

## CPE217 – Homework 5

### Homework: Binary Search Tree Data Structure

Homework Due Date: 1 November 2016

Patiwet Wuttisarnwattana, Ph.D.

Department of Computer Engineering

- คำชี้แจงการส่งงาน
- ให้นักศึกษาส่งงานเข้าอีเมล [class.submissionx@gmail.com](mailto:class.submissionx@gmail.com) เท่านั้น ห้ามส่งงานเข้าเมลของอาจารย์ โปรดจำไว้ว่าอีเมลนี้อาจจะไม่อ่านจนกว่าจะถึงเวลาตรวจ เพราะฉะนั้นถ้านักศึกษามีคำถามจะถามอาจารย์หรือพี่ TA ให้ส่งอีเมลเข้าอีเมลส่วนตัวอาจารย์หรือพี่ TA ตามที่ให้ไว้
- ในการส่งงานให้ระบุในช่อง Subject ของ email ให้เขียน pattern ดังต่อไปนี้ [CPE217][HWx][core person ID] ตรง x ให้ระบุว่าเป็นการบ้านที่เท่าไรและตรง core person ID ให้ระบุรหัสนักศึกษาที่เป็น core person ตัวอย่างเช่น [CPE217][HW5][580610615] หาก core person คือ กนกวรรณ
- แต่ละกลุ่ม ควรให้ core person เป็นคนส่งงาน และในเนื้อความอีเมล ให้ระบุ รหัสประจำตัวนักศึกษาของทุกคนที่เป็นสมาชิกในกลุ่ม หาก core person ไม่สามารถส่งงานได้ ให้สมาชิกคนอื่นก็ได้ส่งงานแทน แต่ core person id ต้องเป็นรหัสเดิมทุกครั้ง
- โค้ดของคุณต้องมีคอมเมนต์ (comment) เพื่ออธิบายว่าโค้ดดังที่เห็นอยู่นี้ทำงานอะไร หรือ if นี่ทำตรวจสอบอะไร หากกลุ่มไหนไม่มีคอมเมนต์ในโค้ดจะไม่ได้รับการตรวจ การเขียนคอมเมนต์ไม่ต้องเขียนแบบละเอียดยิบก็ได้ เท่าที่คุณต้องการให้ผู้ตรวจทราบก็พอ
- อาจารย์เปลี่ยนใจละ ไม่ต้องส่งคะแนน contribution ของเพื่อนในกลุ่ม หากกลุ่มไหนทนไม่ไหวและอยากรายงานความประพฤติของเพื่อนว่าเขาเปรี้ยบ ไม่ช่วยทำงาน ให้ส่งอีเมลแจ้งอาจารย์เป็นกรณีไป
- งานที่ส่งต้องประกอบด้วย Zip file ของ src folder ที่สามารถกด F6 รันได้เลย หากมี compile error หรือ runtime exception งานของนักศึกษาจะไม่ได้รับการตรวจ
- สามารถส่งการบ้านช้าได้ แต่หักคะแนนวันละ 10%

การบ้านนี้ให้นักศึกษา implement Binary Search Tree (BST) ตามที่เรียนในห้อง โดยใช้ Java โดยให้มีคลาสดังต่อไปนี้

1. ให้สร้าง class Node โดย class Node นี้โดยตัวมันเองมีคุณสมบัติของการเป็น Binary Search Tree (BST)
  - a. ให้ Node แต่ละ Node สามารถบรรจุ data (key) ได้ค่า ๆ หนึ่ง โดยให้เป็นตัวแปรชนิด integer
  - b. ให้ Node แต่ละ Node สามารถต่อกันเพื่อเป็นโครงสร้างข้อมูล Binary Search Tree ตามที่เรียนในห้องได้
  - c. สมาชิกของ class Node ควรที่จะมี reference ชี้ไปยัง ลูกคนซ้าย (left child) และลูกคนขวา (right child)
  - d. การบ้านนี้กำหนดให้มี parent reference/pointer
2. ให้ class Node มี 1 Constructor คือ Node(int data) ซึ่งทำหน้าที่ กำหนดค่าเริ่มต้นของ key จาก data
3. ให้ class Node (ซึ่งตัว Node เอง สามารถมีคุณสมบัติเป็น BST ได้) มี operation ดังต่อไปนี้
  - a. public int height() ทำหน้าที่หาค่า Node ที่อยู่ความสูงที่เท่าไรเมื่อเทียบกับลูกที่อยู่ลึกที่สุด
  - b. public int size() ทำหน้าที่หาค่า หากกำหนดให้ Node นี้เป็น root node ของ Tree หนึ่ง ๆ แล้ว Tree นี้จะมีลูกอยู่ทั้งหมดกี่ node (ความหมายคือหา Tree size)
  - c. public int depth(Tree tree) ทำหน้าที่หาค่า หากกำหนดให้ Node นี้เป็นส่วนหนึ่งของ Tree tree แล้ว Node นี้จะมีความลึกเป็นเท่าไร เมื่อเทียบกับ tree.root (ความหมายคือหา Node depth)
  - d. public Node findNext() ทำหน้าที่หาค่า Node ที่มีค่า key อยู่มากกว่าขึ้นไปอีกค่าหนึ่งคือ Node ไດ
4. เพื่อความสะดวกแก่การทำ recursive function จึงกำหนดให้คุณทำการ implement “static method” สำหรับ class Node ดังต่อไปนี้
  - a. public static int height(Node node) ทำหน้าที่คืนค่าความสูงของ node
  - b. public static int size(Node node) ทำหน้าที่คืนค่าขนาดของต้นไม้ของ node
  - c. public static Node leftDescendant(Node node) ทำหน้าที่หา descendant Node ที่อยู่ด้านซ้ายสุดของ node
  - d. public static Node rightAncestor(Node node) ทำหน้าที่หา ancestor Node ที่อยู่ด้านขวาแรกของ node
5. ให้สร้าง class Tree ซึ่งทำหน้าที่บรรจุ Node ตามคุณสมบัติของ BST
  - a. ให้ class Tree มีตัวแปรที่ชื่อว่า root (class Node) ทำหน้าที่ชี้ไปยัง root Node ของแผนภาพต้นไม้
  - b. ให้ class Tree ทำการ inherit class BTreePrinter ที่อาจารย์แนบมาให้ด้วย เพื่อที่จะได้แสดงแผนภาพต้นไม้ออกมาอย่างคร่าว ๆ ได้
  - c. ให้ class Tree มี method ดังต่อไปนี้
    - public Node find(int search\_key) ทำหน้าที่หา Node ที่บรรจุใน Tree ที่มี key ดังที่ระบุ
    - public Node findClosest(int search\_key) ทำหน้าที่หา Node ที่บรรจุใน Tree ที่มี Node ใกล้เคียงกับ search\_key ดังที่ระบุ ให้ดำเนินการตามลำดับดังต่อไปนี้ (1) หากมี Node ที่มี key เท่ากับ search\_key พอดีให้คืน Node นั้นเลย (2) ถ้า Node ที่อยู่ใกล้อันดับถัดไป มีค่าน้อยกว่า search\_key ให้คืน Node นั้น (3) ) ถ้า Node ที่อยู่ใกล้อันดับถัดไป มีค่ามากกว่า search\_key ให้คืน Node นั้น
    - public Node findMin() ทำหน้าที่หา Node ที่บรรจุใน Tree ที่มีค่า key น้อยที่สุด
    - public Node findMax() ทำหน้าที่หา Node ที่บรรจุใน Tree ที่มีค่า key มากที่สุด

- `public Node findKthSmallest(int k)` ทำหน้าที่หา Node ที่บรรจุใน Tree ที่มีค่า key เล็กเป็นอันดับที่ k (k=1 แปลว่า มีค่า key เล็กที่สุด, k=2 แปลว่า มีค่า key เล็กเป็นอันดับที่สอง)
  - `public void printPreOrderDFT()` ทำหน้าที่พิมพ์ค่า key ของทุก ๆ Node ตามลำดับ Pre-Order Depth First Traversal ตามที่เรียนในห้อง
    - ให้ pattern การพิมพ์ออกทาง console ให้เป็นไปดังตัวอย่างด้านล่าง เริ่มต้นด้วยคำว่า "PreOrder DFT node sequence [ " ลงท้ายด้วย "]"
  - `public void printInOrderDFT()` ทำหน้าที่พิมพ์ค่า key ของทุก ๆ Node ตามลำดับ In-Order Depth First Traversal ตามที่เรียนในห้อง
    - ให้ pattern การพิมพ์ออกทาง console ให้เป็นไปดังตัวอย่างด้านล่าง เริ่มต้นด้วยคำว่า "InOrder DFT node sequence [ " ลงท้ายด้วย "]"
  - `public void printPostOrderDFT()` ทำหน้าที่พิมพ์ค่า key ของทุก ๆ Node ตามลำดับ Post-Order Depth First Traversal ตามที่เรียนในห้อง
    - ให้ pattern การพิมพ์ออกทาง console ให้เป็นไปดังตัวอย่างด้านล่าง เริ่มต้นด้วยคำว่า "PostOrder DFT node sequence [ " ลงท้ายด้วย "]"
  - `public void insert(int key)` ทำหน้าที่สร้าง Node ใหม่ที่บรรจุค่า key แล้วนำไปต่อใน BST ตามที่เรียนในห้อง ถ้าหาก key ที่บรรจุเข้ามาใหม่มีอยู่แล้วใน Node ใด Node หนึ่งของ Tree ให้พิมพ์ออกทางหน้าจอว่า "Duplicated key!!!" แล้วไม่ต้องทำอะไร
  - `public void delete(int key)` ทำหน้าที่ค้นหา Node ที่บรรจุอยู่แล้วใน BST แล้วทำการลบออกตามที่เรียนในห้อง
    - คำแนะนำ: หาก Node ที่คุณต้องการลบเป็น root Node ให้คุณทำการ implement ใน function นี้เลย แต่ถ้า Node ที่คุณต้องการลบไม่ใช่ root Node ผมแนะนำให้คุณลบด้วยวิธีการของ recursive โดยใช้ Java static function/method ด้านล่าง
  - `public int height()` ทำหน้าที่คำนวณว่าต้นไม้ต้นนี้มีค่าสูงเท่าไร จงเขียนโค้ดในหนึ่งบรรทัด (คำใบ้: implement function height ของ class Node ให้เสร็จก่อน)
  - `public int depth()` ทำหน้าที่คำนวณว่าต้นไม้ต้นนี้มีค่าลึกเท่าไร จงเขียนโค้ดในหนึ่งบรรทัด (คำใบ้: implement function height ของ class Node ให้เสร็จก่อน)
  - `public int size()` ทำหน้าที่คำนวณว่าต้นไม้ต้นนี้มีบรรจุ Node ไปแล้วทั้งหมดกี่ Node จงเขียนโค้ดในหนึ่งบรรทัด (คำใบ้: implement function size ของ class Node ให้เสร็จก่อน)
  - `public List rangeSearch(int x, int y)` ทำหน้าที่ค้นหา Node ที่มี key อยู่ระหว่างค่า x กับค่า y โดยค่า  $x \leq \text{key} \leq y$  ซึ่ง Node ทั้งหมดที่เข้าเงื่อนไขให้คุณบรรจุเข้าไปใน List Data Structure ที่ผมสร้างไว้ให้แล้ว
- d. เพื่อความสะดวกแห่งการทำ recursive function จึงกำหนดให้คุณทำการ implement "static method" สำหรับ class Node ดังต่อไปนี้

- `public static Node find(Node node, int search_key)` ทำหน้าที่หา descendant Node ของ Node node ที่มีค่า key เท่ากับ search\_key
  - `public static Node findClosest(Node node, int search_key)` ทำหน้าที่หา descendant Node ของ Node node ที่มีค่า key ใกล้เคียงกับ search\_key นิยามคำว่าใกล้เคียงให้ใช้เกณฑ์เดียวกับ method `public Node findClosest(int search_key)` ข้างต้น
  - `public static Node findMin(Node node)` ทำหน้าที่หา descendant Node ของ Node node ที่มีค่า key น้อยที่สุด
  - `public static Node findMax(Node node)` ทำหน้าที่หา descendant Node ของ Node node ที่มีค่า key มากที่สุด
  - `public static Node findKthSmallest(Node node, int k)` ทำหน้าที่หา descendant Node ของ Node node ที่มีค่า key เล็กเป็นอันดับที่ k. k=1 แปลว่า มีค่า key น้อยที่สุด
  - `public static void printPreOrderDFT(Node node)` ทำหน้าที่พิมพ์ค่า key ของทุก ๆ descendant ของ Node node ตามลำดับ Pre-Order Depth First Traversal ตามที่เรียนในห้อง Pattern การพิมพ์ออกทางหน้าจอให้เป็นไปตาม non-static function ดังที่กล่าวไปแล้วข้างบน
  - `public static void printInOrderDFT(Node node)` ทำหน้าที่พิมพ์ค่า key ของทุก ๆ descendant ของ Node node ตามลำดับ In-Order Depth First Traversal ตามที่เรียนในห้อง Pattern การพิมพ์ออกทางหน้าจอให้เป็นไปตาม non-static function ดังที่กล่าวไปแล้วข้างบน
  - `public static void printPostOrderDFT(Node node)` ทำหน้าที่พิมพ์ค่า key ของทุก ๆ descendant ของ Node node ตามลำดับ Post-Order Depth First Traversal ตามที่เรียนในห้อง Pattern การพิมพ์ออกทางหน้าจอให้เป็นไปตาม non-static function ดังที่กล่าวไปแล้วข้างบน
  - `public static void delete(Node node)` ทำหน้าที่ค้นหา Node ที่บรรจุอยู่แล้วใน BST แล้วทำการลบออก (แทนที่ด้วย Node ที่มีค่า key เล็กที่สุดของ right subtree) ตามที่เรียนในห้อง
    - คำใบ้: ให้เรียกใช้ function นี้หาก node ไม่ใช่ root
  - ฟังก์ชันลับ (Optional): `public static void replace(Node node1, Node node2)` ทำหน้าที่ แทนที่ node1 ด้วย node2 โดยมีเงื่อนไขคือ node2 ต้องถูกสร้างใหม่มาแล้ว เช่น `new Node(key)` ก่อนเรียกใช้ฟังก์ชันนี้ ฟังก์ชันนี้คุณจะใช้ก็ได้หรือไม่ใช้ก็ได้ (Optional) หากคุณคิดว่าจะเป็นประโยชน์ต่อการทำบ้านของคุณก็ใช้ โค้ดของผมต้องใช้ฟังก์ชันนี้ โค้ดของคุณอาจจะใช้หรือไม่ใช้ก็ได้
6. ใน class Main คุณควรที่จะสร้าง function ที่ชื่อว่า `void printNodeKey(Node node)` เพื่อทำการพิมพ์ค่า key ออกทาง console หาก node ที่รับเข้ามานั้นเป็น null ให้พิมพ์ออก console “Node not found!!!”

## 7. ตัวอย่างการทำงาน

### Java code

```
public static void main(String[] args) {  
    Tree tree = new Tree();  
    tree.printTree();  
  
    int[] keyList = {5, 3, 1, 2, 7, 9, 10, 8};  
    for (int i=0; i<keyList.length; i++)  
        tree.insert(keyList[i]);  
    tree.printTree();  
  
    Node node = tree.find(3); printNodeKey(node);  
    node = tree.find(4); printNodeKey(node);  
    node = tree.findClosest(4); printNodeKey(node);  
    node = tree.findClosest(3); printNodeKey(node);  
    node = tree.findClosest(-999); printNodeKey(node);  
    node = tree.findClosest(999); printNodeKey(node);  
}
```

### Output (แผนภาพต้นไม้ด้านล่างอาจแตกต่างกับผลลัพธ์จริงเล็กน้อย)

Empty tree!!!

```
      5  
     /\  
    /\   
   /\   
  /\   
 /\   
3  7  
/\   
/\   
1  9  
 \ /\   
2  8 10
```

3  
Node not found!!!  
3  
3  
1  
10

#### Java code

```
public static void main(String[] args) {  
    Tree tree = new Tree();  
  
    int[] keyList = {6, 7, 9, 5, 3, 9, 10, 8, 1};  
    for (int i=0; i<keyList.length; i++)  
        tree.insert(keyList[i]);  
    tree.printTree();  
  
    List list = tree.rangeSearch(4, 8);  
    list.printList();  
}
```

#### Output (แผนภาพต้นไม้ด้านล่างอาจแตกต่างกับผลลัพธ์จริงเล็กน้อย)

Duplicated key!!!

```
      6  
     /\  
    /\   
   /\   
  /\   
 5  7  
 /\   
/\   
3   9  
/\  /\  
1   8 10
```

[Head] 5 6 7 8 [Tail]

### Java code

```
public static void main(String[] args) {  
    Tree tree = new Tree();  
  
    int[] keyList = {5, 3, 1, 2, 7, 9, 10, 8};  
    for (int i=0; i<keyList.length; i++)  
        tree.insert(keyList[i]);  
    tree.printTree(); System.out.println("-----");  
  
    tree.delete(7);  
    tree.printTree(); System.out.println("-----");  
    tree.delete(3);  
    tree.printTree();  
    tree.delete(9);  
    tree.delete(1);  
    tree.delete(5);  
    tree.delete(5); System.out.println("-----");  
    tree.printTree();  
  
}
```

### Output (แผนภาพต้นไม้ด้านล่างอาจแตกต่างกับผลลัพธ์จริงเล็กน้อย)

```
      5  
     /\  
    /\   
   /\   
  /\   
 3   7  
 /\   \  
/\     \  
1       9  
 \      /\  
 2      8 10
```

-----

```

      5
    /\
   /\
  /\
 /\
3   9
/\  /\
/\  /\
1   8 10
\
2

```

```

-----
      5
    /\
   /\
  1  9
 \ /\
 2 8 10

```

Key not found!!!

```

-----
      8
    /\
  2 10

```

#### Java code

```

public static void main(String[] args) {
    Tree tree = new Tree();

    int[] keyList = {5, 3, 1, 2, 7, 9, 10, 8};
    for (int i=0; i<keyList.length; i++)
        tree.insert(keyList[i]);
}

```



```

tree.printTree();

Node node = tree.find(4);
printNodeKey(node);
node = tree.findClosest(4);
printNodeKey(node);
node = node.findNext();
printNodeKey(node);
node = node.findNext();
printNodeKey(node);
node = node.findNext();
printNodeKey(node);
}

```

**Output** (แผนภาพต้นไม้ด้านล่างอาจแตกต่างกับผลลัพธ์จริงเล็กน้อย)

```

      5
     /\
    /\ 
   /\ 
  /\ 
 3   7
 /\ 
/\ 
1   9
 \  /\
 2  8 10

```

Node not found!!!

```

3
5
7
8

```

<b>Java code</b>
<pre> public static void main(String[] args) {     Tree tree = new Tree();      int[] keyList = {5, 3, 1, 2, 7, 9, 10, 8};     for (int i=0; i&lt;keyList.length; i++)         tree.insert(keyList[i]);     tree.printTree();      tree.printPreOrderDFT();     tree.printInOrderDFT();     tree.printPostOrderDFT(); } </pre>
<b>Output</b> (แผนภาพต้นไม้ด้านล่างอาจแตกต่างกับผลลัพธ์จริงเล็กน้อย)
<pre>       5      /\     /\     /\    /\   3   7  /\  /\  1   9 \   /\ 2   8 10  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 ] </pre>

<b>Java code</b>
<pre> public static void main(String[] args) {     Tree tree = new Tree(); </pre>

```

int[] keyList = {5, 2, 3, 9, 1, 10, 8, 7};
for (int i=0; i<keyList.length; i++)
    tree.insert(keyList[i]);
tree.printTree();
System.out.println(tree.depth());
System.out.println(tree.height());
Node node = tree.find(9);
System.out.println(node.depth(tree));
System.out.println(node.height());

node = tree.findMax();
printNodeKey(node);
node = tree.findMin();
printNodeKey(node);
node = tree.findKthSmallest(6);
printNodeKey(node);
node = tree.findKthSmallest(3);
printNodeKey(node);
}

```

**Output** (แผนภาพต้นไม้ด้านล่างอาจแตกต่างกับผลลัพธ์จริงเล็กน้อย)

```

      5
     /\
    /\ 
   /\ 
  /\ 
 2   9
 /\   /\
/\    /\
1 3 8 10
   /
  7

```

3
3
1
2
10
1
8
3

8. โปรดใช้ Starter code ที่อาจารย์แนบให้