```
1 //AVL Tree
 2 #include <iostream>
4 using namespace std;
6 struct AVLNode {
7
       int key;
       AVLNode* left, * right;
8
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;
       root->right = Y;
26
27
       fixHeight(root);
28
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) {
       return getHeight(node->left) - getHeight(node->right);
48
```

```
...-HCMUS\Documents\repos\DSA\Final-LyThuyet\AVLTree.cpp
```

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

```
\verb|...-HCMUS\Documents\repos\DSA\Final-LyThuyet\AVLTree.cpp|
```

```
98
         AVLNode* y = node->left;
 99
         while (y->right) y = y->right;
100
         return y;
101 }
102
103 void deleteAVLNode(AVLNode*& root, int key) {
104
         if (!root) return;
105
         else if (key < root->key) deleteAVLNode(root->left, key);
         else if (key > root->key) deleteAVLNode(root->right, key);
106
         else {
107
             if (!root->left) {
108
                 AVLNode* temp = root->right;
109
110
                 delete root;
111
                 root = temp;
112
                 return;
113
             }
114
             else if (!root->right) {
                 AVLNode* temp = root->left;
115
116
                 delete root;
117
                 root = temp;
                 return;
118
119
             }
120
             else {
                 AVLNode* pred = findPredecessor(root);
121
122
                 root->key = pred->key;
123
                 deleteAVLNode(root->left, pred->key);
124
125
             }
         }
126
127
         fixHeight(root);
128
129
130
         int bf = getBalanceFactor(root);
131
         if (bf > 1 && getBalanceFactor(root->left) >= 0) {
132
133
             rightRotate(root);
134
         }
135
         else if (bf < -1 && getBalanceFactor(root->right) <= 0) {</pre>
136
137
             leftRotate(root);
         }
138
139
140
         else if (bf > 1 && getBalanceFactor(root->left) < 0) {</pre>
141
             leftRotate(root->left);
142
             rightRotate(root);
143
         }
144
145
         else if (bf < -1 && getBalanceFactor(root->right) > 0) {
146
             rightRotate(root->right);
```