NOI 2023

18 March 2023

Tasks	Time Limit	Memory Limit
Task 1: Topical	1 s	1024 MB
Task 2: Inspections	2 s	1024 MB
Task 3: Airplane	1 s	1024 MB
Task 4: Curtains	1.5 s	1024 MB
Task 5: Toxic Gene	1 s	1024 MB

Have Fun!

Task 1: `Topical`

Benson the Rabbit is attending pilot school!

He has n modules to complete, numbered from 1 to n. There are k topics in flying numbered from 1 to k. As Benson is new to flying, he starts with zero knowledge in each topic.

Each of these n modules have a knowledge requirement to complete them. In particular, to complete module i, Benson requires at least $r[i][j]$ knowledge of topic j for all topics j.

Completing a module also allows Benson to gain knowledge in some topics. After completing module i, he will gain $u[i][j]$ knowledge in topic j.

Formally, let Benson's knowledge in topic j be $p[j]$. Initially, $p[j] = 0$ for all j. He can only complete a module i if $p[j] \geq r[i][j]$ for every topic j. After completing module i, the value of $p[j]$ increases by $u[i][j]$ for each topic j.

Benson may complete the modules in any order, but each module may only be completed at most once. Help Benson determine the maximum number of modules he can complete.

Input format

Your program must read from standard input.

The first line of input contains 2 space-separated integers, n and k.

Then, n lines will follow. The i-th ($1 \leq i \leq n$) of these lines contains k spaced integers $r[i][1], r[i][2], \ldots, r[i][k]$, denoting the knowledge requirements to complete module i.

Another n lines follow. The i-th ($1 \leq i \leq n$) of these lines contains k spaced integers $u[i][1], u[i][2], \ldots, u[i][k]$, denoting the knowledge gained after completing module i.

Output format

Your program must print to standard output.

The output should contain one integer, the maximum number of modules Benson can complete.

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:

- $1 \leq n, k \leq 10^6$
- $n \cdot k \leq 10^6$
- $0 \leq u[i][j], r[i][j] \leq 10^9$ (for all $1 \leq i \leq n$ and $1 \leq j \leq k$).

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

Subtask	Marks	Additional Constraints
1	12	$n = 1$
2	28	$1 \leq n, k \leq 100$
3	21	$k = 1$
4	39	No additional restrictions

Sample Testcase 1

This testcase is valid for subtasks 2 and 4.

Input	Output
3 3 0 0 0 7 9 2 7 8 9 7 8 2 7 7 7 8 10 9	1

Sample Testcase 1 Explanation

Benson can only complete module 1, which has knowledge requirement $[0, 0, 0]$. After which, he gains 7, 8, 2 knowledge in each of the 3 topics, but $p = [7, 8, 2]$ is insufficient for him to complete any other module. Since no other sequence allows Benson to complete more than 1 module, the final answer is 1.

Sample Testcase 2

This testcase is valid for subtasks 2 and 4.

Input	Output
4 3 5 1 0 0 1 5 0 0 0 7 7 7 0 5 6 1 1 1 8 2 0 8 1 4	4

Sample Testcase 2 Explanation

Benson can complete all 4 modules in the order 3, 1, 2, 4.

With initial knowledge $p = [0, 0, 0]$, he can complete module 3 and his knowledge increases by $u[3] = [8, 2, 0]$.
With knowledge $p = [8, 2, 0]$, he can complete module 1 and his knowledge increases by $u[1] = [0, 5, 6]$.
With knowledge $p = [8, 7, 6]$, he can complete module 2 and his knowledge increases by $u[2] = [1, 1, 1]$.
With knowledge $p = [9, 8, 7]$, he can complete module 4 and his knowledge increases by $u[4] = [8, 1, 4]$.

Since Benson can complete all 4 modules, the answer is 4.

Sample Testcase 3

This testcase is valid for subtasks 2 and 4.

Input	Output
5 5 14 11 15 7 15 0 0 0 0 0 9 9 14 2 13 4 3 6 1 0 2 4 7 0 0 5 5 0 0 13 4 4 7 1 0 4 1 0 2 1 2 5 0 2 1 4 0 7 2 12	4

Sample Testcase 3 Explanation

Benson can only complete 4 modules in the order 2, 4, 5, 3.

With initial knowledge $p = [0, 0, 0, 0, 0]$, he can complete module 2 and his knowledge increases by $u[2] = [4, 4, 7, 1, 0]$.
With knowledge $p = [4, 4, 7, 1, 0]$, he can complete module 4 and his knowledge increases by $u[4] = [2, 5, 0, 2, 1]$.
With knowledge $p = [6, 9, 7, 3, 1]$, he can complete module 5 and his knowledge increases by $u[5] = [4, 0, 7, 2, 12]$.
With knowledge $p = [10, 9, 14, 5, 13]$, he can complete module 3 and his knowledge increases by $u[4] = [4, 1, 0, 2, 1]$.

With that, he has knowledge $p = [14, 10, 14, 7, 14]$ which is insufficient to complete any other module. Since no other sequence allows Benson to complete more than 4 modules, the final answer is 4.

Task 2: `Inspections`

Benson the Rabbit now has to build a plane!

Benson the Rabbit has a factory with n machines numbered from 1 to n. Each machine runs for one day and only one machine can run at a time. He has m tasks to complete, numbered from 1 to m. Each task i is represented by two positive integers $l[i]$ and $r[i]$ where $l[i] \leq r[i]$.

To complete task i, Benson needs to run machines $l[i], l[i]+1, \ldots, r[i]$ in that order. Once a machine has finished running, the next machine starts immediately. Once task i is complete, Benson immediately starts task $i+1$ until task m is complete.

In order to comply with safety regulations, the factory must have an safety value s. If a machine with safety value s was not run in the past s or more days, this machine needs to be inspected before it can be run. Machines do not need to be inspected the first time they are run. See the samples for more details.

Benson has q different candidate safety values $s[1], s[2], \ldots, s[q]$. For each safety value $s[j]$, help him compute the number of inspections that need to be done if the safety value is $s[j]$.

Input format

Your program must read from standard input.

The first line of input will contain 3 spaced integers n, m and q, representing the number of machines, tasks and safety values respectively.

The next m lines of input will contain 2 spaced integers each. The i-th of these lines will contain $l[i]$ and $r[i]$ respectively, describing task i.

The next line of input will contain q spaced integers $s[1], s[2], \ldots, s[q]$, which represent the q safety values to be tested.

Output format

Your program must print to standard output.

Output one line with q spaced integers, the j-th integer representing the number of inspections that need to be done if the safety value is $s[j]$.

Subtasks

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

- $1 \leq n, m, q \leq 200\,000$
- $1 \leq l[i] \leq r[i] \leq n$
- $0 \leq s[j] \leq 10^{12}$

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

Subtask	Marks	Additional Constraints
1	11	$1 \leq n, m, q \leq 200$
2	18	$1 \leq n, m \leq 2000$
3	22	$l[i] = 1$
4	26	$m \leq 2000$
5	23	No additional restrictions

Sample Testcase 1

This testcase is valid for subtasks 1, 2, 4 and 5.

Input	Output
5 3 7 1 3 3 5 2 3 0 1 2 3 4 5 6	3 2 2 2 1 0 0

Sample Testcase 1 Explanation

The machines will be run in the following order: $1, 2, 3, 3, 4, 5, 2, 3$.

On the 4-th day, machine 3 will be run 0 days after it was last run.

On the 7-th day, machine 2 will be run 4 days after it was last run.

On the 8-th day, machine 3 will be run 3 days after it was last run.

If the safety value is 0, then machine 3 would need to be inspected on day 4 and day 8, while machine 2 would need to be inspected on day 7.

If the safety value is 2, then machine 3 would only need to be inspected on day 8. Machine 2 would still need to be inspected on day 7.

Sample Testcase 2

This testcase is valid for all subtasks.

Input	Output
6 6 7 1 6 1 5 1 4 1 3 1 2 1 1 1 2 3 4 5 6 7	15 14 12 9 5 0 0

Task 3: `Airplane`

Benson the Rabbit wants to fly an airplane!

There are n regions that Benson can fly in, numbered from 1 to n. For each region i, there is a minimum altitude $a[i]$ that Benson must fly at within the region due to terrain constraints.

Additionally, Benson can only fly between certain pairs of regions due to prevailing wind conditions and Benson's lack of flying experience (he is a rabbit after all). There are m such pairs numbered from 1 to m, and the j-th pair $u[j]$ and $v[j]$ indicates that Benson can fly between regions $u[j]$ and $v[j]$ in both directions. It is always possible to travel from any region to all other regions using only the allowed pairs.

Initially, Benson is at region 1 at height 0. He wants to travel to region n, and to land he must end at height 0.

In a minute, Benson can choose to stay at his current region or travel to another region. In that same minute, his altitude can increase by 1, decrease by 1 or remain the same. However, when Benson arrives at a region, his height must be at least the minimum altitude required for that region. What is the minimum time Benson needs to land at region n?

Input format

Your program must read from standard input.

The first line of input will contain 2 spaced integers n and m, which represent the number of regions and the number of pairs of regions that Benson can fly between.

The next line contains n spaced integers $a[1], a[2], \ldots, a[n]$, representing the minimum required altitude at each region.

The next m lines of input will contain 2 spaced integers each. The j-th of these lines contains $u[j]$ and $v[j]$, indicating that Benson can fly between regions $u[j]$ and $v[j]$ in both directions.

Output format

Your program must print to standard output.

The output should contain one integer, the minimum time required to land at region n.

Subtasks

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

- $1 \leq n \leq 200\,000$
- $1 \leq m \leq 400\,000$
- $0 \leq a[i] \leq 10^8$
- $a[1] = a[n] = 0$
- $1 \leq u[j], v[j] \leq n, u[j] \neq v[j]$

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

Subtask	Marks	Additional Constraints
1	22	$m = n - 1, u[j] = j, v[j] = j + 1$
2	10	$n \leq 2000, m \leq 4000, a[i] \leq 2000$
3	31	$n \leq 2000, m \leq 4000$
4	37	No additional restrictions

Sample Testcase 1

This testcase is valid for all subtasks.

Input	Output
3 2 0 2 0 1 2 2 3	4

Sample Testcase 1 Explanation

Benson is travelling from region 1 to region 3. He can do so in 4 minutes.

- In minute 1, Benson stays at region 1. He increases his altitude from 0 to 1.

- In minute 2, Benson travels from region 1 to 2. He increases his altitude from 1 to 2.
- In minute 3, Benson travels from region 2 to 3. He decreases his altitude from 2 to 1.
- In minute 4, Benson stays at region 3. He decreases his altitude from 1 to 0.

Sample Testcase 2

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

Input	Output
11 12 0 0 0 0 0 0 2 2 1 5 0 1 2 2 3 3 4 4 5 5 6 6 11 1 7 7 8 8 9 9 11 1 10 10 11	5

Sample Testcase 2 Explanation

Benson is travelling from region 1 to region 11. He can do so in 5 minutes.

- In minute 1, Benson stays at region 1. He increases his altitude from 0 to 1.
- In minute 2, Benson travels from region 1 to 7. He increases his altitude from 1 to 2.
- In minute 3, Benson travels from region 7 to 8. He maintains his altitude at 2
- In minute 4, Benson travels from region 8 to 9. He decreases his altitude from 2 to 1.
- In minute 5, Benson travels from region 9 to 11. He decreases his altitude from 1 to 0.

Task 4: `Curtains`

Benson the Rabbit is organizing a performance on his plane!

He has a stage with n sections numbered 1 to n from left to right. He also has m curtains numbered from 1 to m.

Each of these m curtains can be lowered. Lowering curtain i covers sections $l[i]$ to $r[i]$. A **curtain configuration** is a set of lowered curtains. Given a curtain configuration, a section x ($1 \leq x \leq n$) is **covered** if and only if there exists a **lowered** curtain i such that $l[i] \leq x \leq r[i]$.

Benson wants to give a total of q performances, numbered from 1 to q. For each performance j, Benson requires a curtain configuration such that the sections $s[j]$ to $e[j]$ are covered and nothing else is covered. More formally, for each $1 \leq x \leq n$,

- If $s[j] \leq x \leq e[j]$, section x is covered.
- Otherwise, section x is not covered.

For each of these q performances, help Benson to determine if there exists a curtain configuration satisfying his requirements.

Input format

Your program must read from standard input.

The first line of input will contain 3 spaced integers n, m and q, representing the number of sections, curtains and performances respectively.

The next m lines of input will contain 2 spaced integers each. The i-th of these lines will contain $l[i]$ and $r[i]$ respectively, describing the range of sections that curtain i can cover.

The next q lines of input will contain 2 spaced integers each. The j-th of these lines will contain $s[j]$ and $e[j]$ respectively, describing the range of sections that need to be covered for performance j.

Output format

Output q lines, the j-th of which should contain `YES` if it is possible to cover the required sections for the j-th performance using the curtains, and `NO` otherwise.

Subtasks

For all subtasks, it is guaranteed that:

- $1 \leq n, m, q \leq 500\,000$
- $1 \leq l[i] \leq r[i] \leq n$ (for all $1 \leq i \leq m$)
- $1 \leq s[j] \leq e[j] \leq n$ (for all $1 \leq j \leq q$)

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

Subtask	Marks	Additional Constraints
1	3	$1 \leq n, m, q \leq 200$
2	6	$1 \leq n, m, q \leq 2000$
3	15	$1 \leq n \leq 2000$
4	20	$s[j] = 1$
5	36	$1 \leq n, m, q \leq 100\,000$
6	20	No additional restrictions

Sample Testcase 1

This testcase is valid for all subtasks.

Input	Output
6 2 3 1 2 3 4 1 3 1 4 1 5	NO YES NO

Sample Testcase 1 Explanation

Benson has 6 sections and 2 curtains. Curtain 1 covers sections 1 and 2 while curtain 2 covers sections 3 and 4.

It is not possible to exactly cover sections 1 to 3. It is also not possible to exactly cover sections 1 to 5. However, he can use both curtains to cover sections 1 to 4 exactly.

Sample Testcase 2

Input	Output
10 10 10 6 9 6 7 1 6 10 10 5 9 3 9 2 10 5 7 9 10 5 10 7 8 4 7 1 6 2 7 3 9 7 7 2 9 4 9 6 6 5 7	NO NO YES NO YES NO NO NO NO YES

Task 5: `Toxic Gene`

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

Benson the Rabbit's plane has been overwhelmed by toxic bacteria, and he has to investigate it!

Benson the Rabbit has n species of bacteria. Each of them falls into exactly one of 3 types: Regular, Strong, Toxic. t of them are Toxic. It is guaranteed that there is at least 1 Toxic bacteria, i.e. $t \geq 1$. Note that Benson does not know the value of t.

Benson wants to identify the type of each bacteria. To analyze the bacteria, he can place bacteria specimens into a machine. He can specify the species of each bacteria, and he can add any number of each species into the machine, including 0. This forms a single sample. Due to size constraints, the total number of bacteria in a sample cannot exceed 300.

Each of the 3 types of bacteria have the following properties:

- Regular bacteria will survive if there are no Toxic bacteria in the sample, and will die if there is at least one Toxic bacteria in the sample.
- Strong bacteria will always survive.
- Toxic bacteria produce a toxin which kills all bacteria in the sample that are not Strong bacteria. Toxic bacteria will always die.

After a sample is selected, the machine will tell Benson how many bacteria survived in total. Each use of the machine takes time, and Benson does not have a lot of time. He may only use the machine up to 600 times. Help Benson determine for each bacteria, whether it is Regular, Strong or Toxic in as few samples 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 up to 100 times per testcase. Each call may have a different combination of Regular, Strong and Toxic bacteria. For each testcase, you must solve all calls to the function within the time and memory limits.

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

- `int query_sample(std::vector<int> species)`
- `void answer_type(int x, char c)`

The function `query_sample` is called with a 1-dimensional array `species`, describing the species of each of the bacteria in the sample your program has chosen. The size of `species` cannot be more than 300. Additionally, the array passed to `query_sample` **will not be modified**. In other words, you can expect the array `species` to remain the same after calling `query_sample`.

The function `answer_type` is called with one integer `x` and one character `c`. Your program may call this function when it is sure that the species `x` is of the type represented by `c`. `c` may be equal to `'R'`, `'S'` or `'T'`, representing Regular, Strong and Toxic bacteria types respectively. Your program must call this function for all species of bacteria.

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

- If either `query_sample` or `answer_type` is called with invalid parameters
- If `answer_type` identifies the type of bacteria wrongly
- If by the time `determine_type` terminates, not all n species of bacteria have been identified via the `answer_type` function.
- If `query_sample` is called more than 600 times during a call of `determine_type`

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

Suppose $n = 5$. Bacteria species 1 and 2 are Toxic, bacteria species 3 and 4 are Regular and bacteria species 5 is Strong. This corresponds to the string `TTRRS`.

Your function will be called with the following parameters

```
determine_type(5)
```

A possible interaction could be as follows:

- `query_sample([1,2,3,4,5]) = 1`

  One specimen of each bacteria is used in this query. Only bacteria 5 survives, so the grader returns the value 1.

- `query_sample([3,3,4,5]) = 4`

  Two specimen of bacteria species 3 and one specimen each of bacteria species 4 and 5 are used in this query. Since there are no toxic bacteria, all specimens survive and the grader returns the value 4.

At this point, the program decides that it has enough information to determine the type of each bacteria and makes the following 5 calls.

- `answer_type(1,'T')`
- `answer_type(2,'T')`
- `answer_type(3,'R')`
- `answer_type(4,'R')`
- `answer_type(5,'S')`

None of these calls return any value. As the program has correctly identified all $n = 5$ bacteria species types and used no more than 600 queries, it will be considered correct for this test case.

Do note that this sample test case does not satisfy the input limits below. It is purely for testing purposes and is not required to be solved to obtain full points for this task.

Scoring

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

- $n = 300$
- $1 \leq t \leq 30$

Your score for this task depends on the maximum number of queries you make across all calls to `determine_type` over all testcases, which we will denote here as m.

- If $m > 600$, your score is 0.

- If $340 < m \leq 600$, your score is $2 + 7 \times \frac{600-m}{260}$.
- If $275 < m \leq 340$, your score is $9 + 15 \times \frac{340-m}{65}$
- If $190 < m \leq 275$, your score is $24 + 22 \times \frac{275-m}{85}$
- If $150 < m \leq 190$, your score is $46 + 54 \times \frac{190-m}{40}$.
- If $m \leq 150$, your score is 100.

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 $tc = 100$ and $n = 300$. Please use the script `compile.sh` to compile and run your solution for testing.

Grader Input Format for Testing

The first line contains two integers tc and n. tc is the number of calls to `determine_type`.

Each of the next tc lines contain a string of n characters (`R`, `S` or `T`) describing the types of all species of bacteria in order.

Grader Output Format for Testing

The result of each call to `determine_type` is represented as an integer, printed on a new line.

If your program enters any of the situations that triggers a *Wrong Answer* verdict, the grader outputs `-1` and terminates immediately. Any remaining calls to `determine_type` will not be made.

Otherwise, the grader outputs the number of calls made to `query_sample`.