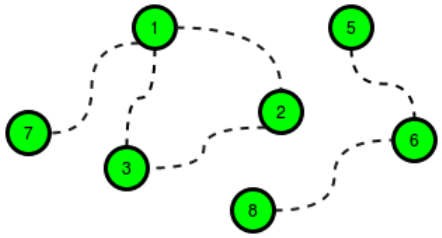
Roads and Libraries

The Ruler of HackerLand believes that every citizen of the country should have access to a library. Unfortunately, HackerLand was hit by a tornado that destroyed all of its libraries and obstructed its roads! As you are the greatest programmer of HackerLand, the ruler wants your help to repair the roads and build some new libraries efficiently.

HackerLand has n cities numbered from 1 to n. The cities are connected by m bidirectional roads. A citizen has access to a library if:

- Their city contains a library.
- They can travel by road from their city to a city containing a library.

The following figure is a sample map of HackerLand where the dotted lines denote obstructed roads:



The cost of repairing any road is c_{road} dollars, and the cost to build a library in any city is c_{lib} dollars. If in the above example $c_{road} = 2$ and $c_{lib} = 3$, we would build 5 roads at a cost of $5 \times 2 = 10$ and 2 libraries for a cost of 6. We don't need to rebuild one of the roads in the cycle $1 \rightarrow 2 \rightarrow 3 \rightarrow 1$.

You are given q queries, where each query consists of a map of HackerLand and value of c_{lib} and c_{road}. For each query, find the minimum cost of making libraries accessible to all the citizens and print it on a new line.

Function Description

Complete the function *roadsAndLibraries* in the editor below. It must return the minimal cost of providing libraries to all, as an integer.

roadsAndLibraries has the following parameters:

- *n*: integer, the number of cities
- *c_lib*: integer, the cost to build a library
- *c_road*: integer, the cost to repair a road
- *cities*: 2D array of integers where each $cities[i]$ contains two integers representing cities connected by an obstructed road .

Input Format

The first line contains a single integer q, denoting the number of queries.

The subsequent lines describe each query in the following format:
- The first line contains four space-separated integers describing the respective values of n, m, c_{lib} and c_{road}, the number of cities, number of roads, cost of a library and cost of a road.
- Each of the next m lines contains two space-separated integers, $u[i]$ and $v[i]$, describing a bidirectional road connecting cities $u[i]$ and $v[i]$.

Constraints

- $1 \leq q \leq 10$

- $1 \leq n \leq 10^5$
- $0 \leq m \leq min(10^5, \frac{n \cdot (n-1)}{2})$
- $1 \leq c_{road}, c_{lib} \leq 10^5$
- $1 \leq u[i], v[i] \leq n$
- Each road connects two distinct cities.

Output Format

For each query, print an integer denoting the minimum cost of making libraries accessible to all the citizens on a new line.

Sample Input

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

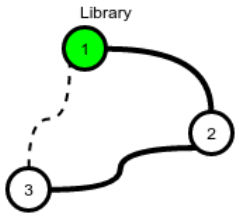
Sample Output

```
4
12
```

Explanation

We perform the following $q = 2$ queries:

1. HackerLand contains $n = 3$ cities connected by $m = 3$ bidirectional roads. The price of building a library is $c_{lib} = 2$ and the price for repairing a road is $c_{road} = 1$.

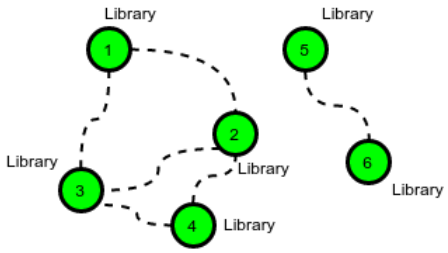
   

   The cheapest way to make libraries accessible to all is to:

   - Build a library in city 1 at a cost of $x = 2$.
   - Repair the road between cities 1 and 2 at a cost of $y = 1$.
   - Repair the road between cities 2 and 3 at a cost of $y = 1$.

   This gives us a total cost of $2 + 1 + 1 = 4$. Note that we don't need to repair the road between cities 3 and 1 because we repaired the roads connecting them to city 2.

2. In this scenario it's optimal to build a library in each city because the cost of building a library ($c_{lib} = 2$) is less than the cost of repairing a road ($c_{road} = 5$).

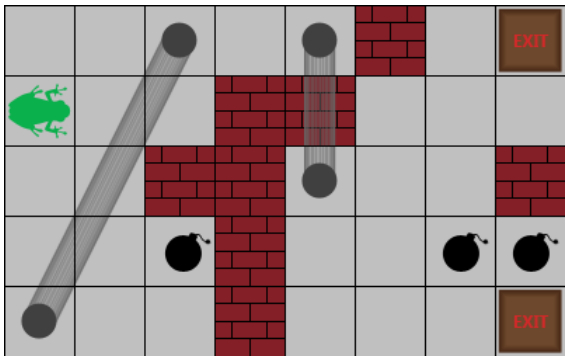


There are 6 cities, so the total cost is $6 \times 2 = 12$.

Frog in Maze

Alef the Frog is in an $n \times m$ two-dimensional maze represented as a table. The maze has the following characteristics:

- Each cell can be *free* or can contain an *obstacle*, an *exit*, or a *mine*.
- Any two cells in the table considered *adjacent* if they share a side.
- The maze is surrounded by a solid wall made of obstacles.
- Some pairs of free cells are connected by a bidirectional *tunnel*.



When Alef is in any cell, he can randomly and with equal probability choose to move into one of the adjacent cells that don't contain an obstacle in it. If this cell contains a mine, the mine explodes and Alef dies. If this cell contains an exit, then Alef escapes the maze.

When Alef lands on a cell with an entrance to a *tunnel*, he is immediately transported through the tunnel and is thrown into the cell at the other end of the tunnel. Thereafter, he won't fall again, and will now randomly move to one of the adjacent cells again. (He could possibly fall in the same tunnel later.)

It's possible for Alef to get stuck in the maze in the case when the cell in which he was thrown into from a tunnel is surrounded by obstacles on all sides.

Your task is to write a program which calculates and prints a probability that Alef escapes the maze.

Input Format

The first line contains three space-separated integers n, m and k denoting the dimensions of the maze and the number of bidirectional tunnels.

The next n lines describe the maze. The i'th line contains a string of length m denoting the i'th row of the maze. The meaning of each character is as follows:

- `#` denotes an obstacle.
- `A` denotes a free cell where Alef is initially in.
- `*` denotes a cell with a mine.
- `%` denotes a cell with an exit.
- `O` denotes a free cell (which may contain an entrance to a tunnel).

The next k lines describe the tunnels. The i'th line contains four space-separated integers i_1, j_1, i_2, j_2. Here, (i_1, j_1) and (i_2, j_2) denote the coordinates of both entrances of the tunnel. (i, j) denotes the row and column number, respectively.

Constraints

- $1 \leq n, m \leq 20$

- $0 \leq 2 \cdot k \leq n \cdot m$
- $1 \leq i_1, i_2 \leq n$
- $1 \leq j_1, j_2 \leq m$
- (i_1, j_1) and (i_2, j_2) are distinct.
- A appears exactly once.
- Each free cell contains at most one entrance to a tunnel.
- If a cell contains an entrance to a tunnel, then it doesn't contain an obstacle, mine or exit, and Alef doesn't initially stand in it.
- Tunnels don't connect adjacent cells.

Output Format

Print one real number denoting the probability that Alef escapes the maze. Your answer will be considered to be correct if its (absolute) difference from the true answer is not greater than 10^{-6}.

Sample Input 0

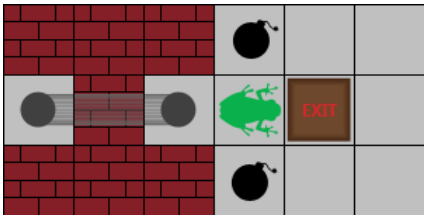
```
3 6 1
###*OO
O#OA%O
###*OO
2 3 2 1
```

Sample Output 0

```
0.25
```

Explanation 0

The following depicts this sample case:



In this case, Alef will randomly choose one of four adjacent cells. If he goes up or down, he will explode and die. If he goes right, he will escape. If he goes left, he will go through a tunnel and get stuck in cell $(2, 1)$. So the probability of Alef escaping is $\frac{1}{4}$.

Journey to the Moon

The member states of the UN are planning to send 2 people to the Moon. They want to choose two from different countries. You will be given a list of pairs of astronaut id's. Each pair is made of astronauts from the same country. Determine how many pairs of astronauts from different countries they can choose from.

For example, we have the following data on 2 pairs of astronauts, and 4 astronauts total, numbered 0 through 3.

```
ID  Country ID
1   2
2   3
```

Astronauts by country are $[0]$ and $[1, 2, 3]$. There are 3 pairs to choose from: $[0, 1]$, $[0, 2]$ and $[0, 3]$.

Input Format

The first line contains two integers n and p, the number of astronauts and the number of pairs.
Each of the next p lines contains 2 space-separated integers denoting astronaut ID's of two who share the same nationality.

Constraints

- $1 \leq n \leq 10^5$
- $1 \leq p \leq 10^4$

Output Format

An integer that denotes the number of ways to choose a pair of astronauts from different coutries.

Sample Input 0

```
5 3
0 1
2 3
0 4
```

Sample Output 0

```
6
```

Explanation 0

Persons numbered $[0, 1, 4]$ belong to one country, and those numbered $[2, 3]$ belong to another. The UN has 6 ways of choosing a pair:

$[0, 2], [0, 3], [1, 2], [1, 3], [4, 2], [4, 3]$

Sample Input 1

```
4 1
0 2
```

Sample Output 1

```
5
```

Explanation 1

Persons numbered $[0, 2]$ belong to the same country, but persons 1 and 3 don't share countries with anyone else. The UN has 5 ways of choosing a pair:

$[0, 1], [0, 3], [1, 2], [1, 3], [2, 3]$

Synchronous Shopping

Bitville is a seaside city that has N shopping centers connected via M bidirectional roads. Each road connects exactly two distinct shopping centers and has a travel time associated with it.

There are K different types of fish sold in Bitville. Historically, any shopping center has a fishmonger selling certain types of fish. Buying any amount of fish from any fishmonger takes no time.

Our heroes, *Big Cat* and *Little Cat*, are standing at Bitville shopping center number 1. They have a list of the types of fish sold at each fishmonger, and they want to collectively purchase all K types of fish in a minimal amount of time. To do this, they decide to split the shopping between themselves in the following way:

- Both cats choose their own paths, starting at shopping center 1 and ending at shopping center N. It should be noted that Little Cat's path is not necessarily the same as Big Cat's.
- While traveling their respective paths, each cat will buy certain types of fish at certain shops.
- When the cats reach shopping center N, they must have collectively purchased all K types of fish in a minimal amount of time.
- If one cat finishes shopping before the other, he waits at shopping center N for his partner to finish; this means that the total shopping time is the maximum of Little and Big Cats' respective shopping times.

It is to be noted that any of the cats can visit the shopping center N in between, but they both *have* to finish their paths at the shopping center N.

Given the layout for Bitville and the list of fish types sold at each fishmonger, what is the minimum amount of time it will take for Big and Little Cat to purchase all K types of fish and meet up at shopping center N?

Input Format

The first line contains 3 space-separated integers: N (the number of shopping centers), M (the number of roads), and K (the number of types of fish sold in Bitville), respectively.

Each line i of the N subsequent lines ($1 \leq i \leq n$) describes a shopping center as a line of space-separated integers. Each line takes the following form:

- The first integer, T_i, denotes the number of types of fish that are sold by the fishmonger at the i^{th} shopping center.
- Each of the T_i subsequent integers on the line describes a type of fish sold by that fishmonger. Which is denoted by $A_{i,j}$.

Each line j of the M subsequent lines ($1 \leq j \leq m$) contains 3 space-separated integers describing a road. The first two integers, X_j and Y_j, describe the two shopping centers it connects. The third integer, Z_j, denotes the amount of time it takes to travel the road (i.e., travel time).

Constraints

- $2 \leq N \leq 10^3$
- $1 \leq M \leq 2 \times 10^3$
- $1 \leq K \leq 10$

- $0 \leq T_i \leq K$
- $1 \leq A_{i,j} \leq K$
- All $A_{i,j}$ are different for every fixed i.
- $1 \leq X_j, Y_j \leq N$
- $1 \leq Z_j \leq 10^4$
- Each road connectes 2 distinct shopping centers (i.e., no road connects a shopping center to itself).
- Each pair of shopping centers is directly connected by no more than 1 road.
- It is possible to get to any shopping center from any other shopping center.
- Each type of fish is always sold by at least one fishmonger.

Output Format

Print the minimum amount of time it will take for the cats to collectively purchase all K fish and meet up at shopping center N.

Sample Input

```
5 5 5
1 1
1 2
1 3
1 4
1 5
1 2 10
1 3 10
2 4 10
3 5 10
4 5 10
```

Sample Output

```
30
```

Explanation

Big Cat can choose the following route: $1 \rightarrow 2 \rightarrow 4 \rightarrow 5$, and buy fish at all of the shopping centers on his way.

Little Cat can choose the following route: $1 \rightarrow 3 \rightarrow 5$, and buy fish from the fishmonger at the 3^{rd} shopping center only.

Subset Component

You are given an array with n 64-bit integers: $d[0], d[1], \ldots, d[n-1]$.

BIT(x, i) = (x >> i) & 1. (where $B(x, i)$ is the i^{th} lower bit of x in binary form.)

If we regard every bit as a vertex of a graph G, there exists one undirected edge between vertex i and vertex j if there exists at least one k such that BIT(d[k], i) == 1 && BIT(d[k], j) == 1.

For every subset of the input array, how many connected-components are there in that graph?

The number of connected-components in a graph are the sets of nodes, which are accessible to each other, but not to/from the nodes in any other set.

For example if a graph has six nodes, labelled $\{1, 2, 3, 4, 5, 6\}$. And contains the edges $(1, 2), (2, 4) and (3, 5)$. There are three connected-components: $\{1, 2, 4\}$, $\{3, 5\}$ and $\{6\}$. Because $\{1, 2, 4\}$ can be accessed from each other through one or more edges, $\{3, 5\}$ can access each other and $\{6\}$ is isolated from everone else.

You only need to output the sum of the number of connected-component(S) in every graph.

Input Format

```
n
d[0] d[1] ... d[n - 1]
```

Constraints

$1 \leq n \leq 20$
$0 \leq d[i] \leq 2^{63} - 1$

Output Format

Print the value of S.

Sample Input 0

```
3
2 5 9
```

Sample Output 0

```
504
```

Explanation 0

There are 8 subset of $\{2, 5, 9\}$.

{}
=> We don't have any number in this subset => no edge in the graph => Every node is a component by itself => Number of connected-components = 64.

{2}
=> The Binary Representation of 2 is 00000010. There is a bit at only one position. => So there is no edge in the graph, every node is a connected-component by itself => Number of connected-components = 64.

{5}

=> The Binary Representation of 5 is 00000101. There is a bit at the 0^{th} and 2^{nd} position. => So there is an edge: (0, 2) in the graph => There is one component with a pair of nodes (0,2) in the graph. Apart from that, all remaining 62 vertices are indepenent components of one node each (1,3,4,5,6...63) => Number of connected-components = 63.

{9}
=> The Binary Representation of 9 is 00001001. => There is a 1-bit at the 0^{th} and 3^{rd} position in this binary representation. => edge: (0, 3) in the graph => Number of components = 63

{2, 5}
=> This will contain the edge (0, 2) in the graph which will form one component
=> Other nodes are all independent components
=> Number of connected-component = 63

{2, 9}
=> This has edge (0,3) in the graph
=> Similar to examples above, this has 63 connected components

{5, 9}
=> This has edges (0, 2) and (0, 3) in the graph
=> Similar to examples above, this has 62 connected components

{2, 5, 9}
=> This has edges(0, 2) (0, 3) in the graph. All three vertices (0,2,3) make one component => Other 61 vertices are all independent components
=> Number of connected-components = 62

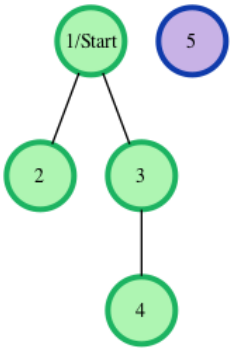
$$S = 64 + 64 + 63 + 63 + 63 + 63 + 62 + 62 = 504$$

Breadth First Search: Shortest Reach

Consider an undirected graph where each edge is the same weight. Each of the nodes is labeled consecutively.

You will be given a number of queries. For each query, you will be given a list of edges describing an undirected graph. After you create a representation of the graph, you must determine and report the shortest distance to each of the other nodes from a given starting position using the *breadth-first search* algorithm (BFS). Distances are to be reported in node number order, ascending. If a node is unreachable, print -1 for that node. Each of the edges weighs 6 units of distance.

For example, given a graph with 5 nodes and 3 edges, $[1, 2], [1, 3], [3, 4]$, a visual representation is:



The start node for the example is node 1. Outputs are calculated for distances to nodes 2 through 5: $[6, 6, 12, -1]$. Each edge is 6 units, and the unreachable node 5 has the required return distance of -1.

Function Description

Complete the *bfs* function in the editor below. It must return an array of integers representing distances from the start node to each other node in node ascending order. If a node is unreachable, its distance is -1.

bfs has the following parameter(s):

- *n*: the integer number of nodes
- *m*: the integer number of edges
- *edges*: a 2D array of start and end nodes for edges
- *s*: the node to start traversals from

Input Format

The first line contains an integer q, the number of queries. Each of the following q sets of lines has the following format:

- The first line contains two space-separated integers n and m, the number of nodes and edges in the graph.
- Each line i of the m subsequent lines contains two space-separated integers, u and v, describing an edge connecting node u to node v.
- The last line contains a single integer, s, denoting the index of the starting node.

Constraints

- $1 \leq q \leq 10$

- $2 \leq n \leq 1000$
- $1 \leq m \leq \frac{n \cdot (n-1)}{2}$
- $1 \leq u, v, s \leq n$

Output Format

For each of the q queries, print a single line of $n-1$ space-separated integers denoting the shortest distances to each of the $n-1$ other nodes from starting position s. These distances should be listed sequentially by node number (i.e., $1, 2, \ldots, n$), but *should not* include node s. If some node is unreachable from s, print -1 as the distance to that node.

Sample Input

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

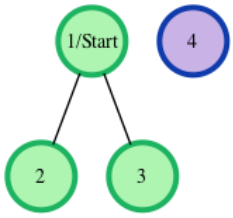
Sample Output

```
6 6 -1
-1 6
```

Explanation

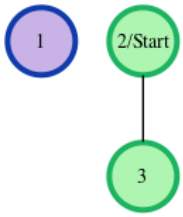
We perform the following two queries:

1. The given graph can be represented as:



   where our *start* node, s, is node 1. The shortest distances from s to the other nodes are one edge to node 2, one edge to node 3, and an infinite distance to node 4 (which it's not connected to). We then print node 1's distance to nodes 2, 3, and 4 (respectively) as a single line of space-separated integers: `6, 6, -1`.

2. The given graph can be represented as:



   where our *start* node, s, is node 2. There is only one edge here, so node 1 is unreachable from node 2 and node 3 has one edge connecting it to node 2. We then print node 2's distance to nodes 1 and 3 (respectively) as a single line of space-separated integers: `-1 6`.

Note: Recall that the actual length of each edge is 6, and we print -1 as the distance to any node that's unreachable from s.

Kruskal (MST): Really Special Subtree

Given an undirected weighted connected graph, it is required to find the Really Special SubTree in it. The Really Special SubTree is defined as a subgraph consisting of all the nodes in the graph and

- There is only one exclusive path from a node to every other node.
- The subgraph is of minimum overall weight (sum of all edges) among all such subgraphs.
- No cycles are formed

To create the Really Special SubTree, always picking the edge with smallest weight. Determine if it will create a cycle. If so, ignore the edge. If there are edges of equal weight available:

- Choose the edge that minimizes the sum $u + v + wt$ where u and v are vertices and wt is the edge weight.
- If there is still a collision, choose any of them.
- While doing the above, ensure that no cycle is formed while picking up edges.

Print the overall weight of the Tree so formed using above rules.

For example, given the following edges:

```
u v wt
1 2 2
2 3 3
3 1 5
```

First we would choose $1 \rightarrow 2$ at weight 2. Next we would choose $2 \rightarrow 3$ at weight 3. All nodes are connected without cycles for a total weight of $3 + 2 = 5$.

Input Format

The first line has two integers g_nodes and g_edges, the number of nodes and edges in the graph.

The next g_edges lines each consist of three space separated integers g_from, g_to g_weight, where g_from and g_to denote the two nodes between which the **undirected** edge exists and g_weight denotes the weight of that edge.

Constraints

- $2 \leq g\_nodes \leq 3000$
- $1 \leq g\_edges \leq (N * (N-1))/2$
- $1 \leq g\_from, g\_to \leq N$
- $0 \leq g\_weight \leq 10^5$

**Note: ** If there are edges between the same pair of nodes with different weights, they are to be considered as is, like multiple edges.

Output Format

Print a single integer denoting the total weight of the Really Special SubTree.

Sample Input 0

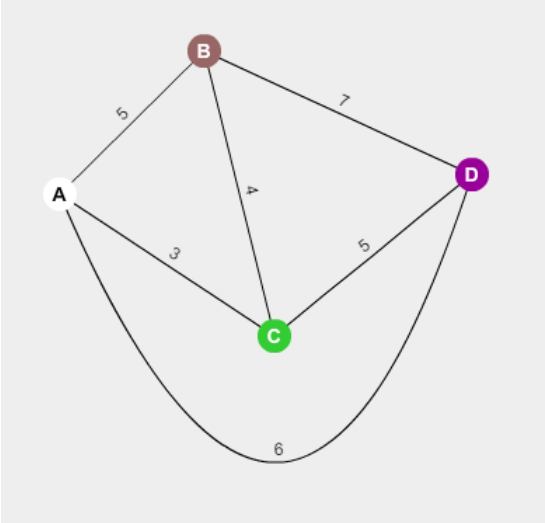
```
4 6
1 2 5
1 3 3
4 1 6
2 4 7
3 2 4
3 4 5
```

Sample Output 0

```
12
```

Explanation 0

The graph given in the test case is shown as :



- The nodes A,B,C and D denote the 1,2,3 and 4 node numbers.
- The starting node is A or 1 in the given test case.

Applying Kruskal's algorithm, all the edges are sorted in ascending order of weight.

After sorting, the edge choices are available as :

A->C (WT. 3) , B->C (WT. 4) , A->B (WT. 5) , C->D (WT. 5) , A->D (WT. 6) and B->D (WT. 7)

Picking these edges and finalizing only if it doesnt create a cycle :

A->C : B->C

The edge **A->B** would form a cycle so it is ignored.

The edge **C->D** is chosen to finish the MST:

A->C : B->C : C->D

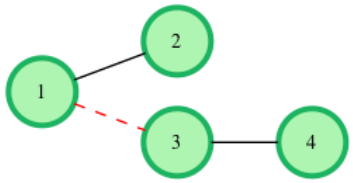
The total weight of the Really Special SubTree is : **12**

Even Tree

You are given a tree (a simple connected graph with no cycles).

Find the maximum number of edges you can remove from the tree to get a forest such that each connected component of the forest contains an even number of nodes.

As an example, the following tree with 4 nodes can be cut at most 1 time to create an even forest.



Function Description

Complete the *evenForest* function in the editor below. It should return an integer as described.

evenForest has the following parameter(s):

- *t_nodes*: the number of nodes in the tree
- *t_edges*: the number of undirected edges in the tree
- *t_from*: start nodes for each edge
- *t_to*: end nodes for each edge, (Match by index to *t_from*.)

Input Format

The first line of input contains two integers t_nodes and t_edges, the number of nodes and edges.
The next t_edges lines contain two integers $t_from[i]$ and $t_to[i]$ which specify nodes connected by an edge of the tree. The root of the tree is node 1.

Constraints

- $2 \leq n \leq 100$
- $n \in \mathbb{Z}^{+}_{\text{even}}$

Note: The tree in the input will be such that it can always be decomposed into components containing an even number of nodes. $\mathbb{Z}^{+}_{\text{even}}$ is the set of positive even integers.

Output Format

Print the number of removed edges.

Sample Input 0

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

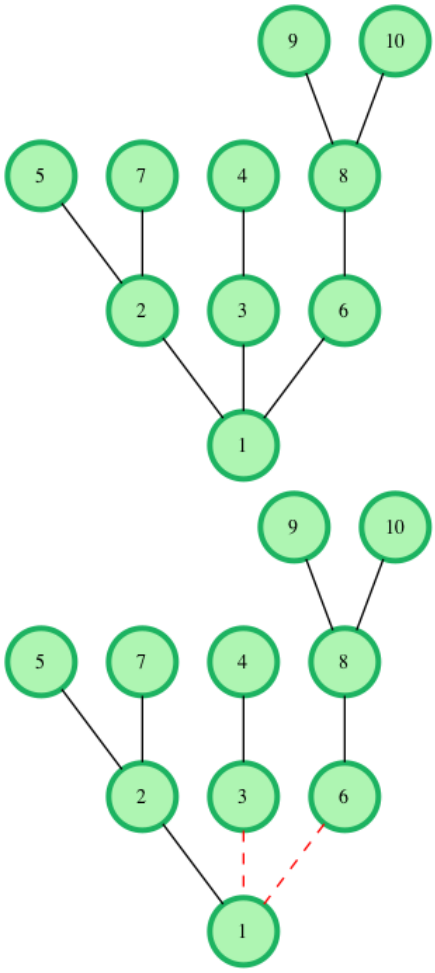
Sample Output 0

```
2
```

Explanation 0

Remove edges $(1, 3)$ and $(1, 6)$ to get the desired result.

. Original tree Decomposed tree



No more edges can be removed.

Snakes and Ladders: The Quickest Way Up

Markov takes out his Snakes and Ladders game and stares at the board, and wonders: If he had absolute control on the die (singular), and could get it to generate any number (in the range $1 - 6$) he desired, what would be the least number of rolls of the die in which he'd be able to reach the destination square (Square Number 100) after having started at the base square (Square Number 1)?

Rules

1. Markov has total control over the die and the face which shows up every time he tosses it. You need to help him figure out the minimum number of moves in which he can reach the target square (100) after starting at the base (Square 1).
2. A die roll which would cause the player to land up at a square greater than 100, goes wasted, and the player remains at his original square. Such as a case when the player is at Square Number 99, rolls the die, and ends up with a 5.
3. If a person reaches a square which is the base of a ladder, (s)he has to climb up that ladder, and he cannot come down on it. If a person reaches a square which has the mouth of the snake, (s)he has to go down the snake and come out through the tail - there is no way to climb down a ladder or to go up through a snake.

Constraints

The board is always of the size 10×10 and Squares are always numbered 1 to 100.

$1 <= T <= 10$
$1 <= \text{Number of Ladders} <= 15$
$1 <= \text{Number of Snakes} <= 15$

Square number 1 and 100 will not be the starting point of a ladder or a snake.
No square will have more than one of the starting or ending point of a snake or ladder (e.g. snake 56 to 44 and ladder 44 to 97 is not possible because 44 has both ending of a snake and a starting of a ladder)

Input Format

The first line contains the number of tests, T. T testcases follow.

For each testcase, the first line contain N(Number of ladders) and after that N line follow. Each of the N line contain 2 integer representing the starting point and the ending point of a ladder respectively.

The next line contain integer M(Number of snakes) and after that M line follow. Each of the M line contain 2 integer representing the starting point and the ending point of a snake respectively.

Output Format

For each of the T test cases, output one integer, each in a new line, which is the least number of moves (or rolls of the die) in which the player can reach the target square (Square Number 100) after starting at the base (Square Number 1). This corresponds to the ideal sequence of faces which show up when the die is rolled.
If there is no solution, print `-1`.

Sample Input

```
2
3
32 62
42 68
12 98
7
95 13
97 25
93 37
79 27
75 19
49 47
67 17
4
8 52
6 80
26 42
2 72
9
51 19
39 11
37 29
81 3
59 5
79 23
53 7
43 33
77 21
```

Sample Output

```
3
5
```

Explanation

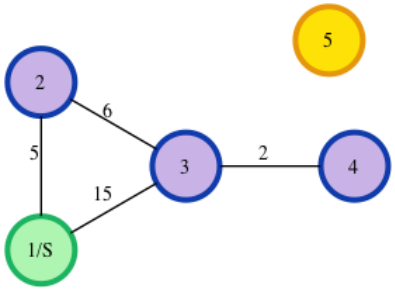
For the first test: To traverse the board via the shortest route, the player first rolls the die to get a 5, and ends up at square 6. He then rolls the die to get 6. He ends up at square 12, from where he climbs the ladder to square 98. He then rolls the die to get '2', and ends up at square 100, which is the target square. So, the player required 3 rolls of the die for this shortest and best case scenario. So the answer for the first test is 3.

Dijkstra: Shortest Reach 2

Given an undirected graph and a starting node, determine the lengths of the shortest paths from the starting node to all other nodes in the graph. If a node is unreachable, its distance is -1. Nodes will be numbered consecutively from 1 to n, and edges will have varying distances or lengths.

For example, consider the following graph of 5 nodes:

```
Begin End Weight
1 2 5
2 3 6
3 4 2
1 3 15
```



Starting at node 1, the shortest path to 2 is direct and distance 5. Going from 1 to 3, there are two paths: $1 \rightarrow 2 \rightarrow 3$ at a distance of $5 + 6 = 11$ or $1 \rightarrow 3$ at a distance of 15. Choose the shortest path, 11. From 1 to 4, choose the shortest path through 3 and extend it: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ for a distance of $11 + 2 = 13$ There is no route to node 5, so the distance is -1.

The distances to all nodes in increasing node order, omitting the starting node, are `5 11 13 -1`.

Function Description

Complete the *shortestReach* function in the editor below. It should return an array of integers that represent the shortest distance to each node from the start node in ascending order of node number.

shortestReach has the following parameter(s):

- *n*: the number of nodes in the graph
- *edges*: a 2D array of integers where each $edges[i]$ consists of three integers that represent the start and end nodes of an edge, followed by its length
- *s*: the start node number

Input Format

The first line contains t, the number of test cases.

Each test case is as follows:
- The first line contains two space-separated integers n and m, the number of nodes and edges in the graph.
- Each of the next m lines contains three space-separated integers x, y, and r, the beginning and ending nodes of an edge, and the length of the edge.
- The last line of each test case has an integer s, denoting the starting position.

Constraints

$1 \leq t \leq 10$
$2 \leq n \leq 3000$

$1 \leq m \leq \frac{N \times (N-1)}{2}$
$1 \leq x, y, s \leq N$
$1 \leq r \leq 10^5$

If there are edges between the same pair of nodes with different weights, they are to be considered as is, like multiple edges.

Output Format

For each of the t test cases, print a single line consisting $n-1$ space separated integers denoting the shortest distance to the $n-1$ nodes from starting position s in increasing order of their labels, excluding s.

For unreachable nodes, print -1.

Sample Input

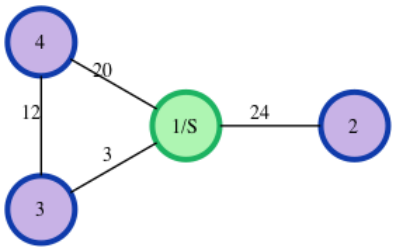
```
1
4 4
1 2 24
1 4 20
3 1 3
4 3 12
1
```

Sample Output

```
24 3 15
```

Explanation

The graph given in the test case is shown as :



* The lines are weighted edges where weight denotes the length of the edge.

The shortest paths followed for the three nodes 2, 3 and 4 are as follows :

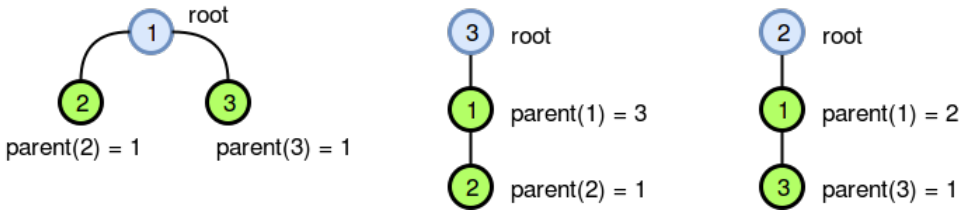
1/S->2 - Shortest Path Value : 24

1/S->3 - Shortest Path Value : 3

1/S->3->4 - Shortest Path Value : 15

The Story of a Tree

One day Bob drew a tree, T, with n nodes and $n-1$ edges on a piece of paper. He soon discovered that parent of a node depends on the root of the tree. The following images shows an example of that:



Learning the fact, Bob invented an exciting new game and decided to play it with Alice. The rules of the game is described below:

1. Bob picks a random node to be the tree's *root* and keeps the identity of the chosen node a secret from Alice. Each node has an equal probability of being picked as the root.
2. Alice then makes a list of g guesses, where each guess is in the form `u v` and means Alice guesses that $parent(v) = u$ is *true*. It's guaranteed that an undirected edge connecting u and v exists in the tree.
3. For each correct guess, Alice earns one point. Alice wins the game if she earns at least k points (i.e., at least k of her guesses were *true*).

Alice and Bob play q games. Given the tree, Alice's guesses, and the value of k for each game, find the probability that Alice will win the game and print it on a new line as a reduced fraction in the format `p/q`.

Input Format

The first line contains an integer, q, denoting the number of different games. The subsequent lines describe each game in the following format:

1. The first line contains an integer, n, denoting the number of nodes in the tree.
2. The $n-1$ subsequent lines contain two space-separated integers, u and v, defining an undirected edge between nodes u and v.
3. The next line contains two space-separated integers describing the respective values of g (the number of guesses) and k (the minimum score needed to win).
4. Each of the g subsequent lines contains two space-separated integers, u and v, indicating Alice guesses $parent(v) = u$.

Constraints

- $1 \leq q \leq 5$
- $1 \leq n \leq 10^5$
- $1 \leq u, v \leq n$
- $1 \leq g, k \leq 10^5$
- The sum of n over all test cases won't exceed 2×10^5.
- No two guesses will be identical.

Scoring

- For 25% of the maximum score, $1 \leq n \leq 10^3$.
- For 100% of the maximum score, $1 \leq n \leq 10^5$.

Output Format

Print the probability as a reduced fraction in the format `p/q`.

Note: Print `0/1` if the probability is 0 and print `1/1` if the probability is 1.

Sample Input 0

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

Sample Output 0

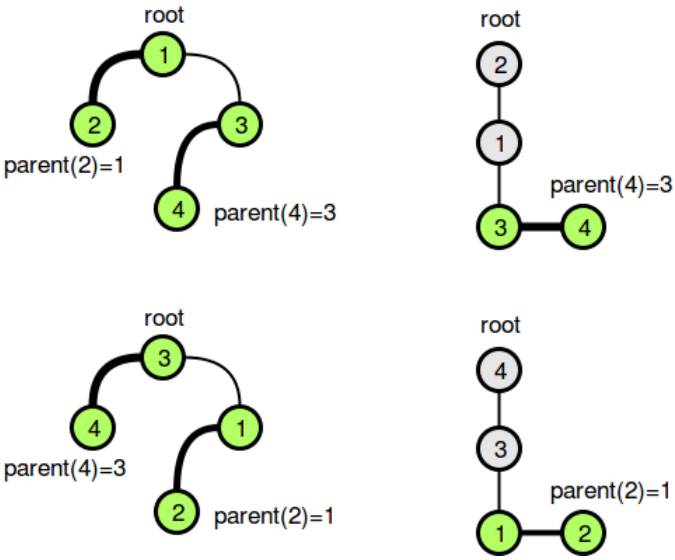
```
1/2
1/3
```

Explanation 0

Alice and Bob play the following $g = 2$ games:

1. Alice makes two guesses, $(1\ 2)$ and $(3\ 4)$, meaning she guessed that $parent(2) = 1$ and $parent(4) = 3$. To win the game, at least $k = 2$ of her guesses must be *true*.

   In the diagrams below, you can see that at least 2 guesses are *true* if the root of the tree is either node 1 or 3:

   There are 4 nodes in total and the probability of picking node 1 or 3 as the root is $\frac{2}{4}$, which reduces to $\frac{1}{2}$.

2. In this game, Alice only wins if node 1 is the root of the tree. There are 3 nodes in total, and the probability of picking node 1 as the root is $\frac{1}{3}$.

Prim's (MST) : Special Subtree

Given a graph which consists of several edges connecting its nodes, find a subgraph of the given graph with the following properties:

- The subgraph contains all the nodes present in the original graph.
- The subgraph is of minimum overall weight (sum of all edges) among all such subgraphs.
- It is also required that there is **exactly one, exclusive** path between any two nodes of the subgraph.

One specific node S is fixed as the starting point of finding the subgraph using Prim's Algorithm.
Find the total weight or the sum of all edges in the subgraph.

For example, consider a graph with 3 nodes. Edges are $1 \leftrightarrow 2$ weight 2, $2 \leftrightarrow 3$ weight 3 and $1 \leftrightarrow 3$ weight 3. Starting from 1, we select the lowest weight path, i.e. $1 \leftrightarrow 2$. From 2, there is only one path $2 \leftrightarrow 3$. We have all nodes connected at a cost of $2 + 3 = 5$.

Input Format

The first line has two space-separated integers n and m, the number of nodes and edges in the graph.

Each of the next m lines contains three space-separated integers x, y and r, the end nodes of $edges[i], and the edge's weight. The last line has an integer$ start$, denoting the starting node.

Constraints

$2 \leq n \leq 3000$
$1 \leq m \leq (n * (n-1))/2$
$1 \leq x, y, start \leq n$
$0 <= r <= 10^5$
There may be multiple edges between two nodes.

Output Format

Print a single integer denoting the total weight of the subgraph.

Sample Input 0

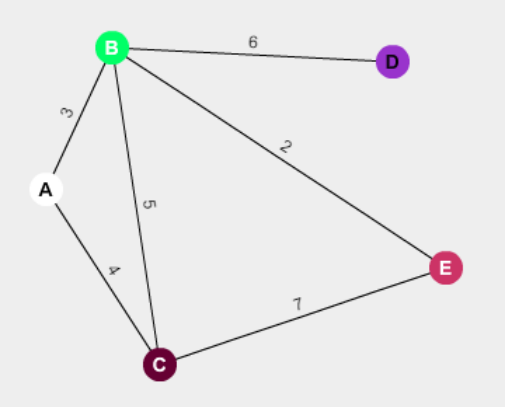
```
5 6
1 2 3
1 3 4
4 2 6
5 2 2
2 3 5
3 5 7
1
```

Sample Output 0

```
15
```

Explanation 0

The graph given in the test case is shown as :



- The nodes A,B,C,D and E denote the obvious 1,2,3,4 and 5 node numbers.
- The starting node is A or 1 (in the given test case)

Applying the Prim's algorithm, edge choices available at first are :

A->B (**WT. 3**) and A->C (**WT. 4**) , out of which A->B is chosen (smaller weight of edge).

Now the available choices are :

A->C (**WT. 4**) , B->C (**WT. 5**) , B->E (**WT. 2**) and B->D (**WT. 6**) , out of which B->E is chosen by the algorithm.

Following the same method of the algorithm, the next chosen edges , sequentially are :

A->C and B->D.

Hence the overall sequence of edges picked up by prims are:

A->B : B->E : A->C : B->D

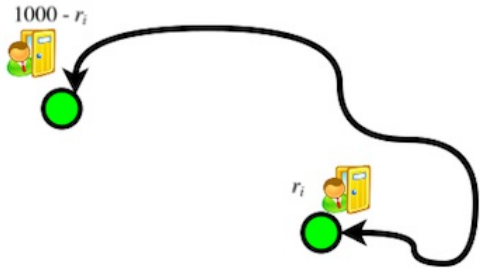
and the total weight of the MST (minimum spanning tree) is : **3+2+4+7=15**

Toll Cost Digits

The mayor of Farzville is studying the city's road system to find ways of improving its traffic conditions. Farzville's road system consists of n junctions connected by e bidirectional toll roads, where the i^{th} toll road connects junctions x_i and y_i. In addition, some junctions may not be reachable from others and there may be multiple roads connecting the same pair of junctions.

Each toll road has a toll rate that's paid each time it's used. This rate varies depending on the direction of travel:

- If traveling from x_i to y_i, then the toll rate is r_i.
- If traveling from y_i to x_i, then the toll rate is $1000 - r_i$. It is guaranteed that $0 < r_i < 1000$.



For each digit $d \in \{0, 1, \ldots, 9\}$, the mayor wants to find the number of ordered pairs of (x, y) junctions such that $x \neq y$ and a path exists from x to y where the total cost of the tolls (i.e., the sum of all toll rates on the path) ends in digit d. Given a map of Farzville, can you help the mayor answer this question? For each digit d from 0 to 9, print the the number of valid ordered pairs on a new line.

Note: Each toll road can be traversed an unlimited number of times in either direction.

Input Format

The first line contains two space-separated integers describing the respective values of n (the number of junctions) and e (the number of roads).
Each line i of the e subsequent lines describes a toll road in the form of three space-separated integers, x_i, y_i, and r_i.

Constraints

- $1 \leq n \leq 10^5$
- $1 \leq e \leq 2 \cdot 10^5$
- $1 \leq x_i, y_i \leq n$
- $x_i \neq y_i$
- $0 < r_i < 1000$

Output Format

Print ten lines of output. Each line j (where $0 \leq j \leq 9$) must contain a single integer denoting the answer for $d = j$. For example, the first line must contain the answer for $d = 0$, the second line must contain the answer for $d = 1$, and so on.

Sample Input 0

```
3 3
1 3 602
1 2 256
```

```
2 3 411
```

Sample Output 0

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

Explanation 0

The table below depicts the distinct pairs of junctions for each d:

d	(x, y)	path	total cost
0	none		
1	$(1, 2)$	$1 \to 3 \to 2$	1191
	$(2, 3)$	$2 \to 3$	411
2	$(1, 3)$	$1 \to 3$	602
3	$(3, 1)$	$3 \to 2 \to 1$	1333
4	$(2, 1)$	$2 \to 1$	744
	$(3, 2)$	$3 \to 1 \to 2$	654
5	none		
6	$(1, 2)$	$1 \to 2$	256
	$(2, 3)$	$2 \to 1 \to 3$	1346
7	$(1, 3)$	$1 \to 2 \to 3$	667
8	$(3, 1)$	$3 \to 1$	398
9	$(2, 1)$	$2 \to 3 \to 1$	809
	$(3, 2)$	$3 \to 2$	589

Note the following:

- There may be multiple paths between each pair of junctions.
- Junctions and roads may be traversed multiple times. For example, the path $2 \to 3 \to 1 \to 2 \to 3$ is also valid, and it has total cost of $411 + 398 + 256 + 411 = 1476$.
- An ordered pair can be counted for more than one d. For example, the pair $(2, 3)$ is counted for $d = 1$ and $d = 6$.
- Each ordered pair must only be counted once for each d. For example, the paths $2 \to 1 \to 3$ and $2 \to 3 \to 1 \to 2 \to 3$ both have total costs that end in $d = 6$, but the pair $(2, 3)$ is only counted once.

Real Estate Broker

You are a real estate broker in ancient Knossos. You have m unsold houses, and each house j has an area, x_j, and a minimum price, y_j. You also have n clients, and each client i wants a house with an area greater than a_i and a price less than or equal to p_i.

Each client can buy *at most* one house, and each house can have *at most* one owner. What is the maximum number of houses you can sell?

Input Format

The first line contains two space-separated integers describing the respective values of n (the number of clients) and m (the number of houses).
Each line i of the n subsequent lines contains two space-separated integers describing the respective values of a_i and p_i for client i.
Each line j of the m subsequent lines contains two space-separated integers describing the respective values of x_j and y_j for house j.

Constraints

- $1 \leq n, m \leq 1000$
- $1 \leq a_i, p_i \leq 10^9$, where $0 \leq i < n$.
- $1 \leq x_j, y_j \leq 10^9$, where $0 \leq j < m$.

Output Format

Print a single integer denoting the maximum number of houses you can sell.

Sample Input 0

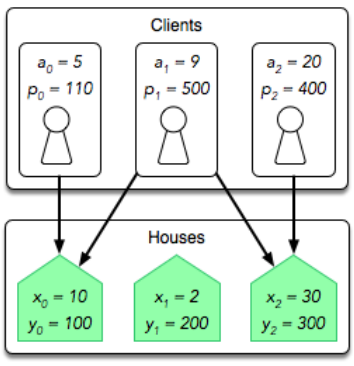
```
3 3
5 110
9 500
20 400
10 100
2 200
30 300
```

Sample Output 0

```
2
```

Explanation 0

Recall that each client i is only interested in some house j where $x_j > a_i$ and $y_j \leq p_i$. The diagram below depicts which clients will be interested in which houses:



- Client 0 will be interested in house 0 because it has more than $a_0 = 5$ units of space and costs less than $p_0 = 110$. Both of the other houses are outside of this client's price range.
- Client 1 will be interested in houses 0 and 2, as both these houses have more than $a_1 = 9$ units of space and cost less than $p_1 = 500$. They will not be interested in the remaining house because it's too small.
- Client 2 will be interested in house 2 because it has more than $a_2 = 20$ units of space and costs less than $p_2 = 400$. They will not be interested in the other two houses because they are too small.

All three clients are interested in the same two houses, so you can sell *at most* two houses in the following scenarios:

- Client 0 buys house 0 and client 1 buys house 2.
- Client 1 buys house 0 and client 2 buys house 2.
- Client 0 buys house 0 and client 2 buys house 2.

Thus, we print the maximum number of houses you can sell, 2, on a new line.

Clique

A clique in a graph is set of nodes such that there is an edge between any two distinct nodes in the set. Finding the largest clique in a graph is a computationally tough problem. Currently no polynomial time algorithm is known for solving this. However, you wonder what is the minimum size of the largest clique in any graph with N nodes and M edges.

Input Format

The first line contains T the number of test cases. Each of the next T lines contain 2 integers : N, M

Constraints

- $1 <= T <= 100000$
- $2 <= N <= 10000$
- $1 <= M <= N \times (N-1)/2$

Output Format

Output T lines, one for each test case, containing the desired answer for the corresponding test case.

Sample Input

```
3
3 2
4 6
5 7
```

Sample Output

```
2
4
3
```

Explanation

For the second test case, the only valid graph having 4 nodes and 6 edges is one where each pair of nodes is connected. So the size of the largest clique cannot be smaller than 4.
For the third test case, it is easy to verify that any graph with 5 nodes and 7 edges will surely have a clique of size 3 or more.

Hints Turan's theorem gives us an upper bound on the number of edges a graph can have if we wish that it should not have a clique of size x. Though the bound is not exact, it is easy to extend the statement of the theorem to get an exact bound in terms of n and x. Once this is done, we can binary search for the largest x such that $f(n, x) <= m$. See: Turan's Theorem

Minimum Penalty Path

Consider an undirected graph containing N nodes and M edges. Each edge M_i has an integer *cost*, C_i, associated with it.

The *penalty* of a path is the *bitwise OR* of every edge cost in the path between a pair of nodes, A and B. In other words, if a path contains edges $M_1, M_2, \ldots, M_k$, then the penalty for this path is C_1 **OR** C_2 **OR** ... **OR** C_k.

Given a graph and two nodes, A and B, find the path between A and B having the *minimal possible penalty* and print its penalty; if no such path exists, print -1 to indicate that there is no path from A to B.

Note: Loops and multiple edges are allowed. The bitwise OR operation is known as **or** in Pascal and as **|** in C++ and Java.

Input Format

The first line contains two space-separated integers, N (the number of nodes) and M (the number of edges), respectively.

Each line i of the M subsequent lines contains three space-separated integers U_i, V_i, and C_i, respectively, describing edge M_i connecting the nodes U_i and V_i and its associated penalty (C_i).

The last line contains two space-separated integers, A (the starting node) and B (the ending node), respectively.

Constraints

- $1 \leq N \leq 10^3$
- $1 \leq M \leq 10^4$
- $1 \leq C_i < 1024$
- $1 \leq U_i, V_i \leq N$
- $1 \leq A, B \leq N$
- $A \neq B$

Output Format

Print the minimal penalty for the optimal path from node A to node B; if no path exists from node A to node B, print -1.

Sample Input

```
3 4
1 2 1
1 2 1000
2 3 3
1 3 100
1 3
```

Sample Output

```
3
```

Explanation

The optimal path is $1 \to 2 \to 3$.
$C_{(1,2)} = 1$ and $C_{(2,3)} = 3$.
The penalty for this path is: 1 **OR** $3 = 3$, so we print 3.

Demanding Money

Killgrave wants to use his mind control powers to get money from the Justice League superheroes living in N houses in Happy Harbor that are numbered sequentially from 1 to N. There are M roads, and each road j connects two different houses, A_j and B_j. Each superhero house i (where $1 \leq i \leq N$) has C_i dollars stashed away for a rainy day.

As long as a superhero is home at house i, Killgrave knows they will hand over all of their saved money, C_i. Once he gets money from them, he moves on to the next house. However, the superheroes are cunning; when Killgrave comes to house X, every neighbor immediately connected to house X by a single road skips town for a couple of days (making it impossible for Killgrave to get money from them). In other words, after Killgrave visits all the superheroes he wants, there will be no road in which he was able to get money from both houses on either end of the road.

What is the maximum amount of money Killgrave can collect from the superheroes, and how many *different* ways can Killgrave get that amount of money? Two ways are considered to be different if the sets of visited houses are different.

Note: Killgrave can start at an arbitrary house and doesn't have to only use the roads.

Input Format

The first line contains two space-separated integers, N (the number of houses) and M (the number of roads), respectively.
The second line contains N space-separated integers, where each integer i describes the amount of money, C_i, at house i.
Each line j of the M subsequent lines contains two space-separated integers defining a road connecting houses A_j and B_j. Every road connects a different pair of houses.

Constraints

- $1 \leq N \leq 34$
- $0 \leq M \leq N \cdot \frac{(N-1)}{2}$
- $0 \leq C_i \leq 100$
- $1 \leq A_j, B_j \leq N$, where $A_j \neq B_j$
- No unordered pair (A_j, B_j) will appear more than once.

Output Format

Print two space-separated integers:

1. The first integer must denote the maximum amount of money Killgrave can get out of the Justice League.
2. The second integer must denote the number of different ways he can collect that amount of money.

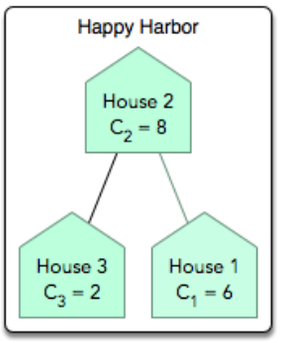
Sample Input

```
3 2
6 8 2
1 2
3 2
```

Sample Output

```
8 2
```

Explanation



Killgrave has two possible courses of action:

1. Visit house 2 and get 8 dollars.
2. Visit houses 1 and 3 and get $2 + 6 = 8$ dollars.

Both of these options result in 8 dollars, so we know that this is maximal. Thus, we print the maximum amount of money (8) followed by the number of ways he can get that amount of money (2) as two space-separated values on a single line.

The Value of Friendship

You're researching friendships between groups of n new college students where each student is distinctly numbered from 1 to n. At the beginning of the semester, no student knew any other student; instead, they met and formed individual friendships as the semester went on. The friendships between students are:

- *Bidirectional.* If student a is friends with student b, then student b is also friends with student a.
- *Transitive.* If student a is friends with student b and student b is friends with student c, then student a is friends with student c. In other words, two students are considered to be friends even if they are only indirectly linked through a network of mutual (i.e., directly connected) friends.

The purpose of your research is to find the maximum total value of a group's friendships, denoted by $total$. Each time a direct friendship forms between two students, you sum the number of friends that *each* of the n students has and add the sum to $total$.

You are given q queries, where each query is in the form of an unordered list of m distinct direct friendships between n students. For each query, find the maximum value of $total$ among all possible orderings of formed friendships and print it on a new line.

Input Format

The first line contains an integer, q, denoting the number of queries. The subsequent lines describe each query in the following format:

1. The first line contains two space-separated integers describing the respective values of n (the number of students) and m (the number of distinct *direct* friendships).
2. Each of the m subsequent lines contains two space-separated integers describing the respective values of x and y (where $x \neq y$) describing a friendship between student x and student y.

Constraints

- $1 \leq q \leq 16$
- $1 \leq n \leq 10^5$
- $1 \leq m \leq \min(\frac{n \cdot (n-1)}{2}, 2 \times 10^5)$

Output Format

For each query, print the maximum value of $total$ on a new line.

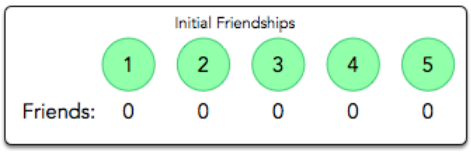
Sample Input 0

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

Sample Output 0

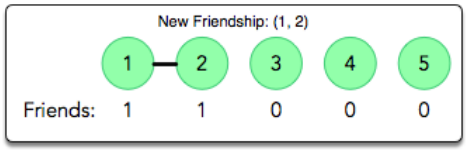
```
32
```

Explanation 0



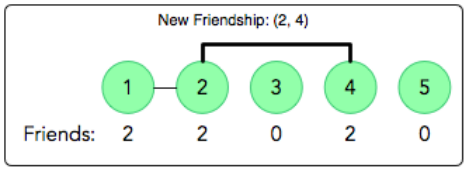
The value of $total$ is maximal if the students form the $m = 4$ direct friendships in the following order:

1. Students 1 and 2 become friends:



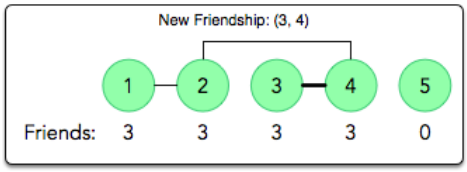
   We then sum the number of friends that each student has to get $1 + 1 + 0 + 0 + 0 = 2$.

2. Students 2 and 4 become friends:



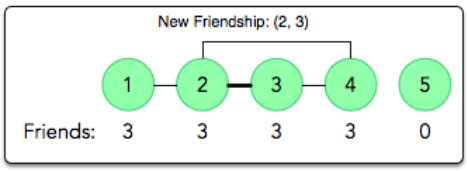
   We then sum the number of friends that each student has to get $2 + 2 + 0 + 2 + 0 = 6$.

3. Students 3 and 4 become friends:



   We then sum the number of friends that each student has to get $3 + 3 + 3 + 3 + 0 = 12$.

4. Students 3 and 2 become friends:



   We then sum the number of friends that each student has to get $3 + 3 + 3 + 3 + 0 = 12$.

When we add the sums from each step, we get $total = 2 + 6 + 12 + 12 = 32$. We then print 32 on a new line.

Coprime Paths

You are given an undirected, connected graph, G, with n nodes and m edges where $m = n - 1$. Each node i is initially assigned a value, $node_i$, that has *at most* 3 prime divisors.

You must answer q queries in the form `u v`. For each query, find and print the *number of* (x, y) *pairs* of nodes on the path between u and v such that $gcd(node_x, node_y) = 1$ and the length of the path between u and v is minimal among all paths from u to v.

Input Format

The first line contains two space-separated integers describing the respective values of n and q.
The second line contains n space-separated integers describing the respective values of $node_1, node_2, \ldots, node_n$.
Each of the $n - 1$ subsequent lines contains two space-separated integers, u and v, describing an edge between nodes u and v.
Each of the q subsequent lines contains two space-separated integers, u and v, describing a query.

Constraints

- $1 \leq n, q \leq 25 \times 10^3$
- $1 \leq node_i \leq 10^7$
- $1 \leq u, v \leq n$

Output Format

For each query, print an integer on a new line denoting the *number of* (x, y) *pairs* of nodes on the path between u and v such that $gcd(node_x, node_y) = 1$ and the length of the path between u and v is minimal among all paths from u to v.

Sample Input 0

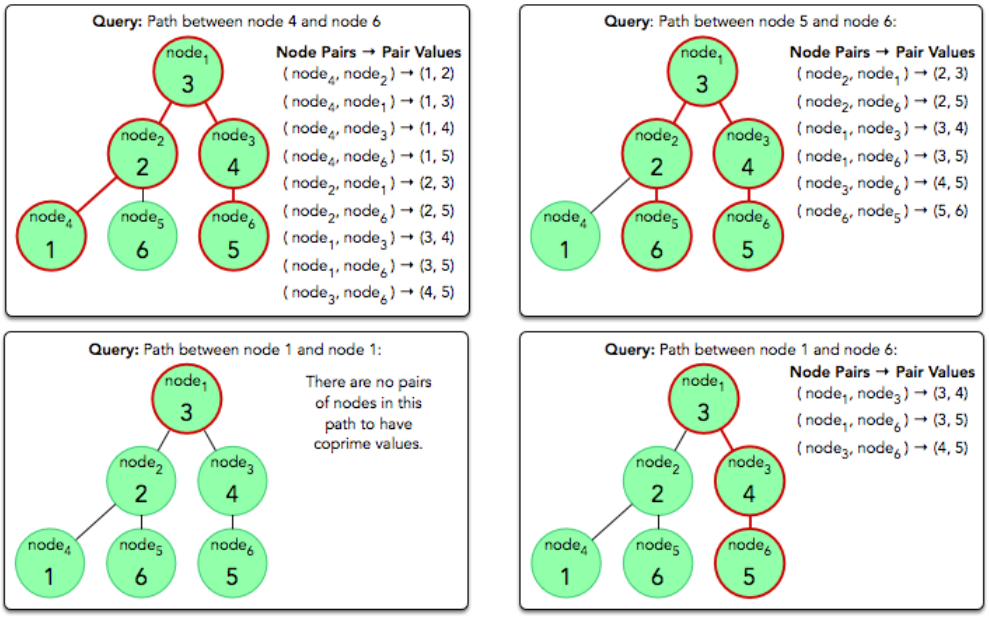
```
6 5
3 2 4 1 6 5
1 2
1 3
2 4
2 5
3 6
4 6
5 6
1 1
1 6
6 1
```

Sample Output 0

```
9
6
0
3
3
```

Explanation 0

The diagram below depicts graph G and the $u \leftrightarrow v$ paths specified by each query, as well as the *Pair Values* for each path in the form $(node_x, node_y)$:



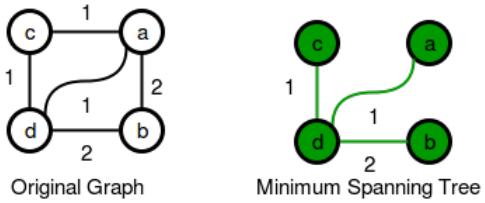
Recall that, for each queried path, we want to find and print the number of (x, y) pairs of nodes such that $gcd(node_x, node_y) = 1$.

Minimum MST Graph

Allison loves graph theory and just started learning about Minimum Spanning Trees(MST). She has three integers, n, m, and s, and uses them to construct a graph with the following properties:

- The graph has n nodes and m undirected edges where *each edge has a positive integer length*.
- No edge may directly connect a node to itself, and each pair of nodes can only be directly connected by *at most* one edge.
- The graph is *connected*, meaning each node is reachable from any other node.
- The *value* of the minimum spanning tree is s. Value of the MST is the sum of all the lengths of all edges of which are part of the tree.
- The sum of the lengths of all edges is as small as possible.

For example, let's say $n = 4$, $m = 5$ and $s = 4$. We need to construct a graph with 4 nodes and 5 edges. The value of minimum spanning tree must be 4. The diagram belows shows a way to construct such a graph while keeping the lengths of all edges is as small as possible:



Here the sum of lengths of all edges is 7.

Given n, m, and s for g graphs satisfying the conditions above, find and print the minimum sum of the lengths of all the edges in each graph on a new line.

Note: It is guaranteed that, for all given combinations of n, m, and s, we can construct a valid graph.

Input Format

The first line contains an integer, g, denoting the number of graphs.
Each of the g subsequent lines contains three space-separated integers describing the respective values of n (the number of nodes in the graph), m (the number of edges in the graph), and s (the value of the MST graph).

Constraints

For 20% of the maximum score:

- $1 \leq g \leq 100$
- $2 \leq n \leq 10$
- $1 \leq m \leq 50$
- $1 \leq s \leq 20$

For 50% of the maximum score:

- $1 \leq g \leq 100$
- $2 \leq n \leq 50$
- $1 \leq m \leq 2000$
- $1 \leq s \leq 200$

For 70% of the maximum score:

- $1 \leq g \leq 100$
- $2 \leq n \leq 10^5$
- $1 \leq m \leq 10^{10}$
- $1 \leq s \leq 10^6$

For 100% of the maximum score:

- $1 \leq g \leq 1000$
- $2 \leq n \leq 10^8$
- $1 \leq m \leq 10^{16}$
- $1 \leq s \leq 10^{10}$

Output Format

For each graph, print an integer on a new line denoting the minimum sum of the lengths of all edges in a graph satisfying the given conditions.

Sample Input

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

Sample Output

```
7
6
```

Explanation

- Graph 1:

The answer for this sample is already explained the problem statement.

- Graph 2:

We must construct a graph with $n = 4$ nodes, $m = 3$ edges, and an MST value of $s = 6$. Recall that a connected graph with n nodes and $n - 1$ edges is already a tree, so the MST will contain all $m = 3$ edges and the total length of all the edges of the graph will be equal to the value of the minimum spanning tree. So the answer is 6.

Jack goes to Rapture

Jack has just moved to a new city called Rapture. However, he is confused by Rapture's public transport system. The rules of the public transport are as follows:

1. Every pair of connected stations has a fare assigned to it.
2. If a passenger travels from station A to station B, he only has to pay the difference between the fare from A to B and the cumulative fare that he has paid to reach station A [*fare(A,B) - total fare to reach station A*]. If the difference is negative, he can travel free of cost from A to B.

Since Jack is new to the city, he is unemployed and low on cash. He needs your help to figure out the most cost efficient way to go from the first station to the last station. You are given the number of stations N (numbered from 1 to N), and the fare between the E pair of stations that are connected.

Input Format

The first line contains two integers, N and E, followed by E lines containing three integers each: the two stations that are connected to each other and the fare between them (C).

Constraints

- $1 \leq N \leq 50000$
- $1 \leq E \leq 500000$
- $1 \leq C \leq 10^7$

Output Format

The minimum fare to be paid to reach station N from station 1. If the station N cannot be reached from station 1, print "NO PATH EXISTS" (without quotes).

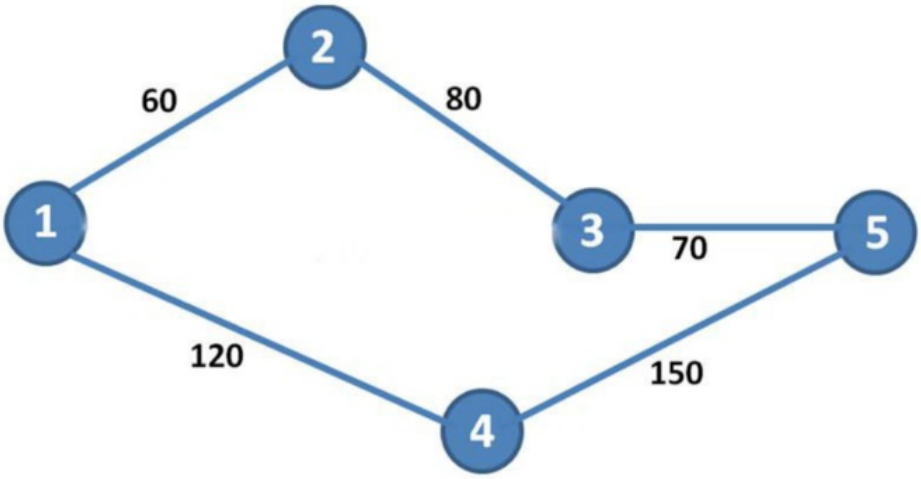
Sample Input 0

```
5 5
1 2 60
3 5 70
1 4 120
4 5 150
2 3 80
```

Sample Output 0

```
80
```

Explanation 0



There are two ways to go from first station to last station.

- 1 -> 2 -> 3-> 5
- 1 -> 4 -> 5

For the first path, Jack first pays 60 units of fare to go from station 1 to 2. Next, Jack has to pay 80-60 = 20 units to go from 2 to 3. Now, to go from 3 to 5, Jack has to pay 70-(60+20) = -10 units, but since this is a negative value, Jack pays 0 units to go from 3 to 5. Thus the total cost of this path is (60+20) = 80 units.

For the second path, Jack pays 120 units to reach station 4 from station 1. To go from station 4 to 5, Jack has to pay 150-120 = 30 units. Thus the total cost becomes (120+30) = 150 units. So, the first path is the most cost efficient, with a cost of 80.

Crab Graphs

A crab is an undirected graph which has two kinds of vertices: 1 head, and K feet , and exactly K edges which join the head to each of the feet.(1 <= K <= T, where T is given)

Given an undirected graph, you have to find in it some vertex-disjoint subgraphs where each one is a crab . The goal is to select those crabs in such a way that the total number of vertices covered by them is maximized.

Note: two graphs are vertex-disjoint if they do not have any vertices in common.

Input Format

The first line of input contains a single integer C. C test-cases follow. The first line of each test-case contains three integers N, T, and M (the number of nodes, max number of feet in the crab graph, and number of edges, respectively). Each of next M lines contains two space separated values v1i, v2i meaning that the there is an edge between vertices v1i and v2i. Note that the graph doesn't have parallel edges or loops.

Constraints

- 1 <= C <= 10
- 2 <= T <= 100
- 2 <= N <= 100
- 0 <= M <= N * (N-1)/2
- 1 <= v1i <= N
- 1 <= v2i <= N

Output Format

For each test-case, output a single integer indicating the maximum number of vertices which can be covered by vertex-disjoint sub-graphs of crab- graphs.

Sample Input

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

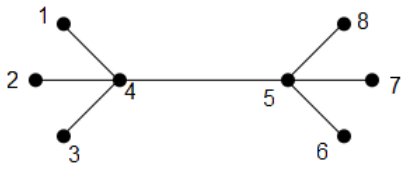
Sample Output

```
6
6
```

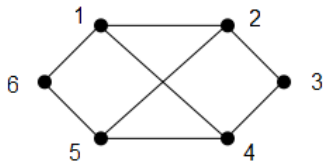
Explanation

Test #1: The graph for this test-case below. Because T = 2, each crab can have a maximum of 2 feet => each crab can cover a maximum of 3 nodes. We can cover 6 nodes of this graph with these two crabs: One of the crabs has 4 as its head and 1 and 3 as its feet, the other crab has 5 as its head and 7 and 8 as its feet. No additional crabs can be added.

The above is not a unique solution: any combination of two crabs, with one head at 4 and one head at 5, will suffice. We could have also chosen Head[4]feet[1,2] and Head[5]feet[6,7] as our two crabs.



Test #2: The graph for this test-case below. We can cover all 6 nodes using two crabs. One of the crabs has 2 as its head and 1 and 3 as its feet, the other crab has 5 as its head and 4 and 6 as its feet.



Bead Ornaments

There are N colors of beads. You have b_i beads of the i^{th} color. You want to make an ornament by joining all the beads together. You create the ornament by using the following algorithm:

- Step $\#1$ Arrange all the beads in any order such that beads of the same color are placed together.
- Step $\#2$ The ornament initially consists of only the first bead from the arrangement.
- Step $\#3$ For each subsequent bead in order, join it to a bead of the same color in the ornament. If there is no bead of the same color, it can be joined to any bead in the ornament.

All beads are distinct, even if they have the same color. How many different ornaments can be formed by following the above algorithm? Two ornaments are considered different if two beads are joined by a thread in one configuration, but not in the other.

Update/clarification

Think of the bead formation as a tree and not as a straight line. Any number of beads can be connected to a bead.

Input Format

The first line contains the number of test cases T. T test cases follow. Each test case contains N on the first line - the number of colors of beads. The next line contains N integers, where the i^{th} integer b_i denotes the number of beads of the i^{th} color.

Constraints

- $1 <= T <= 20$
- $1 <= N <= 10$
- $1 <= b_i <= 30$

Output Format

Output T lines, one for each test case. All answers should be output modulo 1000000007.

Sample Input

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

Sample Output

```
2
4
16
9
125
```

Explanation

Testcase 1:

Let us label the beads A1,A2 and B1. Initially, they can be arranged in 4 ways - "A1,A2,B1", "A2,A1,B1", "B1,A1,A2", and "B1,A2,A1".

For each of the first two arrangements, an ornament can be formed in 2 ways (A1-A2-B1 or B1-A1-A2 from the first one and A2-A1-B1 or B1-A2-A1 from the second one).

For each of the last two arrangements, an ornament can be formed in 1 way.

However, of the total 6 possible ornaments, there are only 2 unique ones : A1 - A2 - B1, and A2 - A1 - B1.

Testcase 2:

The possible unique ornaments are A1 - A2 - B1 - B2, A1 - A2 - B2 - B1, A2 - A1 - B1 - B2, and A2 - A1 - B2 - B1.

Testcase 3:

For the third test-case, it might be easier to see there are only 2 types of graphs on 4 vertices: the path or the star. It's not hard to see that there are 12 paths and 4 stars (explanation courtesy: zlangley)

Testcase 5:

For the fifth test-case, a lot of people claimed that the total number of possible ways is $5!/2 = 60$. But that is wrong. The correct answer is 125. Here's the hint: Once again, you've to think of it as a tree.

So one possible arrangement can be:

A is a root node and has two edges (A-B and A-C). Now, think of B as a sub-root node with two edges (B-D and B-E). Similarly, you can figure out the other possible bead arrangements. This will lead you to the correct answer.

Jeanie's Route

Byteland has N cities (numbered from 1 to N) and $N-1$ bidirectional roads. It is guaranteed that there is a route from any city to any other city.

Jeanie is a postal worker who must deliver K letters to various cities in Byteland. She can start and end her delivery route in any city. Given the destination cities for K letters and the definition of each road in Byteland, find and print the minimum distance Jeanie must travel to deliver all K letters.

Note: The letters can be delivered in any order.

Input Format

The first line contains two space-separated integers, N (the number of cities) and K (the number of letters), respectively.
The second line contains K space-separated integers describing the delivery city for each letter.
Each line i of the $N-1$ subsequent lines contains 3 space-separated integers describing a road as $u_i\ v_i\ d_i$, where d_i is the distance (length) of the bidirectional road between cities u_i and v_i.

Constraints

- $2 \leq K \leq N \leq 10^5$
- $1 \leq d_i \leq 10^3$
- $Byteland\ is\ a\ weighted\ undirected\ acyclic\ graph.$

Output Format

Print the minimum distance Jeanie must travel to deliver all K letters.

Sample Input 0

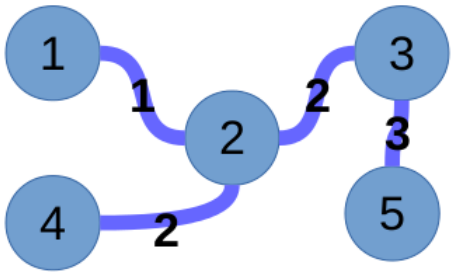
```
5 3
1 3 4
1 2 1
2 3 2
2 4 2
3 5 3
```

Sample Output 0

```
6
```

Explanation 0

Jeanie has 3 letters she must deliver to cities 1, 3, and 4 in the following map of Byteland:



One of Jeanie's optimal routes is $\underbrace{3 \to 2}_{2} \overbrace{\to 1}^{1} \underbrace{\to 2}_{1} \overbrace{\to 4}^{2}$, for a total distanced traveled of $2+1+1+2=6$.

Thus, we print 6 on a new line.

Floyd : City of Blinding Lights

Given a directed, weighted graph, consisting of N nodes and there are edges ,of specified length between some of them in the graph.

Given Q questions, inquring the shortest distance between a queried pair of nodes in the graph.

Answer all these questions as quickly as possible !

Input Format

First line has two integers N, denoting the number of nodes in the graph and M, denoting the number of edges in the graph.
The next M lines each consist of three space separated integers x y r, where x and y denote the two nodes between which the *directed* edge $(x->y)$ exists, r denotes the length of the edge between the corresponding edges.
The next line contains a single integer Q, denoting number of queries.
The next Q lines each, contain two space separated integers a and b, denoting the node numbers specified according to the question.

Constraints
$2 \leq N \leq 400$
$1 \leq M \leq \frac{N \times (N-1)}{2}$
$1 \leq Q \leq 10^5$
$1 \leq x, y, \leq N$
$1 \leq r \leq 350$

If there are edges between the same pair of nodes with different weights, the last one (most recent) is to be considered as the only edge between them.

Output Format

Print Q lines, each containing a single integer, specifying the shortest distance between the nodes specified for that query in the input.

If the distance between a pair of nodes is infinite (not reachable), then print -1 as the shortest distance.

Sample Input

```
4 5
1 2 5
1 4 24
2 4 6
3 4 4
3 2 7
3
1 2
3 1
1 4
```

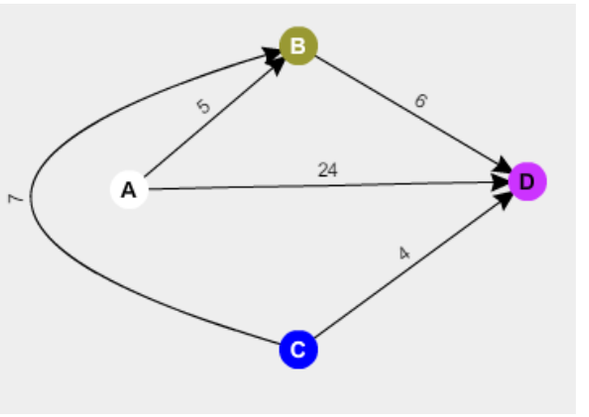
Sample Output

```
5
```

```
-1
11
```

Explanation

The graph given in the test case is shown as :



- The nodes A,B,C and D denote the 1,2,3 and 4 node numbers.

The shortest paths for the 3 queries are :

- **A->B** (Direct Path is shortest with weight 5)
- **-1** (There is no way of reaching node 1 from node 3, hence unreachable)
- **A->B->D** (Indirect path is shortest with weight (5+6) = 11 units, the direct path is longer with 24 units length)

Roads in HackerLand

John lives in HackerLand, a country with N cities and M bidirectional roads. Each of the roads has a distinct length, and each length is a *power of two* (i.e., 2 raised to some exponent). It's possible for John to reach any city from any other city.

Given a map of HackerLand, can you help John determine the sum of the minimum distances between each pair of cities? Print your answer in binary representation.

Input Format

The first line contains two space-seperated integers denoting N (the number of cities) and M (the number of roads), respectively.
Each line i of the M subsequent lines contains the respective values of A_i, B_i, and C_i as three space-separated integers. These values define a bidirectional road between cities A_i and B_i having length 2^{C_i}.

Constraints

- $1 \leq N \leq 10^5$
- $1 \leq M \leq 2 \times 10^5$
- $1 \leq A_i, B_i \leq N$, $A_i \neq B_i$
- $0 \leq C_i < M$
- If $i \neq j$, then $C_i \neq C_j$.

Output Format

Find the sum of minimum distances of each pair of cities and print the answer in binary representation.

Sample Input

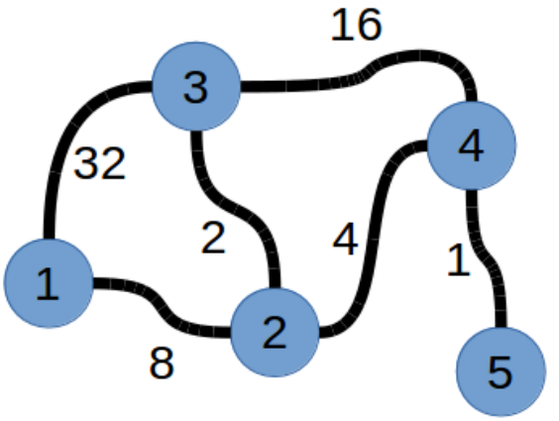
```
5 6
1 3 5
4 5 0
2 1 3
3 2 1
4 3 4
4 2 2
```

Sample Output

```
1000100
```

Explanation

In the sample, the country looks like this:



Let $d(x, y)$ be the minimum distance between city x and city y.

$d(1,2) = 8$
$d(1,3) = 10$
$d(1,4) = 12$
$d(1,5) = 13$
$d(2,3) = 2$
$d(2,4) = 4$
$d(2,5) = 5$
$d(3,4) = 6$
$d(3,5) = 7$
$d(4,5) = 1$

$$Sum = 8 + 10 + 12 + 13 + 2 + 4 + 5 + 6 + 7 + 1 = (68)_{10} = (1000100)_2$$

Kingdom Connectivity

It has been a prosperous year for King Charles and he is rapidly expanding his empire. In fact, he recently invaded his neighboring country and set up a new kingdom! This kingdom has many cities connected by **one-way roads.** To ensure higher connectivity, two cities are sometimes directly linked by more than one road also.

In the new kingdom, King Charles has made one of the cities his financial capital and another city his warfare capital. He wants a better connectivity between these two capitals. The connectivity of a pair of cities, A and B, is defined as the number of different paths from city A to city B. A path may use a road more than once if possible. Two paths are considered different if they do not use the same sequence of roads.

There are N cities numbered 1 to N in the new kingdom and M **one-way roads**. City 1 is the financial capital and city N is the warfare capital.

What is the connectivity of the financial capital and the warfare capital, i.e., how many different paths are there from city 1 to city N?

Input Format
The first line contains two integers N and M.

M lines follow, each containing two integers x and y, indicating there is a road from city x to city y $(1 \leq x, y \leq N)$.

Constraints

- $2 \leq N \leq 10^4$
- $1 \leq M \leq 10^5$

Output Format
Print the number of different paths from city 1 to city N modulo 10^9. If there are infinitely many different paths, print `INFINITE PATHS`.

Two roads may connect the same cities, but they are still considered distinct for path connections.

Sample Input 0

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

Sample Output 0

```
2
```

Sample Input 1

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

Sample Output 1

```
INFINITE PATHS
```

Computer Game

Sophia is playing a game on the computer. There are two random arrays A & B, each having the same number of elements. The game begins with Sophia removing a pair (A_i, B_j) from the array if they are not co-prime. She keeps a count on the number of times this operation is done.

Sophia wants to find out the maximal number of times(S) she can do this on the arrays. Could you help Sophia find the value?

Input Format

The first line contains an integer n. 2 lines follow, each line containing n numbers separated by a single space. The format is shown below.

```
n
A[0] A[1] ... A[n - 1]
B[0] B[1] ... B[n - 1]
```

Constraints

$0 < n <= 10^5$
$2 <= A[i], B[i] <= 10^9$
Each element in both arrays are generated randomly between 2 and 10^9

Output Format

Output S which is the maximum number of times the above operation can be made.

Sample Input

```
4
2 5 6 7
4 9 10 12
```

Sample Output

```
3
```

Explanation

You can remove:

```
(2, 4)
(5, 10)
(6, 9)
```

hence 3.

Rust & Murderer

Detective Rust is investigating a homicide and he wants to chase down the murderer. The murderer knows he would definitely get caught if he takes the main roads for fleeing, so he uses the village roads (or side lanes) for running away from the crime scene.

Rust knows that the murderer will take village roads and he wants to chase him down. He is observing the city map, but it doesn't show the village roads (or side lanes) on it and shows only the main roads.

The map of the city is a graph consisting N nodes (labeled 1 to N) where a specific given node S represents the current position of Rust and the rest of the nodes denote other places in the city, and an edge between two nodes is a main road between two places in the city. It can be suitably assumed that *an edge that doesn't exist/isn't shown on the map is a village road (side lane)* . That means, there is a village road between two nodes a and b iff(if and only if) there is no city road between them.

In this problem, distance is calculated as number of village roads (side lanes) between any two places in the city.

Rust wants to calculate the shortest distance from his position (Node S) to all the other places in the city if he travels only using the village roads (side lanes).

Note: The graph/map of the city is ensured to be a sparse graph.

Input Format

The first line contains T, denoting the number of test cases. T testcases follow.
First line of each test case has two integers N, denoting the number of cities in the map and M, denoting the number of roads in the map.
The next M lines each consist of two space-separated integers x and y denoting a main road between city x and city y. The last line has an integer S, denoting the current position of Rust.

Constraints

- $1 \leq T \leq 10$
- $2 \leq N \leq 2 \times 10^5$
- $0 \leq M \leq 120000$
- $1 \leq x, y, S \leq N$

Note

- *No nodes will have a road to itself.*
- *There will not be multiple edges between any pair of nodes i.e. there is at most one undirected edge between them.*
- *Graph is guaranteed to be sparse.*
- *It is guranteed that there will be a path between any pair of nodes using the side lanes.*

Output Format

For each of T test cases, print a single line consisting of N-1 space separated integers, denoting the shortest distances of the remaining N-1 places from Rust's position (that is all distances, except the source node to itself) using the village roads/side lanes in ascending order based on vertex number.

Sample Input 0

```
2
```

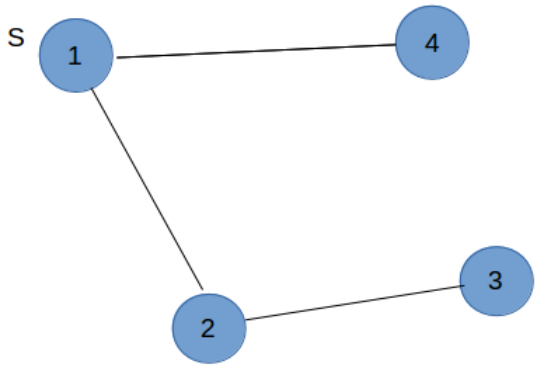
```
4 3
1 2
2 3
1 4
1
4 2
1 2
2 3
2
```

Sample Output 0

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

Explanation 0

The graph in the first testcase can be shown as:



Here the source node is 1 (marked S).
The distance from 1 to 2 is 3. Path: 1 -> 3 -> 4 -> 2
The distance from 1 to 3 is 1. Path: 1 -> 3
The distance from 1 to 4 is 2. Path: 1 -> 3 -> 4

Problem solving

There are N problems numbered 1..N which you need to complete. You've arranged the problems in increasing difficulty order, and the i^{th} problem has estimated difficulty level *i*. You have also assigned a rating *vi* to each problem. Problems with similar *vi* values are similar in nature. On each day, you will choose a subset of the problems and solve them. You've decided that each subsequent problem solved on the day should be tougher than the previous problem you solved on that day. Also, to make it less boring, consecutive problems you solve should differ in their *vi* rating by at least K. What is the least number of days in which you can solve all problems?

Input Format

The first line contains the number of test cases T. T test cases follow. Each case contains an integer N and K on the first line, followed by integers v1,...,vn on the second line.

Constraints

1 <= T <= 100
1 <= N <= 300
1 <= vi <= 1000
1 <= K <= 1000

Output Format

Output T lines, one for each test case, containing the minimum number of days in which all problems can be solved.

Sample Input

```
2
3 2
5 4 7
5 1
5 3 4 5 6
```

Sample Output

```
2
1
```

Explanation

For the first example, you can solve the problems with rating 5 and 7 on the first day and the problem with rating 4 on the next day. Note that the problems with rating 5 and 4 cannot be completed consecutively because the ratings should differ by at least K (which is 2). Also, the problems cannot be completed in order 5,7,4 in one day because the problems solved on a day should be in increasing difficulty level.

For the second example, all problems can be solved on the same day.

Journey Scheduling

Fedya is a seasoned traveller and is planning his trip to Treeland. Treeland is a country with an ancient road system which is in the form of a tree structure. N cities of Treeland are numbered by N positive integers: $1, 2, 3, \ldots, N$.

Fedya has not yet decided the starting point (city) of his journey and the cities he will visit. But there are a few things you know about Fedya's trip:

- Fedya is fond of travelling to great distances. So if he is currently located in city V, his destination will be a city which is most distant from city V.
- There might be more than 1 such cities. In that case, Fedya will choose a city that was already visited as less times as possible in this journey.
- There still might be more than 1 such cities. In that case, Fedya will go to the city with the smallest number.

Fedya has prepared a list of M possible journeys. Each one is characterized by two integers - the starting city V and the total number of cities to be visited, K. For each of them, he is keen to know the total distance travelled by him.

Input Format

The first line of input will contain two space separated integers N and M - the number of cities and the number of possible journeys.

Then, there will be $(N-1)$ lines, each of them will contain two space separated integers $X\ Y$, denoting the bi-directional road between the cities with numbers X and Y with the unitary length.

Then there will be M lines, each of them will have two space separated integers V and K, denoting a journey.

Constraints

$1 \leq N, M \leq 10^5$
$1 \leq V, X, Y \leq N$
$1 \leq K \leq 10^9$

Output Format

For each journey, output the travelled distance on a separate line.

Sample Input

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

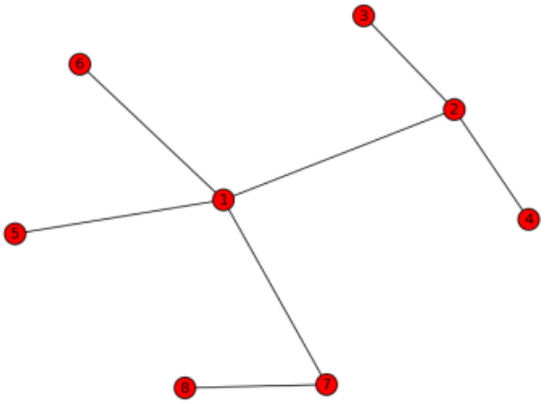
```
7 1
2 6
```

Sample Output

```
24
16
11
23
24
3
23
```

Explanation

The tree in question is given in the picture below.



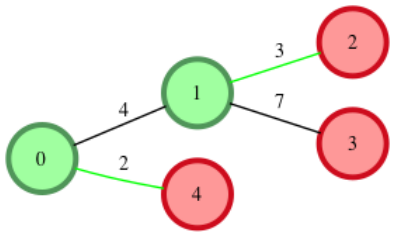
- `4 6` indicates that Fedya starts at 4. Now we see that the most distant city from 4 is 8. Fedya now travels to city 8. From 8, the most distance cities are [4, 3]. As 4 is already visited, he chooses to visit city 3. From city 3, he revisits city 8 and so on. The cities in the order of visit is 4 - > 8 -> 3 -> 8 -> 4 - > 8 -> 3 which sums to 24. Hence, the answer.
- `6 3` indicates that Fedya starts at city 6. From 6, the most distant cities are [3,4,8]. In this leg of the journey, no city is visited and hence Fedya chooses to visit the city with the smallest number 3. From 3, he visits 8 and then he ends his trip at city 4 which sums to 3 + 4 + 4 = 11. Hence, the answer.

Matrix

The kingdom of Zion has cities connected by bidirectional roads. There is a unique path between any pair of cities. Morpheus has found out that the machines are planning to destroy the whole kingdom. If two machines can join forces, the will attack. Neo has to destroy roads connecting cities with machines in order to stop them from joining forces. There must not be any path connecting two machines.

Each of the roads takes an amount of time to destroy, and only one can be worked on at a time. Given a list of edges and times, determine the minimum time to stop the attack.

For example, there are $n = 5$ cities called $0 - 4$. Three of them have machines and are colored red. The time to destroy is shown next to each road. If we cut the two green roads, there are no paths between any two machines. The time required is $3 + 2 = 5$.



Function Description

Complete the function *minTime* in the editor below. It must return an integer representing the minimum time to cut off access between the machines.

minTime has the following parameter(s):

- *roads*: a two-dimensional array of integers, each $roads[i] = [city1, city2, time]$ where cities are connected by a road that takes $time$ to destroy
- *machines*: an array of integers representing cities with machines

Input Format

The first line of the input contains two space-separated integers, n and k, the number of cities and the number of machines.

Each of the following $n - 1$ lines contains three space-separated integers, $city1$, $city2$, and $time$. There is a bidirectional road connecting $city1$ and $city2$, and to destroy this road it takes $time$ units.

Each of the last k lines contains an integer, $machine[i]$, the label of a city with a machine.

Constraints

- $2 \leq n \leq 10^5$
- $2 \leq k \leq n$
- $1 \leq time[i] \leq 10^6$

Output Format

Return an integer representing the minimum time required to disrupt the connections among all machines.

Sample Input

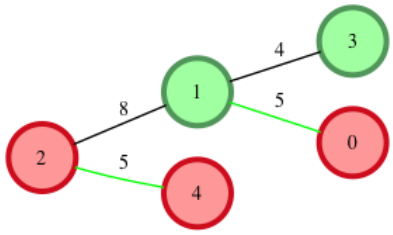
```
5 3
2 1 8
```

```
1 0 5
2 4 5
1 3 4
2
4
0
```

Sample Output

```
10
```

Explanation



The machines are located at the cities 0, 2 and 4. Neo can destroy the green roads resulting in a time of $5 + 5 = 10$. Destroying the road between cities 2 and 1 instead of between 1 and 0 would work, but it's not minimal.

Recording Episodes

Dave is a die-hard fan of a show called "HackerRank", in which a young programmer uses her problem-solving abilities to solve crimes. He splurged on a Digital Video Recorder (DVR) so that he can record HackerRank episodes and watch them later. Luckily, Dave managed to get his hands on schedules for all the episodes in each upcoming season.

Each season has n episodes numbered from 1 to n. Each episode airs twice; the first time it's called "live", and the second time it's called "repeat". So, for each episode, we have 4 integers, $(s_{live}$ and $e_{live})$ for the live airing and $(s_{repeat}$ and $e_{repeat})$ for the repeat airing, where s is episode's start time and and e is its end time. All times are given as integers representing the number of minutes passed since the start of the season.

Episodes broadcast on multiple channels, so some of the air times overlap and the episodes may not broadcast sequentially. It's possible that both the live and repeat broadcasts of some episode i are held before episode j, even though $i > j$. In addition, live and repeat broadcasts of the same episode may differ in length due to the number of advertisements during the broadcast.

Dave only has one TV with a DVR attached to it, and the DVR is capable of recording one episode at a time. For each episode in a season, Dave first decides whether or not he will record it. If he decides to record it, he will either record it during (s_{live}, e_{live}) or (s_{repeat}, e_{repeat}). Dave will only ever record one of the two airings of an episode, and he always records *full* episodes. This means that once he starts recording an episode, he will always record it until the end (i.e., he never records partial episodes).

Dave realizes that it might not be possible for him to record all episodes successfully, so instead of focusing on recording all episodes of HackerRank (which may be impossible), he decides to record all consecutively airing episodes whose episode number occurs in some inclusive $[L, R]$ interval such that $R - L$ (i.e., the number of consecutive episodes recorded) is as large as possible.

Given the programming schedule for each season, find L and R episode numbers for largest range of consecutive episodes Dave can record during that season and print these respective values as two space-separated integers on a new line. If two or more such intervals exist, choose the one having the smallest L value.

Input Format

The first line contains a single positive integer, q, denoting number of seasons of HackerRank.
The subsequent lines describe each of the q seasons in the following format:

1. The first line contains an integer, n, denoting the number of episodes in the season.
2. Each line i of the n subsequent line contains four space-separated integers describing the respective values of s_{live}, e_{live}, s_{repeat}, and e_{repeat}.

Constraints

- $1 \leq q \leq 100$
- $1 \leq n \leq 100$
- $1 \leq s_l, e_l, s_r, e_r \leq 10^4$
- $s_{live} \leq e_{live}$
- $s_{repeat} \leq e_{repeat}$
- $s_{live} \leq s_{repeat}$

Output Format

On a new line for each season, print two space-separated integers denoting the respective L and R (inclusive) values for the maximum possible range of consecutive episodes Dave can record such that $R - L$ is as large as possible. If more than one such interval exists, choose the interval having the smallest L.

Sample Input

```
3
3
10 20 30 40
20 35 21 35
14 30 35 50
1
10 20 30 40
3
11 19 31 39
12 38 13 37
10 20 30 40
```

Sample Output

```
1 2
1 1
1 1
```

Explanation

For the first season, Dave records the live airing of episode 1 and the repeat airing of episode 2. Note that it is *not* possible to record episodes 1, 2 and 3 simultaneously.

For the second season, there is only one episode so Dave records from episode 1 to episode 1 and we print `1 1` on a new line.

For the third season, Dave must choose to record either episode 1 or episode 3 (episode 2 starts while episode 1 is still airing and ends after episode 3 starts); he cannot record both, because he only wants to record consecutive episodes. Thus, we pick the episode with the smallest L value, which is episode 1, and print `1 1` as we are only recording episode 1.

Repair Roads

The country of Byteland contains N cities and $N-1$ bidirectional roads. There is a path between any two cities. The roads in Byteland were built long ago, and now they are in need of repair. You have been hired to fix all the roads. You intend to do this by dispatching robots on some of the roads. Each robot will repair the road he is currently on and then moves to one of the adjacent unrepaired roads. After repairing that, it will move to another adjacent unrepaired road, repair that and so on.

Two roads are adjacent if they have the same city at one of their endpoints. For the process to be efficient, no two robots will ever repair the same road, and no road can be visited twice. What is the minimum number of robots needed to accomplish the task?

Input Format

The first line contains the number of test cases T. T test cases follow. The first line of each test case contains N, the number of cities in Byteland. The cities are numbered $0..N-1$. The following $N-1$ lines contain the description of the roads. The i^{th} line contains two integers a_i and b_i, meaning that there is a road connecting cities with numbers a_i and b_i.

Constraints

- $1 \leq T \leq 20$
- $1 \leq N \leq 10000$
- $0 \leq a_i, b_i < N$

Output Format

Print T lines, one corresponding to each test case containing the required answer for that test case.

Sample Input

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

Sample Output

```
1
1
2
```

Explanation

For the first case, one robot is enough to repair all roads: $(0,1) \rightarrow (0,2) \rightarrow (0,3)$
For the second case, one robot is again enough: $(0,1) \rightarrow (1,2) \rightarrow (2,3) \rightarrow (2,4) \rightarrow (4,5)$
The the third case, there is no way to repair all the roads with one robot and at least two are needed.

Kth Ancestor

A tree of P nodes is an un-directed connected graph having $P-1$ edges. Let us denote R as the root node. If A is a node such that it is at a distance of L from R, and B is a node such that it is at at distance of $L+1$ from R and A is connected to B, then we call A as the parent of B.

Similarly, if A is at a distance of L from R and B is at a distance of $L+K$ from R and there is a path of length K from A to B, then we call A as the K^{th} parent of B.

Susan likes to play with graphs and Tree data structure is one of her favorites. She has designed a problem and wants to know if anyone can solve it. Sometimes she adds or removes a leaf node. Your task is to figure out the K^{th} parent of a node at any instant.

Input Format

The first line contain an integer T denoting the number of test cases. T test cases follow. First line of each test case contains an integer P, the number of nodes in the tree. P lines follows each containing two integers X and Y separated by a single space denoting Y as the parent of X. If Y is 0, then X is the root node of the tree. (0 is for namesake and is not in the tree).
The next line contains an integer Q, the number of queries.
Q lines follow each containing a query.

- $0\ Y\ X$: X is added as a new leaf node whose parent is Y . X is not in the tree while Y is in.
- $1\ X$: This tells that leaf node X is removed from the tree. X is a leaf in the tree.
- $2\ X\ K$: In this query output the K^{th} parent of X . X is a node in the tree.

Note

- Each node index is any number between 1 and 10^5 i.e., a tree with a single node can have its root indexed as 10^5

Constraints

$1 \leq T \leq 3$
$1 \leq P \leq 10^5$
$1 \leq Q \leq 10^5$
$1 \leq X \leq 10^5$
$0 \leq Y \leq 10^5$
$1 \leq K \leq 10^5$

Output Format

For each query of type 2, output the K^{th} parent of X. If K^{th} parent doesn't exist, output 0 and if the node doesn't exist, output 0.

Sample Input

```
2
7
2 0
5 2
3 5
7 5
9 8
8 2
6 8
10
0 5 15
2 15 2
1 3
```

```
0 15 20
0 20 13
2 13 4
2 13 3
2 6 10
2 11 1
2 9 1
1
10000 0
3
0 10000 4
1 4
2 4 1
```

Sample Output

```
2
2
5
0
0
8
0
```

Explanation

There are 2 test cases. The first test case has 7 nodes with 2 as its root. There are 10 queries

- 0 5 15 -> 15 is added as a leaf node to 5.
- 2 15 2 -> 2nd parent of 15 is 15->5->2 is 2.
- 1 3 -> leaf node 3 is removed from the tree.
- 0 15 20 -> 20 is added as a leaf node to 15.
- 0 20 13 -> 13 is added as a leaf node to 20.
- 2 13 4 -> 4th parent of 13 is 2.
- 2 13 3 -> 3rd parent of 13 is 5.
- 2 6 10 -> there is no 10th parent of 6 and hence 0.
- 2 11 1 -> 11 is not a node in the tree, hence 0.
- 2 9 1 -> 9's parent is 8.

the second testcase has a tree with only 1 node (10000).

- 0 10000 4 -> 4 is added as a leaf node to 10000.
- 1 4 -> 4 is removed.
- 2 4 1 -> as 4 is already removed, answer is 0.

ByteLandian Tours

The country of Byteland contains N cities and N - 1 bidirectional roads between them such that there is a path between any two cities. The cities are numbered (0,...,N - 1). The people were very unhappy about the time it took to commute, especially salesmen who had to go about every city selling goods. So it was decided that new roads would be built between any two "somewhat near" cities. Any two cities in Bytleland that can be reached by traveling on exactly two old roads are known as "somewhat near" each other.

Now a salesman situated in city 0, just like any other typical salesman, has to visit all cities exactly once and return back to city 0 in the end. In how many ways can he do this?

Input Format

The first line contains the number of test cases T. T test cases follow. The first line contains N, the number of cities in Byteland. The following N - 1 lines contain the description of the roads. The ith line contains two integers ai and bi, meaning that there was originally a road connecting cities with numbers ai and bi.

Constraints

1 <= T <= 20
1 <= N <= 10000
0 <= ai,bi < N

Output Format

Output T lines, one corresponding to each test case containing the required answer for that test case. Since the answers can be huge, output them modulo 1000000007.

Sample Input

```
2
3
0 1
1 2
5
0 1
1 2
2 3
2 4
```

Sample Output

```
2
4
```

Explanation

For the first case, a new road was build between cities 0 and 2. Now, the salesman has two tour possibilities: 0-1-2-0 or 0-2-1-0.

Find the Path

You are given a table, a, with n rows and m columns. The top-left corner of the table has coordinates $(0, 0)$, and the bottom-right corner has coordinates $(n-1, m-1)$. The i^{th} cell contains integer $a_{i,j}$.

A path in the table is a sequence of cells $(r_1, c_1), (r_2, c_2), \ldots, (r_k, c_k)$ such that for each $i \in \{1, \ldots, k-1\}$, cell (r_i, c_i) and cell (r_{i+1}, c_{i+1}) share a side.

The weight of the path $(r_1, c_1), (r_2, c_2), \ldots, (r_k, c_k)$ is defined by $\sum_{i=1}^{k} a_{r_i, c_i}$ where a_{r_i, c_i} is the weight of the cell (r_i, c_i).

You must answer q queries. In each query, you are given the coordinates of two cells, (r_1, c_1) and (r_2, c_2). You must find and print the minimum possible weight of a path connecting them.

Note: A cell can share sides with at most 4 other cells. A cell with coordinates (r, c) shares sides with $(r-1, c)$, $(r+1, c)$, $(r, c-1)$ and $(r, c+1)$.

Input Format

The first line contains 2 space-separated integers, n (the number of rows in a) and m (the number of columns in a), respectively.
Each of n subsequent lines contains m space-separated integers. The j^{th} integer in the i^{th} line denotes the value of $a_{i,j}$.
The next line contains a single integer, q, denoting the number of queries.
Each of the q subsequent lines describes a query in the form of 4 space-separated integers: r_1, c_1, r_2, and c_2, respectively.

Constraints

- $1 \leq n \leq 7$
- $1 \leq m \leq 5 \times 10^3$
- $0 \leq a_{i,j} \leq 3 \times 10^3$
- $1 \leq q \leq 3 \times 10^4$

For each query:

- $0 \leq r_1, r_2 < n$
- $0 \leq c_1, c_2 < m$

Output Format

On a new line for each query, print a single integer denoting the minimum possible weight of a path between (r_1, c_1) and (r_2, c_2).

Sample Input

```
3 5
0 0 0 0 0
1 9 9 9 1
0 0 0 0 0
3
0 0 2 4
```

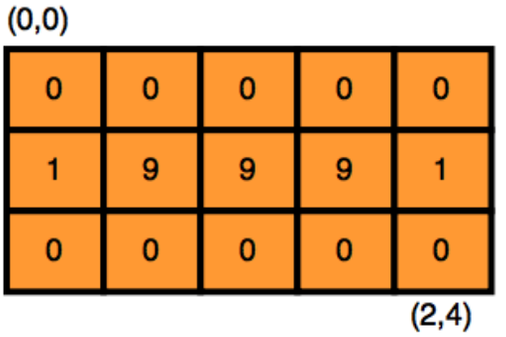
```
0 3 2 3
1 1 1 3
```

Sample Output

```
1
1
18
```

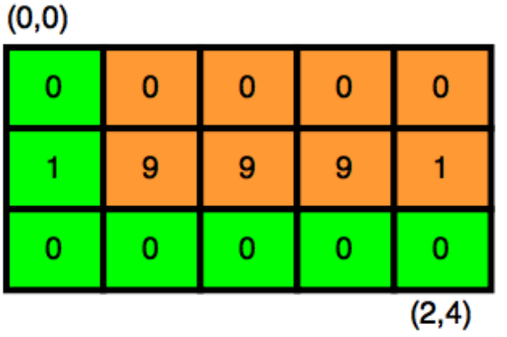
Explanation

The input table looks like this:



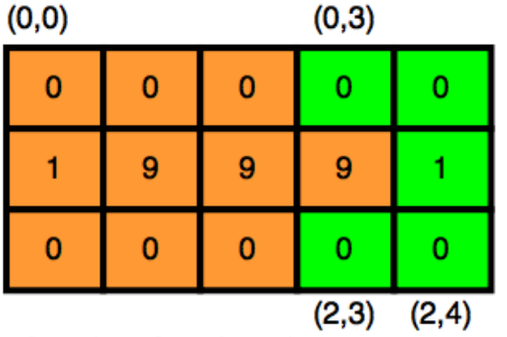
The first two queries are explained below:

1. In the first query, we have to find the minimum possible weight of a path connecting $(0, 0)$ and $(2, 4)$. Here is one possible path:



   The total weight of the path is $0 + 1 + 0 + 0 + 0 + 0 + 0 = 1$.

2. In the second query, we have to find the minimum possible weight of a path connecting $(0, 3)$ and $(2, 3)$. Here is one possible path:



   The total weight of the path is $0 + 0 + 1 + 0 + 0 = 1$.

Savita And Friends

After completing her final semester, Savita is back home. She is excited to meet all her friends. Her N friends live in different houses spread across the city.

There are M roads connecting the houses. The road network formed is connected and does not contain self loops and multiple roads between same pair of houses. Savita and Friends decide to meet.

Savita wants to choose a point(not necessarily an integer) P on the road numbered K, such that, the maximum of $dist(i)$ for all $1 \leq i \leq N$ is minimised,
where $dist(i)$ is the shortest distance between the i^{th} friend and P.

If K'th road connects friend A and friend B you should print distance of chosen point from A. Also, print the $max(dist(i))$ for all $1 \leq i \leq N$. If there is more than one solution, print the one in which the point P is closest to A.

Note:

- Use scanf/printf instead of cin/cout. Large input files.
- Order of A and B as given in the input must be maintained. If P is at a distance of 8 from A and 2 from B, you should print 8 and not 2.

Input Format

First line contain T, the number of testcases.
T testcases follow.
First Line of each testcase contains 3 space separated integers N, M, K .
Next M lines contain description of the i^{th} road : three space separated integers A, B, C, where C is the length of road connecting A and B.

Constraints

$1 \leq T \leq 10$
$2 \leq N, M \leq 10^5$
$N - 1 \leq M \leq N * (N - 1)/2$
$1 \leq A, B \leq N$
$1 \leq C \leq 10^9$
$1 \leq K \leq M$

Output Format

For each testcase, print two space separated values in one line. The first value is the distance of P from the point A and the second value is the maximum of all the possible shortest paths between P and all of Savita's and her friends' houses. Round both answers to 5 decimal digits and print exactly 5 digits after the decimal point.

Sample Input

```
2
2 1 1
1 2 10
4 4 1
1 2 10
2 3 10
3 4 1
4 1 5
```

Sample Output

```
5.00000 5.00000
2.00000 8.00000
```

Explanation

First testcase:
As K = 1, they will meet at the point P on the road that connects friend 1 with friend 2. If we choose mid point then distance for both of them will be 5. In any other position the maximum of distance will be more than 5.

Second testcase:
As K = 1, they will meet at a point P on the road connecting friend 1 and friend 2. If we choose point at a distance of 2 from friend 1: Friend 1 will have to travel distance 2.
Friend 2 will have to travel distance 8.
Friend 3 will have to travel distance 8.
Friend 4 will have to travel distance 7.
So, the maximum will be 8.
In any other position of point choosen, the maximum distance will be more than 8.

Timelimits

Timelimits for this problem is 2 times the environment limit.

Liars

You have **N** soldiers numbered from 1 to N. Each of your soldiers is either a liar or a truthful person. You have **M** sets of information about them. Each set of information tells you the number of liars among a certain range of your soldiers. Let **L** be the total number of your liar soldiers. Since you can't find the exact value of L, you want to find the minimum and maximum value of L.

Input Format

- The first line of the input contains two integers **N** and **M**.
- Each of next **M** lines contains three integers:
  A B C where the set of soldiers numbered as {A, A+1, A+2, ..., B}, exactly C of them are liars. (1 <= Ai <= Bi <= n) and (0 <= Ci <= Bi-Ai).

Note: **N** and **M** are not more than 101, and it is guaranteed the given informations is satisfiable.

Output Format
Print two integers Lmin and Lmax to the output.

Sample Input #1

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

Sample Output #1

```
1 2
```

Sample Input #2

```
20 11
3 8 4
1 9 6
1 13 9
5 11 5
4 19 12
8 13 5
4 8 4
7 9 2
10 13 3
7 16 7
14 19 4
```

Sample Output #2

```
13 14
```

Explanation
In the first input, the initial line is "3 2", i.e. that there are 3 soldiers and we have 2 sets of information. The next line says there is one liar in the set of soldiers {1, 2}. The final line says there is one liar in the set {2,3}. There are two possibilities for this scenario: Soldiers number 1 and 3 are liars or soldier number 2 is liar.
So the minimum number of liars is 1 and maximum number of liars is 2. Hence the answer, 1 2.

Jumping Rooks

Nina has an $n \times n$ chessboard and k jumping rooks. Every cell of the chessboard is either *blocked* or *free*, and Nina can only put a *single* rook in any *free* cell.

Two jumping rooks beat each other if they are either in the same row or in the same column *and* all cells between them are free (note that it's possible that there are some other rooks between them). More formally, if the first rook is in cell (x, y_1) and the second rook is in cell (x, y_2) (where $y_1 \leq y_2$), then these two rooks beat each other if and only if $(x, y_1), (x, y_1 + 1), \ldots, (x, y_2)$ are free. If the rooks are in cells (x_1, y) and (x_2, y), then cells $(x_1, y), (x_1 + 1, y), \ldots, (x_2, y)$ must all be free.

Given the configuration of the chessboard and some k, help Nina place k jumping rooks in the chessboard's free cells such that the number of pairs of rooks that beat each other is minimal. Then print a single integer denoting the number of rooks that beat each other.

Input Format

The first line contains two space-separated integers describing the respective values of n (the size of the chessboard) and k (the number of rooks to place).
Each line i of the n subsequent lines contains a string of n characters describing each row in the chessboard. The j^{th} character of the i^{th} line is `#` if cell (i, j) is blocked or `.` if the cell is free.

Constraints

- $1 \leq n \leq 50$
- It is guaranteed that k is less than the number of free cells in the chessboard.

Output Format

Print a single integer denoting the minimum possible number of pairs of rooks that beat each other.

Sample Input 0

```
3 4
...
...
...
```

Sample Output 0

```
2
```

Explanation 0
For this input, one possible arrangement is:

```
o.o
.o.
..o
```

where each `o` is a jumping rook.

Sample Input 1

```
5 10
..#..
..#..
#####
..#..
```

```
..#..
```

Sample Output 1

```
4
```

Explanation 1

For this input, one possible arrangement is:

```
.o#o.
oo#oo
#####
.o#o.
o.#.o
```

where each `o` is a jumping rook.

Tripartite Matching

You are given 3 unweighted, undirected graphs, G_1, G_2, and G_3, with n vertices each, where the k^{th} graph has m_k edges and the vertices in each graph are numbered from 1 through n. Find the number of ordered triples (a, b, c), where $1 \leq a, b, c \leq n$, $a \neq b, b \neq c, c \neq a$, such that there is an edge (a, b) in G_1, an edge (b, c) in G_2, and an edge (c, a) in G_3.

Input Format

The first line contains single integer, n, denoting the number of vertices in the graphs. The subsequent lines define G_1, G_2, and G_3. Each graph is defined as follows:

1. The first line contains an integer, m, describing the number of edges in the graph being defined.
2. Each line i of the m subsequent lines (where $1 \leq i \leq m$) contains 2 space-separated integers describing the respective nodes, u_i and v_i connected by edge i.

Constraints

- $n \leq 10^5$
- $m_k \leq 10^5$, and $k \in \{1, 2, 3\}$
- Each graph contains no cycles and any pair of directly connected nodes is connected by a maximum of 1 edge.

Output Format

Print a single integer denoting the number of distinct (a, b, c) triples as described in the *Problem Statement* above.

Sample Input

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

Sample Output

```
3
```

Explanation

There are three possible triples in our *Sample Input*:

1. $(1, 2, 3)$
2. $(2, 1, 3)$

3. $(3, 2, 1)$

Thus, we print 3 as our output.

Tree Flow

Recall that a tree is an undirected, connected acyclic graph. We have a weighted tree, T, with n vertices; let $dist_{u,v}$ be the total sum of edge weights on the path between nodes u and v.

Let's consider all the matrices, $A_{u,v}$, such that:

- $A_{u,v} = -A_{v,u}$
- $0 \leq |A_{u,v}| \leq dist_{u,v}$
- $\sum_{i=1}^{n} A_{u,i} = 0$ for each $u \neq 1$ and $u \neq n$

We consider the *total value* of matrix A to be:

$$\sum_{i=1}^{n} A_{1,i}$$

Calculate and print the maximum total value of A for a given tree, T.

Input Format

The first line contains a single positive integer, n, denoting the number of vertices in tree T.
Each line i of the $n-1$ subsequent lines contains three space-separated positive integers denoting the respective a_i, b_i, and c_i values defining an edge connecting nodes a_i and b_i (where $1 \leq a_i, b_i \leq n$) with edge weight c_i.

Constraints

- $2 \leq n \leq 500000$
- $1 \leq c_i \leq 10^4$
- Test cases with $n \leq 10$ have 30% of total score
- Test cases with $n \leq 500$ have 60% of total score

Output Format

Print a single integer denoting the maximum total value of matrix A satisfying the properties specified in the *Problem Statement* above.

Sample Input

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

Sample Output

```
3
```

Explanation

In the sample case, matrix A is:

$$A = \begin{pmatrix} 0 & 2 & 1 \\ -2 & 0 & 2 \\ -1 & -2 & 0 \end{pmatrix}$$

The sum of the elements of the first row is equal to 3.

DAG Queries

You are given a Directed Acyclic Graph (DAG) with n vertices and m edges. Each vertex v has an integer, a_v, associated with it and the initial value of a_v is 0 for all vertices. You must perform q queries on the DAG, where each query is one of the following types:

1. `1 u x`: Set a_v to x for all v such that there is a path in the DAG from u to v.
2. `2 u x`: Set a_v to x for all v such that there is a path from u to v and $a_v > x$.
3. `3 u`: Print the value of a_u on a new line.

Input Format

The first line contains three space-separated integers describing the respective values of n (the number of vertices in the DAG), m (the number of edges in the DAG), and q (the number of queries to perform).
Each of the m subsequent lines contains two space-separated integers describing the respective values of u and v (where $1 \leq u, v \leq n$, $u \neq v$) denoting a directed edge from vertex u to vertex v in the graph.
Each of the q subsequent lines contains a query in one of the three formats described above.

Constraints

- $2 \leq n \leq 10^5$
- $1 \leq m, q \leq 10^5$
- $0 \leq x \leq 10^9$
- $0 \leq a_v \leq 10^9$
- It's guaranteed that the graph is acyclic, but there may be more than one edge connecting two nodes.

Output Format

For each query of type 3 (i.e., `3 u`), print the value of a_u on a new line.

Sample Input 0

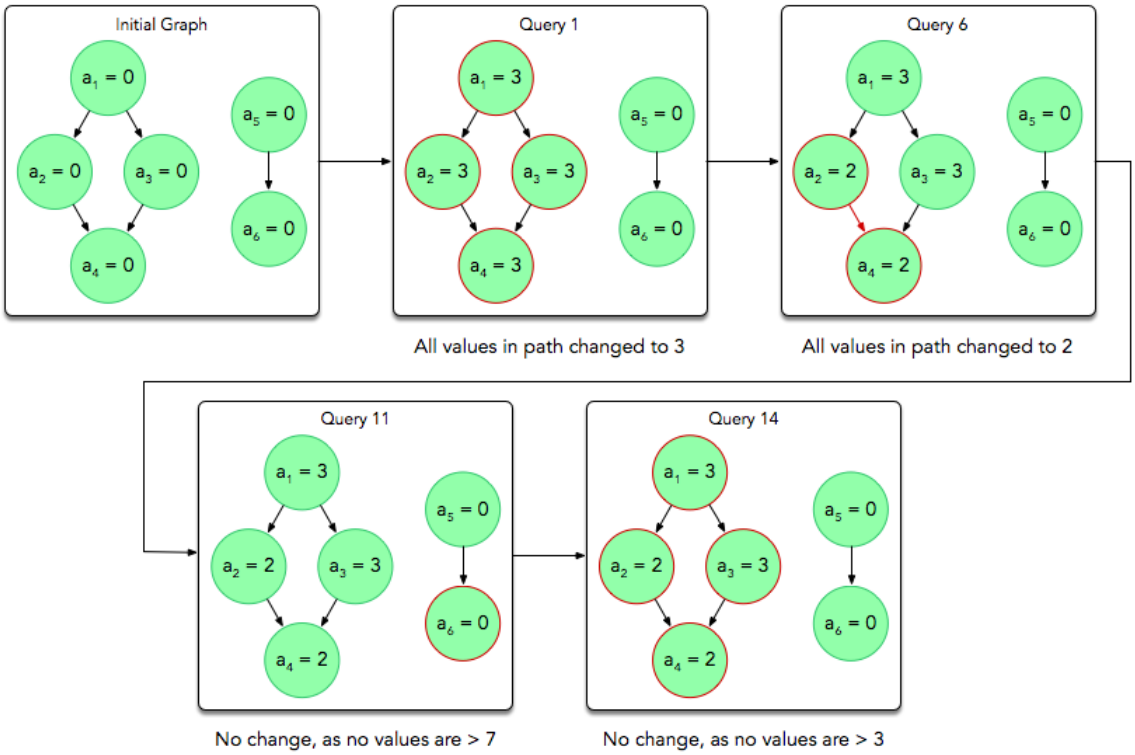
```
6 5 18
1 2
1 3
3 4
2 4
5 6
1 1 3
3 1
3 2
3 3
3 4
1 2 2
3 1
3 2
3 3
3 4
2 6 7
3 5
3 6
2 1 3
3 1
3 2
3 3
3 4
```

Sample Output 0

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

Explanation 0

The diagram below depicts the changes to the graph after all type 1 and type 2 queries:



Favorite sequence

Johnny, like every mathematician, has his favorite sequence of **distinct** natural numbers. Let's call this sequence M. Johnny was very bored, so he wrote down N copies of the sequence M in his big notebook. One day, when Johnny was out, his little sister Mary erased some numbers(possibly zero) from every copy of M and then threw the notebook out onto the street. You just found it. Can you reconstruct the sequence?

In the input there are N sequences of natural numbers representing the N copies of the sequence M after Mary's prank. In each of them all numbers are **distinct**. Your task is to construct the shortest sequence S that might have been the original M. If there are many such sequences, return the lexicographically smallest one. It is guaranteed that such a sequence exists.

Note
Sequence $A[1 \ldots n]$ is lexicographically less than sequence $B[1 \ldots n]$ if and only if there exists $1 \leq i \leq n$ such that for all $1 \leq j < i, A[j] = B[j] \ \ and \ \ A[i] < B[i]$.

Input Format

In the first line, there is one number N denoting the number of copies of M.
This is followed by K
and in next line a sequence of length K representing one of sequences after Mary's prank. All numbers are separated by a single space.

Constraints
$1 \leq N \leq 10^3$
$2 \leq K \leq 10^3$
All values in one sequence are **distinct** numbers in range $[1, 10^6]$.

Output Format

In one line, write the space-separated sequence S - the shortest sequence that might have been the original M. If there are many such sequences, return the lexicographically smallest one.

Sample Input

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

Sample Output

```
1 2 3 4
```

Explanation

You have 2 copies of the sequence with some missing numbers: $[1, 3]$ and $[2, 3, 4]$. There are two candidates for the original sequence M **:** $[1, 2, 3, 4] \ and \ [2, 1, 3, 4]$, where the first one is lexicographically least.

Jogging Cats

It's almost summertime, so Big Cat and Little Cat are getting in shape. They decide the core of their fitness plan is to start jogging every day.

Their city consists of N intersections connected by M bidirectional roads. The cats decide that their jogging route should be cyclic (i.e., starting and ending at the same intersection) and consist of 4 different roads.

The cats also love exploring new places, so each day they want to choose a new route to jog on that is not equal to any of their previous routes. Two routes are considered to be equal if their sets of component roads are equal.

Given a map of the city, can you help our heroic cats determine the maximum number of days they can go jogging so that every route traveled is different?

Input Format

The first line contains a pair of space-separated integers, N (the number of intersections) and M (the number of roads), respectively.

Each line i of the M subsequent lines contains a pair of space-separated integers, X_i and Y_i, defining a bidirectional road connecting intersections X_i and Y_i.

Constraints

- $1 \leq N \leq 5 \cdot 10^4$
- $1 \leq M \leq 10^5$
- $1 \leq X_i, Y_i \leq N$
- Each bidirectional road connects 2 distinct intersections (i.e., no road connects an intersection to itself).
- Each pair of intersections is directly connected by no more than 1 road.

Output Format

Print the maximum number of days for which the cats can go jogging without repeating a route.

Sample Input

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

Sample Output

```
3
```

Explanation

There are 3 different routes:

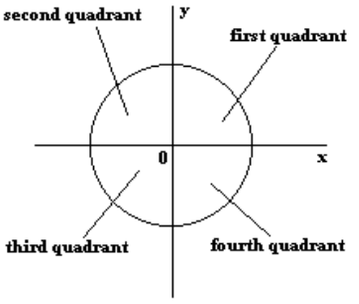
1. $1 \to 2 \to 3 \to 4 \to 1$
2. $1 \to 3 \to 2 \to 4 \to 1$
3. $1 \to 2 \to 4 \to 3 \to 1$

Recall that each route is a *set* of intersections forming a cycle, so each unique route is the same regardless of which city on the route the cats start out at. Thus, we print 3 (the number of routes) as our answer.

Quadrant Queries

There are n *2D* points on a plane, and the i^{th} point, p_i, has coordinates (x_i, y_i), where $1 \leq i \leq n$. There are three types of queries:

1. `X i j` Reflect all points in the inclusive range between points p_i and p_j along the x-axis.
2. `Y i j` Reflect all points in the inclusive range between points p_i and p_j along the y-axis.
3. `C i j` Count the number of points in the inclusive range between points p_i and p_j in each of the 4 quadrants. Then print a single line of four space-separated integers describing the respective numbers of points in the first, second, third, and fourth quadrants. Recall that the four quadrants of a graph are labeled as follows:



Given a set of n points (where each point p is indexed from 1 to n) and q queries, perform each query in order.

Input Format

The first line contains a single integer, n, denoting the number of points.
Each line i of the n subsequent lines contains two space-separated integers describing the respective x_i and y_i values for point p_i on the *2D* plane.
The next line contains a single integer, q, denoting the number of queries.
Each of the q subsequent lines contains three space-separated values describing a query in one of the three forms defined above. You must process each query in the same order as it's read from stdin.

Constraints

- $1 \leq n \leq 10^5$
- $1 \leq q \leq 10^6$
- It is guaranteed that no point lies on the x or y axes.
- All (x_i, y_i) points will fit in a 32-bit signed integer.
- In all queries, $1 \leq i \leq j \leq n$.

Output Format

For each query of type `C i j`, print four space-separated integers describing the number of points having indices in the inclusive range between points p_i and p_j in the respective first, second, third, and fourth graph quadrants.

Sample Input

```
4
1 1
-1 1
-1 -1
1 -1
5
C 1 4
X 2 4
C 3 4
Y 1 2
C 1 3
```

Sample Output

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

Explanation

Query `C 1 4` asks you to consider the set of points having indices in $\{1, 2, 3, 4\}$, meaning $p_1 = (1, 1)$, $p_2 = (-1, 1)$, $p_3 = (-1, -1)$, and $p_4 = (1, -1)$; amongst those points, how many of them lie in the respective first, second, third, and fourth quadrants? Because we have one point in each quadrant, we print `1 1 1 1` on a new line.

Recall that queries in the form `X i j` and `Y i j` are telling us to take all the points in the inclusive range between indices i and j (i.e., points p_i and p_j) and reflect them along the axis specified by the first character of the query. Note that i and j here refer to actual point numbers/subscripts and *not* coordinates on the plane.

So when we process query `X 2 4`, we reflect the points in the inclusive range between indices 2 and 4 (i.e., points p_2, p_3, and p_4) along the x-axis. This means our coordinates are now $p_1 = (1, 1)$, $p_2 = (-1, -1)$, $p_3 = (-1, 1)$, and $p_4 = (1, 1)$.

Next, `C 3 4` tells us to consider the set of points in the inclusive range between indices 3 and 4 (i.e., points p_3 and p_4) and print the number of points in this range falling on each respective axis. Point $p_3 = (-1, 1)$ lies in quadrant 2 and point $p_4 = (1, 1)$ lies in quadrant 1, so we print `1 1 0 0` on a new line.

Next, `Y 1 2` tells us to reflect the points in the inclusive range between indices 1 and 2 (i.e., points p_1 and p_2) along the y-axis. This means our coordinates are now $p_1 = (-1, 1)$, $p_2 = (1, -1)$, $p_3 = (-1, 1)$, and $p_4 = (1, 1)$.

Finally, `C 1 3` tells us to count the number of points in each quadrant that fall in the inclusive range between indices 1 and 3 (i.e., points p_1, p_2, and p_3). Point $p_1 = (-1, 1)$ is in quadrant 2, point $p_2 = (1, -1)$ is in quadrant 4, and point $p_3 = (-1, 1)$ is in quadrant 2. Thus, we print `0 2 0 1` on a new line.

Hacker Country

There are *N* cities in *Hacker Country*. Each pair of cities are directly connected by a unique directed road, and each road has its own toll that must be paid every time it is used. You're planning a road trip in *Hacker Country*, and its itinerary must satisfy the following conditions:

- You can start in any city.
- You must use 2 or more different roads (meaning you will visit 2 or more cities).
- At the end of your trip, you should be back in your city of origin.
- The average cost (sum of tolls paid per road traveled) should be minimum.

Can you calculate the *minimum average cost* of a trip in *Hacker Country*?

Time Limits
Time limits for this challenge are provided here.

Input Format

The first line is an integer, N (number of cities).
The N subsequent lines of N space-separated integers each describe the respective tolls or traveling from city i to city j; in other words, the j^{th} integer of the i^{th} line denotes the toll for traveling from city i to city j.

Note: As there are no roads connecting a city to itself, the i^{th} integer of line i will always be 0.

Constraints
$1 < N \leq 500$
$0 < toll\ cost \leq 200$
$roads\ traveled \geq 2$

Output Format

Print the *minimum cost* as a rational number $p \ / \ q$ (tolls paid over roads traveled). The *greatest common divisor* of p and q should be 1.

Sample Input

```
2
0 1
2 0
```

Sample Output

```
3/2
```

Explanation

The toll from city c_0 to city c_1 is 1. The toll from c_1 to c_0 is 2. Your travel cost $p = 1 + 2 = 3$. Your number of roads traveled is $q = 2$. Thus, we print `3/2` as our answer.

HackerX

Update: A slight modification in the problem statement (see below)

Evil Nation A is angry and plans to launch **N** guided-missiles at the peaceful Nation B in an attempt to wipe out all of Nation B's people. Nation A's missile *i* will arrive in nation B at time t_i. Missile *i* communicates with its headquarters by unique radio signals with a frequency equal to f_i. Can you help the peaceful Nation B survive by building a defensive system that will stop the missiles dead in the sky?

Defensive system:

The only way to defend Nation B from the attacking missile is by counter attacking them with a *hackerX* missile. You have a lot of *hackerX* missiles and each one of them has its own radio frequency. An individual *hackerX* missile can destroy Evil Nation A's attacking missile if the radio frequency of both of the missiles match. Each *hackerX* missile can be used an indefinite number of times. Its invincible and doesn't get destroyed in the collision.

The good news is you can adjust the frequency of the *hackerX* missile to match the evil missiles' frequency. When changing the *hackerX* missile's initial frequency fA to the new defending frequency fB, you will need \|fB - fA\| units of time to do.

~~Each *hackerX* missile can only destroy one of Nation A's missile at a time. So if two evil missiles with same frequency arrive at the same time, you need at least two *hackerX* missiles with the same frequency as the evil missiles to avoid damage.~~

If two evil missles with same frequency arrive at the same time, we can destroy them both with one *hackerX* missile. You can set the frequency of a *hackerX* missile to any value when its fired.

What is the minimum number of *hackerX* missiles you must launch to keep Nation B safe?

Input Format:
The first line contains a single integer **N** denoting the number of missiles.
This is followed by **N** lines each containing two integers t_i and f_i denoting the time & frequency of the i^{th} missile.

Output Format:
A single integer denoting the minimum number of *hackerX* missiles you need to defend the nation.

Constraints:
$1 <= N <= 100000$
$0 <= t_i <= 100000$
$0 <= f_i <= 100000$
$t_1 <= t_2 <= ... <= t_N$

Sample Input #00

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

Sample Output #00

```
1
```

Explanation #00

A *HackerX* missile is launched at t = 1 with a frequency f = 1, and destroys the first missile. It re-tunes its frequency to f = 2 in 1 unit of time, and destroys the missile that is going to hit Nation B at t = 2. It re-tunes its frequency back to 1 in 1 unit of time and destroys the missile that is going to hit the nation at t = 3. It is relaunched at t = 5 with f = 1 and destroys the missile that is going to hit nation B at t = 5. Hence, you need only 1 *HackerX* to protect nation B.

Sample Input #01

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

Sample Output #01

```
2
```

Explanation #01

Destroy 1 missile at t = 1, f = 1. now at t = 2, there is a missile with frequency 3. The launched missile takes 2 units of time to destroy this, hence we need a new hackerX missile to destroy this one. The first hackerX missile can destroy the 3rd missile which has the same frequency as itself. The same hackerX missile destroys the missile that is hitting its city at t = 5. Thus, we need atleast 2 hackerX missiles.

Huarongdao

Huarongdao is a well-known game in China. The purpose of this game is to move the Cao Cao block out of the board.

Acme is interested in this game, and he invents a similar game. There is a N*M board. Some blocks in this board are movable, while some are fixed. There is only one empty position. In one step, you can move a block to the empty position, and it will take you one second. The purpose of this game is to move the Cao Cao block to a given position. Acme wants to finish the game as fast as possible.

But he finds it hard, so he cheats sometimes. When he cheats, he spends K seconds to pick a block and put it in an empty position. However, he is not allowed to pick the Cao Cao block out of the board .

Note

1. Immovable blocks cannot be moved while cheating.
2. A block can be moved only in the directions UP, DOWN, LEFT or RIGHT.

Input Format

The first line contains four integers N, M, K, Q separated by a single space. N lines follow.
Each line contains M integers 0 or 1 separated by a single space. If the j_{th} integer is 1, then the block in i_{th} row and j_{th} column is movable. If the j_{th} integer is 0 then the block in i_{th} row and j_{th} column is fixed. Then Q lines follows, each line contains six integers EX_i, EY_i, SX_i, SY_i, TX_i, TY_i separated by a single space. The i_{th} query is the Cao Cao block is in row SX_i column SY_i, the exit is in TX_i, TY_i, and the empty position is in row EX_i column EY_i. It is guaranteed that the blocks in these positions are movable. Find the minimum seconds Acme needs to finish the game. If it is impossible to finish the game, you should answer -1.

Constraints

$N,M \leq 200$
$1 \leq Q \leq 250$
$10 \leq K \leq 15$
$1 \leq EX_i, SX_i, TX_i \leq N$
$1 \leq EY_i, SY_i, TY_i \leq M$

Output Format

You should output Q lines, i-th line contains an integer which is the answer to i-th query.

Sample Input

```
5 5 12 1
1 1 1 1 1
1 1 1 1 1
0 1 1 1 1
1 1 1 1 1
0 1 0 1 1
1 5 4 3 4 1
```

Sample Output

```
20
```

Explanation

Move the block in (1, 4) to (1, 5);
Move the block in (1, 3) to (1, 4);

Move the block in (1, 2) to (1, 3);
Move the block in (2, 2) to (1, 2);
Move the block in (3, 2) to (2, 2);
Move the block in (4, 2) to (3, 2);
Move the block in (4, 3) to (4, 2);
Move the block in (4, 1) to (4, 3) by cheating;
Move the block in (4, 2) to (4, 1).

So, 1 + 1 + 1 + 1 + 1 + 1 + 1 + 12 + 1 = 20.

Training the army

In the magical kingdom of Kasukabe, people strive to possess skillsets. Higher the number of skillset present among the people, the more content people will be.

There are N types of skill set present and initially there exists C_i people possessing i^{th} skill set, where $i \in [1, N]$.

There are T wizards in the kingdom and they have the ability to transform the skill set of a person into another skill set. Each of the these wizards has two **lists** of skill sets associated with them, A and B. He can only transform the skill set of person whose initial skill set belongs to the list A to one of the final skill set which belongs to the list B. That is, if $A = [2, 3, 6]$ and $B = [1, 2]$ then following transformation can be done by that trainer.

$$2 \rightarrow 1$$
$$2 \rightarrow 2$$
$$3 \rightarrow 1$$
$$3 \rightarrow 2$$
$$6 \rightarrow 1$$
$$6 \rightarrow 2$$

Once a transformation is done, both skill is removed from the respective lists. In the above example, if he perform $3 \rightarrow 1$ transformation on a person, list A will be updated to $[2, 6]$ and list B will be $[2]$. This updated list will be used for further transformations.

Few points to note are:

- One person can possess only one skill set.
- A wizard can perform zero or more transformation as long as they satisfies the above criteria.
- A person can go through multiple transformation of skill set.
- Same class transformation is also possible. That is a person' skill set can be transformed into his current skill set. Eg. $2 \rightarrow 2$ in the above example.

Your goal is to design a series of transformation which results into maximum number of skill set with non-zero number of people knowing it.

Input Format

The first line contains two numbers, $N\ T$, where N represent the number of skill set and T represent the number of wizards.
Next line contains N space separated integers, $C_1\ C_2\ \ldots\ C_N$, where C_i represents the number of people with i^{th} skill. Then follows $2 \times T$ lines, where each pair of line represent the configuration of each wizard.
First line of the pair will start with the length of list A and followed by list A in the same line. Similarly second line of the pair starts with the length of list B and then the list B.

Constraints

- $1 \leq N \leq 200$
- $0 \leq T \leq 30$
- $0 \leq C_i \leq 10$
- $0 \leq |A| \leq 50$
- $1 \leq A_i \leq N$

- $A_i \neq A_j, 1 \leq i < j \leq |A|$
- $0 \leq |B| \leq 50$
- $1 \leq B_i \leq N$
- $B_i \neq B_j, 1 \leq i < j \leq |B|$

Output Format

The output must consist of one number, the maximum number of distinct skill set that can the people of country learn, after making optimal transformation steps.

Sample Input

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

Sample Output

```
2
```

Explanation

There are 3 types of skill sets present along with 3 wizards. Initially, all three people know the 1^{st} skill set but no one knows the 2^{nd} and 3^{rd} skill sets.

The 1^{st} wizard's initial lists are: $A = [1]$ and $B = [2, 3]$. Suppose, he performs $1 \rightarrow 2$ transformation one any one of person with the 1^{st} skill set, then it's list A will be updated to an empty list [] and list B will be $[3]$.
Now, we have two people knowing the 1^{st} skill set and one person knowing the 2^{nd} skill set.

The 3^{rd} wizard's initial lists are: $A = [1]$ and $B = [2]$. He will use the transformation $1 \rightarrow 2$ one of the person with the 1^{st} skill set, then it's lists will also be updated to an empty lists A: [] and B: [].

Now, we have 1 person with 1^{st} skillset and and 2 people knowing the 2^{nd} skillset.

The 2^{nd} wizard's initial lists are: $A = [2]$ and $B = [3]$. He will transform one of the person with 2^{nd} skillset to 3^{rd} one using the transformation $2 \rightarrow 3$. It's lists will also be updated to an empty lists A: [] and B: [].
At this point, no further transformations are possible and we have achieved our maximum possible answer. Thus, each of the skill set, is known by 1 person.. This means there are three skill sets available in the kingdom.

Jim and his LAN Party

During the Steam Summer Sale, Jim's $N-1$ friends have purchased M games, which are numbered from 1 to M. The games are multiplayer. Jim has invited his friends to his basement where they will play by making a LAN-Party.

Each friend has already decided the game he would like to play for the rest of the day. So there will be a group of friends who will play the same game together.

But then, they face a problem: Currently, none of the friends' PCs are connected. So they have to be connected using the available Q wires. Jim decides to connect friends u_i and v_i with the i^{th} wire one by one. So he starts with wire 1, then with wire 2 and so on.

A group can start playing their game, only if all the members are connected (if not directly, then there must exist a path connecting them). They want to start playing as soon as possible.

For each game, find out the wire after adding which the group can start playing. It is also possible that a group will never get connected. In such a case, this group starts crying and you should display `-1`.

Input Format

On the first line there will be N, M and Q each separated by a single space. On the second line we will give you N integers separated by a single space: The i-th integer denotes the game friend i wants to play (all between 1 and M). The next Q lines will denote Q wires: i$^{\text{th}}$ line denotes i$^{\text{th}}$ wire and is denoted by u_i and v_i pairs each separated by a single space.

Constraints

$1 \leq N, M \leq 10^5$ For each game i, the number of players playing i will be positive.
$0 \leq Q \leq 10^5$

Note Each game is chosen by at least one player. If a group consists of only one member, then print `0`, since this lucky (?) lone player can start right away!

Output Format

Print on the i^{th} line the answer for the i^{th} game.

Sample Input

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

Sample Output

```
3
4
```

Explanation

The group with the game 1 can play after the 3$^{\text{rd}}$ wire is added. The group with game 2 can play only after the 4$^{\text{th}}$ wire has been added because after adding the 4$^{\text{th}}$ wire, a path between (2,3) (3,4) and (2,4) gets created.

Travel in HackerLand

HackerLand is a country with n beautiful cities and m undirected roads. Like every other beautiful country, HackerLand has traffic jams.

Each road has a *crowd value*. The crowd value of a path is defined as the maximum crowd value for all roads in the path. For example, if the crowd values for all roads are $[1, 4, 5, 6, 3]$, then the crowd value for the path will be $max(1, 4, 5, 6, 3) = 6$.

Each city i has a type value, T_i, denoting the type of buildings in the city.

David just started his vacation in HackerLand. He wants to travel from city x to city y. He also wants to see at least k different types of buildings along the path from x to y. When choosing a path from city x to city y that has at least k different types of buildings along the path, David always selects the one with the *minimum* crowd value.

You will be given q queries. Each query takes the form of 3 space-separated integers, x_i, y_i, and k_i, denoting the respective values for starting city, destination city, and minimum number of unique buildings that David wants to see along the way. For each query, you must print the minimum crowd value for a path between x_i and y_i that has at least k_i different buildings along the route. If there is no such path, print `-1`.

Note: A path may contain cycles (i.e., the same roads or cities may be traveled more than once).

Input Format

The first line contains 3 space-separated integers denoting the respective values for n (the number of cities), m (the number of roads), and q (the number of queries).

The second line contains n space-separated integers describing the respective building type for each city in array T (where the i-th value is T_i and $1 \leq i \leq n$).

Each of the m subsequent lines defines a road in the form of 3 space-separated integers, x_i, y_i, and u_i, defining an undirected road with crowd value u_i that connects cities x_i and y_i.

Each of the q subsequent lines defines a query in the form of 3 space-separated integers, x_j, y_j, and k_j (where $0 \leq j < q$), respectively.

Constraints

- $1 \leq n, m, q \leq 10^5$
- $1 \leq T_i, u_i \leq 10^9$
- $1 \leq x_j, y_j, k_j \leq n$
- Each road connect 2 distinct cities, meaning no road starts and ends in the same city.

Output Format

For each query, print its answer on a new line.

Sample Input

```
7 6 1
1 1 4 5 1 3 2
```

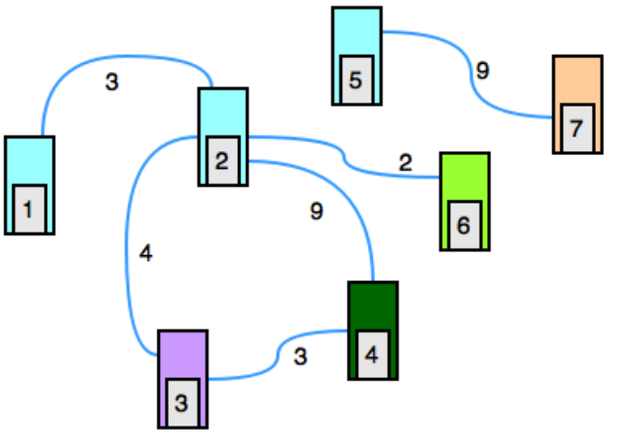
```
1 2 3
2 6 2
2 3 4
3 4 3
2 4 9
5 7 9
1 2 4
```

Sample Output

```
4
```

Explanation

The diagram below depicts the country given as *Sample Input*. Different values of T_i are shown in different colors.



The path for the last query (`1 2 4`) will be $1 \to 2 \to 3 \to 4 \to 3 \to 2 \to 6 \to 2$. David sees his first type of building in cities 1 and 2, his second type of building in city 3, his third type of building in city 4, and his fourth type of building in city 6. The crowd values for each road traveled on this path are $[3, 4, 3, 3, 4, 2, 2]$; the maximum of these values is 4. Thus, we print 4 on a new line.

Alex vs Fedor

In order to entertain themselves during the long flight, Alex and Fedor invented the following very simple two players game. The game is:

- First, Alex draws some graph with bidirectional weighted edges. There are possibly multiple edges (probably, with different or same weights) in this graph.
- Then Fedor picks some spanning tree of this graph. If the tree appears to be the minimal spanning tree, then the winner is Fedor. Otherwise, the winner is Alex.

We consider two trees different if the sets of the numbers of edges that are included in these trees are different. We consider two sets A and B different if there is at least one element that is present in A and not present in B or vice versa.

We should also mention that the graphs with enormous number of spanning trees upset Alex, as well as Fedor, so they will never have a graph that has more than 10^{18} spanning trees.

At some point, Fedor became too lazy to look for minimal spanning trees and now he just picks some arbitrary spanning tree from the Alex's graph. Each spanning tree has equal probability to be picked by Fedor. What is the probability of Fedor's victory now?

Input Format

The first line of input consists of two single space separated integers N and M - the number of nodes in Alex's graph and the number of edges in that graph, respectively.

Then there are M lines, where the i^{th} line has three numbers X_i Y_i Z_i with the meaning that the edge with the number i connects the nodes X_i and Y_i and has the weight of Z_i.

Constraints

- The graph is always connected.
- $1 \leq N, M \leq 100$
- $1 \leq weight < 2^{31}$

Output Format

Output the probability of Fedor's victory, if he picks the spanning tree randomly, as an irreducible fraction.

Sample Input

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

Sample Output

```
1/4
```

Vertical Paths

You have a rooted tree with n vertices numbered from 1 through n where the root is vertex 1.

You are given m triplets, the j^{th} triplet is denoted by three integers u_j, v_j, c_j. The j^{th} triplet represents a simple path in the tree with endpoints in u_i and v_i such that u_j is ancestor of v_j. The cost of the path is c_j.

You have to select a subset of the paths such that the sum of path costs is maximum and the i^{th} edge of the tree belongs to at most d_i paths from the subset. Print the sum as the output.

Input Format

The first line contains a single integer, T, denoting the number of testcases. Each testcase is defined as follows:

- The first line contains two space-separated integers, n (the number of vertices) and m (the number of paths), respectively.
- Each line i of the $n-1$ subsequent lines contains three space-separated integers describing the respective values of a_i, b_i, and d_i where (a_i, b_i) is an edge in the tree and d_i is maximum number of paths which can include this edge.
- Each line of the m subsequent lines contains three space-separated integers describing the respective values of u_j, v_j, and c_j ($u_j \neq v_j$) that define the j^{th} path and its cost.

Constraints

- Let M be the sum of m over all the trees.
- Let D be the sum of $n \times m$ over all the trees.
- $1 \leq T \leq 10^3$
- $1 \leq M, m \leq 10^3$
- $1 \leq D, n \leq 5 \times 10^5$
- $1 \leq c_i \leq 10^9$
- $1 \leq d_j \leq m$

Output Format

You must print T lines, where each line contains a single integer denoting the answer for the corresponding testcase.

Sample Input

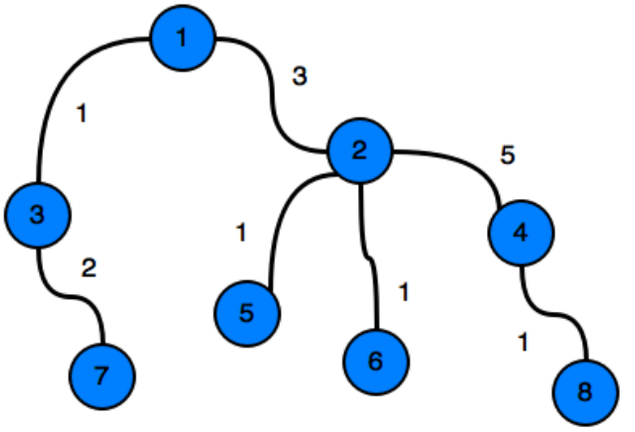
```
1
8 8
1 2 3
1 3 1
2 4 5
2 5 1
2 6 1
3 7 2
4 8 1
1 2 3
2 8 5
```

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

Sample Output

```
37
```

Explanation



One of the possible subsets contains paths $1, 2, 4, 5, 6, 7$. Its total cost is $3 + 5 + 8 + 10 + 5 + 6 = 37$.

Drive

HackerRank is starting a bus service in MountainView, California. The bus starts at time T = 0 at $station_1$ and goes through $station_2$, $station_3$, $station_4$ in that order and reaches the headquarters located at $station_n$. At every station, the bus waits for various commuters to arrive before it departs to the next station. Ignoring the acceleration, the bus moves at 1 meter / second. i.e., if $station_i$ and $station_j$ are 1000 meters apart, the bus takes 1000 seconds to travel from $station_i$ to $station_j$.

The bus is equipped with **K** units of Nitro (N_2O). If going from $station_i$ to $station_j$ takes x seconds, then using t units of nitro can decrease the time taken to max(x-t, 0) seconds where max(a,b) denotes the greater of the two values between a & b. The Nitro can be used all at once or in multiples of 1 unit.

If the bus driver travels optimally, what is the minimum sum of travelling time for all commuters? The travelling time equals to the time he/she arrived at the destination minus the time he/she arrived the start station.

Please remember that the driver must take all passengers to their destination.

Input Format

The first line contains 3 space separated integers n, m and K which indicate the number of stations, total number of people who board the bus at various stations and the total units of Nitro (N_2O) present in the bus.

The second line contains n-1 space separated integers where the i^{th} integer indicates the distance between $station_{(i-1)}$ to $station_i$.

m lines follow each containing 3 space separated integers. The i^{th} line contains t_i, s_i and e_i in that order indicating the arrival time of the commuter at s_i at time t_i with his destination being e_i.

```
n m K
d1 d2 ... dn-1   // di: the distance between station_i to station_(i+1).
t1 s1 e1         // commuter 1 arrives at his boarding point at s1 and his destination is e1
t2 s2 e2
...
tm sm em
```

Constraints

$0 < n <= 100000$
$0 < m <= 100000$
$0 <= K <= 10000000$
$0 < d_i <= 100$
$0 <= t_i <= 10000000$
$1 <= s_i < e_i <= n$

Output Format

The minimal total travel time.

Sample Input

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

Sample Output

```
9
```

Explanation

The bus waits for the 1^{st} and the 2^{nd} commuter to arrive at $station_1$ and travels to $station_2$ carrying 2 passengers. The travel time from $station_1$ to $station_2$ is 1 second. It then waits for the 3^{rd} commuter to board the bus at time = 5, 2^{nd} commuter deboards the bus. The 3^{rd} commuter boards the bus at t = 5. The bus now uses 2 units of nitro, this reduces the commute time to travel to $station_3$ from 4 to 2.

Hence, the total time spent by each of the passengers on the bus is

1. 1 (time spent waiting for commuter 2) + 1 (travel time from $station_1$ to $station_2$) + 2 (time spent waiting for commuter 3) + 2 (travel time from $station_2$ to $station_3$) = 6
2. 1 (travel time from $station_1$ to $station_2$)
3. 2 (travel time from $station_2$ to $station_3$)

   6+1+2 = 9

hence the answer.

Timelimits

Timelimits for this challenge can be seen here

Travelling Salesman in a Grid

The travelling salesman has a map containing m*n squares. He starts from the top left corner and visits every cell exactly once and returns to his initial position (top left). The time taken for the salesman to move from a square to its neighbor might not be the same. Two squares are considered adjacent if they share a common edge and the time taken to reach square *b* from square *a* and vice-versa are the same. Can you figure out the shortest time in which the salesman can visit every cell and get back to his initial position?

Input Format

The first line of the input is 2 integers m and n separated by a single space. m and n are the number of rows and columns of the map.
Then m lines follow, each of which contains (n – 1) space separated integers. The j^{th} integer of the i^{th} line is the travel time from position (i,j) to (i,j+1) (index starts from 1.)
Then (m-1) lines follow, each of which contains n space integers. The j^{th} integer of the i^{th} line is the travel time from position (i,j) to (i + 1, j).

Constraints

$1 \leq m, n \leq 10$
Times are non-negative integers no larger than 10000.

Output Format

Just an integer contains the minimal time to complete his task. Print 0 if its not possible to visit each cell exactly once.

Sample Input

```
2 2
5
8
6 7
```

Sample Output

```
26
```

Explanation

As its a 2*2 square, all cells are visited. 5 + 7 + 8 + 6 = 26

Road Network

Chinese

Fedor is a research scientist, who has recently found a road map of Ancient Berland.

Ancient Berland consisted of N cities that were connected by M bidirectional roads. The road builders weren't knowledgable. Hence, the start city and the end city for each road were always chosen *randomly and independently*. As a result, there were more than one road between some pairs of cities. Nevertheless, by luck, the country remained connected (i.e. you were able to get from one city to another via these M roads). And for any road, the start and the end city were not the same.

Moreover, each road had it's own *value of importance*. This value was assigned by the Road Minister of Ancient Berland. The Road Minister also was not knowledgable, so these numbers were assigned to the roads *randomly and independently* from the other roads.

When there was a war with the neighboring countries (usually it was with Ancient Herland), it was important to estimate *separation number* for some pairs of cities.

The separation number for a pair of cities - let's call these cities A and B - is explained below:

Consider a set of roads that were built. The subset of this set is *good*, if after removing all roads from this set, there's no longer a way from A to B. The minimal possible sum of roads' *value of importance* of any good subset is a *separation number* for the pair of cities (A, B).

For a research, Fedor would like to know the product of *separation values* over all unordered pairs of cities. Please, find this number. It can be huge, so we ask you to output its product modulo 10^9+7.

Input Format

The first line of input consist of two integers N and M, separated by a single space.
Then, M lines follow. Each of these lines consist of three integers X_i, Y_i, Z_i separated by a single space.
It means that there was a road between the city X_i and the city Y_i with a value of importance equal to Z_i.

Constraints

$3 \leq N \leq 500$
$3 \leq M \leq 10^4$
$1 \leq$ *value of importance* $\leq 10^5$
The cities are indexed from 1 to N.

Scoring

In the 25% of the test data $N = 50$ and $M = 300$.

In another 25% of the test data $N = 200$ and $M = 10000$

In the rest of the test data $N = 500$ and $M = 10000$

Output Format

An integer that represents the value, Fedor needs, modulo 10^9+7.

Sample Input 0

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

Sample Output 0

```
36
```

Explanation 0

There are three unordered pairs of cities: (1, 2), (1, 3) and (2, 3). Let's look at the *separation numbers*:

- For (1, 2) we have to remove the first and the second roads. The sum of the importance values is 4.
- For (1, 3) we have to remove the second and the third roads. The sum of the importance values is 3.
- For (2, 3) we have to remove the second and the third roads. The sum of the importance values is 3.

  So, we get 4 * 3 * 3 = 36.

Going to the Office

Ms.Kox enjoys her job, but she does not like to waste extra time traveling to and from her office. After working for many years, she knows the shortest-distance route to her office on a regular day.

Recently, the city began regular maintenance of various roads. Every day a road gets blocked and no one can use it that day, but all other roads can be used. You are Ms. Kox's new intern and she needs some help. Every day, you need to determine the minimum distance that she has to travel to reach her office.

Input Format

There are N cities numbered 0 to N-1 and M bidirectional roads.

- The first line of the input contains two integers N and M.
- M lines follow, each containing three space-separated integers u , v and w, where u and v are cities connected by a bi-directional road and w is the length of this road. There is at most one road between any two cities and no road connects a city to itself.
- The next line contains two integers S and D. S is the city where Ms. Kox lives and D is the city where her office is located.
- The next line contains an integer Q, the number of queries.
- Q lines follow, each containing two integers u and v, where the road between u and v has been blocked that day.

Constraints

$0 < N < 200,000$
$0 < M < 200,000$
$0 < Q < 200,000$
$0 < w < 1001$
$0 \leq S, D < N$

Output Format

Output Q lines, with each line containing the minimum distance Ms.Kox has to travel on that day. If there is no path, print "**Infinity**".

Sample Input

```
6 9
0 1 1
1 2 1
2 3 1
3 4 1
4 5 1
2 4 5
3 5 8
1 3 3
0 2 4
0 5
9
0 1
1 2
2 3
3 4
4 5
2 4
3 5
1 3
0 2
```

Sample Output

```
7
6
6
8
11
5
5
5
5
```

Tree Splitting

Given a tree with vertices numbered from 1 to n, perform m queries. Each query is in the form of a vertex number. For each query, v:

1. Print the size of the connected component containing v.
2. Remove vertex v and all edges connected to v.

Input Format

The first line contains a single integer, n, denoting the number of vertices in the tree.
Each line i of the $n-1$ subsequent lines (where $0 \leq i < n$) contains 2 space-separated integers describing the respective nodes, u_i and v_i, connected by edge i.
The next line contains a single integer, m, denoting the number of queries.
Each line j of the m subsequent lines contains a single integer, vertex number m_j.

Queries are encoded in the following way. Let $ans_0 = 0$ and ans_j be the answer for the j^{th} query. Then $v_j = ans_{j-1} \oplus m_j$. We are assure that v_j is between 1 and n, and hasn't removed before.

Note: $\oplus$ is the bitwise XOR operator.

Constraints

- $1 \leq n, m \leq 2 \cdot 10^5$.

Output Format

For each query, print the size of the corresponding connected component on a new line.

Sample Input 0

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

Sample Output 0

```
3
1
1
```

Sample Input 1

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

Sample Output 1

```
4
3
2
1
```

Explanation

Sample Case 0:
Queries are $\mathbf{1}$, $\mathbf{2}$, $\mathbf{3}$, in order.

Sample Case 1:
Queries are $\mathbf{3}$, $\mathbf{2}$, $\mathbf{1}$, $\mathbf{4}$, in order.

Ticket

There are n people at the railway station, and each one wants to buy a ticket to go to one of k different destinations. The n people are in a queue.

There are m ticket windows from which tickets can be purchased. The n people will be distributed in the windows such that *the order is maintained*. In other words, suppose we number the people 1 to n from front to back. If person i and person j go to the same window and $i < j$, then person i should still be ahead of person j in the window.

Each ticketing window has an offer. If a person in the queue shares the same destination as the person immediately in front of him/her, a 20% reduction in the ticket price is offered to him/her.

For example, suppose there are 3 people in the queue for a single ticket window, all with the same destination which costs 10 bucks. Then the first person in the queue pays 10 bucks, and the 2nd and 3rd persons get a discount of 20% on 10 bucks, so they end up paying 8 bucks each instead of 10 bucks.

Try to distribute the n people across the m windows such that the total cost S paid by all n people is minimized.

Input Format

The first line contains 3 integers:

- n is the number of people
- m is the number of ticket windows
- k is the number of destinations separated by a single space (in the same order)

Then k lines follow. The i^{th} line contains an alphanumeric string place_i and an integer price_i:

- place_i is the i^{th} destination
- price_i is the ticket price for place_i

Then n lines follow. The i^{th} line contains an alphanumeric string destination_i which is the destination of the i^{th} person.

Constraints

- $1 \leq n \leq 500$
- $1 \leq m \leq 10$
- $1 \leq k \leq 100$
- The k available destinations have nonempty and distinct names.
- Each person's destination appears in the list of k available destinations.
- $0 \leq \text{price}_i \leq 100$

Output Format

Output $n + 1$ lines. The first line contains S, the total cost that is to be minimized. In the i^{th} following line, print the ticket window which the i^{th} person goes to. The windows are indexed 1 to m. There may be multiple ways to distribute the people among the windows such that the total cost is minimized; any one will be accepted.

The answer S will be accepted if it is within an error of 10^{-3} of the true answer.

Sample Input

```
5 2 3
CALIFORNIA 10
HAWAII 8
NEWYORK 12
NEWYORK
NEWYORK
CALIFORNIA
NEWYORK
HAWAII
```

Sample Output

```
49.2
1
1
2
1
1
```

Explanation

At the beginning, all the people are in the same queue, and will go to the ticket windows one by one in the initial order.

- $\{1, 2, 4, 5\}$ will buy ticket in the first window.
- $\{3\}$ will buy ticket in the second window.

In the first ticket window, #1 will pay 12 bucks to go to `NEWYORK`, and #2 and #4 have the same destination with the person in front of them, so they will get 20% off, and will pay 9.6 bucks each. #5 has a different destination, so it will cost him 8 bucks to go to `HAWAII`.

In the second ticket window, #3 will pay 10 bucks to go to `CALIFORNIA`.

DFS Edges

Let G be a connected, directed graph with vertices numbered from 1 to n such that any vertex is reachable from vertex 1. In addition, any two distinct vertices, u and v, are connected by *at most* one edge (u, v).

Consider the standard *DFS* (Depth-First Search) algorithm starting from vertex 1. As every vertex is reachable, each edge (u, v) of G is classified by the algorithm into one of four groups:

1. *tree edge*: If v was discovered for the first time when we traversed (u, v).
2. *back edge*: If v was already on the stack when we tried to traverse (u, v).
3. *forward edge*: If v was already discovered *while* u was on the stack.
4. *cross edge*: Any edge that is not a *tree*, *back*, or *forward* edge.

To better understand this, consider the following C++ pseudocode:

```
// initially false
bool discovered[n];

// initially false
bool finished[n];

vector<int> g[n];

void dfs(int u) {
    // u is on the stack now
    discovered[u] = true;
    for (int v: g[u]) {
        if (finished[v]) {
            // forward edge if u was on the stack when v was discovered
            // cross edge otherwise
            continue;
        }
        if (discovered[v]) {
            // back edge
            continue;
        }
        // tree edge
        dfs(v);
    }
    finished[u] = true;
    // u is no longer on the stack
}
```

Given four integers, t, b, f, and c, construct any graph G having exactly t *tree edges*, exactly b *back edges*, exactly f *forward edges*, and exactly c *cross edges*. Then print G according to the *Output Format* specified below.

Input Format

A single line of four space-separated integers describing the respective values of t, b, f, and c.

Constraints

- $0 \leq t, b, f, c \leq 10^5$

Output Format

If there is no such graph G, print `-1`; otherwise print the following:

1. The first line must contain an integer, n, denoting the number of vertices in G.
2. Each line i of the n subsequent lines must contain the following space-separated integers:

- The first integer is the outdegree, d_i, of vertex i.
- This is followed by d_i distinct numbers, $v_{i,j}$, denoting edges from u to $v_{i,j}$ for $1 \leq j \leq d_i$. The order of each $v_{i,j}$ should be the order in which a *DFS* considers edges.

Sample Input 0

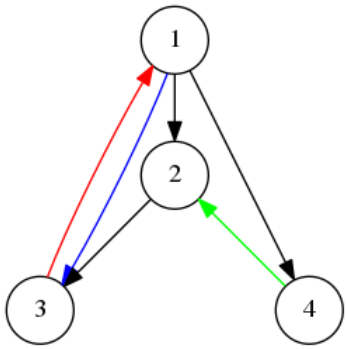
```
3 1 1 1
```

Sample Output 0

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

Explanation 0

The *DFS* traversal order is: $1, 2, 3, 2, 1, 4, 1$. Thus, $(1, 2)$, $(2, 3)$ and $(1, 4)$ are *tree edges*; $(3, 1)$ is a *back edge*; $(1, 3)$ is a *forward edge*; and $(4, 2)$ is a *cross edge*. This is demonstrated by the diagram below, in which *tree edges* are black, *forward edges* are blue, *back edges* are red, and *cross edges* are green.



Sample Input 1

```
1 10 20 30
```

Sample Output 1

```
-1
```

Explanation 1

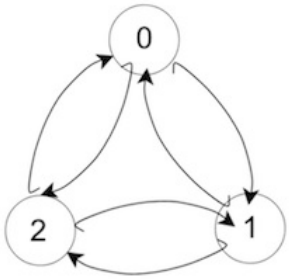
No such graph exists satisfying the given values.

Diameter Minimization

We define the diameter of a strongly-connected oriented graph, $G = (V, E)$, as the minimum integer d such that for each $u, v \in G$ there is a path from u to v of length $\leq d$ (recall that a path's length is its number of edges).

Given two integers, n and m, build a strongly-connected oriented graph with n vertices where each vertex has outdegree m and *the graph's diameter is as small as possible* (see the *Scoring* section below for more detail). Then print the graph according to the *Output Format* specified below.

Here's a sample strongly-connected oriented graph with 3 nodes, whose outdegree is 2 and diameter is 1.



Note: Cycles and multiple edges between vertices are allowed.

Input Format

Two space-separated integers describing the respective values of n (the number of vertices) and m (the outdegree of each vertex).

Constraints

- $2 \leq n \leq 1000$
- $2 \leq m \leq \min(n, 5)$

Scoring

We denote the diameter of your graph as d and the diameter of the graph in the author's solution as s. Your score for each test case (as a real number from 0 to 1) is:

- 1 if $d \leq s + 1$
- $\frac{s}{d}$ if $s + 1 < d \leq 5 \times s$
- 0 if $5 \times s < d$

Output Format

First, print an integer denoting the diameter of your graph on a new line.
Next, print n lines where each line i ($0 \leq i < n$) contains m space-separated integers in the inclusive range from 0 to $n - 1$ describing the endpoints for each of vertex i's outbound edges.

Sample Input 0

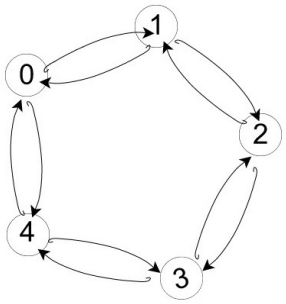
```
5 2
```

Sample Output 0

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

Explanation 0

The diagram below depicts a strongly-connected oriented graph with $n = 5$ nodes where each node has an outdegree of $m = 2$:



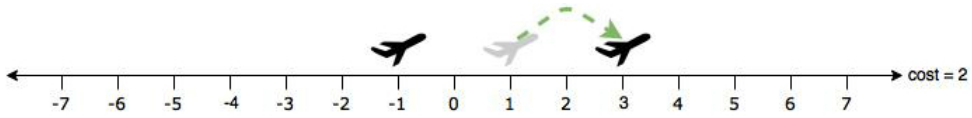
The diameter of this graph is $d = 2$, which is minimal as the outdegree of each node must be m. We cannot construct a graph with a smaller diameter of $d = 1$ because it requires an outbound edge from each vertex to each other vertex in the graph (so the outdegree of that graph would be $n - 1$).

Airports

Airports are being built on a straight road according to a new construction plan. For convenience, imagine a number line on which at different points airports can be positioned. Because a plane can't take off and start landing immediately, there will be flight between two airports in locations x and y if and only if $|x - y| \geq d$, where d is a constant.

Changing the position of an airport from x to y costs $|x - y|$. The cost to fix a certain plan is the minimum total cost of changing the positions of airports. After the changes, it should be possible to travel between any pair of airports, possibly taking flights through some intermediate airports. Note that it's possible that two airports have the same initial position, and this can be the case after changes too.

On i^{th} day, a plan to build a new airport with position x_i is announced. On each day that a new airport is announced, print the smallest cost to fix the set of airports announced so far . Note that you should not change the positions of any airports, just calculate the cost to do it.



Input Format

Input contains multiple queries.
The first line consists of an integer q which is the number of queries. Each query is given as follows.
The first line of each query contains two integers n and d, the number of days, and the minimum distance respectively.
The second line of each test case contains n space-separated integers x_i denoting the position of the airport that was announced on i^{th} day.

Constraints

- $3 \leq n \leq 2 \cdot 10^5$
- $|x_i| \leq 10^8$
- $0 \leq d \leq 10^8$
- the sum of n over all test cases in a file will not exceed $2 \cdot 10^5$

Output Format

Print one line for each query.
A line for a query with n airports should have n numbers on it where the i^{th} one should be the minimum cost to fix airports in positions $x_1, x_2, \cdots, x_i$.

Sample Input 0

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

Sample Output 0

```
0 1 1
```

Explanation 0

The answer for a single airport is always zero. When we have many airports in the same position, it's enough to move only one of them to satisfy the condition from the statement.

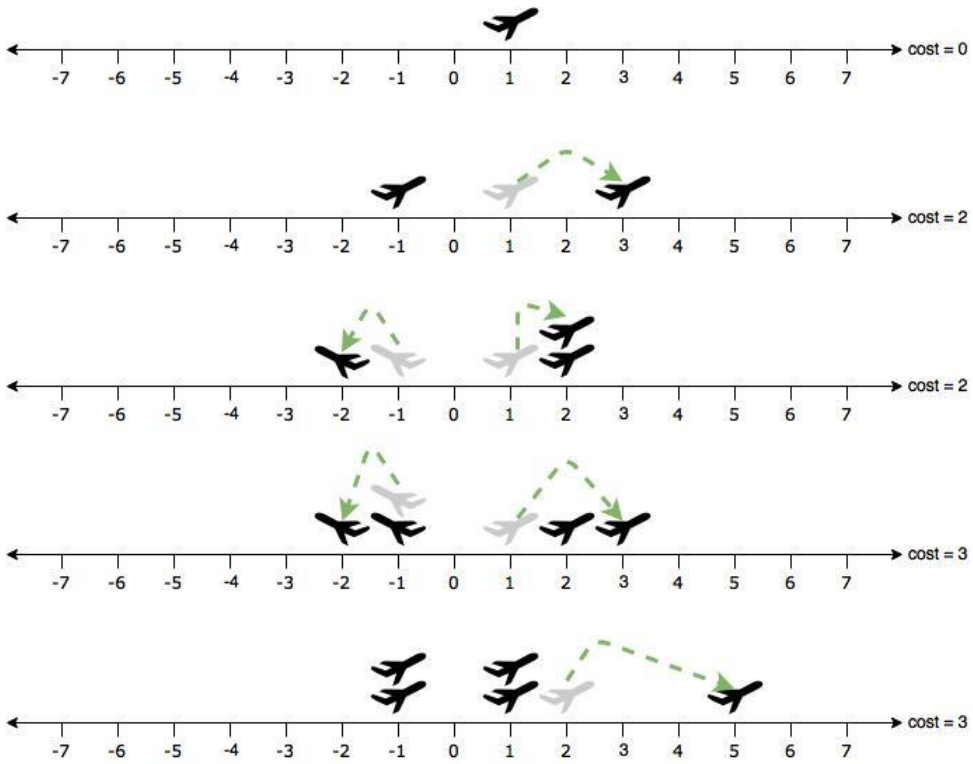
Sample Input 1

```
1
5 4
1 -1 2 -1 1
```

Sample Output 1

```
0 2 2 3 3
```

Explanation 1



For each new day that an airport is inserted, the cheapest rearranging of existing airports is shown on the diagram above. Note that cost changes for every day and travelling between airports can be done possibly flying through some intermediate ones. Costs are calculated without changing actual positions of the airports.

Definite Random Walks

Alex has a board game consisting of:

- A *chip* for marking his current location on the board.
- n *fields* numbered from 1 to n. Each position i has a value, f_i, denoting the *next* position for the chip to jump to from that field.
- A *die* with m faces numbered from 0 to $m-1$. Each face j has a probability, p_j, of being rolled.

Alex then performs the following actions:

- Begins the game by placing the chip at a position in a field randomly and with equiprobability.
- Takes k turns; during each turn he:
  - Rolls the die. We'll denote the number rolled during a turn as d.
  - Jumps the chip d times. Recall that each field contains a value denoting the *next* field number to jump to.
- After completing k turns, the game ends and he must calculate the respective probabilities for each field as to whether the game ended with the chip in that field.

Given n, m, k, the game board, and the probabilities for each *die* face, print n lines where each line i contains the probability that the chip is on field i at the end of the game.

Note: All the probabilities in this task are rational numbers modulo $M = 998244353$. That is, if the probability can be expressed as the irreducible fraction $\frac{p}{q}$ where $q \bmod M \neq 0$, then it corresponds to the number $(p \times q^{-1}) \bmod M$ (or, alternatively, $p \times q^{-1} \equiv x (\bmod M)$). Click here to learn about *Modular Multiplicative Inverse*.

Input Format

The first line contains three space-separated integers describing the respective values of n (the number of positions), m (the number of die faces), and k (the number of turns).
The second line contains n space-separated integers describing the respective values of each f_i (i.e., the index of the field that field i can transition to).
The third line contains m space-separated integers describing the respective values of each p_j (where $0 \leq p_j < M$) describing the probabilities of the faces of the m-sided die.

Constraints

- $1 \leq n \leq 6 \times 10^4$
- $4 \leq m \leq 10^5$
- $1 \leq k \leq 1000$
- $1 \leq i, f_i \leq n$
- $0 \leq p_j < M$
- The sum of $p_j \bmod M$ is 1

Note: The time limit for this challenge is doubled for *all* languages. Read more about standard time limits at our environment page.

Output Format

Print n lines of output in which each line i contains a single integer, x_i (where $0 \leq x_i < M$), denoting the probability that the chip will be on field i after k turns.

Sample Input 0

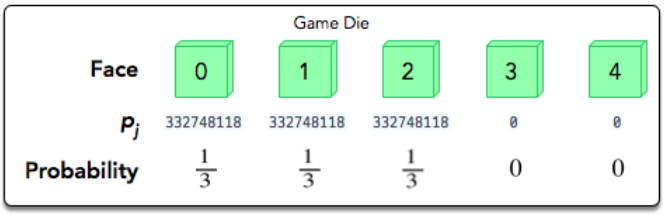
```
4 5 1
2 3 2 4
332748118 332748118 332748118 0 0
```

Sample Output 0

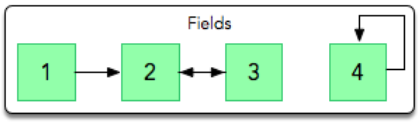
```
582309206
332748118
332748118
748683265
```

Explanation 0

The diagram below depicts the respective probabilities of each *die* face being rolled:



The diagram below depicts each field with an arrow pointing to the *next* field:



There are four equiprobable initial fields, so each field has a $\frac{1}{4}$ probability of being the chip's initial location. Next, we calculate the probability that the chip will end up in each field after $k = 1$ turn:

1. The only way the chip ends up in this field is if it never jumps from the field, which only happens if Alex rolls a 0. So, this field's probability is $\frac{1}{4} \cdot \frac{1}{3} = \frac{1}{12}$. We then calculate and print the result of $\frac{1}{12} \bmod 998244353 = 582309206$ on a new line.
2. The chip can end up in field 2 after one turn in the following scenarios:
   - Start in field 1 and roll a 1, the probability for which is $\frac{1}{4} \cdot \frac{1}{3} = \frac{1}{12}$.
   - Start in field 2 and roll a 0 or a 2, the probability for which is $\frac{1}{4} \cdot \frac{2}{3} = \frac{2}{12}$.
   - Start in field 3 and roll a 1, the probability for which is $\frac{1}{4} \cdot \frac{1}{3} = \frac{1}{12}$.

   After summing these probabilities, we get a total probability of $\frac{1}{12} + \frac{2}{12} + \frac{1}{12} = \frac{1}{3}$ for the field. We then calculate and print the result of $\frac{1}{3} \bmod 998244353 = 332748118$ on a new line.
3. The chip can end up in field 3 after one turn in the following scenarios:
   - Start in field 1 and roll a 2, the probability for which is $\frac{1}{4} \cdot \frac{1}{3} = \frac{1}{12}$.
   - Start in field 2 and roll a 1, the probability for which is $\frac{1}{4} \cdot \frac{1}{3} = \frac{1}{12}$.
   - Start in field 3 and roll a 0 or a 2, the probability for which is $\frac{1}{4} \cdot \frac{2}{3} = \frac{2}{12}$.

After summing these probabilities, we get a total probability of $\frac{1}{12} + \frac{1}{12} + \frac{2}{12} = \frac{1}{3}$ for the field. We then calculate and print the result of $\frac{1}{3} \bmod 998244353 = 332748118$ on a new line.

4. If the chip is initially placed in field 4, it will always end up in field 4 regardless of how many turns are taken (because this field loops back onto itself). Thus, this field's probability is $\frac{1}{4}$. We then calculate and print the result of $\frac{1}{4} \bmod 998244353 = 748683265$ on a new line.