Mock Test > sammetamanogna82@gmail.com

Full Name:	Sammeta Manogna
Email:	sammetamanogna82@gmail.com
Test Name:	**Mock Test**
Taken On:	4 Sep 2025 21:44:08 IST
Time Taken:	45 min 19 sec/ 105 min
Invited by:	Ankush
Invited on:	4 Sep 2025 21:43:51 IST
Skills Score:	
Tags Score:	Algorithms 175/255
	Core CS 175/255
	Data Structures 60/60
	Disjoint Set 60/60
	Graph Theory 100/100
	Medium 115/195
	Search 15/95
	problem-solving 115/195

68.6%

175/255

scored in **Mock Test** in 45 min 19 sec on 4 Sep 2025 21:44:08 IST

Recruiter/Team Comments:

No Comments.

Plagiarism flagged

We have marked questions with suspected plagiarism below. Please review it in detail here -

	Question Description	Time Taken	Score	Status
Q1	Breadth First Search: Shortest Reach > **Coding**	27 min 33 sec	100/ 100	!
Q2	Components in a graph > **Coding**	9 min 27 sec	60/ 60	!
Q3	Cut the Tree > **Coding**	8 min 8 sec	15/ 95	!

QUESTION 1

!

Needs Review

Score 100

Breadth First Search: Shortest Reach > Coding

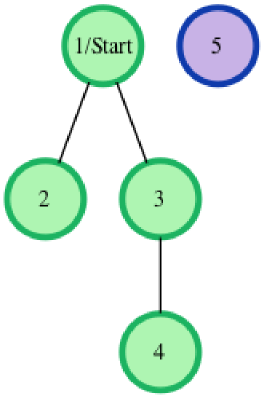
Graph Theory Algorithms Medium problem-solving Core CS

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
```

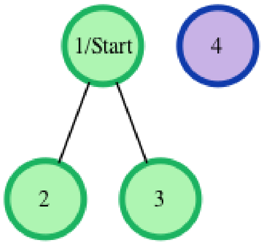
Sample Output

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

Explanation

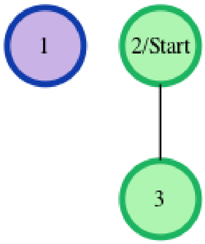
We perform the following two queries:

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: **Python 3**

```
#
# 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
#
from collections import deque, defaultdict
```

```python
def bfs(n, m, edges, s):
    # Write your code here
    graph = defaultdict(list)
    for u, v in edges:
        graph[u].append(v)
        graph[v].append(u)

    dist = [-1] * (n + 1)
    dist[s] = 0

    queue = deque([s])
    while queue:
        node = queue.popleft()
        for neighbor in graph[node]:
            if dist[neighbor] == -1:
                dist[neighbor] = dist[node] + 6
                queue.append(neighbor)

    result = []
    for i in range(1, n + 1):
        if i != s:
            result.append(dist[i])
    return result
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 1	Easy	Sample case	✓ Success	0	0.0372 sec	10 KB
Testcase 2	Medium	Hidden case	✓ Success	5	0.0298 sec	10.6 KB
Testcase 3	Medium	Hidden case	✓ Success	5	0.1118 sec	14 KB
Testcase 4	Hard	Hidden case	✓ Success	15	0.0281 sec	10.1 KB
Testcase 5	Hard	Hidden case	✓ Success	15	0.0306 sec	10.5 KB
Testcase 6	Hard	Hidden case	✓ Success	30	0.5223 sec	27.7 KB
Testcase 7	Hard	Hidden case	✓ Success	30	0.0447 sec	11.4 KB
Testcase 8	Easy	Sample case	✓ Success	0	0.0304 sec	10 KB

No Comments

QUESTION 2

! Needs Review

Score 60

Components in a graph › Coding

Algorithms | Data Structures | Disjoint Set | Core CS

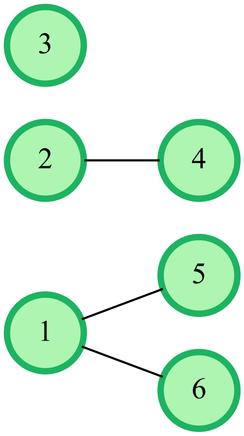
QUESTION DESCRIPTION

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

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

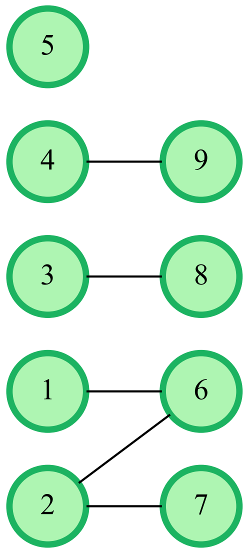
Sample Input

```
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: **Python 3**

```

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

def componentsInGraph(gb):
    # Write your code here
    from collections import defaultdict, deque
    graph = defaultdict(list)
    nodes = set()
    for u, v in gb:
        graph[u].append(v)
        graph[v].append(u)
        nodes.add(u)
        nodes.add(v)
    visited = set()
    component_sizes = []

    for node in nodes:
        if node not in visited:
            queue = deque([node])
            visited.add(node)
            size = 1
            while queue:
                curr = queue.popleft()
                for neighbor in graph[curr]:
                    if neighbor not in visited:
                        visited.add(neighbor)
                        queue.append(neighbor)
                        size += 1
            if size > 1:
                component_sizes.append(size)
    return [min(component_sizes), max(component_sizes)]

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 1	Medium	Hidden case	✓ Success	0	0.0264 sec	10.3 KB
Testcase 2	Medium	Hidden case	✓ Success	0	0.0279 sec	10 KB
Testcase 3	Medium	Hidden case	✓ Success	0	0.0352 sec	10.6 KB
Testcase 4	Medium	Hidden case	✓ Success	0	0.0277 sec	10.4 KB
Testcase 5	Medium	Hidden case	✓ Success	0	0.0318 sec	10.4 KB
Testcase 6	Medium	Hidden case	✓ Success	0	0.0275 sec	10.5 KB
Testcase 7	Medium	Hidden case	✓ Success	0	0.0364 sec	11 KB
Testcase 8	Medium	Hidden case	✓ Success	0	0.0457 sec	11.1 KB
Testcase 9	Medium	Hidden case	✓ Success	0	0.0354 sec	11.3 KB
Testcase 10	Medium	Hidden case	✓ Success	0	0.0327 sec	11.5 KB
Testcase 11	Medium	Hidden case	✓ Success	0	0.0369 sec	11.4 KB
Testcase 12	Medium	Hidden case	✓ Success	0	0.0448 sec	13.4 KB
Testcase 13	Medium	Hidden case	✓ Success	0	0.0549 sec	13.4 KB
Testcase 14	Medium	Hidden case	✓ Success	0	0.0394 sec	13 KB
Testcase 15	Medium	Hidden case	✓ Success	0	0.0412 sec	12 KB
Testcase 16	Medium	Hidden case	✓ Success	0	0.0548 sec	12.8 KB
Testcase 17	Medium	Hidden case	✓ Success	0	0.0495 sec	10.5 KB
Testcase 18	Medium	Hidden case	✓ Success	0	0.0436 sec	12.1 KB
Testcase 19	Easy	Sample case	✓ Success	0	0.0275 sec	10.3 KB
Testcase 20	Medium	Hidden case	✓ Success	0	0.0753 sec	16.4 KB
Testcase 21	Medium	Hidden case	✓ Success	0	0.0722 sec	16.4 KB
Testcase 22	Medium	Hidden case	✓ Success	0	0.0916 sec	16.5 KB
Testcase 23	Medium	Hidden case	✓ Success	0	0.0712 sec	16.5 KB
Testcase 24	Medium	Hidden case	✓ Success	0	0.0927 sec	16.4 KB
Testcase 25	Medium	Hidden case	✓ Success	0	0.0833 sec	16.5 KB
Testcase 26	Medium	Hidden case	✓ Success	0	0.1304 sec	16.3 KB
Testcase 27	Medium	Hidden case	✓ Success	0	0.0764 sec	16.4 KB
Testcase 28	Medium	Hidden case	✓ Success	0	0.0699 sec	16.4 KB
Testcase 29	Medium	Hidden case	✓ Success	0	0.0701 sec	16.4 KB
Testcase 30	Medium	Hidden case	✓ Success	0	0.0731 sec	16.5 KB
Testcase 31	Medium	Hidden case	✓ Success	0	0.1149 sec	16.5 KB
Testcase 32	Medium	Hidden case	✓ Success	0	0.0823 sec	16.4 KB
Testcase 33	Medium	Hidden case	✓ Success	0	0.0663 sec	15.8 KB
Testcase 34	Hard	Hidden case	✓ Success	10	0.0636 sec	16.3 KB
Testcase 35	Hard	Hidden case	✓ Success	10	0.0716 sec	16.3 KB
Testcase 36	Hard	Hidden case	✓ Success	10	0.0699 sec	16.1 KB

Testcase 37	Hard	Hidden case	✓ Success	10	0.0859 sec	16.3 KB
Testcase 38	Hard	Hidden case	✓ Success	10	0.0677 sec	16.3 KB
Testcase 39	Hard	Hidden case	✓ Success	10	0.0681 sec	16.3 KB

No Comments

QUESTION 3

Needs Review

Score 15

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.

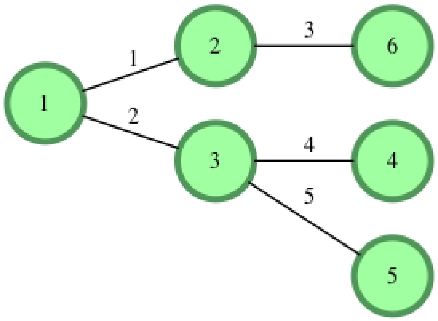
Given a tree, determine which edge to cut so that the resulting trees have a minimal *difference* between them, then return that difference.

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

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

Sample Input

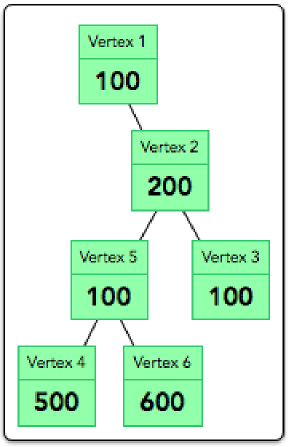
```
STDIN                           Function
-----                           --------
6                               data[] size n = 6
100 200 100 500 100 600         data = [100, 200, 100, 500, 100, 600]
1 2                             edges = [[1, 2], [2, 3], [2, 5], [4, 5], [5,
6]]
2 3
2 5
4 5
5 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 $d_{5\leftrightarrow 6} = 1000 - 600 = 400$

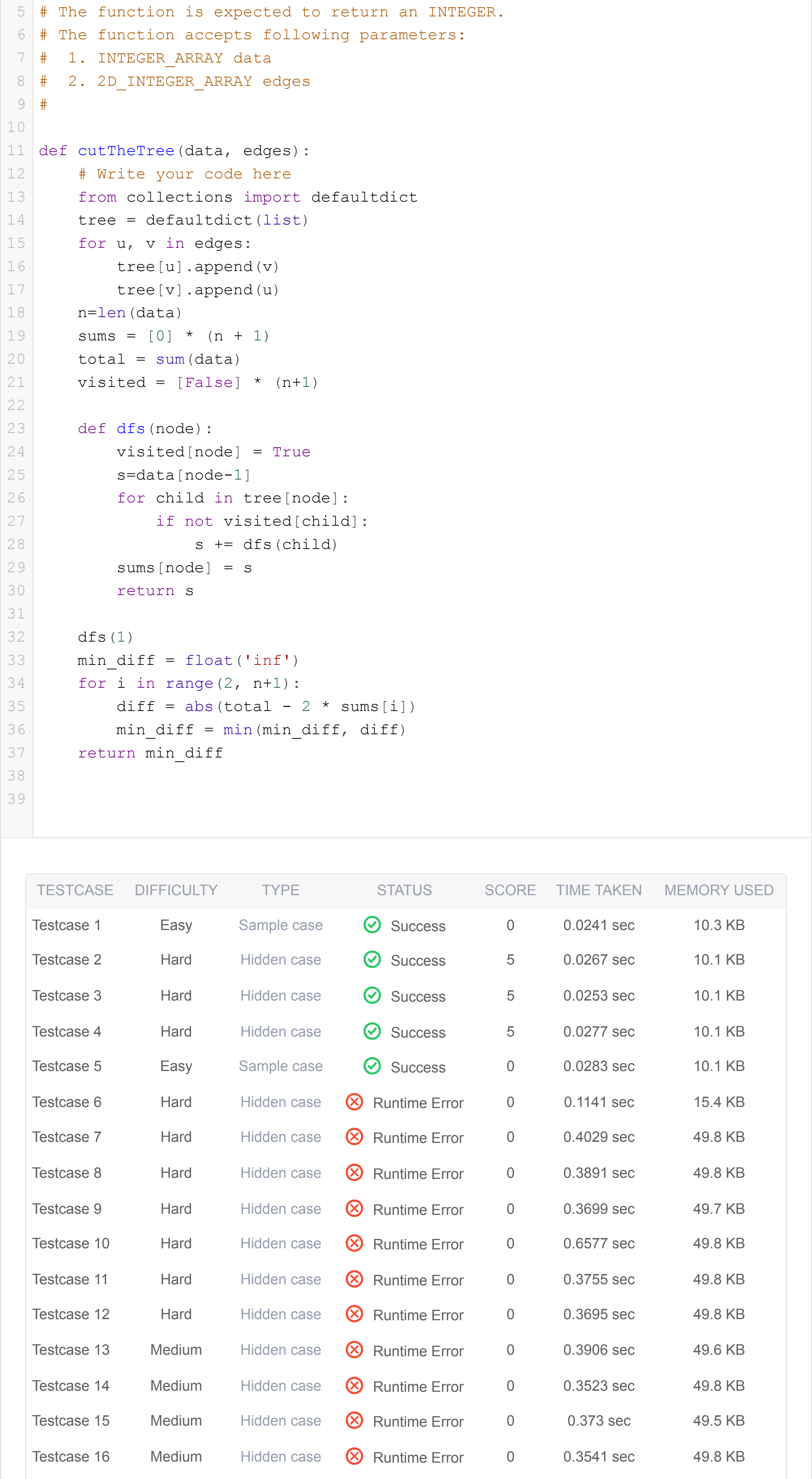
The minimum *difference* is 400.

CANDIDATE ANSWER

Language used: **Python 3**

```

#
# Complete the 'cutTheTree' function below.
#
```



```python
# The function is expected to return an INTEGER.
# The function accepts following parameters:
#  1. INTEGER_ARRAY data
#  2. 2D_INTEGER_ARRAY edges
#

def cutTheTree(data, edges):
    # Write your code here
    from collections import defaultdict
    tree = defaultdict(list)
    for u, v in edges:
        tree[u].append(v)
        tree[v].append(u)
    n=len(data)
    sums = [0] * (n + 1)
    total = sum(data)
    visited = [False] * (n+1)

    def dfs(node):
        visited[node] = True
        s=data[node-1]
        for child in tree[node]:
            if not visited[child]:
                s += dfs(child)
        sums[node] = s
        return s

    dfs(1)
    min_diff = float('inf')
    for i in range(2, n+1):
        diff = abs(total - 2 * sums[i])
        min_diff = min(min_diff, diff)
    return min_diff
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 1	Easy	Sample case	✓ Success	0	0.0241 sec	10.3 KB
Testcase 2	Hard	Hidden case	✓ Success	5	0.0267 sec	10.1 KB
Testcase 3	Hard	Hidden case	✓ Success	5	0.0253 sec	10.1 KB
Testcase 4	Hard	Hidden case	✓ Success	5	0.0277 sec	10.1 KB
Testcase 5	Easy	Sample case	✓ Success	0	0.0283 sec	10.1 KB
Testcase 6	Hard	Hidden case	⊗ Runtime Error	0	0.1141 sec	15.4 KB
Testcase 7	Hard	Hidden case	⊗ Runtime Error	0	0.4029 sec	49.8 KB
Testcase 8	Hard	Hidden case	⊗ Runtime Error	0	0.3891 sec	49.8 KB
Testcase 9	Hard	Hidden case	⊗ Runtime Error	0	0.3699 sec	49.7 KB
Testcase 10	Hard	Hidden case	⊗ Runtime Error	0	0.6577 sec	49.8 KB
Testcase 11	Hard	Hidden case	⊗ Runtime Error	0	0.3755 sec	49.8 KB
Testcase 12	Hard	Hidden case	⊗ Runtime Error	0	0.3695 sec	49.8 KB
Testcase 13	Medium	Hidden case	⊗ Runtime Error	0	0.3906 sec	49.6 KB
Testcase 14	Medium	Hidden case	⊗ Runtime Error	0	0.3523 sec	49.8 KB
Testcase 15	Medium	Hidden case	⊗ Runtime Error	0	0.373 sec	49.5 KB
Testcase 16	Medium	Hidden case	⊗ Runtime Error	0	0.3541 sec	49.8 KB

Testcase 17	Medium	Hidden case	⊗ Runtime Error	0	0.4007 sec	49.7 KB
Testcase 18	Medium	Hidden case	⊗ Runtime Error	0	0.4115 sec	49.7 KB
Testcase 19	Medium	Hidden case	⊗ Runtime Error	0	0.3627 sec	49.8 KB
Testcase 20	Medium	Hidden case	⊗ Runtime Error	0	0.393 sec	49.5 KB

No Comments

PDF generated at: 4 Sep 2025 17:01:33 UTC