Горен изглед

locked

Problem Submissions Leaderboard Discussions

Дадено ви е двоично дърво с N на брой върха номерирани от 0 до $N-1$, разположено в координатната система. Коренът е с индекс 0 и е разположен в точката $(0,0)$.

Ако даден връх е на координати (x,y), то лявото му дете е на координати $(x-1,y-1)$, а дясното - на координати $(x+1,y-1)$.

Вашата задача е да намерите кои върхове ще се виждат ако погледнем дървото отгоре (един връх се вижда ако е на координати (x,y) и за всеки друг връх с координати $(x,y_1) : y_1 \leq y$).

Ако има два върха на едни и същи координати и влизат в горния изглед, то трябва да изкарата само този с по-малък индекс в preorder индексация (този, който се среща първи в preorder обхождане)

Input Format

Въвежда се N. На i-тия ред се въвеждат по две числа - индексите съответно на лявото и дясното дете на върха с индекс i или -1 ако той няма съответното дете.

Constraints

$1 \leq N \leq 10^6$

Output Format

Извеждат се всички индекси на върхове, които са видими ако погледнем дървото отгоре подредени по техните x координати в нарастващ ред.

Sample Input 0

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

Sample Output 0

```
10 9 3 1 0 2 6
```

Sample Input 1

```
5
1 2
-1 3
-1 -1
-1 4
-1 -1
```

Sample Output 1

```
1 0 2
```

Submissions: 67
Max Score: 100
Difficulty: Medium

Rate This Challenge:

More

Current Buffer (saved locally, editable) C++14

```cpp
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
#include <queue>
using namespace std;

struct Point {
    int x, y;
    Point(int x=0, int y=0) {
        this->x = x;
        this->y = y;
    }
};

struct Node {
    int data;
    Point pt;
    Node* left, * right;

    Node(int data, Point pt = { 0,0 }, Node* left = nullptr, Node* right = nullptr) {
        this->data = data;
        this->left = left;
        this->right = right;
        this->pt = pt;
    }
};

class Tree {
    Node* root;
public:
    Tree() {
        root = nullptr;
    }

    Node* getRoot() { return root; }

    Node* insertRoot() {
        return new Node(0);
    }

    Node* insertL(Node* parent,Point pt,int data) {
        parent->left = new Node(data, pt);

        return parent->left;
    }

    Node* insertR(Node* parent, Point pt, int data) {
        parent->right = new Node(data, pt);
        return parent->right;
    }

    Node* remove(Node* root, int data) {
        if (root == NULL)
```

```
            return root;

        if (data < root->data)
            root->left = remove(root->left, data);
        else if (data > root->data)
            root->right = remove(root->right, data);
        else {
            if (root->left == NULL) {
                Node* temp = root->right;
                delete root;
                return temp;
            }
            else if (root->right == NULL) {
                Node* temp = root->left;
                delete root;
                return temp;
            }
            Node* temp = minValueNode(root->right);
            root->data = temp->data;
            root->right = remove(root->right, temp->data);
        }
        return root;
    }

    Node* minValueNode(Node* node) {
        Node* current = node;
        while (current->left != NULL) {
            current = current->left;
        }
        return current;
    }

    void print(Node* root) {
        if (root == NULL)
            return;

        queue<Node*> q;
        q.push(root);

        while (!q.empty()) {
            Node* current = q.front();


            printf("%d - (%d,%d)\n", current->data,current->pt.x,current->pt.y);
            q.pop();


            if (current->left != NULL)
                q.push(current->left);

            if (current->right != NULL)
                q.push(current->right);

        }
    }
};

vector<Node*>nodes;


int main() {

    int n,left,right;
    cin >> n;

    Tree tree;
    Node* root = NULL;
    queue<Node*> q;

    vector<Node*> visible;

    root = new Node(0);
    nodes.push_back(root);

    int j = 0;
```

```
    for (int i = 1; i < n; i++) {
        cin >> left >> right;

        if (left != -1) {
            Node* node = new Node(left, { nodes[j]->pt.x - 1,nodes[j]->pt.y - 1 });
            nodes[j]->left = node;
            nodes.push_back(node);
        }

        if (right != -1) {
            Node* node = new Node(right, { nodes[j]->pt.x + 1,nodes[j]->pt.y - 1 });
            nodes[j]->right = node;
            nodes.push_back(node);
        }
        j++;
    }

    sort(nodes.begin(), nodes.end(), [](const auto& lhs, const auto& rhs) {
        if (lhs->pt.x == rhs->pt.x && lhs->pt.y == rhs->pt.y && lhs->pt.x<0)
            return lhs->data > rhs->data;
        else if(lhs->pt.x == rhs->pt.x && lhs->pt.y == rhs->pt.y && lhs->pt.x>0)
            return lhs->data < rhs->data;

        if (lhs->pt.x == rhs->pt.x && lhs->pt.x<0)
            return lhs->pt.y < rhs->pt.y;

        if (lhs->pt.x == rhs->pt.x && lhs->pt.x > 0)
            return lhs->pt.y > rhs->pt.y;

        return lhs->pt.x < rhs->pt.x;
        });


    /*printf("\n\nSORTED: \n");
    for(int i=0;i<nodes.size();i++)
        printf("%d - (%d,%d)\n", nodes[i]->data, nodes[i]->pt.x, nodes[i]->pt.y);*/

    for (int i = 0; i < nodes.size(); i++) {
        if (nodes[i]->pt.x < 0) {
            if (nodes[i]->pt.x == nodes[i + 1]->pt.x)
                continue;
        }

        if (nodes[i]->pt.x == 0 && nodes[i]->pt.y != 0)
            continue;

        if (nodes[i]->pt.x > 0) {
            if (nodes[i]->pt.x == nodes[i - 1]->pt.x)
                continue;
        }

        visible.push_back(nodes[i]);
    }


    //printf("\n\nVISIBLE: \n");
    for (int i = 0; i < visible.size(); i++)
        //printf("%d - (%d,%d)\n", visible[i]->data, visible[i]->pt.x, visible[i]->pt.y);
        printf("%d ", visible[i]->data);

    //tree.print(root);

    return 0;
}
```

Line: 1 Col: 1

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

Contest Calendar | Interview Prep | Blog | Scoring | Environment | FAQ | About Us | Support | Careers | Terms Of Service | Privacy Policy | Request a Feature