HackerRank

HackerRank | Prepare > Data Structures > Trees > Swap Nodes [Algo]

Exit Full Screen View

Problem | Submissions | Leaderboard | Discussions

- it is the root of the tree, the first time visited

Swapping: Swapping subtrees of a node means that if initially node has left subtree L and right subtree R, then after swapping, the left subtree will be R and the right subtree, L.

For example, in the following tree, we swap children of node 1.

```
      1          1
     / \        / \
    2   3  ->  3   2
     \   \      \   \
      4   5      5   4
```

In-order traversal of left tree is 2 4 1 3 5 and of right tree is 3 5 1 2 4.

Swap operation:

We define depth of a node as follows:

- The root node is at depth 1.
- If the depth of the parent node is d, then the depth of current node will be d+1.

Given a tree and an integer, k, in one

Change Theme | Language C++11

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

using namespace std;

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

/*
 * Complete the 'swapNodes' function below.
 *
 * The function is expected to return a 2D_INTEGER_ARRAY.
 * The function accepts following parameters:
 *  1. 2D_INTEGER_ARRAY indexes
 *  2. INTEGER_ARRAY queries
 */
struct TreeNode {
    int data;
    TreeNode* left;
    TreeNode* right;

    TreeNode(int val) : data(val), left(nullptr), right(nullptr) {}
};

// Function to perform in-order traversal of the binary tree
void inOrderTraversal(TreeNode* root, vector<int>& result) {
    if (root == nullptr)
        return;
```

Line: 190 Col: 1

Upload Code as File | Test against custom input | Run Code | Submit Code

HackerRank | Prepare > Data Structures > Trees > Swap Nodes [Algo]

Exit Full Screen View

Problem | Submissions | Leaderboard | Discussions

- it is the root of the tree, the first time visited

Swapping: Swapping subtrees of a node means that if initially node has left subtree L and right subtree R, then after swapping, the left subtree will be R and the right subtree, L.

For example, in the following tree, we swap children of node 1.

```
      1          1
     / \        / \
    2   3  ->  3   2
     \   \      \   \
      4   5      5   4
```

In-order traversal of left tree is 2 4 1 3 5 and of right tree is 3 5 1 2 4.

Swap operation:

We define depth of a node as follows:

- The root node is at depth 1.
- If the depth of the parent node is d, then the depth of current node will be d+1.

Given a tree and an integer, k, in one

Change Theme | Language C++11

```cpp
// Function to perform in-order traversal of the binary tree
void inOrderTraversal(TreeNode* root, vector<int>& result) {
    if (root == nullptr)
        return;

    inOrderTraversal(root->left, result);
    result.push_back(root->data);
    inOrderTraversal(root->right, result);
}

// Function to swap subtrees at specified depths
void swapSubtrees(TreeNode* root, int k, int depth) {
    if (root == nullptr)
        return;

    if (depth % k == 0) {
        TreeNode* temp = root->left;
        root->left = root->right;
        root->right = temp;
    }

    swapSubtrees(root->left, k, depth + 1);
    swapSubtrees(root->right, k, depth + 1);
}

// Function to build the binary tree from the given indexes
TreeNode* buildTree(vector<vector<int>>& indexes) {
    queue<TreeNode*> nodes;
    TreeNode* root = new TreeNode(1);
```

Line: 190 Col: 1

Upload Code as File | Test against custom input | Run Code | Submit Code

Problem

Submissions

Leaderboard

Discussions

- it is the root of the tree, the first time visited

Swapping: Swapping subtrees of a node means that if initially node has left subtree `L` and right subtree `R`, then after swapping, the left subtree will be `R` and the right subtree, `L`.

For example, in the following tree, we swap children of node `1`.

```
                          De
  1            1          [
 / \          / \
2   3    ->  3   2
 \   \        \   \
  4   5        5   4
```

In-order traversal of left tree is `2 4 1 3 5` and of right tree is `3 5 1 2 4`.

Swap operation:

We define depth of a node as follows:

- The root node is at depth 1.
- If the depth of the parent node is `d`, then the depth of current node will be `d+1`.

Given a tree and an integer, `k`, in one

Change Theme Language C++11

```cpp
// Function to build the binary tree from the given indexes
TreeNode* buildTree(vector<vector<int>>& indexes) {
    queue<TreeNode*> nodes;
    TreeNode* root = new TreeNode(1);
    nodes.push(root);

    for (auto& idx : indexes) {
        TreeNode* curr = nodes.front();
        nodes.pop();

        if (idx[0] != -1) {
            curr->left = new TreeNode(idx[0]);
            nodes.push(curr->left);
        }

        if (idx[1] != -1) {
            curr->right = new TreeNode(idx[1]);
            nodes.push(curr->right);
        }
    }

    return root;
}
vector<vector<int>> swapNodes(vector<vector<int>> indexes, vector<int> queries) {
vector<vector<int>> result;
    TreeNode* root = buildTree(indexes);

    for (int k : queries) {
```

Line: 38 Col: 16

Upload Code as File Test against custom input Run Code Submit Code

HackerRank | Prepare > Data Structures > Trees > Swap Nodes [Algo] Exit Full Screen View

Problem

Submissions

Leaderboard

Discussions

- it is the root of the tree, the first time visited

Swapping: Swapping subtrees of a node means that if initially node has left subtree `L` and right subtree `R`, then after swapping, the left subtree will be `R` and the right subtree, `L`.

For example, in the following tree, we swap children of node `1`.

```
                          De
  1            1          [
 / \          / \
2   3    ->  3   2
 \   \        \   \
  4   5        5   4
```

In-order traversal of left tree is `2 4 1 3 5` and of right tree is `3 5 1 2 4`.

Swap operation:

We define depth of a node as follows:

- The root node is at depth 1.
- If the depth of the parent node is `d`, then the depth of current node will be `d+1`.

Given a tree and an integer, `k`, in one

Change Theme Language C++11

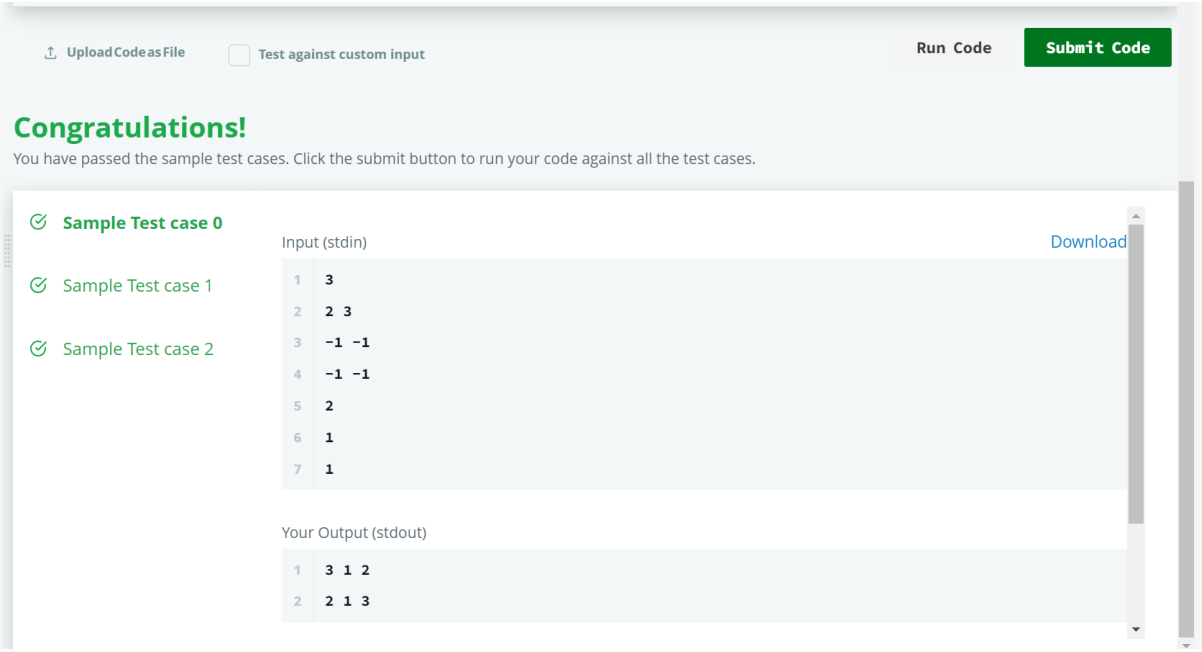
```cpp
}
vector<vector<int>> swapNodes(vector<vector<int>> indexes, vector<int> queries) {
vector<vector<int>> result;
    TreeNode* root = buildTree(indexes);

    for (int k : queries) {
        swapSubtrees(root, k, 1);
        vector<int> traversal;
        inOrderTraversal(root, traversal);
        result.push_back(traversal);
    }

    return result;
}

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

    string n_temp;
    getline(cin, n_temp);

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

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

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

Line: 62 Col: 36

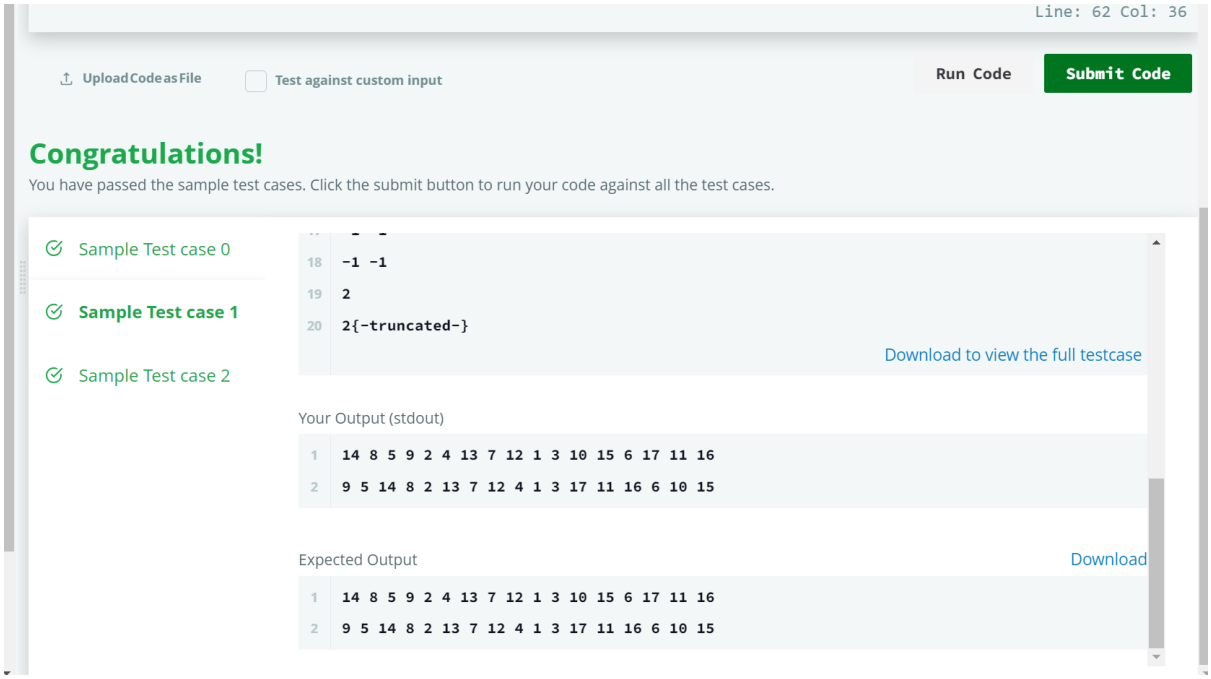
Upload Code as File Test against custom input Run Code Submit Code

Output

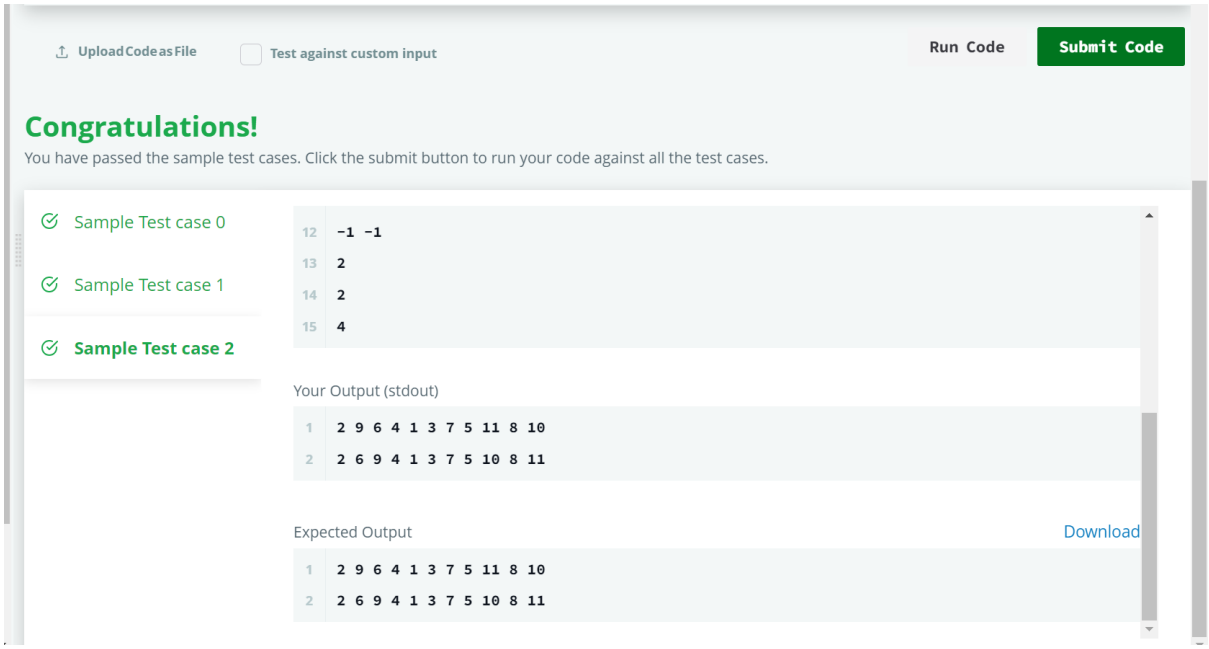
Testcase:1



Testcaase:2



Testcase:3



Code

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

using namespace std;

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

/*
 * Complete the 'swapNodes' function below.
 *
 * The function is expected to return a 2D_INTEGER_ARRAY.
 * The function accepts following parameters:
 *  1. 2D_INTEGER_ARRAY indexes
 *  2. INTEGER_ARRAY queries
 */
struct TreeNode {
    int data;
    TreeNode* left;
    TreeNode* right;
```

```
    TreeNode(int val) : data(val), left(nullptr), right(nullptr)
{}
};

// Function to perform in-order traversal of the binary tree
void inOrderTraversal(TreeNode* root, vector<int>& result) {
    if (root == nullptr)
        return;

    inOrderTraversal(root->left, result);
    result.push_back(root->data);
    inOrderTraversal(root->right, result);
}

// Function to swap subtrees at specified depths
void swapSubtrees(TreeNode* root, int k, int depth) {
    if (root == nullptr)
        return;

    if (depth % k == 0) {
        TreeNode* temp = root->left;
        root->left = root->right;
        root->right = temp;
    }

    swapSubtrees(root->left, k, depth + 1);
    swapSubtrees(root->right, k, depth + 1);
}

// Function to build the binary tree from the given indexes
TreeNode* buildTree(vector<vector<int>>& indexes) {
    queue<TreeNode*> nodes;
    TreeNode* root = new TreeNode(1);
    nodes.push(root);

    for (auto& idx : indexes) {
        TreeNode* curr = nodes.front();
        nodes.pop();

        if (idx[0] != -1) {
            curr->left = new TreeNode(idx[0]);
            nodes.push(curr->left);
        }

        if (idx[1] != -1) {
            curr->right = new TreeNode(idx[1]);
```

```
            nodes.push(curr->right);
        }
    }

    return root;
}
vector<vector<int>> swapNodes(vector<vector<int>> indexes, vector
<int> queries) {
vector<vector<int>> result;
    TreeNode* root = buildTree(indexes);

    for (int k : queries) {
        swapSubtrees(root, k, 1);
        vector<int> traversal;
        inOrderTraversal(root, traversal);
        result.push_back(traversal);
    }

    return result;
}

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

    string n_temp;
    getline(cin, n_temp);

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

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

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

        string indexes_row_temp_temp;
        getline(cin, indexes_row_temp_temp);

        vector<string> indexes_row_temp = split(rtrim(indexes_row
_temp_temp));

        for (int j = 0; j < 2; j++) {
            int indexes_row_item = stoi(indexes_row_temp[j]);

            indexes[i][j] = indexes_row_item;
        }
```

```
    }

    string queries_count_temp;
    getline(cin, queries_count_temp);

    int queries_count = stoi(ltrim(rtrim(queries_count_temp)));

    vector<int> queries(queries_count);

    for (int i = 0; i < queries_count; i++) {
        string queries_item_temp;
        getline(cin, queries_item_temp);

        int queries_item = stoi(ltrim(rtrim(queries_item_temp)));

        queries[i] = queries_item;
    }

    vector<vector<int>> result = swapNodes(indexes, queries);

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

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

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

    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>(isspac
e)))
    );

    return s;
}

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

    s.erase(
        find_if(s.rbegin(), s.rend(), not1(ptr_fun<int, int>(issp
ace))).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;
}
```