LAPORAN KELOMPOK 7 : BINARY SEARCH TREE

Struktur Data

Oleh:
1. Revina Augriha Firdaus (21091397003)
2. Ummiyatun (21091397039)
3. Imas Dewi Orvala Nathania Insani (21091397053)
4. Riska Octavia Arianto (21091397059)
5. Rendi Nicolas Mahendra (21091397071)

D4 MANAJEMEN INFORMATIKA
UNIVERSITAS NEGERI SURABAYA
2021/2022

C++ Binary Search Tree

```cpp
//Kelompok7
#include<iostream>
#define SPACE 10

using namespace std;

class TreeNode {
  public:
    int value;
  TreeNode * left;
  TreeNode * right;

  TreeNode() {
    value = 0;
    left = NULL;
    right = NULL;
  }
  TreeNode(int v) {
    value = v;
    left = NULL;
    right = NULL;
  }
};

class BST {
  public:
    TreeNode * root;
  BST() {
    root = NULL;
  }
  bool isTreeEmpty() {
    if (root == NULL) {
      return true;
    } else {
      return false;
    }
```

```cpp
  }

  void insertNode(TreeNode * new_node) {
    if (root == NULL) {
      root = new_node;
      cout << "Value Inserted as root node!" << endl;
    } else {
      TreeNode * temp = root;
      while (temp != NULL) {
        if (new_node -> value == temp -> value) {
          cout << "Value Already exist," <<
            "Insert another value!" << endl;
          return;
        } else if ((new_node -> value < temp -> value) && (temp -> left == NULL)) {
          temp -> left = new_node;
          cout << "Value Inserted to the left!" << endl;
          break;
        } else if (new_node -> value < temp -> value) {
          temp = temp -> left;
        } else if ((new_node -> value > temp -> value) && (temp -> right == NULL)) {
          temp -> right = new_node;
          cout << "Value Inserted to the right!" << endl;
          break;
        } else {
          temp = temp -> right;
        }
      }
    }
  }
    TreeNode* insertRecursive(TreeNode *r, TreeNode *new_node)
    {
        if(r==NULL)
        {
            r=new_node;
            cout <<"Insertion successful"<<endl;
            return r;
```

```
70              r=new_node;
71              cout <<"Insertion successful"<<endl;
72              return r;
73          }
74
75          if(new_node->value < r->value)
76          {
77              r->left = insertRecursive(r->left,new_node);
78          }
79          else if (new_node->value > r->value)
80          {
81              r->right = insertRecursive(r->right,new_node);
82          }
83         else
84          {
85              cout << "No duplicate values allowed!" << endl;
86              return r;
87          }
88          return r;
89      }
90
91  void print2D(TreeNode * r, int space) {
92    if (r == NULL) // Base case  1
93      return;
94    space += SPACE; // Increase distance between levels   2
95    print2D(r -> right, space); // Process right child first 3
96    cout << endl;
97    for (int i = SPACE; i < space; i++) // 5
98      cout << " "; // 5.1
99    cout << r -> value << "\n"; // 6
100   print2D(r -> left, space); // Process left child  7
101 }
102
103 void printPreorder(TreeNode * r) //(current node, Left, Right)
104 {
105   if (r == NULL)
```

[*] BST.cpp

```
103 void printPreorder(TreeNode * r) //(current node, Left, Right)
104 {
105   if (r == NULL)
106     return;
107   /* first print data of node */
108   cout << r -> value << " ";
109   /* then recur on left sutree */
110   printPreorder(r -> left);
111   /* now recur on right subtree */
112   printPreorder(r -> right);
113 }
114
115 void printInorder(TreeNode * r) //  (Left, current node, Right)
116 {
117   if (r == NULL)
118     return;
119   /* first recur on left child */
120   printInorder(r -> left);
121   /* then print the data of node */
122   cout << r -> value << " ";
123   /* now recur on right child */
124   printInorder(r -> right);
125 }
126 void printPostorder(TreeNode * r) //(Left, Right, Root)
127 {
128   if (r == NULL)
129     return;
130   // first recur on left subtree
131   printPostorder(r -> left);
132   // then recur on right subtree
133   printPostorder(r -> right);
134   // now deal with the node
135   cout << r -> value << " ";
136 }
137
138 TreeNode * iterativeSearch(int v) {
```

```
}

TreeNode * iterativeSearch(int v) {
  if (root == NULL) {
    return root;
  } else {
    TreeNode * temp = root;
    while (temp != NULL) {
      if (v == temp -> value) {
        return temp;
      } else if (v < temp -> value) {
        temp = temp -> left;
      } else {
        temp = temp -> right;
      }
    }
    return NULL;
  }
}

TreeNode * recursiveSearch(TreeNode * r, int val) {
  if (r == NULL || r -> value == val)
    return r;

  else if (val < r -> value)
    return recursiveSearch(r -> left, val);

  else
    return recursiveSearch(r -> right, val);
}

int height(TreeNode * r) {
  if (r == NULL)
    return -1;
  else {
    /* compute the height of each subtree */
```

```
      return -1;
    else {
      /* compute the height of each subtree */
      int lheight = height(r -> left);
      int rheight = height(r -> right);

      /* use the larger one */
      if (lheight > rheight)
        return (lheight + 1);
      else return (rheight + 1);
    }
  }

  /* Print nodes at a given level */
  void printGivenLevel(TreeNode * r, int level) {
    if (r == NULL)
      return;
    else if (level == 0)
      cout << r -> value << " ";
    else // level > 0
    {
      printGivenLevel(r -> left, level - 1);
      printGivenLevel(r -> right, level - 1);
    }
  }
  void printLevelOrderBFS(TreeNode * r) {
    int h = height(r);
    for (int i = 0; i <= h; i++)
      printGivenLevel(r, i);
  }

  TreeNode * minValueNode(TreeNode * node) {
    TreeNode * current = node;
    /* loop down to find the leftmost leaf */
    while (current -> left != NULL) {
      current = current -> left;
```

```
202       /* Loop down to find the leftmost leaf */
203       while (current -> left != NULL) {
204         current = current -> left;
205       }
206       return current;
207     }
208
209     TreeNode * deleteNode(TreeNode * r, int v) {
210       // base case
211       if (r == NULL) {
212         return NULL;
213       }
214       // If the key to be deleted is smaller than the root's key,
215       // then it lies in left subtree
216       else if (v < r -> value) {
217         r -> left = deleteNode(r -> left, v);
218       }
219       // If the key to be deleted is greater than the root's key,
220       // then it lies in right subtree
221       else if (v > r -> value) {
222         r -> right = deleteNode(r -> right, v);
223       }
224       // if key is same as root's key, then This is the node to be deleted
225       else {
226         // node with only one child or no child
227         if (r -> left == NULL) {
228           TreeNode * temp = r -> right;
229           delete r;
230           return temp;
231         } else if (r -> right == NULL) {
232           TreeNode * temp = r -> left;
233           delete r;
234           return temp;
235         } else {
236           // node with two children: Get the inorder successor (smallest
237           // in the right subtree)
```

```
235         } else {
236           // node with two children: Get the inorder successor (smallest
237           // in the right subtree)
238           TreeNode * temp = minValueNode(r -> right);
239           // Copy the inorder successor's content to this node
240           r -> value = temp -> value;
241           // Delete the inorder successor
242           r -> right = deleteNode(r -> right, temp -> value);
243           //deleteNode(r->right, temp->value);
244         }
245       }
246       return r;
247     }
248
249   };
250
251   int main() {
252     BST obj;
253     int option, val;
254
255     do {
256       cout << "What operation do you want to perform? " <<
257         " Select Option number. Enter 0 to exit." << endl;
258       cout << "1. Insert Node" << endl;
259       cout << "2. Search Node" << endl;
260       cout << "3. Delete Node" << endl;
261       cout << "4. Print/Traversal BST values" << endl;
262       cout << "5. Height of Tree" << endl;
263       cout << "6. Clear Screen" << endl;
264       cout << "0. Exit Program" << endl;
265
266       cin >> option;
267       //Node n1;
268       TreeNode * new_node = new TreeNode();
269
270       switch (option) {
```

```
268         TreeNode * new_node = new TreeNode();
269
270         switch (option) {
271         case 0:
272           break;
273         case 1:
274             cout <<"INSERT"<<endl;
275               cout <<"Enter VALUE of TREE NODE to INSERT in BST: ";
276               cin >> val;
277               new_node->value = val;
278               obj.root= obj.insertRecursive(obj.root,new_node);
279               //obj.insertNode(new_node);
280               cout<<endl;
281                 break;
282
283         case 2:
284           cout << "SEARCH" << endl;
285           cout << "Enter VALUE of TREE NODE to SEARCH in BST: ";
286           cin >> val;
287           //new_node = obj.iterativeSearch(val);
288           new_node = obj.recursiveSearch(obj.root, val);
289           if (new_node != NULL) {
290             cout << "Value found" << endl;
291           } else {
292             cout << "Value NOT found" << endl;
293           }
294           break;
295         case 3:
296           cout << "DELETE" << endl;
297           cout << "Enter VALUE of TREE NODE to DELETE in BST: ";
298           cin >> val;
299           new_node = obj.iterativeSearch(val);
300           if (new_node != NULL) {
301             obj.deleteNode(obj.root, val);
302             cout << "Value Deleted" << endl;
303           } else {
```

```
302             cout << "Value Deleted" << endl;
303           } else {
304             cout << "Value NOT found" << endl;
305           }
306           break;
307         case 4:
308           cout << "PRINT 2D: " << endl;
309           obj.print2D(obj.root, 5);
310           cout << endl;
311           cout << "Print Level Order BFS: \n";
312           obj.printLevelOrderBFS(obj.root);
313           cout << endl;
314           //          cout <<"PRE-ORDER: ";
315           //          obj.printPreorder(obj.root);
316           //          cout<<endl;
317           //          cout <<"IN-ORDER: ";
318           //          obj.printInorder(obj.root);
319           //          cout<<endl;
320           //          cout <<"POST-ORDER: ";
321           //          obj.printPostorder(obj.root);
322           break;
323         case 5:
324           cout << "TREE HEIGHT" << endl;
325           cout << "Height : " << obj.height(obj.root) << endl;
326           break;
327         case 6:
328           system("cls");
329           break;
330         default:
331           cout << "Enter Proper Option number " << endl;
332         }
333
334       } while (option != 0);
335
336       return 0;
337     }
```

INPUT & OUTPUT

```
C:\Users\Asuspro\Documents\Semeseter 2\Tugas\Prak.Struktur data\New folder\BST.exe
What operation do you want to perform?  Select Option number. Enter 0 to exit.
1. Insert Node
2. Search Node
3. Delete Node
4. Print/Traversal BST values
5. Height of Tree
6. Clear Screen
0. Exit Program
1
INSERT
Enter VALUE of TREE NODE to INSERT in BST: 8
Insertion successful

What operation do you want to perform?  Select Option number. Enter 0 to exit.
1. Insert Node
2. Search Node
3. Delete Node
4. Print/Traversal BST values
5. Height of Tree
6. Clear Screen
0. Exit Program
1
INSERT
Enter VALUE of TREE NODE to INSERT in BST: 5
Insertion successful

What operation do you want to perform?  Select Option number. Enter 0 to exit.
1. Insert Node
2. Search Node
3. Delete Node
4. Print/Traversal BST values
5. Height of Tree
6. Clear Screen
0. Exit Program
1
INSERT
Enter VALUE of TREE NODE to INSERT in BST: 10
Insertion successful

What operation do you want to perform?  Select Option number. Enter 0 to exit.
1. Insert Node
2. Search Node
3. Delete Node
```

```
C:\Users\Asuspro\Documents\Semeseter 2\Tugas\Prak.Struktur data\New folder\BST.exe
What operation do you want to perform?  Select Option number. Enter 0 to exit.
1. Insert Node
2. Search Node
3. Delete Node
4. Print/Traversal BST values
5. Height of Tree
6. Clear Screen
0. Exit Program
1
INSERT
Enter VALUE of TREE NODE to INSERT in BST: 2
Insertion successful

What operation do you want to perform?  Select Option number. Enter 0 to exit.
1. Insert Node
2. Search Node
3. Delete Node
4. Print/Traversal BST values
5. Height of Tree
6. Clear Screen
0. Exit Program
1
INSERT
Enter VALUE of TREE NODE to INSERT in BST: 6
Insertion successful

What operation do you want to perform?  Select Option number. Enter 0 to exit.
1. Insert Node
2. Search Node
3. Delete Node
4. Print/Traversal BST values
5. Height of Tree
6. Clear Screen
0. Exit Program
1
INSERT
Enter VALUE of TREE NODE to INSERT in BST: 9
Insertion successful

What operation do you want to perform?  Select Option number. Enter 0 to exit.
1. Insert Node
2. Search Node
3. Delete Node
```

```
C:\Users\Asuspro\Documents\Semeseter 2\Tugas\Prak.Struktur data\New folder\BST.exe
What operation do you want to perform?  Select Option number. Enter 0 to exit.
1. Insert Node
2. Search Node
3. Delete Node
4. Print/Traversal BST values
5. Height of Tree
6. Clear Screen
0. Exit Program
1
INSERT
Enter VALUE of TREE NODE to INSERT in BST: 11
Insertion successful

What operation do you want to perform?  Select Option number. Enter 0 to exit.
1. Insert Node
2. Search Node
3. Delete Node
4. Print/Traversal BST values
5. Height of Tree
6. Clear Screen
0. Exit Program
4
PRINT 2D:

                        11

               10

                        9

     8

                        6

               5

                        2

Print Level Order BFS:
8 5 10 2 6 9 11
What operation do you want to perform?  Select Option number. Enter 0 to exit.
1. Insert Node
2. Search Node
```