Mock Test > utkarshmishra.jyp@gmail.com

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:	

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 38 min on 23 Aug 2025 22:24:39 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**	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

Score 100

Breadth First Search: Shortest Reach > Coding

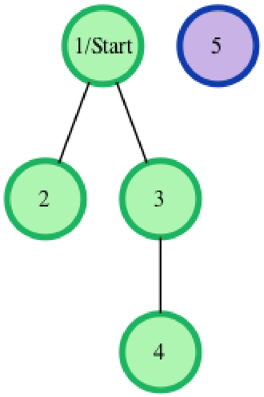
Graph Theory | Algorithms | Medium | problem-solving | Core CS

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
```

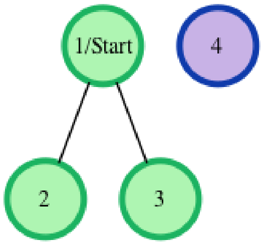
Sample Output

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

Explanation

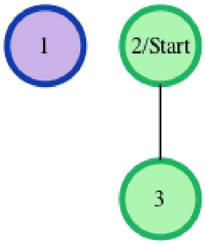
We perform the following two queries:

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: **C++14**

```
#include <bits/stdc++.h>
#include <vector>

using namespace std;

string ltrim(const string &);
string rtrim(const string &);
vector<string> split(const string &);



```

```cpp
/*
 * 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
 */

vector<int> bfs(int n, int m, vector<vector<int>> edges, int s) {
    vector<vector<int>> adj(n+1);
    for(auto &e : edges){
        adj[e[0]].push_back(e[1]);
        adj[e[1]].push_back(e[0]);
    }
    vector<int>dist(n+1,-1);
    dist[s]=0;
    queue<int>q;
    q.push(s);
    while(!q.empty()){
        int u= q.front();
        q.pop();
        for(int v: adj[u]){
            if(dist[v]==-1){
                dist[v]=dist[u]+6;
                q.push(v);
            }
        }
    }
    vector<int>result;
    for(int i=1;i<=n;i++){
        if(i!=s){
            result.push_back(dist[i]);
        }
    }
    return result;
}

int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    string q_temp;
    getline(cin, q_temp);

    int q = stoi(ltrim(rtrim(q_temp)));

    for (int q_itr = 0; q_itr < q; q_itr++) {
        string first_multiple_input_temp;
        getline(cin, first_multiple_input_temp);

        vector<string> first_multiple_input =
split(rtrim(first_multiple_input_temp));

        int n = stoi(first_multiple_input[0]);

        int m = stoi(first_multiple_input[1]);

        vector<vector<int>> edges(m);

        for (int i = 0; i < m; i++) {
            edges[i].resize(2);

            string edges_row_temp_temp;
            getline(cin, edges_row_temp_temp);

            vector<string> edges_row_temp =
split(rtrim(edges_row_temp_temp));

            for (int j = 0; j < 2; j++) {
                int edges_row_item = stoi(edges_row_temp[j]);

                edges[i][j] = edges_row_item;
            }
        }

        string s_temp;
        getline(cin, s_temp);

        int s = stoi(ltrim(rtrim(s_temp)));

        vector<int> result = bfs(n, m, edges, s);

        for (size_t i = 0; i < result.size(); i++) {
            fout << result[i];

            if (i != result.size() - 1) {
                fout << " ";
            }
        }

        fout << "\n";
    }

    fout.close();

    return 0;
}

string ltrim(const string &str) {
    string s(str);

    s.erase(
        s.begin(),
        find_if(s.begin(), s.end(), not1(ptr_fun<int, int>(isspace)))
    );

    return s;
}

string rtrim(const string &str) {
    string s(str);

    s.erase(
        find_if(s.rbegin(), s.rend(), not1(ptr_fun<int, int>
(isspace))).base(),
        s.end()
    );

    return s;
}

vector<string> split(const string &str) {
    vector<string> tokens;
    string::size_type start = 0;
    string::size_type end = 0;

    while ((end = str.find(" ", start)) != string::npos) {
        tokens.push_back(str.substr(start, end - start));

        start = end + 1;
    }

    tokens.push_back(str.substr(start));

    return tokens;
}
```

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

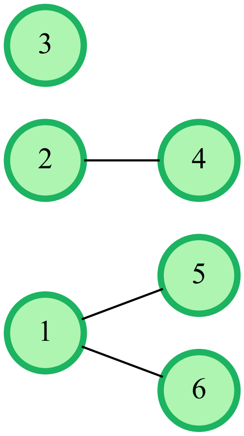
QUESTION DESCRIPTION

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

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

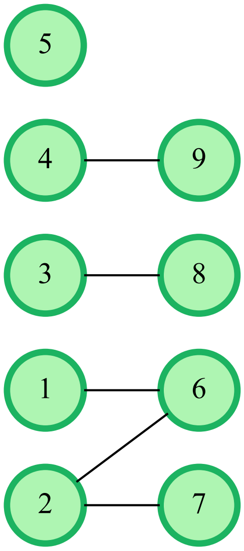
Sample Input

```
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: **C++14**

```
#include <bits/stdc++.h>
#include <vector>

using namespace std;

string ltrim(const string &);
string rtrim(const string &);
vector<string> split(const string &);



/*
 * Complete the 'componentsInGraph' function below.
 *
 * The function is expected to return an INTEGER_ARRAY.
 * The function accepts 2D_INTEGER_ARRAY gb as parameter.
 */
struct DSU{
    vector<int>parent,size;
    DSU(int n){
        parent.resize(n+1);
        size.assign(n+1,1);
        for(int i=1;i<=n;i++)
            parent[i]=i;
    }
        int find(int x){
            if(parent[x]!=x) parent[x]=find(parent[x]);
            return parent[x];
        }
        void unite(int a,int b){
            a=find(a);
            b=find(b);
            if(a!=b){
                if(size[a]<size[b]) swap(a,b);
                parent[b]=a;
                size[a]+=size[b];

        }
    }
};


vector<int> componentsInGraph(vector<vector<int>> gb) {
    int maxNode=0;
    for(auto &e : gb){
        maxNode=max({maxNode,e[0],e[1]});
    }
    DSU dsu(maxNode);
    for(auto &e:gb){
        dsu.unite(e[0],e[1]);
    }
    int mn=INT_MAX,mx=INT_MIN;
    for(int i=1;i<=maxNode;i++){
        if(dsu.find(i)==i && dsu.size[i]>1){
            mn=min(mn, dsu.size[i]);
            mx=max(mx,dsu.size[i]);
        }
    }
    return {mn,mx};
}

int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    string n_temp;
    getline(cin, n_temp);

    int n = stoi(ltrim(rtrim(n_temp)));

    vector<vector<int>> gb(n);

    for (int i = 0; i < n; i++) {
        gb[i].resize(2);

        string gb_row_temp_temp;
        getline(cin, gb_row_temp_temp);

        vector<string> gb_row_temp = split(rtrim(gb_row_temp_temp));

        for (int j = 0; j < 2; j++) {
            int gb_row_item = stoi(gb_row_temp[j]);

            gb[i][j] = gb_row_item;
        }
    }

    vector<int> result = componentsInGraph(gb);

    for (size_t i = 0; i < result.size(); i++) {
        fout << result[i];

        if (i != result.size() - 1) {
            fout << " ";
        }
    }
    fout << "\n";

    fout.close();

    return 0;
}

string ltrim(const string &str) {
    string s(str);

    s.erase(
        s.begin(),
        find_if(s.begin(), s.end(), not1(ptr_fun<int, int>(isspace)))
    );

    return s;
}

string rtrim(const string &str) {
    string s(str);

    s.erase(
        find_if(s.rbegin(), s.rend(), not1(ptr_fun<int, int>
(isspace))).base(),
        s.end()
    );

    return s;
}

vector<string> split(const string &str) {
    vector<string> tokens;

    string::size_type start = 0;
    string::size_type end = 0;

    while ((end = str.find(" ", start)) != string::npos) {
        tokens.push_back(str.substr(start, end - start));

        start = end + 1;
    }

    tokens.push_back(str.substr(start));

    return tokens;
}

```

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

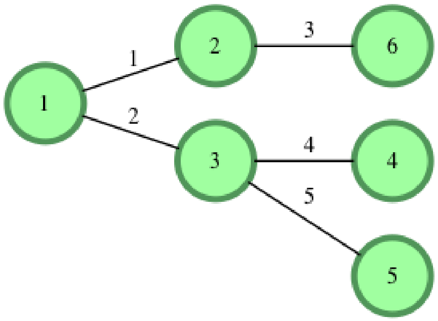
There is an undirected tree where each vertex is numbered from 1 to n, and each contains a data value. The *sum* of a tree is the sum of all its nodes' data values. If an edge is cut, two smaller trees are formed. The *difference* between two trees is the absolute value of the difference in their sums.

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
```

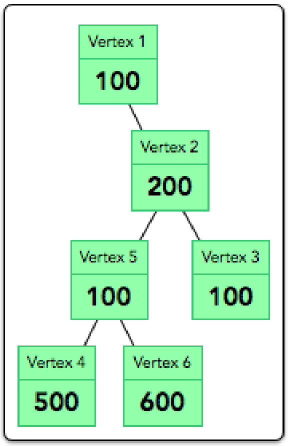
```
4 5
5 6
```

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**

```cpp
#include <bits/stdc++.h>
#include <vector>

using namespace std;

string ltrim(const string &);
string rtrim(const string &);
vector<string> split(const string &);



/*
 * 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
 */
 int n;
 vector<int>values;
 vector<vector<int>>adj;
 vector<int>subtreeSum;
 int totalSum;
 int dfs(int u,int parent){
    int sum=values[u];
    for(int v : adj[u]){
        if(v!=parent){
            sum+=dfs(v,u);
        }
    }
    subtreeSum[u]=sum;
    return sum;
 }

int cutTheTree(vector<int> data, vector<vector<int>> edges) {
    n=data.size();
    values=data;
    adj.assign(n,{});
    subtreeSum.assign(n,0);
    for(auto &e: edges){
        int u=e[0]-1;
        int v=e[1]-1;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    totalSum=dfs(0,-1);
    int ans=INT_MAX;
    for(int i=1;i<n;i++){
        int part1=subtreeSum[i];
        int part2=totalSum-part1;
        ans=min(ans,abs(part1-part2));
    }
    return ans;
}

int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    string n_temp;
    getline(cin, n_temp);

    int n = stoi(ltrim(rtrim(n_temp)));

    string data_temp_temp;
    getline(cin, data_temp_temp);

    vector<string> data_temp = split(rtrim(data_temp_temp));

    vector<int> data(n);

    for (int i = 0; i < n; i++) {
        int data_item = stoi(data_temp[i]);

        data[i] = data_item;
    }

    vector<vector<int>> edges(n - 1);

    for (int i = 0; i < n - 1; i++) {
        edges[i].resize(2);

        string edges_row_temp_temp;
        getline(cin, edges_row_temp_temp);

        vector<string> edges_row_temp = split(rtrim(edges_row_temp_temp));
        for (int j = 0; j < 2; j++) {
            int edges_row_item = stoi(edges_row_temp[j]);

            edges[i][j] = edges_row_item;
        }
    }

    int result = cutTheTree(data, edges);

    fout << result << "\n";

    fout.close();

    return 0;
}

string ltrim(const string &str) {
    string s(str);

    s.erase(
        s.begin(),
        find_if(s.begin(), s.end(), not1(ptr_fun<int, int>(isspace)))
    );

    return s;
}

string rtrim(const string &str) {
    string s(str);

    s.erase(
        find_if(s.rbegin(), s.rend(), not1(ptr_fun<int, int>
(isspace))).base(),
        s.end()
    );

    return s;
}

vector<string> split(const string &str) {
    vector<string> tokens;

    string::size_type start = 0;
    string::size_type end = 0;

    while ((end = str.find(" ", start)) != string::npos) {
        tokens.push_back(str.substr(start, end - start));

        start = end + 1;
    }

    tokens.push_back(str.substr(start));

    return tokens;
}

```

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

PDF generated at: 23 Aug 2025 17:35:07 UTC