

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

```
1
2
3 #include <iostream>
4 #include <string>
5 #include <iomanip>
6 bool flag=1;
7 using namespace std;
8
9 struct Node
10 {
11     double value;
12     Node *left;
13     Node *right;
14     int height;
15
16     Node(double value, Node *left, Node *right)
17     {
18         this->value = value;
19         this->left = left;
20         this->right = right;
21     }
22     void calculateHeight(){
23         height=0;
24         if(left){
25             height=max(height,left->height+1);
26         }
27         if(right){
28             height=max(height,right->height+1);
29         }
30     }
31     int leftHeight(){
32         if(left){
33             return left->height+1;
34         }
35         return 0;
36     }
37     int rightHeight(){
38         if(right){
39             return right->height+1;
40         }
41         return 0;
42     }
43     int balance(){
44         return leftHeight()-rightHeight();
45     }
46     void rotateRight(){
47         if(!left){
48             return;
49         }
50         Node* leftRight=this->left->right;
51         Node* oldRight=this->right;
52         swap(this->value,this->left->value);
53         this->right=this->left;
54         this->left=this->left->left;
55         this->right->right=oldRight;
56         this->right->left=leftRight;
57     }
58     void rotateLeft() {
59         if (!right) {
60             return;
61         }
62     }
63
64     Node* rightLeft = this->right->left;
65     Node* oldLeft = this->left;
66
67     swap(this->value, this->right->value);
68     this->left = this->right;
69     this->right = this->right->right;
70     this->left->left = oldLeft;
71     this->left->right = rightLeft;
72 }
73 void recalculateHeights() {
74     if (left) {
75         left->calculateHeight();
76     }
77     if (right) {
78         right->calculateHeight();
79     }
80     this->calculateHeight();
81 }
82 void fixTree(){
83     if(balance()<=-1){
84         if(right->balance()<=-1){
85             this->rotateLeft();
86             recalculateHeights();
87         }
88         else if(right->balance()>=1){
89             right->rotateRight();
90             this->rotateLeft();
91             recalculateHeights();
92         }
93     }
94     else if(balance()>=1){
95         if(left->balance()>=1){
96             this->rotateRight();
97             recalculateHeights();
98         }
99         else if(left->balance()<=-1){
100             left->rotateLeft();
101             this->rotateRight();
102             recalculateHeights();
103         }
104     }
105 }
106 };
107
108 class AVLTree
109 {
110 private:
111     Node *root;
112
113     bool containsRecursive(Node *current, double value)
114     {
115         if (current == NULL)
116         {
117             return false;
118         }
119
120         if (current->value == value)
121         {
122             return true;
123         }
124
125         if (value < current->value)
126         {
127             return containsRecursive(current->left, value);
128         }
129         else
130         {
131             return containsRecursive(current->right, value);
132         }
133     }
134
135     void printRecursive(Node *current)
136     {
137         if (current == NULL)
138         {
139             return;
140         }
141
142         printRecursive(current->left);
143         cout << current->value << " ";
144         printRecursive(current->right);
145     }
146
147 public:
148     AVLTree()
149     {
150         root = NULL;
151     }
152
153     void add(double value)
154     {
155         root=_add(value,root);
156     }
157     Node* _add(double value,Node* current){
158         if(!current){
159             return new Node(value,nullptr,nullptr);
160         }
161         else if(current->value<value){
162             current->right=_add(value,current->right);
163         }
164         else if(current->value>value){
165             current->left=_add(value,current->left);
166         }
167
168         if(current->value==value){
169             cout<<current->value<<" already added"<<endl;
170         }
171         current->calculateHeight();
172         current->fixTree();
173         return current;
174     }
175
176
177     void remove(double value)
178     {
179         root=_remove(value,root);
180         if(flag){
181             cout<<value<<" not found to remove"<<endl;
182         }
183
184         flag=1;
185     }
186
187     Node* _remove(double value,Node* current){
188         if(current==nullptr){
189             return nullptr;
190         }
191         if(value<current->value){
192             current->left=_remove(value,current->left);
193         }
194         else if(value>current->value){
195             current->right=_remove(value,current->right);
196         }
197         else{
198             flag=0;
199             if(!current->left && !current->right){
200                 free(current);
201                 return nullptr;
202             }
203             else if(!current->left){
204                 Node* tempRight=current->right;
205                 free(current);
206                 return tempRight;
207             }
208             else if(!current->right){
209                 Node* tempLeft=current->left;
210                 free(current);
211                 return tempLeft;
212             }
213             else{
214                 Node* swapWith=current->right;
215                 while(swapWith->left){
216                     swapWith=swapWith->left;
217                 }
218                 current->value=swapWith->value;
219                 current->right=_remove(swapWith->value,current->right);
220             }
221         }
222
223
224         current->calculateHeight();
225         current->fixTree();
226         return current;
227     }
228 }
229 bool contains(double value)
230 {
231     if (root == NULL)
232     {
233         return false;
234     }
235     return containsRecursive(root, value);
236 }
237
238 void print()
239 {
240     if (root == NULL)
241     {
242         return;
243     }
244
245     printRecursive(root);
246     cout << endl;
247 }
248 };
249
250 int main()
251 {
252     AVLTree tree;
253     string operation;
254     double number;
255     int N;
256
257     cin >> N;
258     cout << fixed;
259
260     for (size_t i = 0; i < N; i++)
261     {
262         cin >> operation;
263         if (operation != "print")
264         {
265             cin >> number;
266         }
267     }
268
269     if (operation == "add")
270     {
271         tree.add(number);
272     }
273     else if (operation == "remove")
274     {
275         tree.remove(number);
276     }
277     else if (operation == "contains")
278     {
279         if (tree.contains(number))
280         {
281             cout << "yes" << endl;
282         }
283         else
284         {
285             cout << "no" << endl;
286         }
287     }
288     else if (operation == "print")
289     {
290         tree.print();
291     }
292 }
293
294 return 0;
295 }
```