Graphs, Trees and Recursion

Assignment 1 (shortest path):

Implement the following algorithm:

We have a network of **N** nodes numbered from 1 to **N**. Some of the network nodes are connected with a **bidirectional** connection. Your task is to find an optimal (shortest) route from node **1** to node **N**. Output of your program will show the number of the connections in the optimal route from 1 to N.

For the choice of the algorithm, please look at theory lesson. You can implement the searching algorithm on your own or reuse an implementation from the web. In the latter case, don't forget to mention the references that you used.

Input Format

Input will be read from the standard input (stdin) and will have the following format:

First line contains **T**. **T** test cases follow.
First line of each test case contains two space-separated integers **N**, **M**.
Each of the next **M** lines contains two space-separated integers **X** and **Y**, denoting that there is a connection between node **X** and node **Y**.

Output Format

Answer to each test case in a new line.

Constraints

- $1 \leq \mathbf{T} \leq 10$
- $1 \leq \mathbf{N} \leq 10^4$
- $1 \leq \mathbf{M} \leq 10^5$
- $1 \leq \mathbf{X}, \mathbf{Y} \leq \mathbf{N}$

Example

Input:
```
2
3 2
1 2
2 3
4 4
1 2
```

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

Output:

```
2
2
```

Explanation:
There are 2 test cases, first case has N=3, M=2 followed by 2 [X, Y] entries, second test case has N=4, M=4 followed by 4 [X, Y] entries.
In the first case, the shortest route to 3 is 1 - 2 - 3, so 2 connections are used.
In the second case, the shortest route to 4 is 1 – 2 – 4, so again 2 connections are used.

Tip 1:
You can test your code by using the following input files:
in3_1 should give result stored in file out3_1
in3_2 should give result stored in file out3_2
in3_3 should give result stored in file out3_3

Don't forget to test with your own test data. Passing examples files doesn't guarantee correctness of the algorithm.

Tip 2:
Don't spend too much time with standard input, use scanf.
Example for scanf of T:

```
int t;
scanf("%d", &t);
```

Example for scanf of 2 integers in the same line (like [X, Y] entries):

```
int x,y;
scanf("%d %d", &x, &y);
```

When you implement stdin in this way, you can run the test files e.g. like this:

program_executable < in3_1.

Assignment 2 (recursion):

Implement the following algorithm:

Find the depth of a tree with **N** elements.

For the choice of the algorithm, please look at theory lesson. We expect that you use recursion for this algorithm. You can implement the searching algorithm on your own or reuse an implementation from the web. In the latter case, don't forget to mention the references that you used.

Input Format

Input will be read from the standard input (stdin) and will have the following format:

First line of input contains number of tree elements **N**
Next **N** lines contain 3 space separated integers **X**, **Y**, **Z** where **X** is root, **Y** is left node of **X** and **Z** is right node of **X**

Output Format

Depth of the tree with the root node number 1.

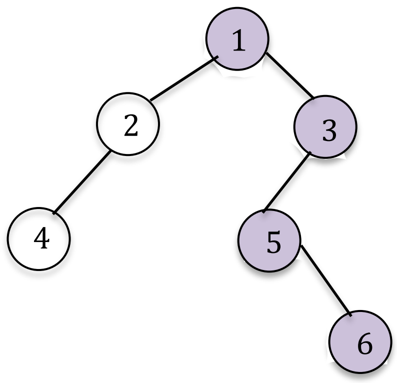
Constraints

- 0 < N < 20

Example

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

Explanation:



The tree defined in the example can be seen in the picture above. The depth of the tree is 4, the deepest path consists of nodes 1,3,5,6.

Tip:

You can test your code by using the following input files:
in4_1 should give result 4
in4_2 should give result 5
in4_3 should give result 6
in4_4 should give result 9

Again, don’t forget to test with your own test data. Passing examples files doesn’t guarantee correctness of the algorithm.