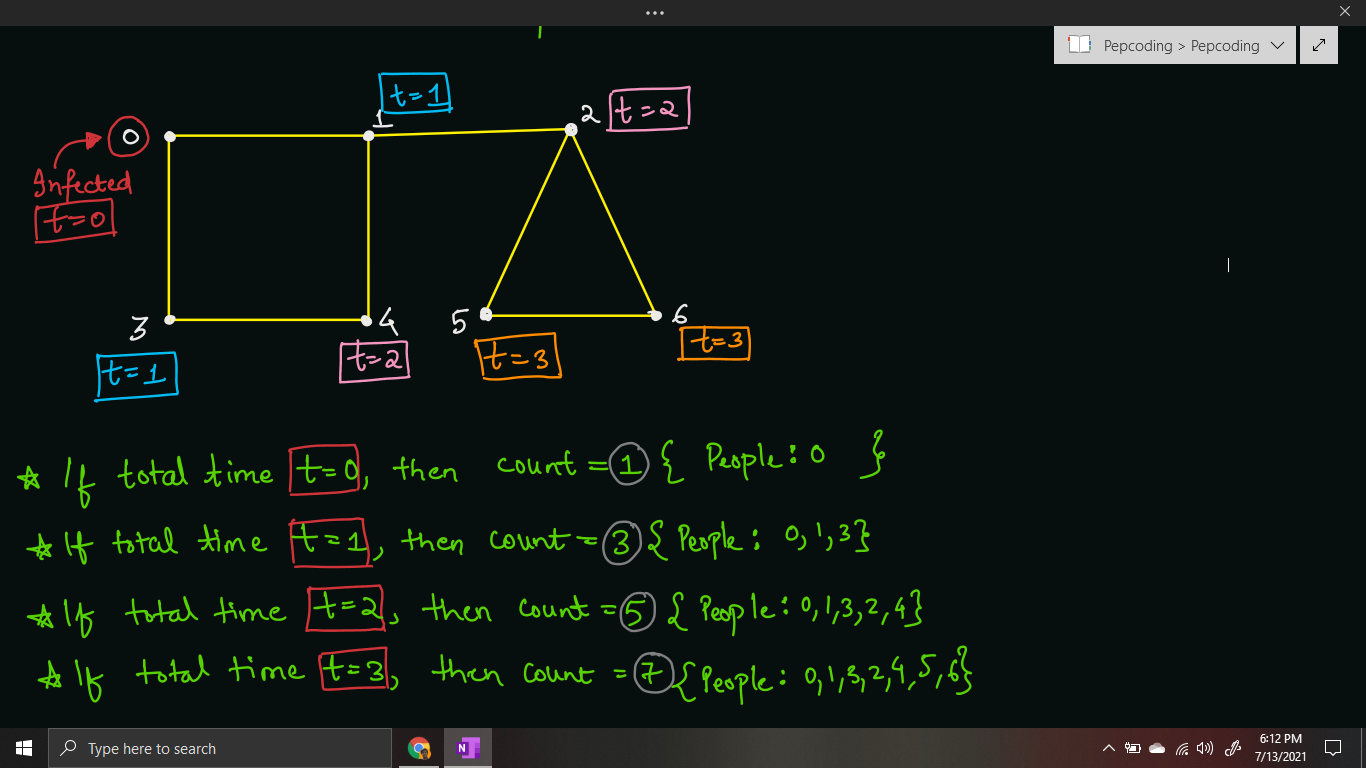
Dear reader, welcome to the article on the problem named **‘**[**Spread of Infection**](https://www.pepcoding.com/resources/online-java-foundation/graphs/infection-spread-official/ojquestion)**’.** It is a simple application of the BFS traversal algorithm.

***Problem Statement:***

* Given an undirected graph representing people and their connectivity. Vertices 0 to n-1 represent different people and edge (u, v) represents that there can be a direct contact between persons u and v.
* You are given a *src* person, who got infected by a disease at time t = 0.
* You are also given the total time t during which the infection can spread among the people.
* If a person is infected, then it takes 1 unit of time to make all of it’s connections get infected.
* You need to find the *count* of people who can get infected by the disease within the time t.

*Note*: Input is given in the form of adjacency list.

***Example:***

****

***Solution:***

* The idea is to use a BFS traversal algorithm with the source node as the *src*.
* Along with the source node’s value, we will also push the visiting time t = 0.
* We will maintain a ***count*** integer variable set to 0 initially, which will finally store the result, i.e. number of infected people within the total time of spread of infection.
* We will run the BFS, until the queue becomes empty (all nodes get visited), or the visiting time of the front element becomes greater than the total time given.
* If the node is already visited (cycle exists and the node was visited with a smaller visiting time), then we will simply pop and continue for the next element (do not visit it’s neighbours).
* Otherwise, we will mark the current node as visited. For each element, who's visiting time < total time, which gets removed (popped) from the queue, we will increment the count by 1.
* Also, we will break from the BFS traversal, if the front element’s visiting time is greater than the total time of spread.

***Note***: We can simply break because all the nodes after the front element present in the queue, will have visiting time greater than (or equal to) the visiting time of the front element. Since, the front element has time > t, hence all the elements after it will also have time > t.

* Now, we will push all the unvisited neighbours of the current node with visiting time one more than the visiting time of the current node. (If visiting time of src is x, then it’s neighbours will have visiting time x + 1).
* Finally, when the BFS algorithm is successfully completed, we can simply print the value of ***count***.

***Implementation***

*Note*: Before reading the Code, we recommend that you must try to come up with the solution on your own. Now, hoping that you have tried by yourself, here is the Java code.

import java.io.\*;

import java.util.\*;

public class Main {

static class Edge {

int src;

int nbr;

Edge(int src, int nbr) {

this.src = src;

this.nbr = nbr;

}

}

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int vtces = Integer.parseInt(br.readLine());

ArrayList<Edge>[] graph = new ArrayList[vtces];

for (int i = 0; i < vtces; i++) {

graph[i] = new ArrayList<>();

}

int edges = Integer.parseInt(br.readLine());

for (int i = 0; i < edges; i++) {

String[] parts = br.readLine().split(" ");

int v1 = Integer.parseInt(parts[0]);

int v2 = Integer.parseInt(parts[1]);

graph[v1].add(new Edge(v1, v2));

graph[v2].add(new Edge(v2, v1));

}

int src = Integer.parseInt(br.readLine());

int t = Integer.parseInt(br.readLine());

int count = 0;

ArrayDeque<Pair> queue = new ArrayDeque<>();

queue.add(new Pair(src, 1));

int[] visited = new int[vtces];

while(queue.size() > 0){

Pair rem = queue.remove();

if(visited[rem.v] > 0){

continue;

}

visited[rem.v] = rem.time;

if(rem.time > t){

break;

} else {

count++;

}

for (Edge e : graph[rem.v]) {

if (visited[e.nbr] == 0) {

queue.add(new Pair(e.nbr, rem.time + 1));

}

}

}

System.out.println(count);

}

static class Pair {

int v;

int time;

Pair(int v, int time){

this.v = v;

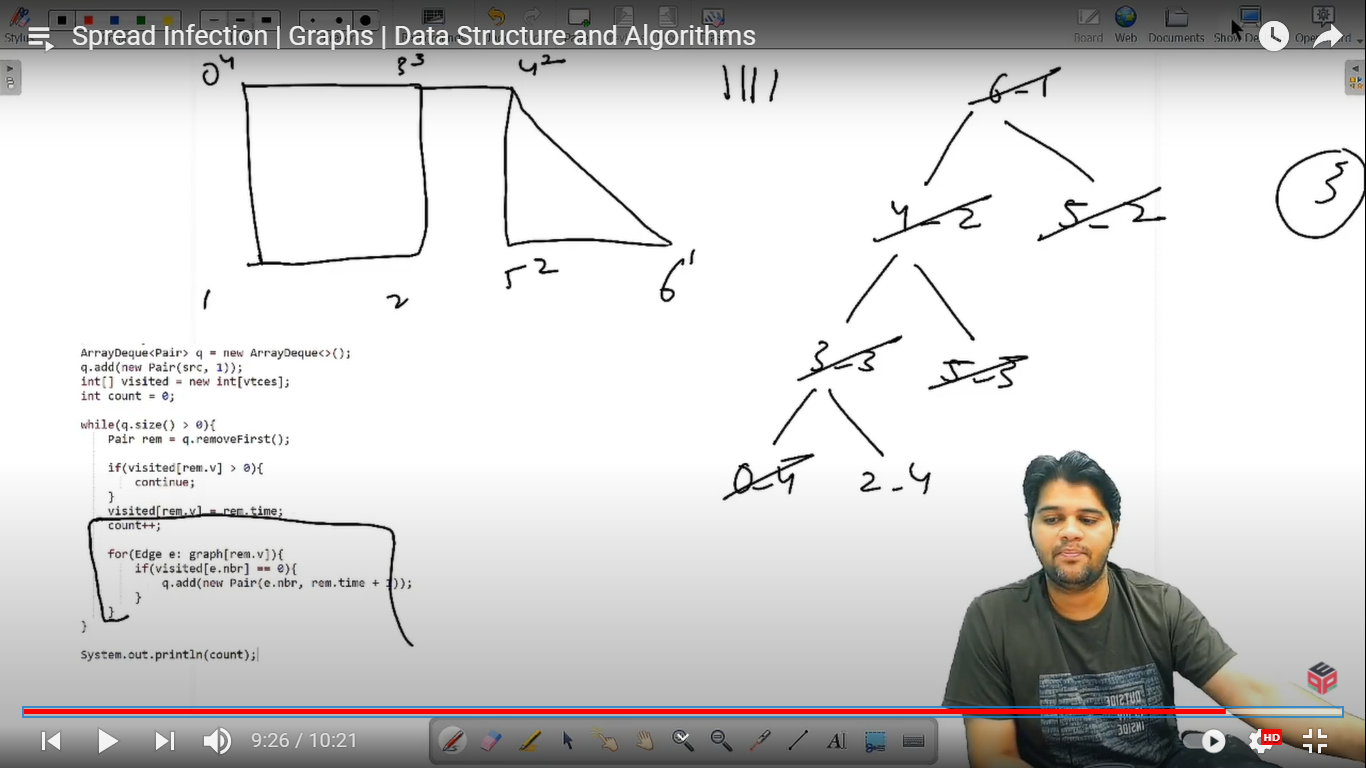
this.time = time;

}

}

}

This code is written and explained by our team in the [solution video](https://www.youtube.com/watch?v=3A1XJbRs_6A&list=TLGG5y3Na_oTBZ8xMzA3MjAyMQ). Do check it out to understand the concept completely.



* What is the ***time complexity*** of the above code?

We are simply doing a BFS traversal, hence the time complexity will be ***O(N + E)*** where N = number of vertices in the graph and E = number of edges in the graph.

* What is the ***space complexity*** of the above code ?

To perform BFS, we use queue data structure, which will take ***O(N)*** auxiliary space.

Please note that we are not taking into account the space taken to build the adjacency list, as it was given to us as an *input*.

**Follow Up:**

* In this problem, only one person was infected in the beginning (at time t = 0). What if there were more than one person who were infected initially?
* This variation is known as ‘**Rotten Oranges**’ and simply, we can start the bfs by pushing all the infected people (rotten oranges) in the queue with time = 0.

Hope that you liked the article on ***’Spread of Infection’***.

Subscribe to Pepcoding’s youtube channel for more such amazing video content on Data Structures & Algorithms.

You can suggest any improvements to the article on our telegram channel, or on the youtube channel’s comment section.

Article Contributed by: [Archit Aggarwal](https://www.linkedin.com/in/archit-aggarwal-6a7716189/)