# Beacons

After getting lost this morning, you have decided to fix the navigation problem plaguing your city. Your city consists of $N$ intersections (numbered from 1 to $N$) connected by $N-1$ bidirectional roads (numbered from 1 to $N-1$). The $i$th road connects intersections $u_i$ and $v_i$, and it is possible to travel between any pair of intersections using some sequence of roads.

There are $K$ beacons in the city (numbered from 1 to $K$), where the $i$th beacon is at intersection $b_i$. When you find yourself lost at an intersection, you ping each beacon to determine your *distance* from that beacon, which is defined as the number of roads on the shortest path between you and that beacon. Using this information, you hope to determine your location. Unfortunately, depending on the placement of the beacons, it may not be possible to determine which intersection you are at!

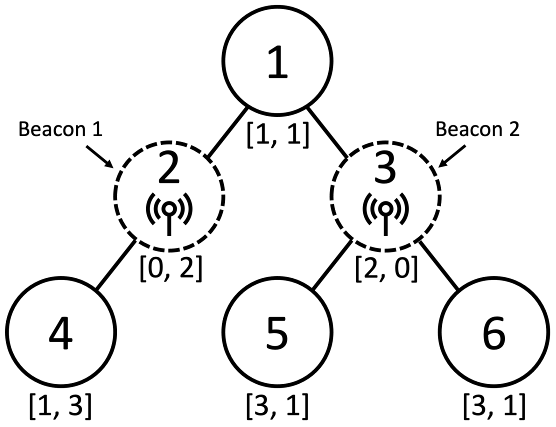

Figure 1: An example city with $N = 6$ intersections and $K = 2$ beacons. The distance from each beacon is written below the intersections. Using the beacons, you cannot determine the difference between intersections 5 and 6. You can fix this by building one more beacon at intersection 5.

You cannot move existing beacons, but you can build more. What is the minimum number of extra beacons you must build so that it is always possible to determine your location?

## Subtasks and Constraints

For all subtasks:

- $2 \leq N \leq 200\,000$.
- $0 \leq K \leq N$.
- $1 \leq b_i \leq N$ for all $i$.
- $b_i \neq b_j$ for all $i \neq j$.
- $1 \leq u_i < v_i \leq N$ for all $i$.
- It is possible to travel between any pair of intersections using the roads.

Additional constraints for each subtask are given below.

| Subtask | Points | Additional constraints |
|---|---|---|
| 1 | 5 | $K = 0$ and your city is a line. That is, $u_i = i$ and $v_i = i + 1$ for all roads. |
| 2 | 9 | Your city is a line. That is, $u_i = i$ and $v_i = i + 1$ for all roads. |
| 3 | 19 | $N \leq 100$ and the answer is 0 or 1. |
| 4 | 24 | The answer is 0 or 1. |
| 5 | 22 | $K = 0$. |
| 6 | 21 | No additional constraints. |

## Input

- The first line of input contains the integers $N$ and $K$.
- The second line of input contains the $K$ integers $b_1, \ldots, b_K$.
- The next $N - 1$ lines of input describe roads. The $i$th line contains two integers $u_i$ and $v_i$.

## Output

Output a single integer: the minimum number of extra beacons you must build so that it is always possible to determine your location.

## Sample Input 1

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

## Sample Output 1

```
1
```

## Sample Input 2

```
6 0

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

## Sample Output 2

```
2
```

## Sample Input 3

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

## Sample Output 3

```
0
```

## Explanation

The first sample case corresponds to the example on the first page.

In the second sample case, there are initially no beacons. You can build 2 beacons, as shown in Figure 2.

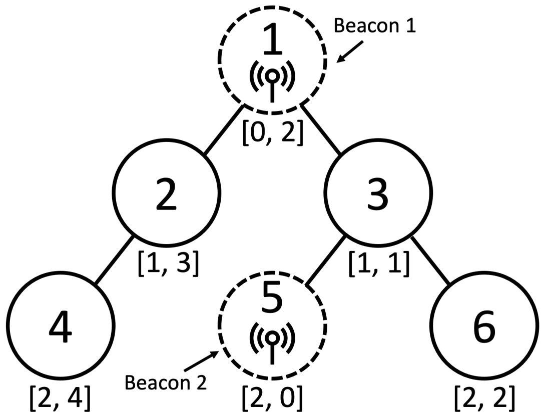

Figure 2: A solution to Sample Input 2.

In the third sample case, no new beacons are needed.

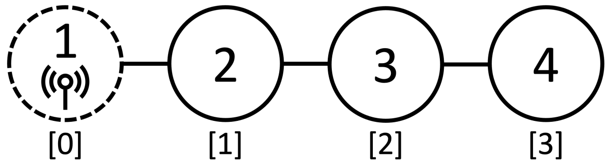

Figure 3: Sample Input 3.

# Maximum Matrix

Neo likes matrices. Neo's favourite type of *matrix* is a grid with $R$ rows and $C$ columns, where each cell in the matrix contains a positive integer.

Neo assigns a score $(A, B)$ to each matrix:

- $A$ is the number of *ascending* rows. A row is ascending if the values in this row are ascending when read from left to right. Specifically, if the values in the row are $v_1, v_2, \ldots, v_C$ from left to right, then the row is ascending if $v_1 \leq v_2 \leq \ldots \leq v_C$.
- $B$ is the number of *constant* columns. A column is constant if the values in this column are all the same.

| | | | | |
|---|---|---|---|---|
| 2 | 3 | 3 | 6 | 8 |
| 2 | 3 | 1 | 6 | 8 |
| 1 | 3 | 6 | 6 | 8 |

Figure 1: An example matrix with $R = 3$ rows and $C = 5$ columns. There are 2 ascending rows (the first and third) and 3 constant columns (the second, fourth, and fifth). Neo's score for this matrix is $(2, 3)$.

A matrix is *better* than another matrix if it has a lexicographically higher score. In particular, assume that you have one matrix with a score $(A, B)$ and another matrix with a score $(A', B')$. The first matrix is better if one of the following conditions holds:

- $A > A'$, or
- $A = A'$ and $B > B'$.

For example,

- A matrix with score $(5, 3)$ is **better** than a matrix with score $(4, 4)$.
- A matrix with score $(5, 3)$ is **better** than a matrix with score $(5, 2)$.
- A matrix with score $(5, 3)$ is **not better** than a matrix with score $(5, 4)$.
- A matrix with score $(5, 3)$ is **not better** than a matrix with score $(6, 1)$.

You have found a matrix with some missing values. To impress Neo, you want to fill in the missing values with **positive integers** in a way that creates the best possible matrix. What is the score of the best matrix you can create?

## Subtasks and Constraints

For all subtasks:

- $1 \le R \le 250\,000$ and $1 \le C \le 250\,000$.
- $R \times C \le 1\,000\,000$.
- All non-missing values in the matrix are positive integers from 1 to 1 000 000, inclusive.

Additional constraints for each subtask are given below.

| Subtask | Points | Additional constraints |
|---|---|---|
| 1 | 7 | $R = 1$. |
| 2 | 18 | The answer has $A = R$. |
| 3 | 10 | $R \le 10$, $C \le 10$, and every column has at least one value that is not missing. |
| 4 | 8 | $R \le 10$ and $C \le 10$. |
| 5 | 17 | $R \le 100$, $C \le 100$, and every column has at least one value that is not missing. |
| 6 | 11 | $R \le 100$ and $C \le 100$. |
| 7 | 14 | $R \le 5\,000$ and $C \le 5\,000$. |
| 8 | 15 | No additional constraints. |

## Input

- The first line of input contains the integers $R$ and $C$.
- The next $R$ lines of input each contain $C$ integers, describing the Matrix. Each value in the matrix is either a positive integer or zero, where zero represents a missing value.

## Output

Output two integers $A$ and $B$ on a single line, representing the score $(A, B)$ of the best matrix that can be created.

### Sample Input 1

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

### Sample Output 1

```
2 3
```

### Sample Input 2

```
2 3
1 0 2
3 0 4
```

### Sample Output 2

```
2 0
```

### Sample Input 3

```
2 4
2 4 0 1
2 0 3 1
```

### Sample Output 3

```
0 4
```

## Explanation

The first sample case has three missing values. One optimal way to fill in the missing values is to create the matrix shown in Figure 1, with $A = 2$ ascending rows and $B = 3$ constant columns.

The second sample case can be filled in as follows, with $A = 2$ ascending rows and $B = 0$ constant columns:

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

The third sample case can be filled in as follows, with $A = 0$ ascending rows and $B = 4$ constant columns:

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

# Hedge Maze II

Perrito has found himself stuck in a hedge maze!

He knows that the maze has $N$ rooms, numbered from 1 to $N$, and that there are $N-1$ passageways that each connect two different rooms. If the passageways are bidirectional, then it is possible to travel between any pair of rooms using the passageways. However, to make the maze more difficult, each passageway **can only be travelled in one direction**. Because of this, it is not necessarily possible to travel between every pair of rooms.

Unfortunately, Perrito does not know where these passageways are and his first goal is to discover all the passageways. That is, he must determine the $N-1$ pairs of rooms that are connected by passageways, and the direction of each passageway.

To do this, the maze designer has allowed Perrito to ask *queries*. In each query, Perrito provides the maze designer with two non-empty arrays $A$ and $B$:

- The arrays $A$ and $B$ each contain integers from 1 to $N$, inclusive.
- The arrays must be disjoint and contain distinct integers. That is, each integer can appear at most once across both arrays.

The maze designer will answer each query with a single integer: the number of pairs of integers $(a, b)$, where $a$ is in the array $A$ and $b$ is in the array $B$, for which there exists a path in the maze from room $a$ to room $b$.

Let $|A|$ represent the number of integers in the array $A$, and $|B|$ represent the number of integers in the array $B$. Then, the *cost* of the query is $|A| \times |B|$. Perrito is allowed to ask queries with a total cost of up to 10 000 000. Additionally, your score depends on the *number* of queries asked. See the scoring section for details.

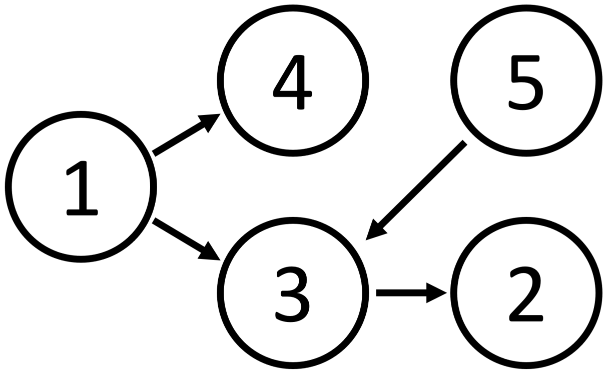

Figure 1: An example maze

Figure 1 has a maze with $N = 5$ rooms. If Perrito asked a query with $A = [1, 3]$ and $B = [2, 4]$, then the maze designer would answer with 3:

- There exists a path from room 1 to room 2, room 1 to room 4, and room 3 to room 2.
- There does not exist a path from room 3 to room 4.

The cost of this query is $2 \times 2 = 4$.

## Implementation Details

In this task, **do not read from standard input nor write to standard output.** Do not interact with any files. Do not implement a `main` function. Instead, begin your program by including the header file `maze.h` (`#include "maze.h"`) and interact with it as described below.

### Functions

You must implement the function `map_maze`, which is called once in every test case:

```
void map_maze(int N);
```

- `N` is the number of rooms in the maze.
- From this function you can call `query`.
- You must call `report_passageway` exactly $N - 1$ times.

From `map_maze`, you can make calls to the following functions:

```
int query(vector<int> A, vector<int> B);
```

- `A` and `B`: Your chosen arrays $A$ and $B$.
- $A$ and $B$ must both be non-empty.
- $A$ and $B$ must contain integers from 1 to $N$ inclusive, and each integer can appear at most once across both of the arrays.
- **The sum of** $|A| \times |B|$ **across all queries can be at most** $10\,000\,000$.
- This function returns the number of pairs of integers $(a, b)$, where $a$ is in the array $A$ and $b$ is in the array $B$, such that there exists a path in the maze from room $a$ to room $b$.
- **Your score depends on the number of times this function is called. See the scoring section for details**.

```
void report_passageway(int i, int j);
```

- You should call this function if there is a directed passageway from room $i$ to room $j$.
- You should call this function exactly $N - 1$ times: once for each passageway in the maze.

If you violate any of the conditions listed above, your program will be judged as incorrect and you will receive 0% of the points for the test case.

The grader is not adaptive. This means that answers to all queries are based on a fixed input.

## Subtasks and Constraints

For all subtasks:

- $3 \le N \le 1\,000$.
- There are $N - 1$ passageways in the maze.
- If passageways are made bidirectional, then it is possible to travel from any room to any other room using the passageways.

Additional constraints for each subtask are given below.

| Subtask | Points | Additional constraints |
|---|---|---|
| 1 | 10 | Each passageway has one of its endpoints at room 1. |
| 2 | 18 | There exists a room $i$, such that every passageway has one of its endpoints at room $i$. |
| 3 | 22 | There exists a room $i$, such that for every other room $j$, there exists a path from room $i$ to room $j$. |
| 4 | 50 | No additional constraints. |

## Scoring

If your solution fails to find all the passageways, or violates any of the conditions in the Implementation Details section, your score will be 0.

Otherwise, let $Q$ be the number of calls that your solution makes to `query`:

- Subtask 1, 2, and 3: You score all the points for these subtasks if $Q \leq 13\,000$, and 0 points otherwise.
- Subtask 4:
  - If $Q \leq 10\,000$, your score is 50. Otherwise,
  - If $Q \leq 11\,000$, your score is 43. Otherwise,
  - If $Q \leq 12\,000$, your score is 36. Otherwise,
  - If $Q \leq 13\,000$, your score is 31. Otherwise,
  - If $Q \leq 999\,000$, your score is 20. Otherwise,
  - Your score is 0.

Your score for a subtask will be the **minimum** score of all test cases in the subtask.

## Experimentation

In order to experiment with your code on your own machine, first downloaded the provided files `maze.cpp`, `maze.h` and `grader.cpp`.

You should modify `maze.cpp`, which contains a basic example implementation.

Compile your solution with:

```
g++ -std=c++17 -O2 -Wall maze.cpp grader.cpp -o maze
```

This will create an executable `maze` which you can run with `./maze`. If you have trouble compiling, please send a message in the Communication section of the contest website.

The provided grader reads from standard input in the following format:

- The first line of input contains $N$.
- The next $N - 1$ lines of input describe the passageways. The $i$th such line contains two integers $a$ and $b$, representing a one-way passageway from room $a$ to room $b$.

Note that the sample grader does not check whether the provided passageways form a valid input.

## Sample Grader Input & Sample Session

The sample grader is supplied with the following input:

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

This corresponds to the diagram on the first page.

One possible interaction is described below:

| Grader | Student | Description |
|---|---|---|
| `map_maze(5)` | | The grader calls your program. |
| | `query({1, 3}, {2, 4})` | You ask a query with $A = \{1, 3\}$ and $B = \{2, 4\}$. |
| returns `3` | | The grader responds with 3. This is explained in the example on the first page. |
| | `report_passageway(1, 4)` | You report that there is a passageway from room 1 to room 4. This is correct. |
| | `query({5}, {1})` | You ask a query with $A = \{5\}$ and $B = \{1\}$. |
| returns `0` | | There is not a path from room 5 to room 1, and so the grader returns 0. |
| | `report_passageway(1, 5)` | You report that there is a passageway from room 1 to room 5. This is incorrect, and your program will score 0. |