

BST and AVL implementation Assignment

Rishi Yadav

November 2020

1. Beginning with an empty binary search tree with complexity analysis, Construct binary search tree by inserting the values in the order given. After constructing a binary tree –
 - (a) Insert new node
 - (b) Find number of nodes in longest path
 - (c) Minimum data value found in the tree
 - (d) How to delete a node from BST with the help of all three cases
 - (e) Search a value

```
1
2
3      #include <iostream>
4      #include <climits>
5      using namespace std;
6
7
8      class Node{
9
10     public:
11         int data;
12         Node *left;
13         Node *right;
14     };
15
16     Node *root;
17
18     Node *getNode(int node_value)
19     {
20         Node *temp = new Node;
21         temp -> data = node_value;
22         temp -> left = NULL;
23         temp -> right = NULL;
24         return temp;
25     }
26
27     void insert(int node_value)
28     {
29         /*
30          case 1 - when tree is empty
31             simply create a node and insert to root;
32          case 2 - when value is less or equal than root and left
33                  pointer points to null
34             create a node
35             just insert to left
36          case 3 - when value is greater than root and right
37                  pointer points to null
38             create a node
39             just insert to right
40          case 4 - when both pointers are not null
41             if value less than root
42                 traverse to left subtree
43             if value greater than or equal than root
44                 traverse to right subtree
45         */
46     }
```

```

43     */
44
45     Node *temp = getNode(node_value);
46
47     if (root == NULL)
48     {
49         root = temp;
50         return;
51     }
52     Node *curr_ptr = root;
53     while(curr_ptr!=NULL)
54     {
55         if(node_value <= curr_ptr->data && curr_ptr -> left ==
56 NULL)
57         {
58             curr_ptr->left = temp;
59             return;
60         }
61         else if(node_value > curr_ptr->data && curr_ptr ->
62 right == NULL)
63         {
64             curr_ptr -> right = temp;
65             return;
66         }
67         else if(node_value <= curr_ptr->data && curr_ptr ->
68 left != NULL)
69         {
70             curr_ptr = curr_ptr -> left;
71             else
72             curr_ptr = curr_ptr -> right;
73         }
74     }
75
76 void inorder(Node *local_root)
77 {
78     if(local_root == NULL) return;
79     inorder(local_root->left);
80     cout<<local_root->data<<" ";
81     inorder(local_root->right);
82 }
83
84 int maxLengthBST(Node *local_root)
85 {
86     if(local_root == NULL) return 0;
87
88     else
89     {
90         int leftNodes = maxLengthBST(local_root->left);
91         int rightNodes = maxLengthBST(local_root->right);
92
93         if (leftNodes > rightNodes) return leftNodes + 1;
94         else return rightNodes + 1;
95     }
96 }
97
98 int minimum(int x, int y, int z) {

```

```

97     int min = x; /* assume x is the smallest */
98
99     if (y < min) { /* if y is smaller than min, assign y to
100 min */
101         min = y;
102     } /* end if */
103
104     if (z < min) { /* if z is smaller than min, assign z to
105 min */
106         min = z;
107     } /* end if */
108
109     return min; /* max is the largest value */
110 }
111
112 int minData(Node *local_root)
113 {
114     if(local_root == NULL) return INT_MAX;
115     else
116     {
117         int minValue = local_root->data;
118
119         minValue = minimum(minValue,
120             minData(local_root->left),
121             minData(local_root->right));
122
123         return minValue;
124     }
125 }
126
127 void search(Node *lRoot, int key)
128 {
129     if(lRoot == NULL)
130     {
131         cout<<"Element not found\n";
132         return;
133     }
134     Node *curr_ptr = lRoot;
135     while(curr_ptr!=NULL)
136     {
137         if(curr_ptr->data == key)
138         {
139             cout<<"\nElement found\n";
140             return;
141             //return curr_ptr;
142         }
143         else if(curr_ptr->data > key)
144             curr_ptr = curr_ptr->left;
145         else
146             curr_ptr = curr_ptr->right;
147     }
148     cout<<"\nElement not found\n";
149     return ;
150 }
151
152 Node *findMin(Node *ptr)
153 {

```

```

152     Node *curr = ptr;
153     while(curr && curr -> left != NULL)
154         curr = curr -> left;
155
156     return curr;
157 }
158
159 Node* deleteNode(Node *lRoot, int value)
160 {
161     if(lRoot == NULL) return lRoot;
162     if(value < lRoot -> data)
163         lRoot -> left = deleteNode(lRoot -> left, value);
164     else if(value > lRoot -> data)
165         lRoot -> right = deleteNode(lRoot -> right, value);
166     else
167     {
168         if(lRoot -> left == NULL)
169         {
170             Node *temp = lRoot -> right;
171             free(lRoot);
172             return temp;
173         }
174         else if(lRoot -> right == NULL)
175         {
176             Node *temp = lRoot -> left;
177             free(lRoot);
178             return temp;
179         }
180
181         //Inorder successor (min value in right subtree)
182         Node *temp = findMin(lRoot->right);
183
184         lRoot -> data = temp -> data;
185
186         lRoot -> right = deleteNode(lRoot -> right, temp ->
data);
187     }
188     return lRoot;
189 }
190
191 int main() {
192
193     int total_nodes; cin>>total_nodes;
194     root = NULL;
195
196     for(int i = 0; i<total_nodes; i++)
197     {
198         int node_val; cin>>node_val;
199         insert(node_val);
200         //insert(5);
201         //insert(6);
202         //insert(1);
203     }
204
205     cout<<"The inorder traversal of BST is :\n";
206     inorder(root);
207

```

```

208     cout<<"\nThe number of nodes in longest path of BST is :
    "<<
209     maxLengthBST(root);
210
211     cout<<"\nMinimum Data value found in the tree is :"<<
212     minData(root);
213
214     int value;
215     cout<<"\nEnter which element you want to delete :";
216     cin>> value;
217
218     //search(root, value);
219
220     deleteNode(root, value);
221     cout<<"\nThe inorder traversal of BST after deletion is
    : \n";
222     inorder(root);
223
224     cout<<"\nEnter the element you want to search for?";
225     cin>>value;
226     search(root, value);
227
228
229     return 0;
230 }
231

```

Output

```

(base) rishi@rishi-X541UAK:~/Programs$ ./bst
5
7 2 3 1 9
The inorder traversal of BST is :
1 2 3 7 9
The number of nodes in longest path of BST is :3
Minimum Data value found in the tree is :1
Enter which element you want to delete :1

The inorder traversal of BST after deletion is :
2 3 7 9
Enter the element you want to search for?:3

Element found
(base) rishi@rishi-X541UAK:~/Programs$ █

```

2. Write a program to implement AVL Tree with various operations of Insertion, deletion and searching.

```

1
2 #include<bits/stdc++.h>
3
4 using namespace std;
5 struct Node {
6     int key;
7     Node * left;
8     Node * right;
9     int height;
10 };
11 int max(int a, int b);
12 int height(Node * N) {

```

```

13     if (N == NULL)
14         return 0;
15     return N -> height;
16 }
17 int max(int a, int b) {
18     return (a > b) ? a : b;
19 }
20 Node * newNode(int key) {
21     Node * node = new Node();
22     node -> key = key;
23     node -> left = NULL;
24     node -> right = NULL;
25     node -> height = 1;
26     return (node);
27 }
28 Node * rightRotate(Node * y) {
29     Node * x = y -> left;
30     Node * T2 = x -> right;
31     x -> right = y;
32     y -> left = T2;
33     y -> height = max(height(y -> left), height(y -> right)) +
34         1;
35     x -> height = max(height(x -> left), height(x -> right)) +
36         1;
37     return x;
38 }
39 Node * leftRotate(Node * x) {
40     Node * y = x -> right;
41     Node * T2 = y -> left;
42     y -> left = x;
43     x -> right = T2;
44     x -> height = max(height(x -> left), height(x -> right)) +
45         1;
46     y -> height = max(height(y -> left), height(y -> right)) +
47         1;
48     return y;
49 }
50 int getBalance(Node * N) {
51     if (N == NULL)
52         return 0;
53     return height(N -> left) - height(N -> right);
54 }
55 Node * insert(Node * node, int key) {
56     if (node == NULL)
57         return (newNode(key));
58     if (key < node -> key)
59         node -> left = insert(node -> left, key);
60     else if (key > node -> key)
61         node -> right = insert(node -> right, key);
62     else
63         return node;
64     node -> height = 1 + max(height(node -> left), height(node
65 -> right));
66     int balance = getBalance(node);
67     if (balance > 1 && key < node -> left -> key)
68         return rightRotate(node);
69     if (balance < -1 && key > node -> right -> key)

```

```

65     return leftRotate(node);
66     if (balance > 1 && key > node -> left -> key) {
67         node -> left = leftRotate(node -> left);
68         return rightRotate(node);
69     }
70     if (balance < -1 && key < node -> right -> key) {
71         node -> right = rightRotate(node -> right);
72         return leftRotate(node);
73     }
74     return node;
75 }
76 Node * minValueNode(Node * node) {
77     Node * current = node;
78     while (current -> left != NULL)
79         current = current -> left;
80     return current;
81 }
82 Node * deleteNode(Node * root, int key) {
83     if (root == NULL)
84         return root;
85     if (key < root -> key)
86         root -> left = deleteNode(root -> left, key);
87     else if (key > root -> key)
88         root -> right = deleteNode(root -> right, key);
89     else {
90         if ((root -> left == NULL) || (root -> right == NULL))
91         {
92             Node * temp = root -> left ? root -> left : root
93             -> right;
94             if (temp == NULL) {
95                 temp = root;
96                 root = NULL;
97             } else
98                 *root = * temp;
99             free(temp);
100         } else {
101             Node * temp = minValueNode(root -> right);
102             root -> key = temp -> key;
103             root -> right = deleteNode(root -> right,
104                                     temp -> key);
105         }
106     }
107     if (root == NULL)
108         return root;
109     root -> height = 1 + max(height(root -> left),
110                             height(root -> right));
111     int balance = getBalance(root);
112     if (balance > 1 && getBalance(root -> left) >= 0)
113         return rightRotate(root);
114     if (balance > 1 && getBalance(root -> left) < 0) {
115         root -> left = leftRotate(root -> left);
116         return rightRotate(root);
117     }
118     if (balance < -1 && getBalance(root -> right) <= 0)
119         return leftRotate(root);
120     if (balance < -1 && getBalance(root -> right) > 0) {
121         root -> right = rightRotate(root -> right);

```



```

120         return leftRotate(root);
121     }
122     return root;
123 }
124 struct Node * search(struct Node * root, int key) {
125     if (root == NULL) {
126         cout << "Element not found";
127         return root;
128     }
129     if (root -> key == key) {
130         cout << "\nElement found";
131         return root;
132     }
133     if (root -> key < key)
134         return search(root -> right, key);
135     return search(root -> left, key);
136 }
137 void inorder(Node * root) {
138     if (root != NULL) {
139         inorder(root -> left);
140         cout << root -> key << " ";
141         inorder(root -> right);
142     }
143 }
144 int main() {
145     Node * root = NULL;
146     char ch;
147     int x;
148     do {
149         cout << "Enter value to be inserted\t\t";
150         cin >> x;
151         root = insert(root, x);
152         cout << "\nInorder traversal of the AVL tree is \n";
153         inorder(root);
154         cout << "\nEnter y to add more elements\t";
155         cin >> ch;
156     } while (ch == 'y');
157     do {
158         cout << "Enter 1 to delete\n2 to search ";
159         cin >> x;
160         switch (x) {
161             case 1:
162                 cout << "\nEnter value to be deleted :";
163                 cin >> x;
164                 root = deleteNode(root, x);
165                 cout << "\nInorder traversal after deletion :";
166                 inorder(root);
167                 break;
168             case 2:
169                 cout << "\nEnter value to be searched :";
170                 cin >> x;
171                 search(root, x);
172                 break;
173             default:
174                 cout << "\nEnter correct option";
175         }
176         cout << "\nEnter y to repeat\t";

```

```

177         cin >> ch;
178     } while (ch == 'y');
179     return 0;
180 }
181

```

Output

```

(base) rtshi@rtshi-X541UAK:~/Programs$ g++ avl.cpp -o avl
(base) rtshi@rtshi-X541UAK:~/Programs$ ./avl
Enter value to be inserted: 5

Inorder traversal of the AVL tree is:
5
Enter y to add more elements: y
Enter value to be inserted: 6

Inorder traversal of the AVL tree is:
5 6
Enter y to add more elements: y
Enter value to be inserted: 7

Inorder traversal of the AVL tree is:
5 6 7
Enter y to add more elements: n
Enter 1 to delete
2 to search :1

Enter value to be deleted :7

Inorder traversal after deletion :5 6
Enter y to repeat      n
(base) rtshi@rtshi-X541UAK:~/Programs$

```