



Full Name:

Utkarsh Mishra

Email:

utkarshmishra.jyp@gmail.com

Test Name:

Mock Test

Taken On:

23 Aug 2025 22:24:39 IST

Time Taken:

38 min/ 105 min

Invited by:

Ankush

Invited on:

23 Aug 2025 22:24:00 IST

Skills Score:

Tags Score:

100%

255/255

scored in **Mock Test** in 38 min  
on 23 Aug 2025 22:24:39 IST

- 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

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	12 min 35 sec	100/ 100	!
Q2	Components in a graph > Coding	14 min 32 sec	60/ 60	✓
Q3	Cut the Tree > Coding	10 min 8 sec	95/ 95	✓

QUESTION 1

!

Needs Review

Breadth First Search: Shortest Reach > Coding

Graph Theory

Algorithms

Medium

problem-solving

Core CS

QUESTION DESCRIPTION

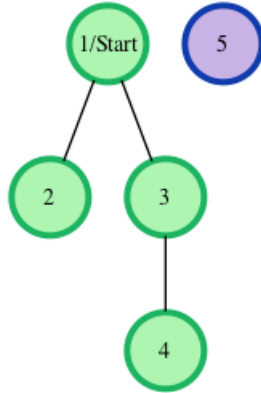
Score 100

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
```

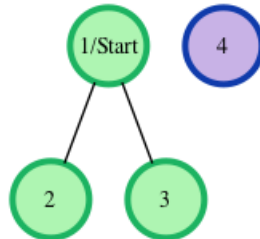
### Sample Output

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

### Explanation

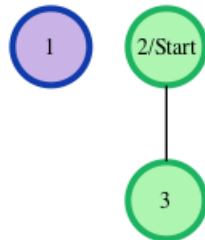
We perform the following two queries:

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: **C++14**

```
1 #include <bits/stdc++.h>
2 #include <vector>
3
4 using namespace std;
5
6 string ltrim(const string &);
7 string rtrim(const string &);
8 vector<string> split(const string &);
9
10
11
```

```

12  /*
13  * Complete the 'bfs' function below.
14  *
15  * The function is expected to return an INTEGER_ARRAY.
16  * The function accepts following parameters:
17  * 1. INTEGER n
18  * 2. INTEGER m
19  * 3. 2D_INTEGER_ARRAY edges
20  * 4. INTEGER s
21  */
22
23  vector<int> bfs(int n, int m, vector<vector<int>> edges, int s) {
24      vector<vector<int>> adj(n+1);
25      for(auto &e : edges){
26          adj[e[0]].push_back(e[1]);
27          adj[e[1]].push_back(e[0]);
28      }
29      vector<int> dist(n+1, -1);
30      dist[s]=0;
31      queue<int> q;
32      q.push(s);
33      while(!q.empty()){
34          int u= q.front();
35          q.pop();
36          for(int v: adj[u]){
37              if(dist[v]==-1){
38                  dist[v]=dist[u]+6;
39                  q.push(v);
40              }
41          }
42      }
43      vector<int> result;
44      for(int i=1; i<=n; i++){
45          if(i!=s){
46              result.push_back(dist[i]);
47          }
48      }
49      return result;
50  }
51
52  int main()
53  {
54      ofstream fout(getenv("OUTPUT_PATH"));
55
56      string q_temp;
57      getline(cin, q_temp);
58
59      int q = stoi(ltrim(rtrim(q_temp)));
60
61      for (int q_itr = 0; q_itr < q; q_itr++) {
62          string first_multiple_input_temp;
63          getline(cin, first_multiple_input_temp);
64
65          vector<string> first_multiple_input =
66  split(rtrim(first_multiple_input_temp));
67
68          int n = stoi(first_multiple_input[0]);
69
70          int m = stoi(first_multiple_input[1]);
71
72          vector<vector<int>> edges(m);
73
74          for (int i = 0; i < m; i++) {

```

```

75     edges[i].resize(2);
76
77     string edges_row_temp_temp;
78     getline(cin, edges_row_temp_temp);
79
80     vector<string> edges_row_temp =
81 split(rtrim(edges_row_temp_temp));
82
83     for (int j = 0; j < 2; j++) {
84         int edges_row_item = stoi(edges_row_temp[j]);
85
86         edges[i][j] = edges_row_item;
87     }
88 }
89
90 string s_temp;
91 getline(cin, s_temp);
92
93 int s = stoi(ltrim(rtrim(s_temp)));
94
95 vector<int> result = bfs(n, m, edges, s);
96
97 for (size_t i = 0; i < result.size(); i++) {
98     fout << result[i];
99
100     if (i != result.size() - 1) {
101         fout << " ";
102     }
103 }
104
105 fout << "\n";
106 }
107
108 fout.close();
109
110 return 0;
111 }
112
113 string ltrim(const string &str) {
114     string s(str);
115
116     s.erase(
117         s.begin(),
118         find_if(s.begin(), s.end(), not1(ptr_fun<int, int>(isspace)))
119     );
120
121     return s;
122 }
123
124 string rtrim(const string &str) {
125     string s(str);
126
127     s.erase(
128         find_if(s.rbegin(), s.rend(), not1(ptr_fun<int, int>
129 (isspace))).base(),
130         s.end()
131     );
132
133     return s;
134 }
135
136 vector<string> split(const string &str) {
137     vector<string> tokens;

```

```

13     string::size_type start = 0;
13     string::size_type end = 0;
18
19     while ((end = str.find(" ", start)) != string::npos) {
19         tokens.push_back(str.substr(start, end - start));
19
19         start = end + 1;
19     }
19
19     tokens.push_back(str.substr(start));
19
19     return tokens;
19 }
9

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 1	Easy	Sample case	✔ Success	0	0.0085 sec	8.63 KB
Testcase 2	Medium	Hidden case	✔ Success	5	0.0089 sec	8.88 KB
Testcase 3	Medium	Hidden case	✔ Success	5	0.024 sec	11 KB
Testcase 4	Hard	Hidden case	✔ Success	15	0.0089 sec	8.5 KB
Testcase 5	Hard	Hidden case	✔ Success	15	0.0096 sec	8.63 KB
Testcase 6	Hard	Hidden case	✔ Success	30	0.1122 sec	18.8 KB
Testcase 7	Hard	Hidden case	✔ Success	30	0.021 sec	9.13 KB
Testcase 8	Easy	Sample case	✔ Success	0	0.0083 sec	8.5 KB

No Comments

## QUESTION 2



Correct Answer

Score 60

## Components in a graph > Coding

Algorithms

Data Structures

Disjoint Set

Core CS

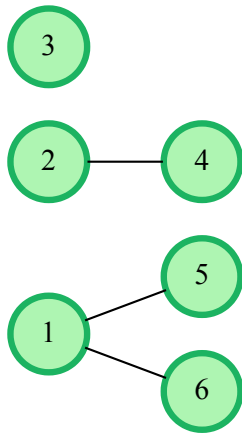
### QUESTION DESCRIPTION

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

- $1 \leq \text{number of nodes } N \leq 15000$
- $1 \leq \text{bg}[i][0] \leq N$
- $N + 1 \leq \text{bg}[i][1] \leq 2N$

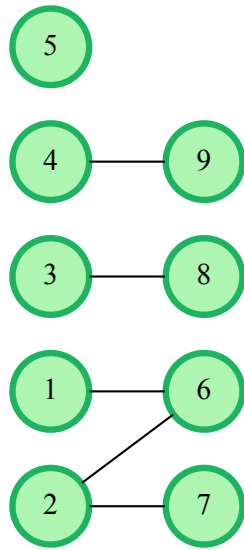
#### Sample Input

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: **C++14**

```

1  #include <bits/stdc++.h>
2  #include <vector>
3
4  using namespace std;
5
6  string ltrim(const string &);
7  string rtrim(const string &);
8  vector<string> split(const string &);
9
10
11
12 /*
13  * Complete the 'componentsInGraph' function below.
14  *
15  * The function is expected to return an INTEGER_ARRAY.
16  * The function accepts 2D_INTEGER_ARRAY gb as parameter.
17  */
18 struct DSU{
19     vector<int>parent,size;
20     DSU(int n){
21         parent.resize(n+1);
22         size.assign(n+1,1);
23         for(int i=1;i<=n;i++){
24             parent[i]=i;
25         }
26     }
27     int find(int x){
28         if(parent[x]!=x) parent[x]=find(parent[x]);
29         return parent[x];
30     }
31     void unite(int a,int b){
32         a=find(a);
33         b=find(b);
34         if(a!=b){

```



```

34         if(size[a]<size[b]) swap(a,b);
35         parent[b]=a;
36         size[a]+=size[b];
37     }
38 }
39 }
40 };
41
42
43 vector<int> componentsInGraph(vector<vector<int>> gb) {
44     int maxNode=0;
45     for(auto &e : gb){
46         maxNode=max({maxNode,e[0],e[1]});
47     }
48     DSU dsu(maxNode);
49     for(auto &e:gb){
50         dsu.unite(e[0],e[1]);
51     }
52     int mn=INT_MAX,mx=INT_MIN;
53     for(int i=1;i<=maxNode;i++){
54         if(dsu.find(i)==i && dsu.size[i]>1){
55             mn=min(mn, dsu.size[i]);
56             mx=max(mx,dsu.size[i]);
57         }
58     }
59     return {mn,mx};
60 }
61
62 int main()
63 {
64     ofstream fout(getenv("OUTPUT_PATH"));
65
66     string n_temp;
67     getline(cin, n_temp);
68
69     int n = stoi(ltrim(rtrim(n_temp)));
70
71     vector<vector<int>> gb(n);
72
73     for (int i = 0; i < n; i++) {
74         gb[i].resize(2);
75
76         string gb_row_temp_temp;
77         getline(cin, gb_row_temp_temp);
78
79         vector<string> gb_row_temp = split(rtrim(gb_row_temp_temp));
80
81         for (int j = 0; j < 2; j++) {
82             int gb_row_item = stoi(gb_row_temp[j]);
83
84             gb[i][j] = gb_row_item;
85         }
86     }
87
88     vector<int> result = componentsInGraph(gb);
89
90     for (size_t i = 0; i < result.size(); i++) {
91         fout << result[i];
92
93         if (i != result.size() - 1) {
94             fout << " ";
95         }
96     }

```

```

97     fout << "\n";
98
99     fout.close();
10
10     return 0;
10 }
10
10 string ltrim(const string &str) {
10     string s(str);
10
10     s.erase(
10         s.begin(),
10         find_if(s.begin(), s.end(), not1(ptr_fun<int, int>(isspace)))
10     );
10
10     return s;
12 }
13
14 string rtrim(const string &str) {
15     string s(str);
15
15     s.erase(
15         find_if(s.rbegin(), s.rend(), not1(ptr_fun<int, int>
15 (isspace))).base(),
15         s.end()
15     );
15
15     return s;
17 }
18
18 vector<string> split(const string &str) {
19     vector<string> tokens;
19
19     string::size_type start = 0;
19     string::size_type end = 0;
19
19     while ((end = str.find(" ", start)) != string::npos) {
19         tokens.push_back(str.substr(start, end - start));
19
19         start = end + 1;
19     }
19
19     tokens.push_back(str.substr(start));
19
19     return tokens;
21 }
22
23

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 1	Medium	Hidden case	 Success	0	0.0085 sec	8.75 KB
Testcase 2	Medium	Hidden case	 Success	0	0.0078 sec	8.63 KB
Testcase 3	Medium	Hidden case	 Success	0	0.0113 sec	8.63 KB
Testcase 4	Medium	Hidden case	 Success	0	0.0101 sec	8.88 KB
Testcase 5	Medium	Hidden case	 Success	0	0.0091 sec	8.63 KB
Testcase 6	Medium	Hidden case	 Success	0	0.0102 sec	8.75 KB
Testcase 7	Medium	Hidden case	 Success	0	0.0143 sec	9 KB
Testcase 8	Medium	Hidden case	 Success	0	0.0135 sec	9.13 KB

Testcase 9	Medium	Hidden case	✔ Success	0	0.01 sec	9.13 KB
Testcase 10	Medium	Hidden case	✔ Success	0	0.0114 sec	9.13 KB
Testcase 11	Medium	Hidden case	✔ Success	0	0.0121 sec	9.13 KB
Testcase 12	Medium	Hidden case	✔ Success	0	0.0223 sec	8.81 KB
Testcase 13	Medium	Hidden case	✔ Success	0	0.0116 sec	8.92 KB
Testcase 14	Medium	Hidden case	✔ Success	0	0.0114 sec	9 KB
Testcase 15	Medium	Hidden case	✔ Success	0	0.0131 sec	9 KB
Testcase 16	Medium	Hidden case	✔ Success	0	0.0152 sec	9.53 KB
Testcase 17	Medium	Hidden case	✔ Success	0	0.0088 sec	8.63 KB
Testcase 18	Medium	Hidden case	✔ Success	0	0.0231 sec	9.52 KB
Testcase 19	Easy	Sample case	✔ Success	0	0.0084 sec	8.63 KB
Testcase 20	Medium	Hidden case	✔ Success	0	0.0163 sec	9.76 KB
Testcase 21	Medium	Hidden case	✔ Success	0	0.0166 sec	9.88 KB
Testcase 22	Medium	Hidden case	✔ Success	0	0.0219 sec	9.88 KB
Testcase 23	Medium	Hidden case	✔ Success	0	0.0199 sec	9.75 KB
Testcase 24	Medium	Hidden case	✔ Success	0	0.0176 sec	9.85 KB
Testcase 25	Medium	Hidden case	✔ Success	0	0.0208 sec	9.75 KB
Testcase 26	Medium	Hidden case	✔ Success	0	0.0167 sec	9.88 KB
Testcase 27	Medium	Hidden case	✔ Success	0	0.0226 sec	10 KB
Testcase 28	Medium	Hidden case	✔ Success	0	0.0166 sec	9.75 KB
Testcase 29	Medium	Hidden case	✔ Success	0	0.0228 sec	9.8 KB
Testcase 30	Medium	Hidden case	✔ Success	0	0.0162 sec	9.5 KB
Testcase 31	Medium	Hidden case	✔ Success	0	0.0252 sec	9.75 KB
Testcase 32	Medium	Hidden case	✔ Success	0	0.0289 sec	9.88 KB
Testcase 33	Medium	Hidden case	✔ Success	0	0.0161 sec	9.75 KB
Testcase 34	Hard	Hidden case	✔ Success	10	0.0175 sec	9.88 KB
Testcase 35	Hard	Hidden case	✔ Success	10	0.031 sec	9.54 KB
Testcase 36	Hard	Hidden case	✔ Success	10	0.018 sec	9.63 KB
Testcase 37	Hard	Hidden case	✔ Success	10	0.018 sec	9.68 KB
Testcase 38	Hard	Hidden case	✔ Success	10	0.015 sec	9.75 KB
Testcase 39	Hard	Hidden case	✔ Success	10	0.0164 sec	9.75 KB

No Comments

### QUESTION 3



Correct Answer

Score 95

## Cut the Tree > Coding

Search

Algorithms

Medium

problem-solving

Core CS

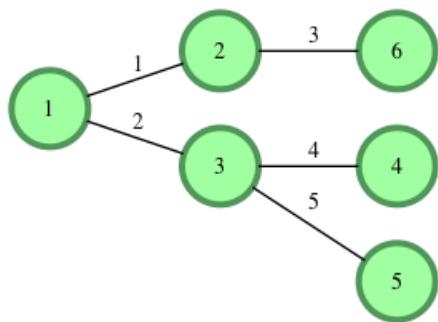
### QUESTION DESCRIPTION

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**

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

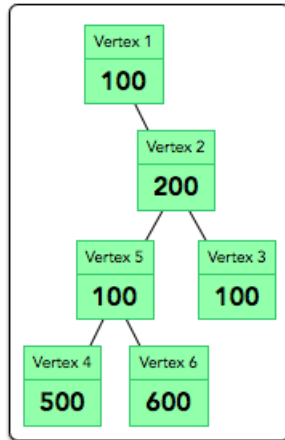
4 5  
5 6

### 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: C++14

```
1 #include <bits/stdc++.h>
2 #include <vector>
3
4 using namespace std;
5
6 string ltrim(const string &);
7 string rtrim(const string &);
8 vector<string> split(const string &);
9
10
11
12 /*
13  * Complete the 'cutTheTree' function below.
14  *
15  * The function is expected to return an INTEGER.
16  * The function accepts following parameters:
17  * 1. INTEGER_ARRAY data
18  * 2. 2D_INTEGER_ARRAY edges
19  */
20 int n;
21 vector<int> values;
22 vector<vector<int>>> adj;
23 vector<int> subtreeSum;
24 int totalSum;
```

```



25     int dfs(int u,int parent){
26         int sum=values[u];
27         for(int v : adj[u]){
28             if(v!=parent){
29                 sum+=dfs(v,u);
30             }
31         }
32         subtreeSum[u]=sum;
33         return sum;
34     }
35
36 int cutTheTree(vector<int> data, vector<vector<int>> edges) {
37     n=data.size();
38     values=data;
39     adj.assign(n,{});
40     subtreeSum.assign(n,0);
41     for(auto &e: edges){
42         int u=e[0]-1;
43         int v=e[1]-1;
44         adj[u].push_back(v);
45         adj[v].push_back(u);
46     }
47     totalSum=dfs(0,-1);
48     int ans=INT_MAX;
49     for(int i=1;i<n;i++){
50         int part1=subtreeSum[i];
51         int part2=totalSum-part1;
52         ans=min(ans,abs(part1-part2));
53     }
54     return ans;
55 }
56
57 int main()
58 {
59     ofstream fout(getenv("OUTPUT_PATH"));
60
61     string n_temp;
62     getline(cin, n_temp);
63
64     int n = stoi(ltrim(rtrim(n_temp)));
65
66     string data_temp_temp;
67     getline(cin, data_temp_temp);
68
69     vector<string> data_temp = split(rtrim(data_temp_temp));
70
71     vector<int> data(n);
72
73     for (int i = 0; i < n; i++) {
74         int data_item = stoi(data_temp[i]);
75
76         data[i] = data_item;
77     }
78
79     vector<vector<int>> edges(n - 1);
80
81     for (int i = 0; i < n - 1; i++) {
82         edges[i].resize(2);
83
84         string edges_row_temp_temp;
85         getline(cin, edges_row_temp_temp);
86
87         vector<string> edges_row_temp = split(rtrim(edges_row_temp_temp));

```

```

88         for (int j = 0; j < 2; j++) {
89             int edges_row_item = stoi(edges_row_temp[j]);
90
91             edges[i][j] = edges_row_item;
92         }
93     }
94
95     int result = cutTheTree(data, edges);
96
97     fout << result << "\n";
98
99     fout.close();
100
101     return 0;
102 }
103
104 string ltrim(const string &str) {
105     string s(str);
106
107     s.erase(
108         s.begin(),
109         find_if(s.begin(), s.end(), not1(ptr_fun<int, int>(isspace)))
110     );
111
112     return s;
113 }
114
115 string rtrim(const string &str) {
116     string s(str);
117
118     s.erase(
119         find_if(s.rbegin(), s.rend(), not1(ptr_fun<int, int>(isspace))).base(),
120         s.end()
121     );
122
123     return s;
124 }
125
126 vector<string> split(const string &str) {
127     vector<string> tokens;
128
129     string::size_type start = 0;
130     string::size_type end = 0;
131
132     while ((end = str.find(" ", start)) != string::npos) {
133         tokens.push_back(str.substr(start, end - start));
134
135         start = end + 1;
136     }
137
138     tokens.push_back(str.substr(start));
139
140     return tokens;
141 }
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 1	Easy	Sample case	 Success	0	0.0093 sec	8.75 KB
Testcase 2	Hard	Hidden case	 Success	5	0.008 sec	8.75 KB

Testcase 3	Hard	Hidden case	✔ Success	5	0.0105 sec	8.63 KB
Testcase 4	Hard	Hidden case	✔ Success	5	0.0149 sec	8.38 KB
Testcase 5	Easy	Sample case	✔ Success	0	0.0136 sec	8.63 KB
Testcase 6	Hard	Hidden case	✔ Success	5	0.0156 sec	10.6 KB
Testcase 7	Hard	Hidden case	✔ Success	10	0.1053 sec	30.3 KB
Testcase 8	Hard	Hidden case	✔ Success	5	0.0871 sec	31 KB
Testcase 9	Hard	Hidden case	✔ Success	5	0.1447 sec	30.4 KB
Testcase 10	Hard	Hidden case	✔ Success	5	0.1198 sec	30.8 KB
Testcase 11	Hard	Hidden case	✔ Success	5	0.1622 sec	30.8 KB
Testcase 12	Hard	Hidden case	✔ Success	5	0.0909 sec	29.9 KB
Testcase 13	Medium	Hidden case	✔ Success	5	0.0966 sec	30.1 KB
Testcase 14	Medium	Hidden case	✔ Success	5	0.1016 sec	30.7 KB
Testcase 15	Medium	Hidden case	✔ Success	5	0.1009 sec	30 KB
Testcase 16	Medium	Hidden case	✔ Success	5	0.1375 sec	30.8 KB
Testcase 17	Medium	Hidden case	✔ Success	5	0.0991 sec	30.9 KB
Testcase 18	Medium	Hidden case	✔ Success	5	0.1297 sec	30.8 KB
Testcase 19	Medium	Hidden case	✔ Success	5	0.0975 sec	29.9 KB
Testcase 20	Medium	Hidden case	✔ Success	5	0.0915 sec	30.1 KB

No Comments