Attacking Vigorously the Leaderboard

locked

Problem | Submissions | Leaderboard | Discussions

Submitted 8 months ago • Score: 100.00 Status: Accepted

✔ Test Case #0	✔ Test Case #1	✔ Test Case #2
✔ Test Case #3	✔ Test Case #4	✔ Test Case #5
✔ Test Case #6	✔ Test Case #7	✔ Test Case #8
✔ Test Case #9		

Submitted Code

Language: C++ Open in editor

```
#include <iostream>
#include <string>
#include <tomantp>
bool flag=1;
using namespace std;

struct Node
{
    double value;
    Node *left;
    Node *right;
int height;

    Node(double value, Node *left, Node *right)
    {
        this->value = value;
        this->left = left;
        this->right = right;
    }
    void calculateHeight(){
            height=0;
        if(left){
        height=max(height,left->height+1);
        }
        if(right){
            height=max(height,right->height+1);
            }
        }
    int leftHeight(){
        if(left){
            return left->height+1;
        }
        return 0;
    }
    int rightHeight(){
        if(right){
            return right->height+1;
        }
        return 0;
    }
    int balance(){
        return leftHeight()-rightHeight();
    }
    void rotateRight(){
        if(!left){
            return;
        }
        Node* leftRight=this->left->right;
        Node* oldRight=this->right;
        swap(this->value,this->left->value);
        this->right=this->left;
        this->left=this->left->left;
        this->right->right=oldRight;
        this->right->left=leftRight;

    }
    void rotateLeft() {
        if (!right) {
            return;
        }

        Node* rightLeft = this->right->left;
        Node* oldLeft = this->left;

        swap(this->value, this->right->value);
        this->left = this->right;
        this->right = this->right->right;
        this->left->left = oldLeft;
        this->left->right = rightLeft;
    }
    void recalculateHeights() {
        if (left) {
            left->calculateHeight();
        }
        if (right) {
            right->calculateHeight();
        }
        this->calculateHeight();
    }
    void fixTree(){
        if(balance()>1){
            if(right->balance()<=-1){
                this->rotateLeft();
                recalculateHeights();
            }
            else if(right->balance()>=1){
                right->rotateRight();
                this->rotateLeft();
                recalculateHeights();
            }
        }
        else if(balance()<1){
            if(left->balance()>=1){
                this->rotateRight();
                recalculateHeights();
            }
            else if(left->balance()<=-1){
                left->rotateLeft();
                this->rotateRight();
                recalculateHeights();
            }
        }
    }
};

class AVLTree
{
private:
    Node *root;

    bool containsRecursive(Node *current, double value)
    {
        if (current == NULL)
        {
            return false;
        }

        if (current->value == value)
        {
            return true;
        }

        if (value < current->value)
        {
            return containsRecursive(current->left, value);
        }
        else
        {
            return containsRecursive(current->right, value);
        }
    }

    void printRecursive(Node *current)
    {
        if (current == NULL)
        {
            return;
        }

        printRecursive(current->left);
        cout << current->value << " ";
        printRecursive(current->right);
    }

public:
    AVLTree()
    {
        root = NULL;
    }

    void add(double value)
    {
        root=_add(value,root);
    }
    Node* _add(double value,Node* current){
        if(current){
            return new Node(value,nullptr,nullptr);
        }
        else if(current->value<value){
            current->right=_add(value,current->right);
        }
        else if(current->value>value){
            current->left=_add(value,current->left);
        }

            if(current->value==value){
                cout<<current->value<<" already added"<<endl;
            }
        current->calculateHeight();
        current->fixTree();
            return current;

    }


    void remove(double value)
    {
        root=_remove(value,root);
        if(flag){
            cout<<value<<" not found to remove"<<endl;

        }
        flag=1;
    }
    Node* _remove(double value,Node* current){
        if(current==nullptr){
            return nullptr;
        }
        if(value<current->value){
            current->left=_remove(value,current->left);
        }
        else if(value>current->value){
            current->right=_remove(value,current->right);
        }
        else{
            flag=0;
            if(!current->left && !current->right){
                free(current);
                return nullptr;
            }
            else if(!current->left){
                Node* tempRight=current->right;
                free(current);
                return tempRight;
            }
            else if(!current->right){
                Node* tempLeft=current->left;
                free(current);
                return tempLeft;
            }
            else{
                Node* swapWith=current->right;
                while(swapWith->left){
                    swapWith=swapWith->left;
                }
                current->value=swapWith->value;
                current->right=_remove(swapWith->value,current->right);
            }
        }



        current->calculateHeight();
        current->fixTree();
        return current;
    }
bool contains(double value)
    {
        if (root == NULL)
        {
            return false;
        }

        return containsRecursive(root, value);
    }

    void print()
    {
        if (root == NULL)
        {
            return;
        }

        printRecursive(root);
        cout << endl;
    }
};

int main()
{
    AVLTree tree;
    string operation;
    double number;
    int N;

    cin >> N;
    cout << fixed;

    for (size_t i = 0; i < N; i++)
    {
        cin >> operation;
        if (operation != "print")
        {
            cin >> number;
        }

        if (operation == "add")
        {
            tree.add(number);
        }
        else if (operation == "remove")
        {
            tree.remove(number);
        }
        else if (operation == "contains")
        {
            if (tree.contains(number))
            {
                cout << "yes" << endl;
            }
            else
            {
                cout << "no" << endl;
            }
        }
        else if (operation == "print")
        {
            tree.print();
        }
    }

    return 0;
}
```