edible plants adjacent to his current location, move there and have a snack. Formally, if he is at the coordinates (x, y), he may move to $(x + 1, y)$, $(x, y + 1)$, $(x - 1, y)$ or $(x, y - 1)$. He can continue his trip as long as it rains, but may also choose to stop at any time at the location of any edible plant, including simply staying at home.

However, it is known that there are n deep puddles in the field. Each puddle covers a rectangular region and is too deep for Stuart to pass through safely. Puddle i prevents Stuart from moving to any location with integer coordinates (x, y) satisfying both $a[i] \le x \le b[i]$ and $c[i] \le y \le d[i]$. Puddles may overlap.

It is not known how long the rain will last. Answer q queries of the same type defined by t.

- If $t = 1$, Stuart would like to visit the plant at the coordinates $(x[j], y[j])$. Find the minimum amount of time (in hours) required for him to reach $(x[j], y[j])$, assuming it never stops raining. If he cannot reach his destination, output -1 instead.
- If $t = 2$, suppose that the rain will last for $m[j]$ hours. Calculate the number of distinct locations that Stuart can end his trip in after at most $m[j]$ hours.

Input format

Your program must read from standard input.

The first line of input contains three integers n, t and q, representing the number of puddles, the type of query and the number of queries respectively.

Each of the next n lines contains four integers $a[i]$, $b[i]$, $c[i]$ and $d[i]$, describing a puddle.

If $t = 1$, each of the next q lines contains two integers $x[j]$ and $y[j]$, describing a destination.

Otherwise, if $t = 2$, each of the next q lines contains a single integer $m[j]$, describing the duration of the rain for one query.

Output format

Your program must print to standard output.

For each of the q queries, output a single integer on a new line representing the answer.

Subtasks

For all testcases, the input will satisfy the following bounds:

- $1 \le n \le 400$
- $1 \le t \le 2$
- $1 \le q \le 200\,000$
- $-10^9 \le a[i] \le b[i] \le 10^9$
- $-10^9 \le c[i] \le d[i] \le 10^9$
- $(0, 0)$ is not covered by any puddle.
- If $t = 1$, then $-10^9 \le x[j], y[j] \le 10^9$
- If $t = 2$, then $1 \le m[j] \le 10^9$

Your program will be tested on input instances that satisfy the following restrictions:

Subtask	Marks	Value of t	Additional Constraints
0	0	Sample testcases	
1	5	$t = 1$	$n \le 100$ $-400 \le a[i] \le b[i] \le 400$ $-400 \le c[i] \le d[i] \le 400$ $-400 \le x[j], y[j] \le 400$
2	17		$n \le 100$ $a[i] \equiv c[i] \equiv 0 \pmod{10^7}$ $b[i] \equiv d[i] \equiv -1 \pmod{10^7}$
3	8		$n \le 100$

Subtask	Marks	Value of t	Additional Constraints
4	8	$t = 2$	$n \leq 100, q \leq 400$ $-400 \leq a[i] \leq b[i] \leq 400$ $-400 \leq c[i] \leq d[i] \leq 400$ $m[j] \leq 400$
5	21		$n \leq 100, q \leq 400$ $a[i] \equiv c[i] \equiv 0 \pmod{10^7}$ $b[i] \equiv d[i] \equiv -1 \pmod{10^7}$
6	10		$n \leq 100, q \leq 400$
7	13		$n \leq 100, q \leq 5000$
8	14		$n \leq 100$
9	4		No additional constraints

Sample Testcase 1

This testcase is valid for subtasks 1 and 3.

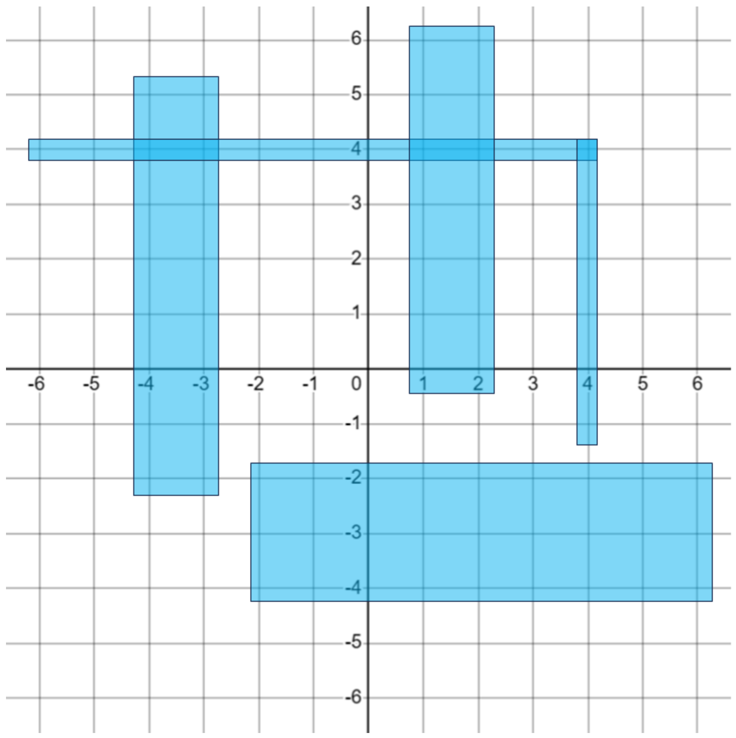
Input	Output

```
5 1 4
-4 -3 -2 5
-6 4 4 4
1 2 0 6
4 4 -1 4
-2 6 -4 -2
-1 2
3 3
0 6
2 -3
```
	```
	3
	8
	-1
	-1
	```

Sample Testcase 1 Explanation

The diagram on the next page describes the area around the origin. The blue rectangles represent puddles which cannot be entered or passed through.

It is not possible to reach the last two destinations without entering puddles. Note that the last destination is covered by a puddle.



Sample Testcase 2

This testcase is valid for subtasks 2 and 3.

Input	Output
`2 1 4`	`2`
`-1000000000 -1 0 999999999`	`-1`
`0 999999999 -1000000000 -1`	`4000000002`
`1 1`	`-1`
`-1 1`	
`-1 -1`	
`1 -1`	

Sample Testcase 3

This testcase is valid for subtasks 4, 6, 7, 8 and 9.

Input	Output
`2 2 6`	`4`
`-2 5 1 1`	`8`
`0 1 -3 -2`	`13`
`1`	`21`
`2`	`32`
`3`	`48`
`4`	
`5`	
`6`	

Sample Testcase 4

This testcase is valid for subtasks 5, 6, 7, 8 and 9.

Input	Output
`2 2 4`	`235`
`0 9999999 -10000000 -1`	`2285986`
`-10000000 -1 10000000 29999999`	`22862261089`
`12`	`231374765559370`
`1234`	
`123456`	
`12345678`	