Full Name:	NGUYEN DANG VINH
Email:	dangvinhprovn@gmail.com
Test Name:	**Mock Test**
Taken On:	25 Feb 2024 14:57:03 IST
Time Taken:	8 min 10 sec/ 105 min
Invited by:	Ankush
Invited on:	25 Feb 2024 14:56:52 IST
Skills Score:	
Tags Score:	Algorithms 255/255 Core CS 255/255 Data Structures 60/60 Disjoint Set 60/60 Graph Theory 100/100 Medium 195/195 Search 95/95 problem-solving 195/195

100%

255/255

scored in **Mock Test** in 8 min 10 sec on 25 Feb 2024 14:57:03 IST

Recruiter/Team Comments:

No Comments.

	Question Description	Time Taken	Score	Status
Q1	Breadth First Search: Shortest Reach > **Coding**	3 min 15 sec	100/ 100	✓
Q2	Components in a graph > **Coding**	1 min 52 sec	60/ 60	✓
Q3	Cut the Tree > **Coding**	2 min 56 sec	95/ 95	✓

QUESTION 1

✓

Correct Answer

Score 100

Breadth First Search: Shortest Reach > Coding

Graph Theory | Algorithms | Medium | problem-solving | Core CS

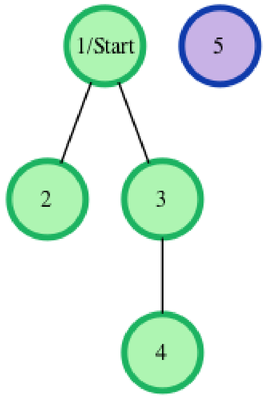
QUESTION DESCRIPTION

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
```

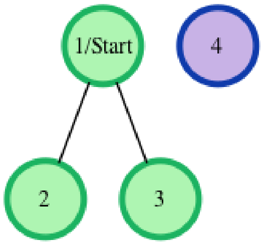
Sample Output

```
6 6 -1
-1 6
```

Explanation

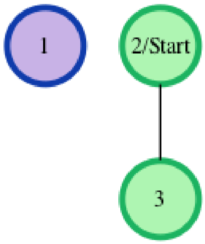
We perform the following two queries:

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: **JavaScript (Node.js)**

```

/*
 * Complete the 'bfs' function below.
 *
 * The function is expected to return an INTEGER_ARRAY.
 * The function accepts following parameters:
 *  1. INTEGER n
 *  2. INTEGER m
 *  3. 2D_INTEGER_ARRAY edges
 *  4. INTEGER s
 */

/**
 *
 * @param {number} numNodes
 * @param {number} numEdges
 * @param {number[][]} edges
 * @param {number} startNodeIdx
 * @returns {number[]}
 */
function bfs(numNodes, numEdges, edges, startNodeIdx) {
  /** @type { Object<number, Set<number>> } */
  const graph = {};

  for (const edge of edges) {
    const [node_1, node_2] = edge;
    if (!graph[node_1]) graph[node_1] = new Set();
    if (!graph[node_2]) graph[node_2] = new Set();
    graph[node_1].add(node_2);
    graph[node_2].add(node_1);
  }

  if (!graph[startNodeIdx]) return new Array(numNodes - 1).fill(-1);

  const distancesMap = {};
  distancesMap[startNodeIdx] = 0;
  const queue = [startNodeIdx];
  while (queue.length) {
    const parentNode = queue.shift();
    const distances = distancesMap[parentNode] + 6;

    for (const neighbor of graph[parentNode] ?? []) {
      if (distancesMap[neighbor]) continue;
      else {
        distancesMap[neighbor] = distances;
        queue.push(neighbor);
      }
    }
  }

  const distances = [];
  for (let idx = 1; idx <= numNodes; idx++) {
    if (idx === startNodeIdx) continue;
    const distancesValue = distancesMap[idx] ?? -1;
    distances.push(distancesValue);
  }
  return distances;
}

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 1	Easy	Sample case	✓ Success	0	0.0312 sec	42 KB
Testcase 2	Medium	Hidden case	✓ Success	5	0.0408 sec	43.6 KB
Testcase 3	Medium	Hidden case	✓ Success	5	0.1126 sec	60.2 KB
Testcase 4	Hard	Hidden case	✓ Success	15	0.0416 sec	42 KB
Testcase 5	Hard	Hidden case	✓ Success	15	0.0413 sec	43.9 KB
Testcase 6	Hard	Hidden case	✓ Success	30	0.4053 sec	90.6 KB
Testcase 7	Hard	Hidden case	✓ Success	30	0.0516 sec	52.2 KB
Testcase 8	Easy	Sample case	✓ Success	0	0.0368 sec	42 KB

No Comments

QUESTION 2

Algorithms | Data Structures | Disjoint Set | Core CS

Algorithms | Data Structures | Disjoint Set | Core CS

✓ Correct Answer

Score 60

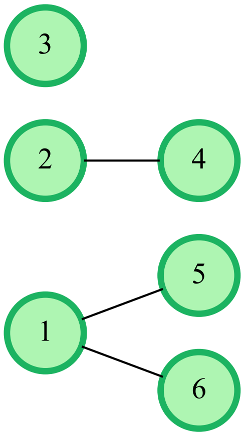
Components in a graph › Coding

QUESTION DESCRIPTION

There are $2 \times N$ nodes in an undirected graph, and a number of edges connecting some nodes. In each edge, the first value will be between 1 and N, inclusive. The second node will be between $N + 1$ and $2 \times N$, inclusive. Given a list of edges, determine the size of the smallest and largest connected components that have 2 or more nodes. A node can have any number of connections. The highest node value will always be connected to at least 1 other node.

Note Single nodes should not be considered in the answer.

Example
$bg = [[1, 5], [1, 6], [2, 4]]$



The smaller component contains 2 nodes and the larger contains 3. Return the array $[2, 3]$.

Function Description
Complete the *connectedComponents* function in the editor below.

connectedComponents has the following parameter(s):
- *int bg[n][2]:* a 2-d array of integers that represent node ends of graph edges

Returns
- *int[2]:* an array with 2 integers, the smallest and largest component sizes

Input Format

The first line contains an integer n, the size of bg.
Each of the next n lines contain two space-separated integers, $bg[i][0]$ and $bg[i][1]$.

Constraints

- $1 \leq numberofnodesN \leq 15000$
- $1 \leq bg[i][0] \leq N$
- $N + 1 \leq bg[i][1] \leq 2N$

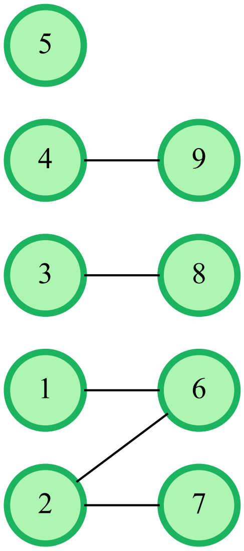
Sample Input

```
STDIN   Function
-----   --------
5       bg[] size n = 5
1 6     bg = [[1, 6],[2, 7], [3, 8], [4,9], [2, 6]]
2 7
3 8
4 9
2 6
```

Sample Output

```
2 4
```

Explanation



Since the component with node 5 contains only one node, it is not considered.

The number of vertices in the smallest connected component in the graph is 2 based on either $(3, 8)$ or $(4, 9)$.

The number of vertices in the largest connected component in the graph is 4 i.e. $1 - 2 - 6 - 7$.

CANDIDATE ANSWER

Language used: **JavaScript (Node.js)**

```

/*
 * Complete the 'componentsInGraph' function below.
 *
 * The function is expected to return an INTEGER_ARRAY.
 * The function accepts 2D_INTEGER_ARRAY gb as parameter.
 */

// @ts-ignore
class DisjSet {
  /** @type { Object<number, {parent: number[], size?: number}> } */
  data = {};
  /** @type {Set<number>} */
  rootList = new Set();

  /**
   *
   * @param {number} value
   * @param {number[]} parentPointer
   * @returns {number}
   */
  find(value, parentPointer = undefined) {
    let parentValue = value;
    const stack = [value];

    if (!this.data[value]) {
      this.data[value] = {
        parent: parentPointer || [value],
      };
      return value;
    }

    let i = 0;
    while (i < stack.length) {
      const node = stack[i];
      if (this.data[node].parent[0] === node) {
        parentValue = node;
        break;
      }
      stack.push(this.data[node].parent[0]);
      i++;
    }

    for (const node of stack) {
      this.data[node].parent[0] = parentValue;
      this.data[node].parent = parentPointer;
    }

    return parentValue;
  }

  /**
   *
   * @param {number} fristNode
   * @param {number} sencondNode
   */
  union(fristNode, sencondNode) {
    const parentPointer = this.data[fristNode]?.parent ?? [fristNode];
    const fristParentNode = this.find(fristNode, parentPointer);
    const sencondParentNode = this.find(sencondNode, parentPointer);

    if (fristParentNode !== sencondParentNode) {
      if (!this.data[fristParentNode].size) this.data[fristParentNode].size =
1;
      this.data[fristParentNode].size +=
        this.data[sencondParentNode]?.size ?? 1;

      this.rootList.add(fristParentNode);
      this.rootList.delete(sencondParentNode);
    }
  }
}

/**
 *
 * @param {number[][]} graphEdges
 * @returns {number[]}
 */
function componentsInGraph(graphEdges) {
  const disjSet = new DisjSet();

  for (const edge of graphEdges) {
    const [fristNode, sencondNode] = edge;
    disjSet.union(fristNode, sencondNode);
  }

  let minComponentSizes = Number.MAX_SAFE_INTEGER;
  let maxComponentSizes = Number.MIN_SAFE_INTEGER;
  for (const root of disjSet.rootList) {
    if (disjSet.data[root].size < minComponentSizes) {
      minComponentSizes = disjSet.data[root].size;
    }
    if (disjSet.data[root].size > maxComponentSizes) {
      maxComponentSizes = disjSet.data[root].size;
    }
  }

  return [minComponentSizes, maxComponentSizes];
}
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 1	Medium	Hidden case	✓ Success	0	0.0368 sec	41.9 KB
Testcase 2	Medium	Hidden case	✓ Success	0	0.0363 sec	42.1 KB
Testcase 3	Medium	Hidden case	✓ Success	0	0.0481 sec	47.3 KB
Testcase 4	Medium	Hidden case	✓ Success	0	0.0398 sec	47.4 KB
Testcase 5	Medium	Hidden case	✓ Success	0	0.0465 sec	47.5 KB
Testcase 6	Medium	Hidden case	✓ Success	0	0.0498 sec	47.2 KB
Testcase 7	Medium	Hidden case	✓ Success	0	0.0615 sec	49.4 KB
Testcase 8	Medium	Hidden case	✓ Success	0	0.0416 sec	49.4 KB
Testcase 9	Medium	Hidden case	✓ Success	0	0.0464 sec	49.5 KB
Testcase 10	Medium	Hidden case	✓ Success	0	0.0555 sec	51 KB
Testcase 11	Medium	Hidden case	✓ Success	0	0.0534 sec	50.6 KB
Testcase 12	Medium	Hidden case	✓ Success	0	0.0629 sec	53.4 KB
Testcase 13	Medium	Hidden case	✓ Success	0	0.059 sec	53.2 KB
Testcase 14	Medium	Hidden case	✓ Success	0	0.0868 sec	52.8 KB
Testcase 15	Medium	Hidden case	✓ Success	0	0.0646 sec	52.5 KB
Testcase 16	Medium	Hidden case	✓ Success	0	0.0636 sec	53.3 KB
Testcase 17	Medium	Hidden case	✓ Success	0	0.0339 sec	43.6 KB
Testcase 18	Medium	Hidden case	✓ Success	0	0.0629 sec	53 KB
Testcase 19	Easy	Sample case	✓ Success	0	0.0326 sec	41.8 KB
Testcase 20	Medium	Hidden case	✓ Success	0	0.1135 sec	59.2 KB
Testcase 21	Medium	Hidden case	✓ Success	0	0.1006 sec	59.4 KB
Testcase 22	Medium	Hidden case	✓ Success	0	0.0887 sec	59.8 KB
Testcase 23	Medium	Hidden case	✓ Success	0	0.0994 sec	59.7 KB
Testcase 24	Medium	Hidden case	✓ Success	0	0.0869 sec	60.8 KB
Testcase 25	Medium	Hidden case	✓ Success	0	0.081 sec	58.5 KB
Testcase 26	Medium	Hidden case	✓ Success	0	0.0905 sec	58.5 KB
Testcase 27	Medium	Hidden case	✓ Success	0	0.0937 sec	58.3 KB
Testcase 28	Medium	Hidden case	✓ Success	0	0.0891 sec	58.8 KB
Testcase 29	Medium	Hidden case	✓ Success	0	0.0975 sec	59.3 KB
Testcase 30	Medium	Hidden case	✓ Success	0	0.0865 sec	59.2 KB
Testcase 31	Medium	Hidden case	✓ Success	0	0.0737 sec	59.6 KB
Testcase 32	Medium	Hidden case	✓ Success	0	0.1317 sec	59.1 KB
Testcase 33	Medium	Hidden case	✓ Success	0	0.077 sec	54 KB
Testcase 34	Hard	Hidden case	✓ Success	10	0.0748 sec	57 KB

Testcase 35	Hard	Hidden case	✓ Success	10	0.0778 sec	58 KB
Testcase 36	Hard	Hidden case	✓ Success	10	0.0685 sec	57.3 KB
Testcase 37	Hard	Hidden case	✓ Success	10	0.0808 sec	57.3 KB
Testcase 38	Hard	Hidden case	✓ Success	10	0.0761 sec	58.3 KB
Testcase 39	Hard	Hidden case	✓ Success	10	0.0978 sec	56.9 KB

No Comments

QUESTION 3

✓

Correct Answer

Score 95

Cut the Tree › Coding

Search | Algorithms | Medium | problem-solving | Core CS

QUESTION DESCRIPTION

There is an undirected tree where each vertex is numbered from 1 to n, and each contains a data value. The *sum* of a tree is the sum of all its nodes' data values. If an edge is cut, two smaller trees are formed. The *difference* between two trees is the absolute value of the difference in their sums.

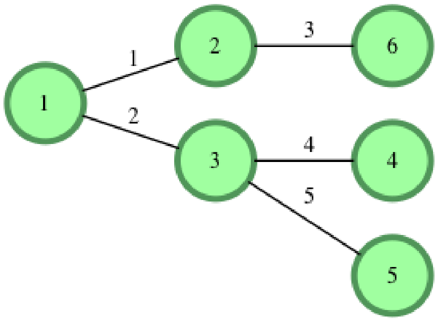
Given a tree, determine which edge to cut so that the resulting trees have a minimal *difference* between them, then return that difference.

Example

$data = [1, 2, 3, 4, 5, 6]$

$edges = [(1, 2), (1, 3), (2, 6), (3, 4), (3, 5)]$

In this case, node numbers match their weights for convenience. The graph is shown below.



The values are calculated as follows:

```
Edge    Tree 1  Tree 2  Absolute
Cut     Sum      Sum     Difference
1        8        13          5
2        9        12          3
3        6        15          9
4        4        17         13
5        5        16         11
```

The minimum absolute difference is 3.

Note: The given tree is *always* rooted at vertex 1.

Function Description

Complete the *cutTheTree* function in the editor below.

cutTheTree has the following parameter(s):

- *int data[n]:* an array of integers that represent node values
- *int edges[n-1][2]:* an 2 dimensional array of integer pairs where each pair represents nodes connected by the edge

Returns

- *int:* the minimum achievable absolute difference of tree sums

Input Format

The first line contains an integer n, the number of vertices in the tree.
The second line contains n space-separated integers, where each integer u denotes the $node[u]$ data value, $data[u]$.
Each of the $n-1$ subsequent lines contains two space-separated integers u and v that describe edge $u \leftrightarrow v$ in tree t.

Constraints

- $3 \le n \le 10^5$
- $1 \le data[u] \le 1001$, where $1 \le u \le n$.

Sample Input

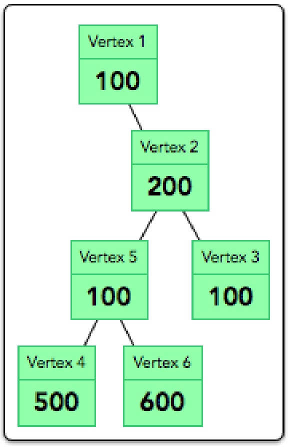
```
STDIN                        Function
-----                        --------
6                            data[] size n = 6
100 200 100 500 100 600      data = [100, 200, 100, 500, 100, 600]
1 2                          edges = [[1, 2], [2, 3], [2, 5], [4, 5], [5,
6]]
2 3
2 5
4 5
5 6
```

Sample Output

```
400
```

Explanation

We can visualize the initial, uncut tree as:



There are $n - 1 = 5$ edges we can cut:

1. Edge $1 \leftrightarrow 2$ results in $d_{1\leftrightarrow 2} = 1500 - 100 = 1400$
2. Edge $2 \leftrightarrow 3$ results in $d_{2\leftrightarrow 3} = 1500 - 100 = 1400$
3. Edge $2 \leftrightarrow 5$ results in $d_{2\leftrightarrow 5} = 1200 - 400 = 800$
4. Edge $4 \leftrightarrow 5$ results in $d_{4\leftrightarrow 5} = 1100 - 500 = 600$
5. Edge $5 \leftrightarrow 6$ results in $d_{5\leftrightarrow 6} = 1000 - 600 = 400$

The minimum *difference* is 400.

CANDIDATE ANSWER

Language used: **JavaScript (Node.js)**

```
/*
 * Complete the 'cutTheTree' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts following parameters:
 *  1. INTEGER_ARRAY data
 *  2. 2D_INTEGER_ARRAY edges
 */

/**
 *
 * @param {number[]} data
 * @param {number[][]} edges
 * @returns {number}
 */
function cutTheTree(data, edges) {
  let minAchievable = Number.MAX_SAFE_INTEGER;
  /** @type {number[][]} */
  const graph = [];
  /** @type {number[]} */
  const sum = [];

  const nodeCut = new Set();
  for (const edge of edges) {
    let [fristNode, sencondNode] = edge;
    fristNode--;
    sencondNode--;
    graph[fristNode]
      ? graph[fristNode].push(sencondNode)
      : (graph[fristNode] = [sencondNode]);
    graph[sencondNode]
      ? graph[sencondNode].push(fristNode)
      : (graph[sencondNode] = [fristNode]);
    sencondNode !== 0 && nodeCut.add(sencondNode);
  }

  /**
   *
   * @param {number} node
   * @param {number} parent
   */
  const dfs = (node, parent) => {
    /** @type {number[][]} */
    const stack = [[node, parent]];
    /** @type {number[][]} */
    const traveledPath = [];

    while (stack.length) {
      const [currentNode, parentNode] = stack.pop();
      traveledPath.push([currentNode, parentNode]);
      if (sum[currentNode] === undefined) sum[currentNode] =
data[currentNode];

      for (const childNode of graph[currentNode]) {
        if (childNode !== parentNode) {
          stack.push([childNode, currentNode]);
        }
      }
    }

    while (traveledPath.length > 1) {
      const [childNode, parentNode] = traveledPath.pop();
      sum[parentNode] += sum[childNode];
    }
  };

  dfs(0, 0);
  const total = sum[0];
  for (const node of nodeCut) {
    const sumNodeTree_1 = total - sum[node];
    const diff = Math.abs(sumNodeTree_1 - sum[node]);
    minAchievable = Math.min(minAchievable, diff);
  }

  return minAchievable;
}
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 1	Easy	Sample case	✓ Success	0	0.0338 sec	42 KB
Testcase 2	Hard	Hidden case	✓ Success	5	0.0502 sec	42.2 KB
Testcase 3	Hard	Hidden case	✓ Success	5	0.0381 sec	42.2 KB
Testcase 4	Hard	Hidden case	✓ Success	5	0.0447 sec	42.4 KB
Testcase 5	Easy	Sample case	✓ Success	0	0.033 sec	42 KB
Testcase 6	Hard	Hidden case	✓ Success	5	0.0743 sec	57.8 KB
Testcase 7	Hard	Hidden case	✓ Success	10	0.2877 sec	111 KB
Testcase 8	Hard	Hidden case	✓ Success	5	0.3304 sec	112 KB
Testcase 9	Hard	Hidden case	✓ Success	5	0.277 sec	112 KB
Testcase 10	Hard	Hidden case	✓ Success	5	0.2693 sec	112 KB
Testcase 11	Hard	Hidden case	✓ Success	5	0.3139 sec	111 KB
Testcase 12	Hard	Hidden case	✓ Success	5	0.3243 sec	111 KB
Testcase 13	Medium	Hidden case	✓ Success	5	0.2651 sec	112 KB
Testcase 14	Medium	Hidden case	✓ Success	5	0.2734 sec	111 KB
Testcase 15	Medium	Hidden case	✓ Success	5	0.3101 sec	113 KB
Testcase 16	Medium	Hidden case	✓ Success	5	0.2988 sec	112 KB
Testcase 17	Medium	Hidden case	✓ Success	5	0.3897 sec	111 KB
Testcase 18	Medium	Hidden case	✓ Success	5	0.2968 sec	113 KB
Testcase 19	Medium	Hidden case	✓ Success	5	0.2733 sec	112 KB
Testcase 20	Medium	Hidden case	✓ Success	5	0.3375 sec	111 KB

No Comments

PDF generated at: 25 Feb 2024 09:36:53 UTC