

HackerRank

Problem

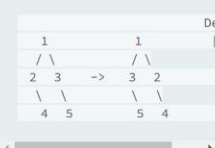
Submissions

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.



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

Exit Full Screen View

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 string ltrim(const string &);
6 string rtrim(const string &);
7 vector<string> split(const string &);
8
9 /*
10  * Complete the 'swapNodes' function below.
11  * The function is expected to return a 2D_INTEGER_ARRAY.
12  * The function accepts following parameters:
13  * 1. 2D_INTEGER_ARRAY indexes
14  * 2. INTEGER_ARRAY queries
15  */
16
17 struct TreeNode {
18     int data;
19     TreeNode* left;
20     TreeNode* right;
21
22     TreeNode(int val) : data(val), left(nullptr), right(nullptr) {}
23 };
24
25 // Function to perform in-order traversal of the binary tree
26 void inOrderTraversal(TreeNode* root, vector<int>& result) {
27     if (root == nullptr)
28         return;
29
30     inOrderTraversal(root->left, result);
31     result.push_back(root->data);
32     inOrderTraversal(root->right, result);
33 }
34
35 // Function to swap subtrees at specified depths
36 void swapSubtrees(TreeNode* root, int k, int depth) {
37     if (root == nullptr)
38         return;
39
40     if (depth % k == 0) {
41         TreeNode* temp = root->left;
42         root->left = root->right;
43         root->right = temp;
44     }
45
46     swapSubtrees(root->left, k, depth + 1);
47     swapSubtrees(root->right, k, depth + 1);
48 }
49
50 // Function to build the binary tree from the given indexes
51 TreeNode* buildTree(vector<vector<int>>& indexes) {
52     queue<TreeNode*> nodes;
53     TreeNode* root = new TreeNode(1);
54     nodes.push(root);
55     int i = 1;
56     while (!nodes.empty() && i < indexes.size()) {
57         TreeNode* node = nodes.front();
58         nodes.pop();
59         if (i < indexes[i][0]) {
60             node->left = new TreeNode(indexes[i][0]);
61             nodes.push(node->left);
62         }
63         if (i < indexes[i][1]) {
64             node->right = new TreeNode(indexes[i][1]);
65             nodes.push(node->right);
66         }
67         i++;
68     }
69     return root;
70 }
```

Line: 190 Col: 1

Upload Code as File

Test against custom input

Run Code

Submit Code

Problem

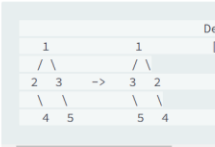
Submissions

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.



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

Exit Full Screen View

```
25 // Function to perform in-order traversal of the binary tree
26 void inOrderTraversal(TreeNode* root, vector<int>& result) {
27     if (root == nullptr)
28         return;
29
30     inOrderTraversal(root->left, result);
31     result.push_back(root->data);
32     inOrderTraversal(root->right, result);
33 }
34
35 // Function to swap subtrees at specified depths
36 void swapSubtrees(TreeNode* root, int k, int depth) {
37     if (root == nullptr)
38         return;
39
40     if (depth % k == 0) {
41         TreeNode* temp = root->left;
42         root->left = root->right;
43         root->right = temp;
44     }
45
46     swapSubtrees(root->left, k, depth + 1);
47     swapSubtrees(root->right, k, depth + 1);
48 }
49
50 // Function to build the binary tree from the given indexes
51 TreeNode* buildTree(vector<vector<int>>& indexes) {
52     queue<TreeNode*> nodes;
53     TreeNode* root = new TreeNode(1);
54     nodes.push(root);
55     int i = 1;
56     while (!nodes.empty() && i < indexes.size()) {
57         TreeNode* node = nodes.front();
58         nodes.pop();
59         if (i < indexes[i][0]) {
60             node->left = new TreeNode(indexes[i][0]);
61             nodes.push(node->left);
62         }
63         if (i < indexes[i][1]) {
64             node->right = new TreeNode(indexes[i][1]);
65             nodes.push(node->right);
66         }
67         i++;
68     }
69     return root;
70 }
```

Line: 190 Col: 1

Upload Code as File

Test against custom input

Run Code

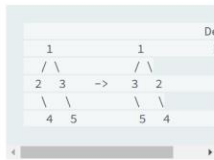
Submit Code

Problem

- 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.



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
50 // Function to build the binary tree from the given indexes
51 Tree* buildTree(vector<vector<int>>& indexes) {
52     queue<Tree*> nodes;
53     Tree* root = new Tree(1);
54     nodes.push(root);
55
56     for (auto& idx : indexes) {
57         Tree* curr = nodes.front();
58         nodes.pop();
59
60         if (idx[0] != -1) {
61             curr->left = new Tree(idx[0]);
62             nodes.push(curr->left);
63         }
64
65         if (idx[1] != -1) {
66             curr->right = new Tree(idx[1]);
67             nodes.push(curr->right);
68         }
69     }
70
71     return root;
72 }
73
74 vector<vector<int>> swapNodes(vector<vector<int>> indexes, vector<int> queries) {
75     vector<vector<int>> result;
76     Tree* root = buildTree(indexes);
77
78     for (int k : queries) {
79         swapSubtrees(root, k, 1);
80     }
81
82     return result;
83 }
84
85 int main() {
86     vector<vector<int>> indexes;
87     vector<int> queries;
88     int n;
89     while (n--) {
90         int x, y;
91         while (x < y) {
92             x = x * 2;
93             y = y * 2;
94         }
95         x = x / 2;
96         y = y / 2;
97         indexes.push_back({x, y});
98     }
99     int q;
100     while (q--) {
101         int k;
102         queries.push_back(k);
103     }
104     vector<vector<int>> result = swapNodes(indexes, queries);
105     for (int i = 0; i < result.size(); i++) {
106         for (int j = 0; j < result[i].size(); j++) {
107             cout << result[i][j] << " ";
108         }
109         cout << endl;
110     }
111     return 0;
112 }
Line: 38 Col: 16
Run Code Submit Code

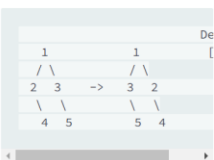
```

Problem

- 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.



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
72 }
73
74 vector<vector<int>> swapNodes(vector<vector<int>> indexes, vector<int> queries) {
75     vector<vector<int>> result;
76     Tree* root = buildTree(indexes);
77
78     for (int k : queries) {
79         swapSubtrees(root, k, 1);
80         vector<int> traversal;
81         inOrderTraversal(root, traversal);
82         result.push_back(traversal);
83     }
84
85     return result;
86 }
87
88 int main()
89 {
90     ofstream fout(getenv("OUTPUT_PATH"));
91
92     string n_temp;
93     getline(cin, n_temp);
94
95     int n = stoi(ltrim(rtrim(n_temp)));
96
97     vector<vector<int>> indexes(n);
98
99     for (int i = 0; i < n; i++) {
100         indexes[i].resize(2);
101         int x, y;
102         while (x < y) {
103             x = x * 2;
104             y = y * 2;
105         }
106         x = x / 2;
107         y = y / 2;
108         indexes[i][0] = x;
109         indexes[i][1] = y;
110     }
111
112     int q;
113     while (q--) {
114         int k;
115         queries.push_back(k);
116     }
117
118     vector<vector<int>> result = swapNodes(indexes, queries);
119     for (int i = 0; i < result.size(); i++) {
120         for (int j = 0; j < result[i].size(); j++) {
121             cout << result[i][j] << " ";
122         }
123         cout << endl;
124     }
125     return 0;
126 }
Line: 62 Col: 36
Run Code Submit Code

```

Output

Testcase:1

⬆️ Upload Code as File

☐ Test against custom input

Run Code

Submit Code

✔️ Sample Test case 0

✔️ Sample Test case 1

✔️ Sample Test case 2

Input (stdin)

Download

13

22 3

3-1 -1

4-1 -1

52

61

71

Your Output (stdout)

13 1 2

22 1 3

Testcaase:2

⬆️ Upload Code as File

☐ Test against custom input

Run Code

Submit Code

✔️ Sample Test case 0

✔️ Sample Test case 1

✔️ Sample Test case 2

Line: 62 Col: 36

17-1 -1

18-1 -1

192

202{-truncated-}

Download to view the full testcase

Your Output (stdout)

114 8 5 9 2 4 13 7 12 1 3 10 15 6 17 11 16

29 5 14 8 2 13 7 12 4 1 3 17 11 16 6 10 15

Expected Output

Download

114 8 5 9 2 4 13 7 12 1 3 10 15 6 17 11 16

29 5 14 8 2 13 7 12 4 1 3 17 11 16 6 10 15

Testcase:3

[Upload Code as File](#) ☐ Test against custom input [Run Code](#) [Submit Code](#)

Congratulations!

You have passed the sample test cases. Click the submit button to run your code against all the test cases.

✔ Sample Test case 0

✔ Sample Test case 1

✔ Sample Test case 2

12	-1 -1
13	2
14	2
15	4

Your Output (stdout)

1	2 9 6 4 1 3 7 5 11 8 10
2	2 6 9 4 1 3 7 5 10 8 11

Expected Output [Download](#)

1	2 9 6 4 1 3 7 5 11 8 10
2	2 6 9 4 1 3 7 5 10 8 11

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>(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;
}

```