

```
1 //AVL Tree
2 #include <iostream>
3
4 using namespace std;
5
6 struct AVLNode {
7     int key;
8     AVLNode* left, * right;
9     int height;
10 };
11
12 int getHeight(AVLNode* root) {
13     if (!root) return 0;
14     return root->height;
15 }
16
17 void fixHeight(AVLNode*& root) {
18     root->height = 1 + std::max(getHeight(root->left), getHeight(root->right));
19 }
20
21 void leftRotate(AVLNode*& root) {
22     AVLNode* B = root->right;
23     AVLNode* Y = B->left;
24
25     B->left = root;
26     root->right = Y;
27
28     fixHeight(root);
29     fixHeight(B);
30
31     root = B;
32 }
33
34 void rightRotate(AVLNode*& root) {
35     AVLNode* B = root->left;
36     AVLNode* Y = B->right;
37
38     B->right = root;
39     root->left = Y;
40
41     fixHeight(root);
42     fixHeight(B);
43
44     root = B;
45 }
46
47 int getBalanceFactor(AVLNode* node) {
48     return getHeight(node->left) - getHeight(node->right);
```

```
49 }
50
51 void insertAVLNode(AVLNode*& root, int key) {
52     if (!root) {
53         root = new AVLNode{ key, nullptr, nullptr, 1 };
54         return;
55     }
56
57     if (key < root->key) {
58         insertAVLNode(root->left, key);
59     }
60     else if (key > root->key) {
61         insertAVLNode(root->right, key);
62     }
63     else return;
64
65     fixHeight(root);
66
67     int bf = getBalanceFactor(root);
68
69     if (bf > 1 && key < root->left->key) {
70         rightRotate(root);
71         return;
72     }
73
74     if (bf < -1 && key > root->right->key) {
75         leftRotate(root);
76         return;
77     }
78
79     if (bf > 1 && key > root->left->key) {
80         leftRotate(root->left);
81         rightRotate(root);
82         return;
83     }
84
85     if (bf < -1 && key < root->right->key) {
86         rightRotate(root->right);
87         leftRotate(root);
88         return;
89     }
90 }
91
92 AVLNode* findPredecessor(AVLNode* node) {
93     if (!node->left) {
94         std::cout << "This node does not have predecessor!";
95         return nullptr;
96     }
97 }
```

```
198     AVLNode* y = node->left;
199     while (y->right) y = y->right;
200     return y;
201 }
202
203 void deleteAVLNode(AVLNode*& root, int key) {
204     if (!root) return;
205     else if (key < root->key) deleteAVLNode(root->left, key);
206     else if (key > root->key) deleteAVLNode(root->right, key);
207     else {
208         if (!root->left) {
209             AVLNode* temp = root->right;
210             delete root;
211             root = temp;
212             return;
213         }
214         else if (!root->right) {
215             AVLNode* temp = root->left;
216             delete root;
217             root = temp;
218             return;
219         }
220         else {
221             AVLNode* pred = findPredecessor(root);
222             root->key = pred->key;
223
224             deleteAVLNode(root->left, pred->key);
225         }
226     }
227
228     fixHeight(root);
229
230     int bf = getBalanceFactor(root);
231
232     if (bf > 1 && getBalanceFactor(root->left) >= 0) {
233         rightRotate(root);
234     }
235
236     else if (bf < -1 && getBalanceFactor(root->right) <= 0) {
237         leftRotate(root);
238     }
239
240     else if (bf > 1 && getBalanceFactor(root->left) < 0) {
241         leftRotate(root->left);
242         rightRotate(root);
243     }
244
245     else if (bf < -1 && getBalanceFactor(root->right) > 0) {
246         rightRotate(root->right);
```

```
147         leftRotate(root);  
148     }  
149 }
```