

Homework: Binary Search Tree Data Structure

Patiwet Wuttisarnwattana, Ph.D.

Department of Computer Engineering

Chiang Mai University

การบ้านนี้ให้นักศึกษา implement Binary Search Tree (BST) โดยใช้ starter code ของอาจารย์ โดยมีคำอธิบายแต่ละส่วนดังต่อไปนี้

1. อาจารย์ได้สร้าง class Node ที่สามารถบรรจุ key ได้ค่า integer ได้ เสร็จเรียบร้อยแล้ว
 - a. Node หนึ่ง Node สามารถชี้ไปยัง ลูกคนซ้าย (left child) และลูกคนขวา (right child) ได้ และสามารถชี้กลับไปยัง parent ด้วยได้
2. อาจารย์ได้ร่าง class Tree ไว้เป็น Template ซึ่งคลาสนี้จะทำหน้าที่ประมวลผล Node ตามคุณสมบัติของ BST
 - a. ขอให้ นศ เริ่มทำการบ้านโดยการให้ class Tree นี้ ทำการสืบทอดคุณสมบัติ (inherit) class BTreePrinter ที่อาจารย์แนบมา เพื่อที่คลาสนี้จะได้สามารถแสดงแผนภาพต้นไม้ออกมาสวย ๆ ดังตัวอย่าง ผ่านฟังก์ชันที่ชื่อว่า printTree()
 - เมื่อ นศ ทำการสืบทอดคุณสมบัติของ class BTreePrinter แล้ว ขอให้ นศ Uncomment โค้ดของอาจารย์ที่ฟังก์ชัน printTree() แล้วเพิ่มเติมให้เหมาะสมและทำงานได้ตรงตาม Testcase
 - b. ใน class Tree ขอให้ นศ สร้าง function ที่ชื่อว่า static void printNode(Node node) เพื่อทำการพิมพ์ค่า node.key ออกทาง console หาก node ที่รับเข้ามานั้นเป็น null ให้พิมพ์ออก console “Node not found!!!”
3. เป็นข้อตกลงของวิชานี้ว่า เมื่อไหร่ก็ตามที่อาจารย์เขียนฟังก์ชันลักษณะเป็น static นำหน้า นั่นจะเป็นสัญญาณให้นักศึกษา รู้ว่า ฟังก์ชันนั้นจะต้องมีการเรียกตัวเองหรือเป็น Recursive function
4. ให้ class Tree มี method ดังต่อไปนี้
 - a. public static Node find(Node node, int search_key) ทำหน้าที่หา node ที่บรรจุ search_key แบบ recursive ตามที่เรียนในห้อง
 - b. public Node find(int search_key) ทำหน้าที่หา Node ที่บรรจุใน Tree ที่มี key ดังที่ระบุ โดยมีการเรียกใช้ static find() อีกทอดหนึ่ง
 - c. public static Node findMin(Node node) ทำหน้าที่หา node ที่มีค่า key น้อยที่สุด แบบ recursive ตามที่เรียนในห้อง
 - d. public Node findMin() ทำหน้าที่หา Node ที่บรรจุใน Tree ที่มีค่า key น้อยที่สุด โดยมีการเรียกใช้ static findMin () อีกทอดหนึ่ง

- e. `public static Node findMax(Node node)` ทำหน้าที่หา node ที่มีค่า key มากที่สุด แบบ recursive ตามที่เรียนในห้อง
- f. `public Node findMax()` ทำหน้าที่หา Node ที่บรรจุใน Tree ที่มีค่า key มากที่สุด โดยมีการเรียกใช้ `static findMax ()` อีกทอดหนึ่ง
- g. `public static Node findClosestLeaf(Node node, int search_key)` ทำหน้าที่หา Node ที่มี null Node ที่สามารถนำ `search_key` ไปห้อยเป็น Node ใหม่ได้ แบบ recursive ตามที่เรียนในห้อง โดยเริ่มที่ node ใด
- h. `public Node findClosestLeaf(int search_key)` ทำหน้าที่หา Node ที่มี null Node ที่สามารถนำ `search_key` ไปห้อยเป็น Node ใหม่ได้ โดยเริ่มจาก root ay ฟังก์ชันนี้จะเรียก `findClosestLeaf ()` อีกทอดหนึ่ง
- i. `public Node findClosest(int search_key)` ทำหน้าที่หา Node ที่มี key ใกล้เคียงกับ `search_key` มากที่สุด
 - เฉพาะฟังก์ชันนี้ในการบ้านนี้ กำหนดให้ใช้ `while loop` ทำ อย่าใช้ Recursive นะครับ
- j. `public void insert(int key)` ทำหน้าที่สร้าง Node ใหม่ที่บรรจุค่า key แล้วนำไปต่อใน BST ตามที่เรียนในห้อง (ให้ทำ `findClosestLeaf` แล้วเอา Key ใหม่ไปห้อย) ถ้าหาก key ที่บรรจุเข้ามาใหม่มีอยู่แล้วใน Node ใด Node หนึ่งของ Tree ให้พิมพ์ออกทางหน้าจอว่า “Duplicated key!!!” แล้วไม่ต้องทำอะไร
 - เฉพาะฟังก์ชันนี้ในการบ้านนี้ กำหนดให้เรียกใช้บริการ `findClosestLeaf()` นะครับ อย่าทำ `insert` แบบ Recursive
- k. `public void printPreOrderDFT()` และคู่หู static function ที่มีชื่อเดียวกัน ให้ทำหน้าที่พิมพ์ค่า key ของทุก ๆ Node ตามลำดับ Pre-Order Depth First Traversal ตามที่เรียนในห้อง
 - ให้ pattern การพิมพ์ออกทาง console ให้เป็นไปดังตัวอย่างด้านล่าง เริ่มต้นด้วยคำว่า “PreOrder DFT node sequence [” ลงท้ายด้วย “]”
- l. `public void printInOrderDFT()` และคู่หู static function ที่มีชื่อเดียวกัน ให้ทำหน้าที่พิมพ์ค่า key ของทุก ๆ Node ตามลำดับ In-Order Depth First Traversal ตามที่เรียนในห้อง
 - ให้ pattern การพิมพ์ออกทาง console ให้เป็นไปดังตัวอย่างด้านล่าง เริ่มต้นด้วยคำว่า “InOrder DFT node sequence [” ลงท้ายด้วย “]”
- m. `public void printPostOrderDFT()` และคู่หู static function ที่มีชื่อเดียวกัน ให้ทำหน้าที่พิมพ์ค่า key ของทุก ๆ Node ตามลำดับ Post-Order Depth First Traversal ตามที่เรียนในห้อง
 - ให้ pattern การพิมพ์ออกทาง console ให้เป็นไปดังตัวอย่างด้านล่าง เริ่มต้นด้วยคำว่า “PostOrder DFT node sequence [” ลงท้ายด้วย “]”
- n. `public static int height(Node node)` ทำหน้าที่หาว่า node นี้อยู่ที่ความสูงที่เท่าไรเมื่อเทียบกับลูกที่อยู่ลึกที่สุด ... นศ จงจำไว้ว่า เมื่อไรก็ตามที่อาจารย์เขียนฟังก์ชันลักษณะเป็น static นำหน้า จะเป็นสัญญาณให้ นศ รู้ว่าฟังก์ชันนี้ควรจะมีการเรียกตัวเอง (Recursion)
- o. `public static int size(Node node)` ทำหน้าที่หาว่า node กับลูก ๆ รวมกันแล้วมีจำนวนเท่าไร (ตามความหมายของการหา Tree size) ... นศ จงจำไว้ว่า เมื่อไรก็ตามที่อาจารย์เขียนฟังก์ชันลักษณะเป็น static นำหน้า จะเป็นสัญญาณให้ นศ รู้ว่าฟังก์ชันนี้ควรจะมีการเรียกตัวเอง (Recursion)

- p. `public static int depth(Node root, Node node)` ทำหน้าที่หาว่า หากกำหนดให้ node นี้เป็นส่วนหนึ่งของ Tree แล้ว node นี้จะมีความลึกเป็นเท่าไร เมื่อเทียบกับ root (ตามความหมายของการหา Node depth)
- q. `public int height()` ทำหน้าที่คำนวณว่าต้นไม้ต้นนี้มีความสูงเท่าไร จงเขียนโค้ดในหนึ่งบรรทัด (คำใบ้: เรียกใช้บริการ `static height(Node node)`)
- r. `public int size()` ทำหน้าที่คำนวณว่าต้นไม้ต้นนี้มีบรรจุ Node ไปแล้วทั้งหมดกี่ Node จงเขียนโค้ดในหนึ่งบรรทัด (คำใบ้: เรียกใช้บริการ `static size(Node node)`)
- s. `public int depth()` ทำหน้าที่คำนวณว่าต้นไม้ต้นนี้มีความลึกเท่าไร จงเขียนโค้ดในหนึ่งบรรทัด (คำใบ้: เรียกใช้บริการ `static depth(Node root, Node node)`)
- t. `public Node findKthSmallest(int k)` และคู่มือ static function ที่มีชื่อเดียวกัน ให้ทำหน้าที่หา Node ที่บรรจุใน Tree ที่มีค่า key เล็กเป็นอันดับที่ k ($k=1$ แปลว่า มีค่า key เล็กที่สุด, $k=2$ แปลว่า มีค่า key เล็กเป็นอันดับที่สอง)
- u. `public static Node findNext(Node node)` ทำหน้าที่หาว่า Node ที่มีค่า key อยู่มากกว่าขึ้นไปอีกค่าหนึ่งของ node (input) คือ Node ไດ
- v. `public static Node leftDescendant(Node node)` ทำหน้าที่หา descendant Node ที่อยู่ด้านซ้ายสุดของ node (ซึ่งทำงานเหมือน `findMin` นั้นแหละ)
- w. `public static Node rightAncestor(Node node)` ทำหน้าที่หา ancestor Node ที่อยู่ด้านขวาแรกของ node
- x. `public List rangeSearch(int x, int y)` ทำหน้าที่ค้นหา Node ที่มี key อยู่ระหว่างค่า x กับค่า y โดยค่า $x \leq \text{key} \leq y$ ซึ่ง Node ทั้งหมดที่เข้าเงื่อนไขนี้ให้คุณบรรจุเข้าไปใน List ผ่านฟังก์ชัน `append()` (เช่น `list.append(node)`) แล้วเสร็จสิ้นกระบวนการแล้วก็ return list นี้ออกไป
- y. `public void delete(int key)` และคู่มือ static function ที่มีชื่อเดียวกัน ให้ทำหน้าที่ค้นหา Node ที่บรรจุอยู่แล้วใน BST แล้วทำการลบออก ตามที่เรียนในห้อง
- ถ้าหาก Node ที่คุณต้องการลบเป็น root Node ให้คุณทำการ implement ใน function นี้เลย
 - แต่ถ้า Node ที่คุณต้องการลบไม่ใช่ root Node ให้คุณเรียกคู่มือ static `delete()` เพื่อทำการลบ node ต่อไป
 - กรณีที่เป็น root จะมีกรณีที่ต้องพิจารณาอยู่ 6 กรณี เช่น Empty Tree, Single Node Root, Root with only left child, Root with only right child, Root with both children
 - กรณีที่เป็น node ไດ ๆ ให้ไปลบใน static `delete()` โดยมีกรณีพิจารณาอยู่ 7 กรณี เช่น Leaf node on parent's left, Leaf node on parent's right, Node with left child on parent's left, ...

5. ตัวอย่างการทำงาน

C# code
<pre>Tree tree = new Tree(); tree.printTree();</pre>
Output
Empty tree!!!

C# code
<pre>Tree tree = constructTree1(); tree.printTree();</pre>
Output (แผนภาพต้นไม้ด้านล่างอาจแตกต่างกับผลลัพธ์จริงเล็กน้อย)
<pre>graph TD 5 --> 3 5 --> 7 3 --> 1 3 --> 2 7 --> 9 7 --> 10 1 --> 2 9 --> 8 9 --> 10</pre>

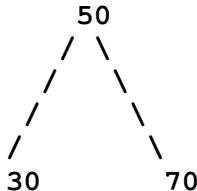
C# code
<pre>Tree tree = constructTree1(); Node node = tree.find(3); Tree.printNode(node);</pre>
Output
3

C# code
<pre>Tree tree = constructTree1(); Node node = tree.find(4); Tree.printNode(node);</pre>
Output

Node not found!!!

C# code
<pre>Tree tree = constructTree1(); Node node = tree.findMin(); Tree.printNode(node); node = tree.findMax(); Tree.printNode(node);</pre>
Output
1
10

C# code
<pre>Tree tree = constructTree1(); Node node = tree.find(3); node = Tree.findMin(node); Tree.printNode(node); node = Tree.findMax(node); Tree.printNode(node);</pre>
Output
1
2

C# code
<pre>Tree tree = constructTree2(); tree.printTree(); Node node = tree.findClosestLeaf(46); Tree.printNode(node); node = tree.findClosest(46); Tree.printNode(node);</pre>
Output (แผนภาพต้นไม้ด้านล่างอาจแตกต่างกับผลลัพธ์จริงเล็กน้อย)
 <pre>graph TD 50 --- 30 50 --- 70</pre>



```

C# code

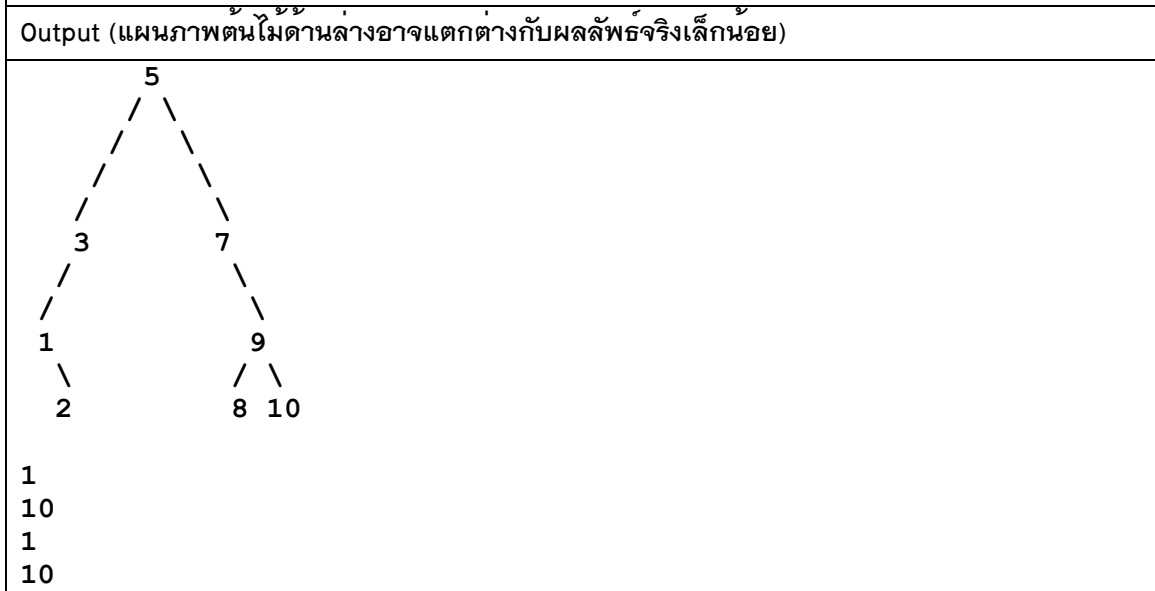
Tree tree = constructTree1();
tree.printTree();

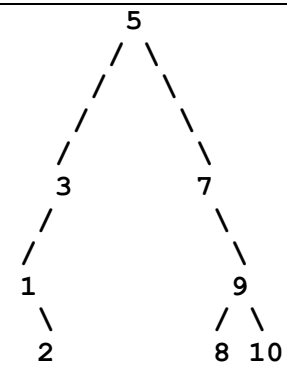
Node node = tree.findClosestLeaf(-999); Tree.printNode(node);

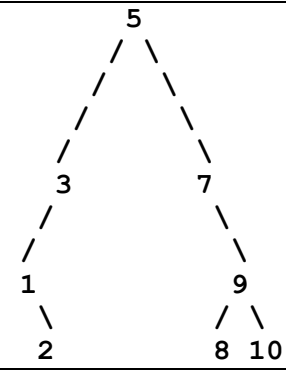
node = tree.findClosestLeaf(999); Tree.printNode(node);

node = tree.findClosest(-999); Tree.printNode(node);

node = tree.findClosest(999); Tree.printNode(node);
  
```



C# code
<pre> Tree tree = constructTree1(); tree.printTree(); Node node = tree.findClosest(6); node = Tree.findMin(node); Tree.printNode(node); node = Tree.findMax(node); Tree.printNode(node); </pre>
Output (แผนภาพต้นไม้ด้านล่างอาจแตกต่างกับผลลัพธ์จริงเล็กน้อย)
 <pre> 5 / \ 3 7 / \ / \ 1 2 9 10 / \ 2 8 </pre> <p>1 2</p>

C# code
<pre> Tree tree = constructTree1(); tree.printTree(); tree.printPreOrderDFT(); tree.printInOrderDFT(); tree.printPostOrderDFT(); </pre>
Output (แผนภาพต้นไม้ด้านล่างอาจแตกต่างกับผลลัพธ์จริงเล็กน้อย)
 <pre> 5 / \ 3 7 / \ / \ 1 2 9 10 / \ 2 8 </pre>

```
PreOrder DFT node sequence [ 5 3 1 2 7 9 8 10 ]
InOrder DFT node sequence [ 1 2 3 5 7 8 9 10 ]
PostOrder DFT node sequence [ 2 1 3 8 10 9 7 5 ]
```

C# code

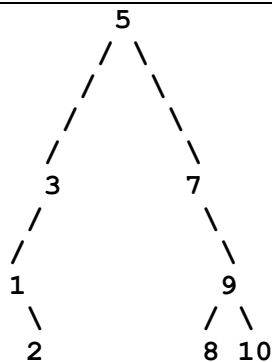
```
Tree tree = constructTree3();
tree.printTree();

Node node = tree.find(7);

Console.WriteLine("Node " + node.key + "'s Size = " + Tree.size(node));
Console.WriteLine("Node " + node.key + "'s Depth = " + Tree.depth(tree.root, node));
Console.WriteLine("Node " + node.key + "'s Height = " + Tree.height(node));

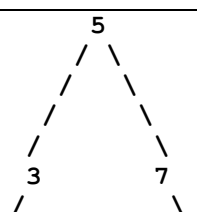
Console.WriteLine("Tree Size = " + tree.size());
Console.WriteLine("Tree Depth = " + tree.depth());
Console.WriteLine("Tree Height = " + tree.height());
```

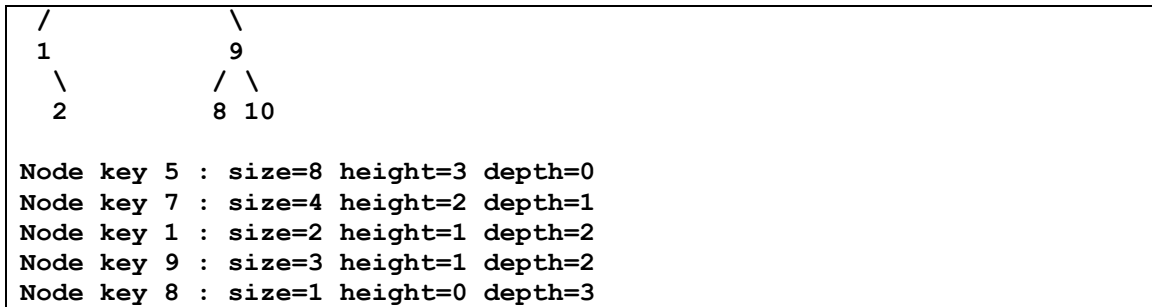
Output (แผนภาพต้นไม้ด้านล่างอาจแตกต่างกับผลลัพธ์จริงเล็กน้อย)



```
Node 7's Size = 4
Node 7's Depth = 1
Node 7's Height = 2
Tree Size = 8
Tree Depth = 3
Tree Height = 3
```


C# code
<pre> Tree tree = new Tree(); Console.WriteLine("Tree Size = " + tree.size()); Console.WriteLine("Tree Depth = " + tree.depth()); Console.WriteLine("Tree Height = " + tree.height()); tree.insert(55); Console.WriteLine("Tree Size = " + tree.size()); Console.WriteLine("Tree Depth = " + tree.depth()); Console.WriteLine("Tree Height = " + tree.height()); </pre>
Output
<pre> Tree Size = 0 Tree Depth = -1 Tree Height = -1 Tree Size = 1 Tree Depth = 0 Tree Height = 0 </pre>

C# code
<pre> Tree tree = constructTree3(); tree.printTree(); Node n; int[] data = { 5, 7, 1, 9, 8 }; int s, h, d; for (int i = 0; i < data.Length; i++) { n = tree.find(data[i]); s = Tree.size(n); h = Tree.height(n); d = Tree.depth(tree.root, n); Console.WriteLine("Node key " + n.key + " : size=" + s + " height=" + h + " depth=" + d); } </pre>
Output
 <pre> graph TD 5 --- 3 5 --- 7 3 --- 7 --- </pre>



```
C# code

Tree tree = new Tree();

int[] keyList = { 5, 2, 3, 9, 1, 10, 8, 7 };

for (int i = 0; i < keyList.Length; i++)
    tree.insert(keyList[i]);

tree.printTree();

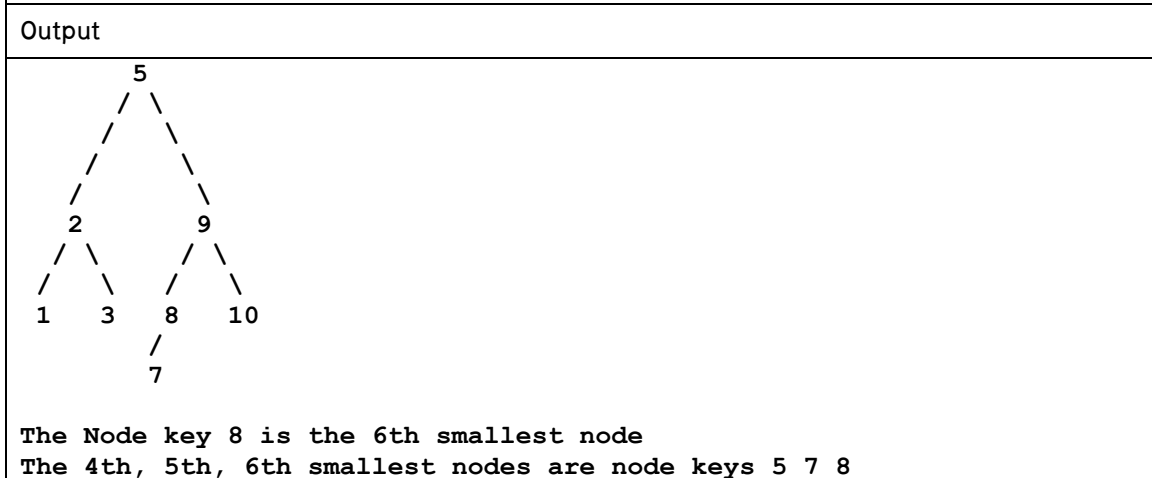
Node node = tree.findKthSmallest(6);

Console.WriteLine("The Node key " + node.key + " is the 6th smallest node");

Console.WriteLine("The 4th, 5th, 6th smallest nodes are node keys ");

for (int i = 4; i <= 6; i++)
{
    node = tree.findKthSmallest(i);
    Console.Write(node.key + " ");
}

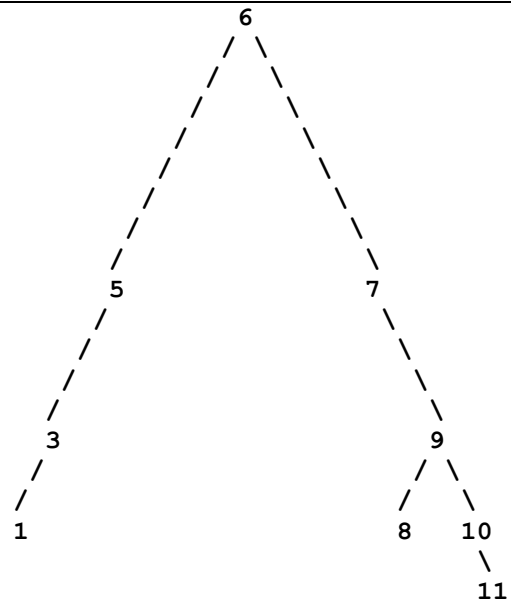
Console.WriteLine();
```



C# code

```
Tree tree = new Tree();  
  
int[] keyList = { 6, 7, 9, 5, 3, 10, 11, 8, 1 };  
for (int i = 0; i < keyList.Length; i++)  
    tree.insert(keyList[i]);  
tree.printTree();  
  
List list = tree.rangeSearch(4, 8);  
list.printList();  
  
list = tree.rangeSearch(-99, 4);  
list.printList();  
  
list = tree.rangeSearch(-99, 99);  
list.printList();
```

Output



[Head] 5 6 7 8 [Tail]

[Head] 1 3 [Tail]

[Head] 1 3 5 6 7 8 9 10 11 [Tail]

C# code

```
Tree tree = new Tree();  
int[] keyList = { 5, 3, 7 };  
for (int i = 0; i < keyList.Length; i++)  
    tree.insert(keyList[i]);  
tree.printTree();  
tree.delete(0);  
Console.WriteLine("-----");  
tree.delete(5); tree.printTree();  
tree.delete(7); tree.printTree();  
tree.delete(3);  
Console.WriteLine("-----");  
tree.printTree();  
tree.delete(0);  
Console.WriteLine("-----");
```

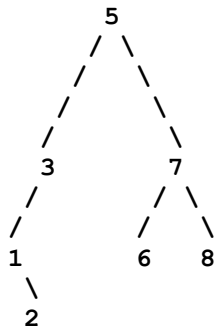
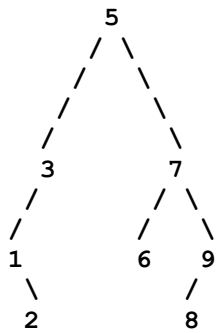
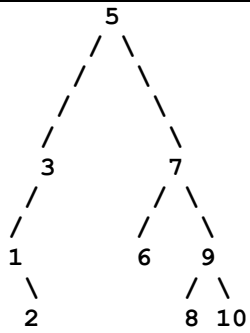
Output

```
5  
/ \  
3 7  
  
Key not found!!!  
-----  
7  
/  
3  
  
3  
  
-----  
Empty tree!!!  
Empty Tree!!!  
-----
```

C# code
<pre> Tree tree = new Tree(); int[] keyList = { 5, 3, 7 }; for (int i = 0; i < keyList.Length; i++) tree.insert(keyList[i]); tree.printTree(); tree.delete(0); Console.WriteLine("-----"); tree.delete(3); tree.printTree(); tree.delete(7); tree.printTree(); tree.delete(5); Console.WriteLine("-----"); tree.printTree(); tree.delete(0); Console.WriteLine("-----"); </pre>
Output
<pre> 5 / \ 3 7 Key not found!!! ----- 5 \ 7 5 ----- Empty tree!!! Empty Tree!!! ----- </pre>

C# code
<pre> Tree tree = constructTree4(); tree.printTree(); tree.delete(10); tree.printTree(); tree.delete(9); tree.printTree(); </pre>

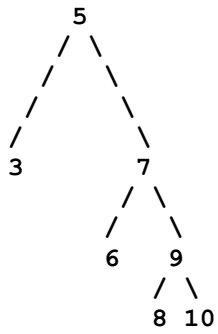
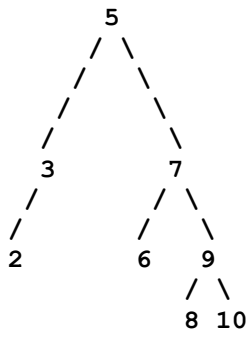
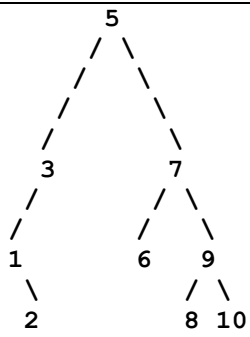
Output



C# code

```
Tree tree = constructTree4();  
tree.printTree();  
tree.delete(1); tree.printTree();  
tree.delete(2); tree.printTree();  
tree.delete(3); tree.printTree();
```

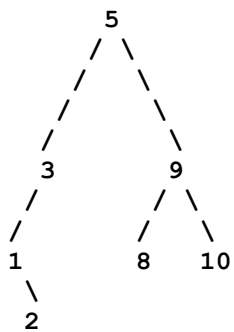
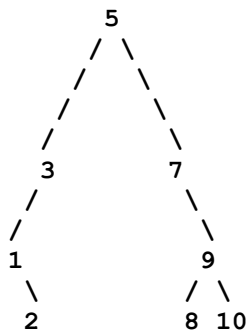
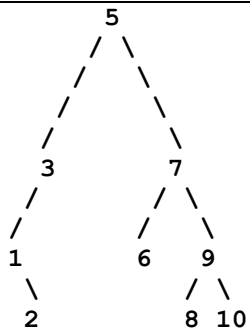
Output



C# code

```
Tree tree = constructTree4();  
tree.printTree();  
tree.delete(6); tree.printTree();  
tree.delete(7); tree.printTree();
```

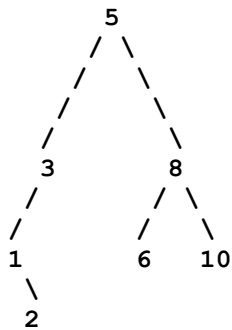
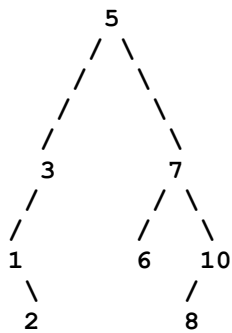
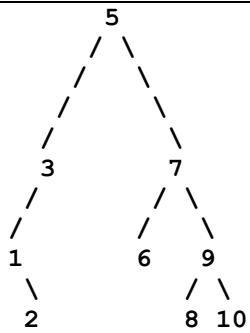
Output



C# code

```
Tree tree = constructTree4();  
tree.printTree();  
tree.delete(9); tree.printTree();  
tree.delete(7); tree.printTree();
```

Output



6. โปรดใช้ Starter code ที่อาจารย์แนบให้