



Full Name: Remi Chartier

Email: remipr.chartier@gmail.com

Test Name: Mock Test

Taken On: 30 Oct 2021 13:52:20 IST

Time Taken: 0 min 35 sec/ 28 min

Contact Number: +14084751573

Linkedin: http://www.linkedin.com/in/remichartier

Invited by: Ankush

Invited on: 30 Oct 2021 13:52:14 IST

Skills Score:

Tags Score:

0%

0/100

scored in **Mock Test** in 0 min 35 sec on 30 Oct 2021 13:52:20 IST

- Algorithms 0/100
- Core CS 0/100
- Graph Theory 0/100
- Medium 0/100
- problem-solving 0/100

Recruiter/Team Comments:

No Comments.

	Question Description	Time Taken	Score	Status
Q1	Breadth First Search: Shortest Reach > Coding	28 sec	0/ 100	⊗

QUESTION 1

⊗

Wrong Answer

Score 0

Breadth First Search: Shortest Reach > Coding

Graph Theory

Algorithms

Medium

problem-solving

Core CS


QUESTION DESCRIPTION

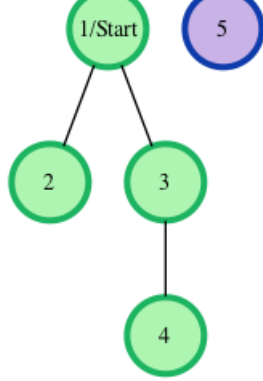
Consider an undirected graph where each edge weighs 6 units. Each of the nodes is labeled consecutively from 1 to n.

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). Return an array of distances from the start node in node number order. If a node is unreachable, return **−1** for that node.

Example

The following graph is based on the listed inputs:





```

n = 5 // number of nodes
m = 3 // number of edges
edges = [1, 2], [1, 3], [3, 4]
s = 1 // starting node

```

All distances are from the start 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. If a node is unreachable, its distance is **-1**.

`bfs` has the following parameter(s):

- `int n`: the number of nodes
- `int m`: the number of edges
- `int edges[m][2]`: start and end nodes for edges
- `int s`: the node to start traversals from

Returns

`int[n-1]`: the distances to nodes in increasing node number order, not including the start node (-1 if a node is not reachable)

### 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**, that describe an edge between nodes **u** and **v**.
- The last line contains a single integer, **s**, the node number to start from.

### Constraints

- $1 \leq q \leq 10$
- $2 \leq n \leq 1000$
- $1 \leq m \leq \frac{n \cdot (n-1)}{2}$
- $1 \leq u, v, s \leq n$

### Sample Input

```

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

```

### Sample Output

```

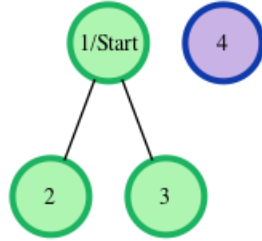
6 6 -1
-1 6

```

## Explanation

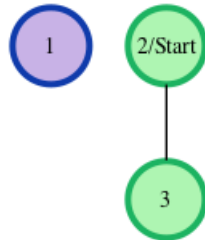
We perform the following two queries:

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 is not connected to). We then return an array of distances from node **1** to nodes **2**, **3**, and **4** (respectively):  $[6, 6, -1]$ .

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 return an array of distances from node **2** to nodes **1**, and **3** (respectively):  $[-1, 6]$ .

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

## CANDIDATE ANSWER


Language used: **Python 3**









```
1 #
2 # Complete the 'bfs' function below.
3 #
4 # The function is expected to return an INTEGER_ARRAY.
5 # The function accepts following parameters:
6 # 1. INTEGER n
7 # 2. INTEGER m
8 # 3. 2D_INTEGER_ARRAY edges
9 # 4. INTEGER s
10 #
11
12 class Node:
13     def __init__(self, value):
14         self.value = value
15         self.edges = []
16         self.seen = False
17
18     def add_edge(self, number):
19         self.edges.append(number)
20
21     def reset_seen(self):
22         self.seen = False
23
```

```

24     def set_seen(self):
25         self.seen = True
26
27 class Graph:
28     def __init__(self):
29         self.nodes = [Node(0)]
30
31     def reset_seen(self):
32         for n in self.nodes:
33             n.reset_seen()
34
35     # start = start node number (1 to n)
36     # target = searched node number (1 to n)
37     def find_bfs(start, target, g):
38         curr = g.nodes[start]
39         links = []
40         if target == curr.value:
41             return 0
42         curr.set_seen()
43         #print(f'node {start} current edges {curr.edges}')
44         for e in curr.edges:
45             #if g.nodes[e].value != start:
46             if g.nodes[e].seen == False:
47                 # dist += find_bfs(g.nodes[e].value, target, g)
48                 # stack the neighbors
49                 links.append(g.nodes[e].value)
50         for l in links:
51             #print(f"find_bfs({l}, {target}, g)")
52             d = find_bfs(l, target, g)
53             #print(f"{d} = find_bfs({l}, {target}, g)")
54             if d != -1:
55                 return 6 + d
56         return -1
57
58
59 def bfs(n, m, edges, s):
60     # Write your code here
61     #returns int[n-1] distances to nodes in increasing node number order,
62     # not including the start node
63     g = Graph()
64     # create nodes, add them to the graph
65     for i in range(1,n+1):
66         g.nodes.append(Node(i))
67     # fill list of edges on the nodes
68     for e in edges:
69         g.nodes[e[0]].add_edge(e[1])
70         #g.nodes[e[1]].add_edge(e[0])
71     # Now the Graph is filled.
72     # We should now be able to implement BFS ...
73     # build a list of distances from node s
74     output = []
75     for i in range(1, n+1):
76         if i != s:
77             g.reset_seen()
78             output.append(find_bfs(s, i, g))
79     #print(output)
80     return output
81
82

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 1	Easy	Sample	 Success	0	0.0301 sec	0.5 KB

Testcase 1	Easy	Sample case	 Success	0	0.0391 sec	9.9 KB
Testcase 2	Medium	Hidden case	 Wrong Answer	0	0.0748 sec	9.79 KB
Testcase 3	Medium	Hidden case	 Wrong Answer	0	0.635 sec	12.7 KB
Testcase 4	Hard	Hidden case	 Wrong Answer	0	0.0435 sec	9.49 KB
Testcase 5	Hard	Hidden case	 Wrong Answer	0	0.0549 sec	9.93 KB
Testcase 6	Hard	Hidden case	 Terminated due to timeout	0	10.0072 sec	18.6 KB
Testcase 7	Hard	Hidden case	 Wrong Answer	0	0.276 sec	10.6 KB
Testcase 8	Easy	Sample case	 Success	0	0.0455 sec	9.51 KB

No Comments

PDF generated at: 30 Oct 2021 08:24:51 UTC