CODEFORCES β
Sponsored by Telegram

15-295 Spring 2017 #5

A. Two Buttons

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Vasya has found a strange device. On the front panel of a device there are: a red button, a blue button and a display showing some positive integer. After clicking the red button, device multiplies the displayed number by two. After clicking the blue button, device subtracts one from the number on the display. If at some point the number stops being positive, the device breaks down. The display can show arbitrarily large numbers. Initially, the display shows number n.

Bob wants to get number m on the display. What minimum number of clicks he has to make in order to achieve this result?

Input
The first and the only line of the input contains two distinct integers n and m ($1 \le n, m \le 10^4$), separated by a space .

Output
Print a single number — the minimum number of times one needs to push the button required to get the number m out of number n.

Examples

input
4 6
output
2

input
10 1
output
9

Note
In the first example you need to push the blue button once, and then push the red button once.

In the second example, doubling the number is unnecessary, so we need to push the blue button nine times.

B. Dorm Water Supply

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

The German University in Cairo (GUC) dorm houses are numbered from 1 to n. Underground water pipes connect these houses together. Each pipe has certain direction (water can flow only in this direction and not vice versa), and diameter (which characterizes the maximal amount of water it can handle).

For each house, there is at most one pipe going into it and at most one pipe going out of it. With the new semester starting, GUC student and dorm resident, Lulu, wants to install tanks and taps at the dorms. For every house with an outgoing water pipe and without an incoming water pipe, Lulu should install a water tank at that house. For every house with an incoming water pipe and without an outgoing water pipe, Lulu should install a water tap at that house. Each tank house will convey water to all houses that have a sequence of pipes from the tank to it. Accordingly, each tap house will receive water originating from some tank house.

In order to avoid pipes from bursting one week later (like what happened last semester), Lulu also has to consider the diameter of the pipes. The amount of water each tank conveys should not exceed the diameter of the pipes connecting a tank to its corresponding tap. Lulu wants to find the maximal amount of water that can be safely conveyed from each tank to its corresponding tap.

Input

The first line contains two space-separated integers n and p ($1 \le n \le 1000$, $0 \le p \le n$) — the number of houses and the number of pipes correspondingly.

Then p lines follow — the description of p pipes. The i-th line contains three integers a_i b_i d_i, indicating a pipe of diameter d_i going from house a_i to house b_i ($1 \le a_i, b_i \le n$, $a_i \ne b_i$, $1 \le d_i \le 10^6$).

It is guaranteed that for each house there is at most one pipe going into it and at most one pipe going out of it.

Output

Print integer t in the first line — the number of tank-tap pairs of houses.

For the next t lines, print 3 integers per line, separated by spaces: $tank_i$, tap_i, and $diameter_i$, where $tank_i \ne tap_i$ ($1 \le i \le t$). Here $tank_i$ and tap_i are indexes of tank and tap houses respectively, and $diameter_i$ is the maximum amount of water that can be conveyed. All the t lines should be ordered (increasingly) by $tank_i$.

Examples

input
3 2 1 2 10 2 3 20
output
1 1 3 10

input
3 3 1 2 20 2 3 10 3 1 5
output
0

input
4 2 1 2 60 3 4 50
output
2 1 2 60 3 4 50

C. The Labyrinth

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a rectangular field of $n \times m$ cells. Each cell is either empty or impassable (contains an obstacle). Empty cells are marked with '`.`', impassable cells are marked with '`*`'. Let's call two empty cells `adjacent` if they share a side.

Let's call a *connected component* any non-extendible set of cells such that any two of them are connected by the path of adjacent cells. It is a typical well-known definition of a connected component.

For each impassable cell (x, y) imagine that it is an empty cell (all other cells remain unchanged) and find the size (the number of cells) of the connected component which contains (x, y). You should do it for each impassable cell independently.

The answer should be printed as a matrix with n rows and m columns. The j-th symbol of the i-th row should be "`.`" if the cell is empty at the start. Otherwise the j-th symbol of the i-th row should contain the only digit —- the answer modulo 10. The matrix should be printed without any spaces.

To make your output faster it is recommended to build the output as an array of n strings having length m and print it as a sequence of lines. It will be much faster than writing character-by-character.

As input/output can reach huge size it is recommended to use fast input/output methods: for example, prefer to use `scanf/printf` instead of `cin/cout` in C++, prefer to use `BufferedReader/PrintWriter` instead of `Scanner/System.out` in `Java`.

Input

The first line contains two integers n, m $(1 \le n, m \le 1000)$ — the number of rows and columns in the field.

Each of the next n lines contains m symbols: "`.`" for empty cells, "`*`" for impassable cells.

Output

Print the answer as a matrix as described above. See the examples to precise the format of the output.

Examples

input
3 3 *.* .*. *.*
output
3.3 .5. 3.3

input
4 5 **..* ..*** .*.*. *.*.*
output
46..3 ..732 .6.4. 5.4.3

Note

In first example, if we imagine that the central cell is empty then it will be included to component of size 5 (cross). If any of the corner cell will be empty then it will be included to component of size 3 (corner).

D. Volleyball

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Petya loves volleyball very much. One day he was running late for a volleyball match. Petya hasn't bought his own car yet, that's why he had to take a taxi. The city has n junctions, some of which are connected by two-way roads. The length of each road is defined by some positive integer number of meters; the roads can have different lengths.

Initially each junction has exactly one taxi standing there. The taxi driver from the i-th junction agrees to drive Petya (perhaps through several intermediate junctions) to some other junction if the travel distance is not more than t_i meters. Also, the cost of the ride doesn't depend on the distance and is equal to c_i bourles. Taxis can't stop in the middle of a road. **Each taxi can be used no more than once. Petya can catch taxi only in the junction, where it stands initially.**

At the moment Petya is located on the junction x and the volleyball stadium is on the junction y. Determine the minimum amount of money Petya will need to drive to the stadium.

Input

The first line contains two integers n and m ($1 \le n \le 1000, 0 \le m \le 1000$). They are the number of junctions and roads in the city correspondingly. The junctions are numbered from 1 to n, inclusive. The next line contains two integers x and y ($1 \le x, y \le n$). They are the numbers of the initial and final junctions correspondingly. Next m lines contain the roads' description. Each road is described by a group of three integers u_i, v_i, w_i ($1 \le u_i, v_i \le n, 1 \le w_i \le 10^9$) — they are the numbers of the junctions connected by the road and the length of the road, correspondingly. The next n lines contain n pairs of integers t_i and c_i ($1 \le t_i, c_i \le 10^9$), which describe the taxi driver that waits at the i-th junction — the maximum distance he can drive and the drive's cost. The road can't connect the junction with itself, but between a pair of junctions there can be more than one road. All consecutive numbers in each line are separated by exactly one space character.

Output

If taxis can't drive Petya to the destination point, print "-1" (without the quotes). Otherwise, print the drive's minimum cost.

Please do not use the %lld specificator to read or write 64-bit integers in C++. It is preferred to use cin, cout streams or the %I64d specificator.

Examples

input
4 4 1 3 1 2 3 1 4 1 2 4 1 2 3 5 2 7 7 2 1 2 7 7

output
9

Note

An optimal way — ride from the junction 1 to 2 (via junction 4), then from 2 to 3. It costs 7+2=9 bourles.

E. The Two Routes

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

In Absurdistan, there are n towns (numbered 1 through n) and m bidirectional railways. There is also an absurdly simple road network — for each pair of different towns x and y, there is a bidirectional road between towns x and y **if and only if** there is no railway between them. Travelling to a different town using one railway or one road always takes exactly one hour.

A train and a bus leave town 1 at the same time. They both have the same destination, town n, and don't make any stops on the way (but they can wait in town n). The train can move only along railways and the bus can move only along roads.

You've been asked to plan out routes for the vehicles; each route can use any road/railway multiple times. One of the most important aspects to consider is safety — in order to avoid accidents at railway crossings, the train and the bus must not arrive at the same town (except town n) simultaneously.

Under these constraints, what is the minimum number of hours needed for both vehicles to reach town n (the maximum of arrival times of the bus and the train)? Note, that bus and train are not required to arrive to the town n at the same moment of time, but are allowed to do so.

Input

The first line of the input contains two integers n and m ($2 \le n \le 400$, $0 \le m \le n(n - 1) / 2$) — the number of towns and the number of railways respectively.

Each of the next m lines contains two integers u and v, denoting a railway between towns u and v ($1 \le u, v \le n$, $u \ne v$).

You may assume that there is at most one railway connecting any two towns.

Output

Output one integer — the smallest possible time of the later vehicle's arrival in town n. If it's impossible for at least one of the vehicles to reach town n, output -1.

Examples

input
4 2 1 3 3 4
output
2

input
4 6 1 2 1 3 1 4 2 3 2 4 3 4
output
-1

input
5 5 4 2 3 5 4 5 5 1 1 2
output
3

Note

In the first sample, the train can take the route $1 \to 3 \to 4$ and the bus can take the route $1 \to 2 \to 4$. Note that they can arrive at town 4 at the same time.

In the second sample, Absurdistan is ruled by railwaymen. There are no roads, so there's no way for the bus to reach town 4.

F. Complete The Graph

time limit per test: 4 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

ZS the Coder has drawn an undirected graph of n vertices numbered from 0 to n - 1 and m edges between them. Each edge of the graph is weighted, each weight is a **positive integer**.

The next day, ZS the Coder realized that some of the weights were erased! So he wants to reassign **positive integer** weight to each of the edges which weights were erased, so that the length of the shortest path between vertices s and t in the resulting graph is exactly L. Can you help him?

Input

The first line contains five integers n, m, L, s, t ($2 \le n \le 1000,\ 1 \le m \le 10\,000,\ 1 \le L \le 10^9,\ 0 \le s, t \le n$ - $1,\ s \ne t$) — the number of vertices, number of edges, the desired length of shortest path, starting vertex and ending vertex respectively.

Then, m lines describing the edges of the graph follow. i-th of them contains three integers, u_i, v_i, w_i ($0 \le u_i, v_i \le n$ - $1,\ u_i \ne v_i,\ 0 \le w_i \le 10^9$). u_i and v_i denote the endpoints of the edge and w_i denotes its weight. If w_i is equal to 0 then the weight of the corresponding edge was erased.

It is guaranteed that there is at most one edge between any pair of vertices.

Output

Print "NO" (without quotes) in the only line if it's not possible to assign the weights in a required way.

Otherwise, print "YES" in the first line. Next m lines should contain the edges of the resulting graph, with weights assigned to edges which weights were erased. i-th of them should contain three integers u_i, v_i and w_i, denoting an edge between vertices u_i and v_i of weight w_i. The edges of the new graph must coincide with the ones in the graph from the input. The weights that were not erased must remain unchanged whereas the new weights can be any **positive integer** not exceeding 10^{18}.

The order of the edges in the output doesn't matter. The length of the shortest path between s and t must be equal to L.

If there are multiple solutions, print any of them.

Examples

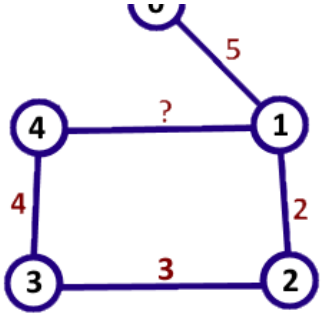
input
5 5 13 0 4 0 1 5 2 1 2 3 2 3 1 4 0 4 3 4
output
YES 0 1 5 2 1 2 3 2 3 1 4 8 4 3 4

input
2 1 123456789 0 1 0 1 0
output
YES 0 1 123456789

input
2 1 999999999 1 0 0 1 1000000000
output
NO

Note

Here's how the graph in the first sample case looks like :



In the first sample case, there is only one missing edge weight. Placing the weight of 8 gives a shortest path from 0 to 4 of length 13.

In the second sample case, there is only a single edge. Clearly, the only way is to replace the missing weight with 123456789.

In the last sample case, there is no weights to assign but the length of the shortest path doesn't match the required value, so the answer is "NO".

G. Generating Sets

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a set Y of n **distinct** positive integers $y_1, y_2, ..., y_n$.

Set X of n **distinct** positive integers $x_1, x_2, ..., x_n$ is said to *generate* set Y if one can transform X to Y by applying some number of the following two operation to integers in X:

1. Take any integer x_i and multiply it by two, i.e. replace x_i with $2 \cdot x_i$.
2. Take any integer x_i, multiply it by two and add one, i.e. replace x_i with $2 \cdot x_i + 1$.

Note that integers in X are not required to be distinct after each operation.

Two sets of distinct integers X and Y are equal if they are equal as sets. In other words, if we write elements of the sets in the array in the increasing order, these arrays would be equal.

Note, that any set of integers (or its permutation) generates itself.

You are given a set Y and have to find a set X that generates Y and the **maximum element of X is mininum possible**.

Input

The first line of the input contains a single integer n ($1 \le n \le 50\,000$) — the number of elements in Y.

The second line contains n integers $y_1, ..., y_n$ ($1 \le y_i \le 10^9$), that are guaranteed to be distinct.

Output

Print n integers — set of distinct integers that generate Y and the maximum element of which is minimum possible. If there are several such sets, print any of them.

Examples

input
5 1 2 3 4 5
output
4 5 2 3 1

input
6 15 14 3 13 1 12
output
12 13 14 7 3 1

input
6 9 7 13 17 5 11
output
4 5 2 6 3 1