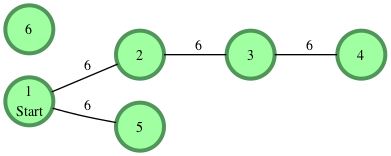
**BFS Shortest Reach in a Graph**

Consider an undirected graph consisting of n nodes where each node is labeled from 1 to n and the edge between any two nodes is always of length 6. We define node s to be the starting position for a BFS. Given a graph, determine the distances from the start node to each of its descendants and return the list in node number order, ascending. If a node is disconnected, it's distance should be -1.

For example, there are n=6 nodes in the graph with a starting node s=1. The list of edges={[1,2],[2,3],[3,4],[1,5]}, and each has a weight of 6.



Starting from node 1 and creating a list of distances, for nodes 2 through 6 we have distances=[6,12,18,6,-1].

**Function Description**

Define a Graph class with the required methods to return a list of distances.

**Input Format**

The first line contains an integer, q, the number of queries.

Each of the following q sets of lines is as follows:

* The first line contains two space-separated integers, n and m, the number of nodes and the number of edges.
* Each of the next m 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, the index of the starting node.

**Constraints**

* 1<=q<=10
* 2<=n<=1000
* 1<=m<= n\*(n-1)/2
* 1<= u,v,s<=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..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

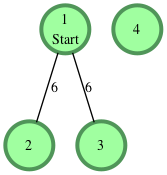
**Sample Output**

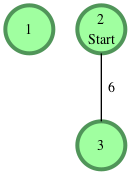
6 6 -1

-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 there is no connection to node 4.
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.

using System.CodeDom.Compiler;

using System.Collections.Generic;

using System.Collections;

using System.ComponentModel;

using System.Diagnostics.CodeAnalysis;

using System.Globalization;

using System.IO;

using System.Linq;

using System.Reflection;

using System.Runtime.Serialization;

using System.Text.RegularExpressions;

using System.Text;

using System;

public class Graph {

private readonly int V;

private readonly int[,] adjMatrix;

public Graph(int size) {

V = size+1;

adjMatrix = new int[V,V];

}

public void addEdge(int k, int l) {

adjMatrix[k,l] = 1;

adjMatrix[l,k] = 1;

}

public int VertexCount

{

get

{

return V;

}

}

public int[] shortestReach(int startId) {

//between 2 point default -1

int[] distances = Enumerable.Repeat(-1, V).ToArray();

Queue<int> queue = new Queue<int>();

queue.Enqueue(startId);

distances[startId] = 0;

while(queue.Count>0){

int v = queue.Dequeue();

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

if(adjMatrix[v,i] == 1 && distances[i] == -1 )

{

queue.Enqueue(i);

//if point is connected then plus 6

distances[i] = distances[v]+6;

}

}

}

int[] result = new int[V - 2];

int j = 0;

for(int i = 0; i < V; i++)

{

if(i!=0 && i != startId)

{

result[j++] = distances[i];

}

}

return result;

}

}

public class Solution {

static void Main(String[] args) {

/\* Enter your code here. Read input from STDIN. Print output to STDOUT. Your class should be named Solution \*/

TextWriter textWriter = new StreamWriter(@System.Environment.GetEnvironmentVariable("OUTPUT\_PATH"), true);

int q = Convert.ToInt32(Console.ReadLine());

int k,m,l,n;

for (int j=0;j<q;j++){

string[] node\_point1 = Console.ReadLine().Split(' ');

n=Convert.ToInt32(node\_point1[0]);

m=Convert.ToInt32(node\_point1[1]);

Graph graph = new Graph(n);

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

string[] node\_point = Console.ReadLine().Split(' ');

k = Convert.ToInt32(node\_point[0]);

l = Convert.ToInt32(node\_point[1]);

graph.addEdge(k,l);

}

int start = Convert.ToInt32(Console.ReadLine());

int[] result= graph.shortestReach(start);

//textWriter.WriteLine(result);

foreach( int i in result){

textWriter.Write( i+" ");

}

textWriter.WriteLine();

}

textWriter.Flush();

textWriter.Close();

}

}

**Congratulations**

You solved this challenge. Would you like to challenge your friends?

[Next Challenge](https://www.hackerrank.com/challenges/matrix?h_l=interview&playlist_slugs%5B%5D=interview-preparation-kit&playlist_slugs%5B%5D=graphs&h_r=next-challenge&h_v=zen)

* **Test case 0**
* **Test case 1**
* **Test case 2**
* **Test case 3**
* **Test case 4**
* **Test case 5**
* **Test case 6**
* **Test case 7**
* **Test case 8**

Compiler Message

**Success**

Input (stdin)

Download

* **2**
* **4 2**
* **1 2**
* **1 3**
* **1**
* **3 1**
* **2 3**
* **2**

Expected Output

**6 6 -1**

**-1 6**