

Homework: AVL Tree Data Structure and Splay Tree Data Structure

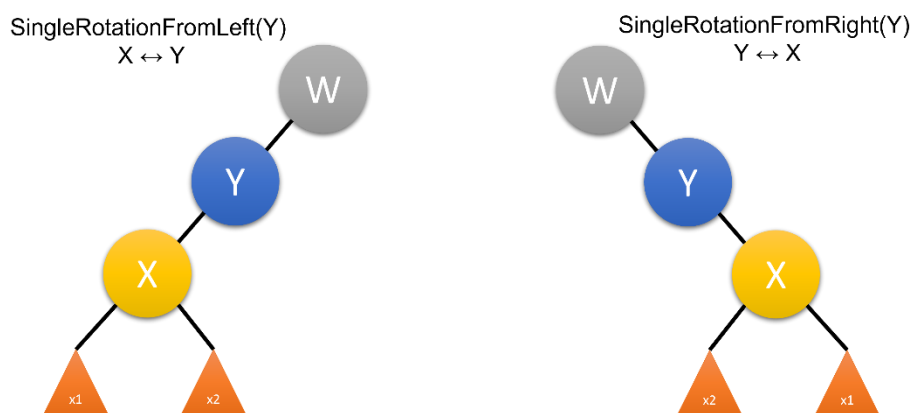
Patiwet Wuttisarnwattana, Ph.D.

Department of Computer Engineering

Chiang Mai University

1. จงปรับปรุงคลาส BSTree (Binary Search Tree) เพื่อให้สามารถทำ Single Rotation ตามที่เรียนในห้องให้สมบูรณ์

สำหรับโจทย์ข้อนี้ ขอให้คุณนึกภาพว่า Node Y คือโหนดที่คุณต้องทำการหมุนแบบ Single Rotation โดยสลับที่กันกับ x (จากซ้ายหรือจากขวาแล้วแต่กรณี) โดย Node Y จะมีหน้าตาประมาณนี้



- Node X กับ Node Y ดังที่เห็นในแผนภาพนี้ ขอให้ นศ ประกาศตัวแปรเอาเอง และที่สำคัญคือ W สามารถที่จะอยู่ด้านซ้ายหรือด้านขวาของ Y ก็ได้ หรือจะไม่มีเลยก็ได้ ขอให้ นศ ตรวจสอบกรณีดังกล่าวด้วย
- ก่อนการหมุนก็ขอให้ตรวจสอบก่อนว่าหมุนได้หรือไม่ เช่น X เป็น null รีเปล่า
- การที่จะโปรโมท X ขึ้นมาแทน Y นั้น ตรวจสอบด้วยว่า ควรผูกเข้ากับด้านซ้ายหรือด้านขวาของ W (อย่าลืมกำหนด parent ใหม่ของ X ด้วย)
- การที่ Node W เป็น null นั้น แปลว่า Node Y เป็นอะไร กรณีนี้เป็นกรณีพิเศษครับ จะต้องแก้ปัญหาแยกกับกรณีอื่น ๆ (นศ ต้องรู้คำตอบนะว่าอาจารย์หมายถึงอะไร)
- หลักของอาจารย์อย่าลืมท่องนะครับ ลูกวงนอกอยู่เหมือนเดิม ลูกที่อยู่วงใน ต้องเปลี่ยน Parent
- เวลาเปลี่ยนให้ Node หนึ่งไปเป็นลูกของอีก Node หนึ่ง อย่าลืมผูก Parent กลับทุกครั้ง เพื่อบอกว่าใครเป็นแม่มัน
- การที่จะโอน ลูกของใคร เอาไปให้ Parent คนใหม่ ก็อย่าลืมเช็คด้วยว่า Node ลูกที่วนั้น เป็น null รีเปล่า ไม่อย่างนั้นจะเกิด NullPointerException นะ จะบอกให้

- สุดท้าย ท้ายสุดอย่าลืม สลับตำแหน่งของ X กับ Y

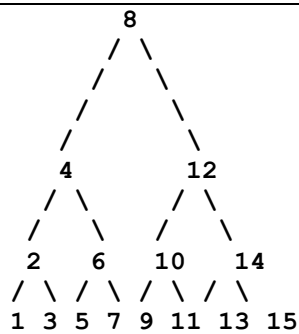
C# code

```

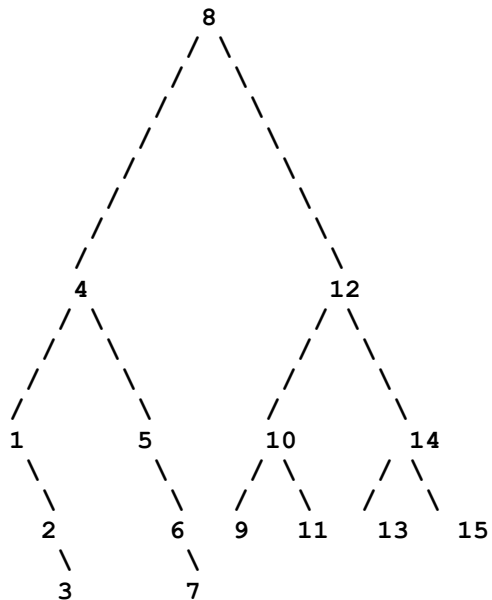
BSTree tree = generateTree1();
tree.printTree();
Console.WriteLine("---- Test1 singleRotateFromLeft at Lv 3 ----");
tree.singleRotateFromLeft(tree.find(6));
tree.singleRotateFromLeft(tree.find(2));
tree.printTree();

```

Output (แผนภาพต้นไม้ด้านล่างอาจแตกต่างกับผลลัพธ์จริงเล็กน้อย)



---- Test1 singleRotateFromLeft at Lv 3 ----

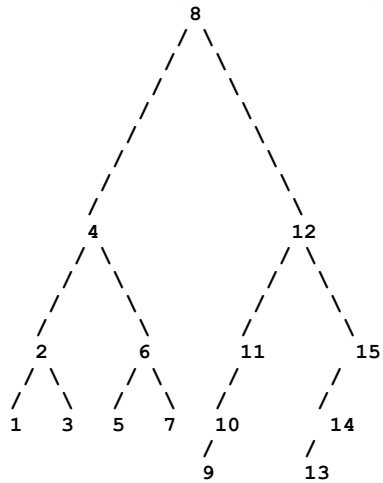


C# code

```
BSTree tree = generateTree1();  
Console.WriteLine("---- Test2 singleRotateFromRight at Lv 3 ----");  
tree.singleRotateFromRight(tree.find(10));  
tree.singleRotateFromRight(tree.find(14));  
tree.printTree();
```

Output (แผนภาพต้นไม้ด้านล่างอาจแตกต่างกับผลลัพธ์จริงเล็กน้อย)

---- Test2 singleRotateFromRight at Lv 3 ----

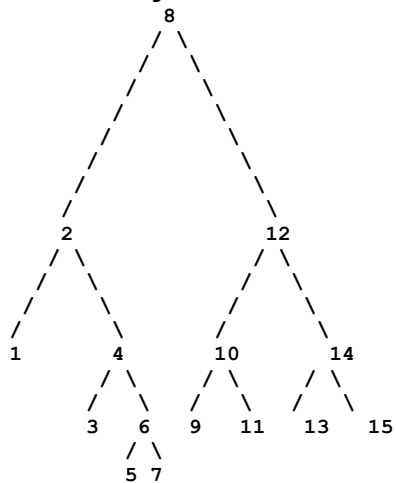


C# code

```
BSTree tree = generateTree1();  
Console.WriteLine("---- Test3 singleRotateFromLeft at Lv 2 ----");  
tree.singleRotateFromLeft(tree.find(4));  
tree.printTree();
```

Output (แผนภาพต้นไม้ด้านล่างอาจแตกต่างกับผลลัพธ์จริงเล็กน้อย)

---- Test3 singleRotateFromLeft at Lv 2 ----



C# code

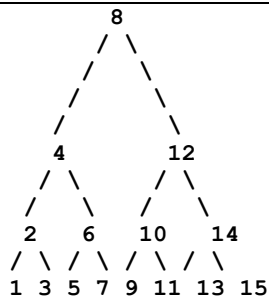
```
BSTree tree = generateTree1();
tree.printTree();

Console.WriteLine("---- Test4 singleRotateFromRight at Lv 2 ----");
tree.singleRotateFromRight(tree.find(12));
tree.printTree();

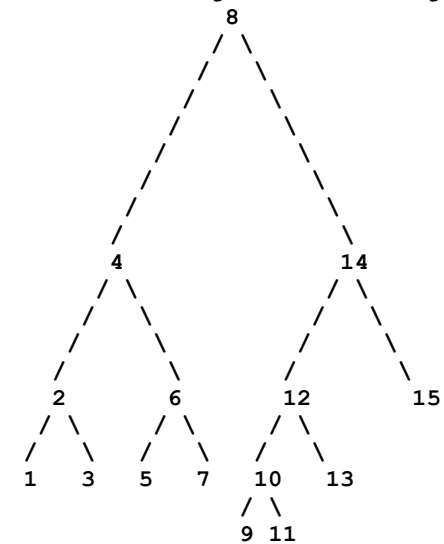
Console.WriteLine("---- Test5 singleRotateFromLeft at Lv 1 ----");
tree.singleRotateFromLeft(tree.find(8));
tree.printTree();

Console.WriteLine("---- Test6 singleRotateFromRight at Lv 1 ----");
tree.singleRotateFromRight(tree.find(4));
tree.singleRotateFromLeft(tree.find(14));
tree.printTree();
```

Output (แผนภาพต้นไม้ด้านล่างอาจแตกต่างกับผลลัพธ์จริงเล็กน้อย)

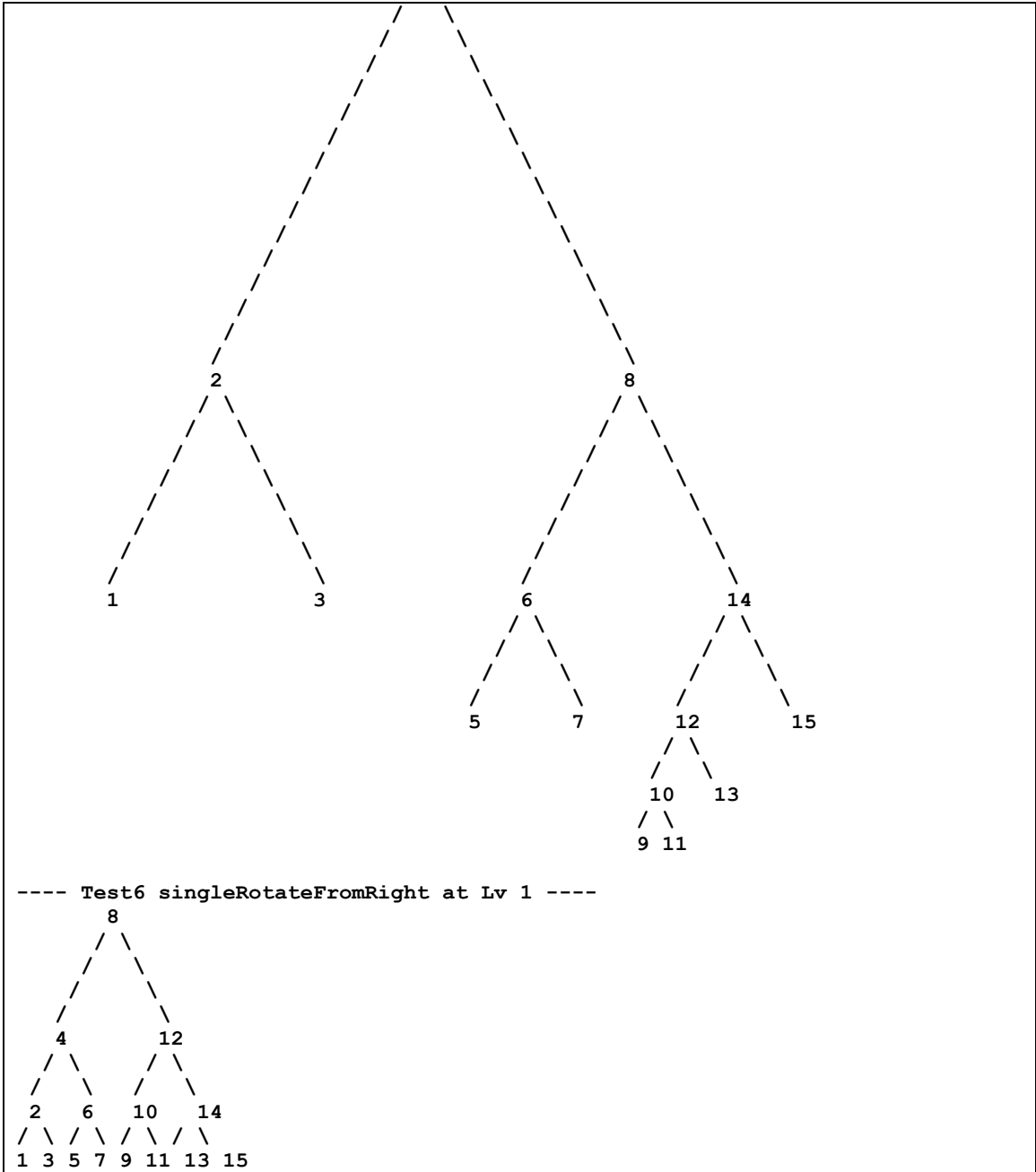


---- Test4 singleRotateFromRight at Lv 2 ----



---- Test5 singleRotateFromLeft at Lv 1 ----

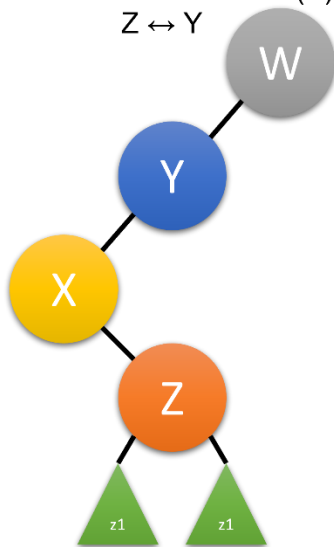




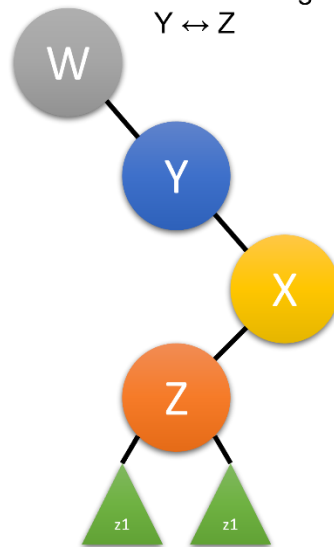
2. จงปรับปรุงคลาส BSTree (Binary Search Tree) เพื่อให้สามารถทำ Double Rotation ตามที่เรียนในห้องให้สมบูรณ์

สำหรับโจทย์ข้อนี้ ขอให้คุณนึกภาพว่า Node Y คือโหนดที่คุณต้องทำการหมุนแบบ Double Rotation โดยสลับที่กันกับ Z (จากซ้ายหรือจากขวาแล้วแต่กรณี) โดย Node Y จะมีหน้าตาประมาณนี้

DoubleRotationFromLeft(Y)
 $Z \leftrightarrow Y$



DoubleRotationFromRight(Y)
 $Y \leftrightarrow Z$

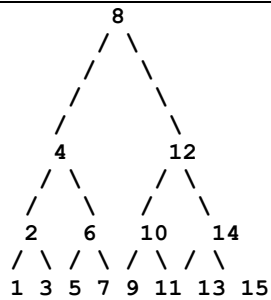


- Node X, Y, Z, W ดังที่เห็นในแผนภาพนี้ ขอให้ นศ ประกาศตัวแปรเอาเอง และที่สำคัญคือ W สามารถที่จะอยู่ด้านซ้ายหรือด้านขวาของ Y ก็ได้ หรือจะไม่มีเลยก็ได้ ขอให้ นศ ตรวจสอบกรณีดังกล่าวด้วย
- ก่อนการหมุนก็ขอให้ตรวจสอบก่อนว่าหมุนได้หรือไม่ เช่น X หรือ Z เป็น null รีบเล่า
- การที่จะโปรโมท Z ขึ้นมาแทน Y นั้น ตรวจสอบด้วยว่า ควรผูกเข้ากับด้านซ้ายหรือด้านขวาของ W (แล้วก็อย่าลืมกำหนด parent ใหม่ของ Z ด้วย)
- การที่ Node W เป็น null นั้น แปลว่า Node Y เป็นอะไร กรณีนี้เป็นกรณีพิเศษครับ จะต้องแก้ปัญหาแยกกับกรณีอื่น ๆ (นศ ต้องรู้คำตอบนะว่าอาจารย์หมายถึงอะไร)
- การที่ Z จะทะลวงขึ้นมาแทน Y นั้น คุณจะต้องโอนลูกของ Z ทั้งสองคนไปให้ X กับ Y ให้เรียบร้อยเสียก่อน
- เวลาเปลี่ยนให้ Node หนึ่งไปเป็นลูกของอีก Node หนึ่ง อย่าลืมผูก Parent กลับทุกครั้ง เพื่อบอกว่าใครเป็นแม่มัน
- การที่จะโอนลูกของใคร เอาไปให้ Parent คนใหม่ ก็อย่าลืมเช็คด้วยว่า Node ลูกที่ว่ามันนั้น เป็น null รีบเล่า ไม่อย่างนั้นจะเกิด NullPointerException นะ จะบอกให้
- สุดท้าย หายสุดอย่าลืม เอา X กับ Y ไปเป็นลูกของ Z ให้ถูกต้องด้วยครับ

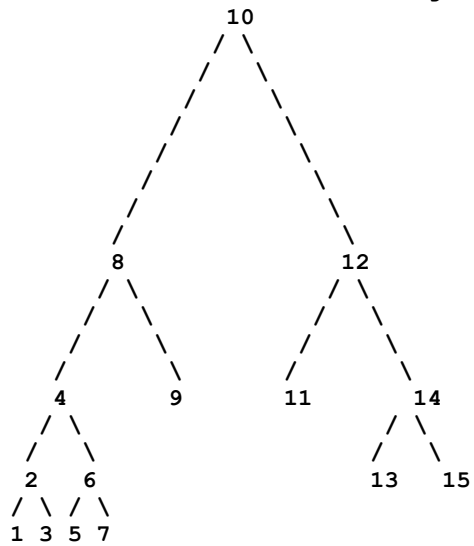
C# code

```
BSTree tree = generateTree1();  
tree.printTree();  
Console.WriteLine("---- Test7 doubleRotateFromRight at Lv 1 ----");  
tree.doubleRotateFromRight(tree.find(8));  
tree.printTree();
```

Output (แผนภาพต้นไม้ด้านล่างอาจแตกต่างกับผลลัพธ์จริงเล็กน้อย)



---- Test7 doubleRotateFromRight at Lv 1 ----



C# code
<pre> BSTree tree = generateTree1(); Console.WriteLine("----- Test8 doubleRotateFromLeft at Lv 1 -----"); tree.doubleRotateFromLeft(tree.find(8)); tree.printTree(); </pre>
Output (แผนภาพต้นไม้ด้านล่างอาจแตกต่างกับผลลัพธ์จริงเล็กน้อย)
<pre> ----- Test8 doubleRotateFromLeft at Lv 1 ----- 6 / \ / \ 4 8 / \ / \ 2 5 7 12 / \ / \ / \ 1 3 9 11 13 15 </pre>

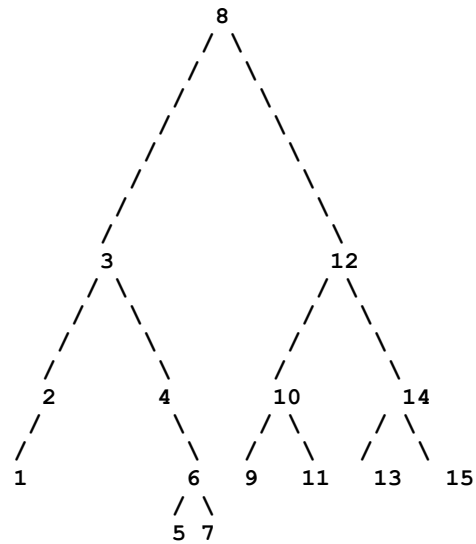
C# code
<pre> BSTree tree = generateTree1(); tree.printTree(); Console.WriteLine("----- Test9.1 doubleRotateFromLeft at Lv 2 -----"); tree.doubleRotateFromLeft(tree.find(4)); tree.printTree(); Console.WriteLine("----- Test9.2 doubleRotateFromLeft at Lv 2 -----"); tree.doubleRotateFromLeft(tree.find(12)); tree.printTree(); </pre>
Output (แผนภาพต้นไม้ด้านล่างอาจแตกต่างกับผลลัพธ์จริงเล็กน้อย)
<pre> 8 / \ 4 12 / \ / \ </pre>


```

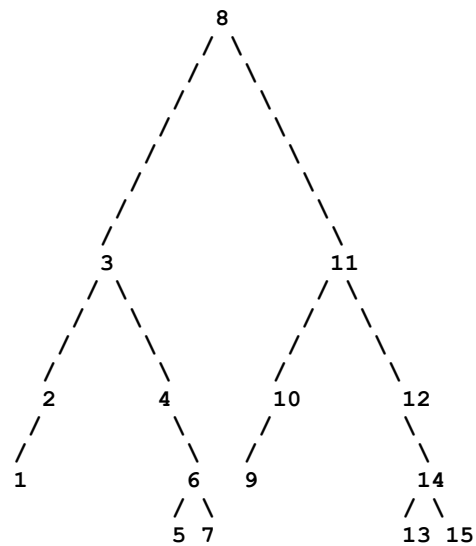
    2    6    10   14
   /\  /\  /\  /\
  1 3 5 7 9 11 13 15

```

---- Test9.1 doubleRotateFromLeft at Lv 2 ----



---- Test9.2 doubleRotateFromLeft at Lv 2 ----



C# code

```

BSTree tree = generateTree1();
tree.printTree();

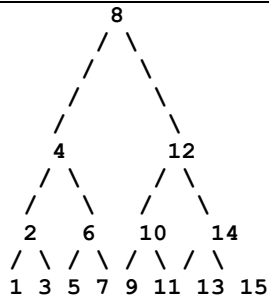
Console.WriteLine("---- Test10.1 doubleRotateFromRight at Lv 2 ----");
tree.doubleRotateFromRight(tree.find(4));
tree.printTree();

Console.WriteLine("---- Test10.2 doubleRotateFromRight at Lv 2 ----");

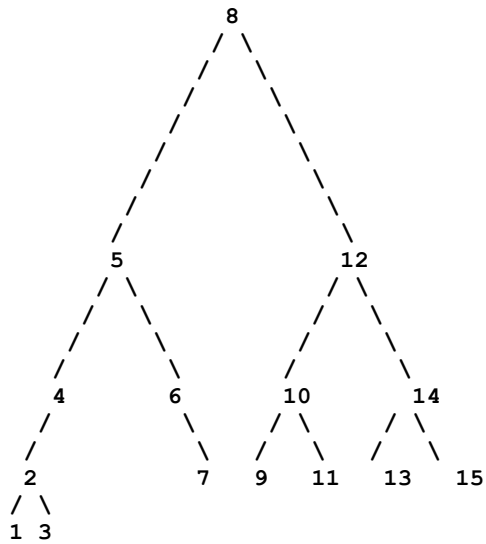
```

```
tree.doubleRotateFromRight(tree.find(12));  
tree.printTree();
```

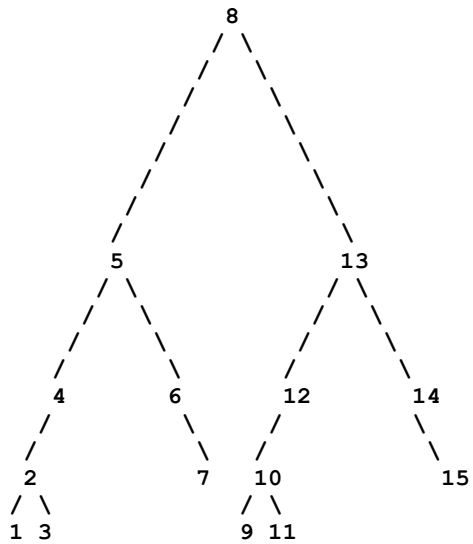
Output (แผนภาพต้นไม้ด้านล่างอาจแตกต่างกับผลลัพธ์จริงเล็กน้อย)



---- Test10.1 doubleRotateFromRight at Lv 2 ----

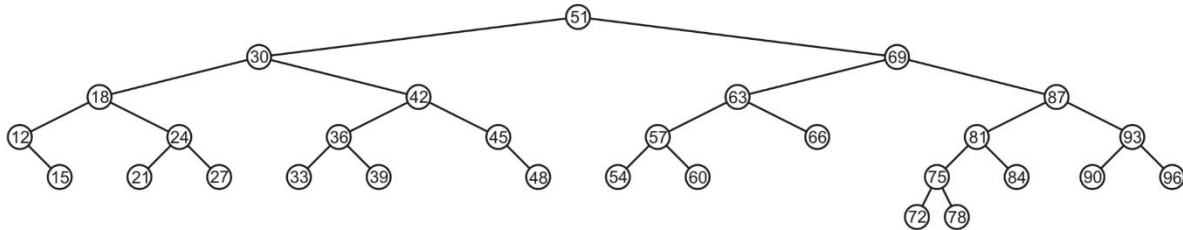


---- Test10.2 doubleRotateFromRight at Lv 2 ----



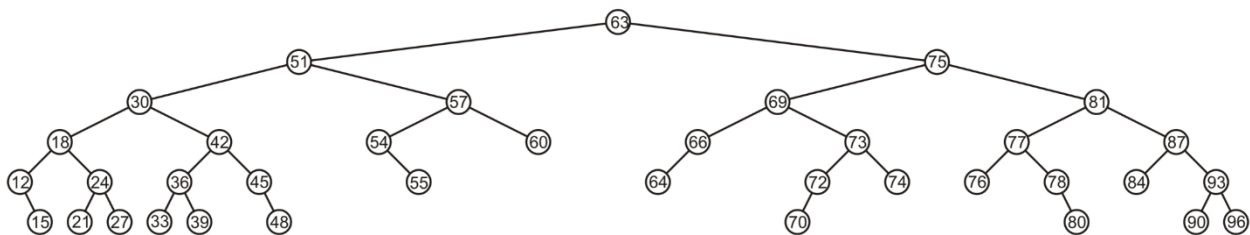
3. จงปรับปรุงคลาส AVLTree (AVL Tree) เพื่อให้สามารถทำการ Insert ข้อมูลโดยคงความเป็น AVLTree ให้ได้ถูกต้อง

สำหรับโจทย์ข้อนี้ โครงสร้างข้อมูลชนิด AVLTree ได้ถูกสร้างขึ้นมาพร้อมข้อมูลเริ่มต้น มีหน้าตาประมาณนี้



ซึ่งรหัส BFT ที่ได้จะมีค่าเป็น BFT [51 30 69 18 42 63 87 12 24 36 45 57 66 81 93 15 21 27 33 39 48 54 60 75 84 90 96 72 78]

ต่อมาโจทย์จะทำการ Insert Node ใหม่ เข้าไปในโครงสร้างข้อมูลชนิด AVL Tree ข้างต้น ตามลำดับ คือ 73, 77, 76, 80, 74, 64, 55, 70 ผลลัพธ์สุดท้ายที่ได้เมื่อทำการ Rebalancing แล้ว จะประมาณนี้



โดยรหัส BFT ที่ได้ คือ BFT [63 51 75 30 57 69 81 18 42 54 60 66 73 77 87 12 24 36 45 55 64 72 74 76 78 84 93 15 21 27 33 39 48 70 80 90 96]

คำอธิบายเพิ่มเติม

- โจทย์ข้อนี้ อาจารย์ได้นำมาจาก Slide ที่เรียนในห้อง ดังนั้น กระบวนการ Insert และ Rebalance จะเป็นไปตาม Slide โดยละเอียดเลย อาจารย์ขออนุญาตไม่อธิบายซ้ำในที่นี้
- โจทย์ข้อนี้ นศ จะต้องการทำ Implement ฟังก์ชันที่ชื่อว่า rebalance(tree, node) ฟังก์ชันนี้จะพิจารณาว่า node ฟังหรือไม่ ถ้าฟังก์ชันทำการ Rotate node นั้นให้ถูกต้องตามกฎที่เรียนในห้อง
- กฎที่ว่าคุณจะต้องเรียกใช้ Single Rotation หรือ Double Rotation ตามเงื่อนไขต่าง ๆ โดยได้ทำการทำ Single Rotation หรือ Double Rotation นั้นให้นำมาจากข้อที่แล้ว Copy มาลงข้อนี้ได้เลย หากพบ Bug ก็อย่าลืมหาค้นหาแก้ไขข้อที่แล้วด้วย
- การหาความสูงของ Node ใด ๆ ให้ นศ เรียกใช้ฟังก์ชัน height() ได้เลย (เบื้องต้น ไม่ต้องสนใจ BigO ครับ)
- เมื่อ นศ แก้ไขฟังก์ชัน rebalance() เสร็จเรียบร้อยแล้ว ให้ นศ ทำการศึกษาการทำงานของฟังก์ชัน Insert ของอาจารย์

- หลังจากนั้น ให้ นศ พิจารณาว่า ควรจะเรียกใช้ฟังก์ชัน `rebalance()` ที่ตำแหน่งใด ในฟังก์ชัน `Insert` เพื่อตรวจสอบและแก้ไข Node ที่พัง
- สำหรับข้อนี้ ต้นไม้จะมีขนาดใหญ่มาก ดังนั้นเราจะไม่ทำการเรียกฟังก์ชัน `printTree()` แต่เราจะสังเกตเอาว่าเกิด Operation ทาง AVL Tree ถูกต้องหรือไม่
- หากข้อนี้ นศ ติด Bug อันมหาศาล อาจารย์ขอแนะนำให้ใช้ Visual Studio ในการพัฒนา แล้วฝึกใช้ระบบ "Debug Project" เพื่อตามหา Bugs ให้เจอ แล้วกำจัดซะ
- อาจารย์ได้สร้างฟังก์ชัน `Main.BFT(tree)` เพื่อทำ Breadth First Traversal ต้นไม้ให้ออกมาอยู่ในรูปแบบที่สามารถตรวจสอบความถูกต้องได้ง่ายขึ้น (หวังว่าจะช่วยครับ)

C# code
<pre>AVLTree tree = generateTree2(); BFT(tree); tree.insert(73); // must perform SingleRotationFromLeft(Node 81) BFT(tree);</pre>
Output
<pre>BFT [51 30 69 18 42 63 87 12 24 36 45 57 66 81 93 15 21 27 33 39 48 54 60 75 84 90 96 72 78] Perform SingleRotationFromLeft(Node 81) BFT [51 30 69 18 42 63 87 12 24 36 45 57 66 75 93 15 21 27 33 39 48 54 60 72 81 90 96 73 78 84]</pre>

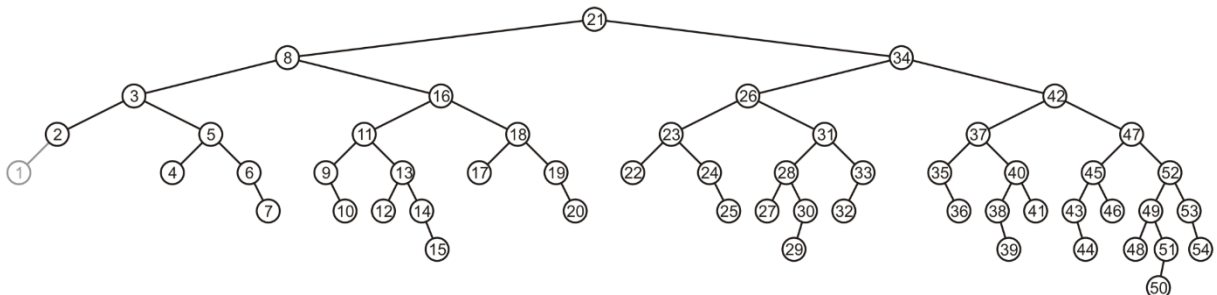
C# code
<pre>AVLTree tree = generateTree2(); BFT(tree); tree.insert(73); // must perform SingleRotationFromLeft(Node 81) tree.insert(77); // must perform DoubleRotationFromLeft(Node 87) tree.insert(76); // must perform SingleRotationFromLeft(Node 78) tree.insert(80); // must perform DoubleRotationFromRight(Node 69) tree.insert(74); // must perform SingleRotationFromRight(Node 72) tree.insert(64); // do nothing tree.insert(55); // must perform SingleRotationFromLeft(Node 69) tree.insert(70); // must perform DoubleRotationFromRight(Node 51) BFT(tree);</pre>
Output
<pre>BFT [51 30 69 18 42 63 87 12 24 36 45 57 66 81 93 15 21 27 33 39 48 54 60 75 84 90 96 72 78] Perform SingleRotationFromLeft(Node 81) Perform DoubleRotationFromLeft(Node 87) Perform SingleRotationFromLeft(Node 78) Perform DoubleRotationFromRight(Node 69) Perform SingleRotationFromRight(Node 72) Perform SingleRotationFromLeft(Node 69)</pre>

Perform DoubleRotationFromRight(Node 51) BFT [63 51 75 30 57 69 81 18 42 54 60 66 73 77 87 12 24 36 45 55 64 72 74 76 78 84 93 15 21 27 33 39 48 70 80 90 96]
--

C# code
<pre> AVLTree tree = generateTree2(); BFT(tree); tree.insert(73); // must perform SingleRotationFromLeft(Node 81) tree.insert(77); // must perform DoubleRotationFromLeft(Node 87) BFT(tree); </pre>
Output
<pre> BFT [51 30 69 18 42 63 87 12 24 36 45 57 66 81 93 15 21 27 33 39 48 54 60 75 84 90 96 72 78] Perform SingleRotationFromLeft(Node 81) Perform DoubleRotationFromLeft(Node 87) BFT [51 30 69 18 42 63 81 12 24 36 45 57 66 75 87 15 21 27 33 39 48 54 60 72 78 84 93 73 77 90 96] </pre>

4. จงปรับปรุงคลาส AVLTree (AVL Tree) เพื่อให้สามารถทำการ Delete ข้อมูล โดยยังคงความเป็น AVLTree ให้ได้ถูกต้อง

สำหรับโจทย์ข้อนี้ โครงสร้างข้อมูลชนิด AVLTree ได้ถูกสร้างขึ้นมาพร้อมข้อมูลเริ่มต้น มีหน้าตาประมาณนี้



BFT [21 8 34 3 16 26 42 2 5 11 18 23 31 37 47 1 4 6 9 13 17 19 22 24 28 33 35 40 45 52 7 10 12 14 20 25 27 30 32 36 38 41 43 46 49 53 15 29 39 44 48 51 54 50]

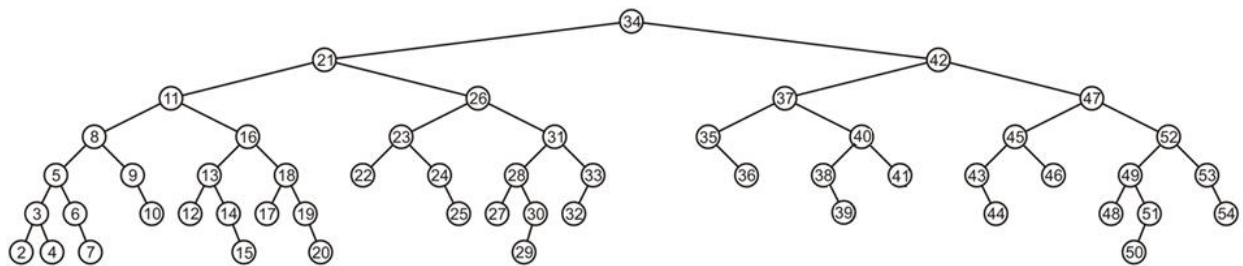
ต่อมาโจทย์จะทำการ Delete Node(1) แต่โหนดเดียวซึ่งนั้นจะทำให้ AVL Tree ต้นนี้ หนักขวาและ Unbalanced ทันที AVL Tree จะต้องทำการแก้ไขตัวเอง โดยไล่เช็ค balanced ตั้งแต่โหนดที่ถูกลบ (จริง ๆ ต้องเป็น node.parent) ไปจนถึง root หากพบโหนดที่พัง ให้แก้ไขโหนดนั้นให้กลับมา balanced ได้อีกครั้ง หากเป็นตัวอย่างข้างต้น จะต้องหมุนเพื่อ แก้ไขโหนดที่พังตามลำดับดังนี้

Perform SingleRotationFromRight(Node 3)

Perform DoubleRotationFromRight(Node 8)

Perform SingleRotationFromRight(Node 21)

ต้นไม้สุดท้ายจะมีหน้าตาดังนี้



BFT [34 21 42 11 26 37 47 8 16 23 31 35 40 45 52 5 9 13 18 22 24 28 33 36 38 41 43 46 49 53 3 6 10 12 14 17 19 25 27 30 32 39 44 48 51 54 2 4 7 15 20 29 50]

คำอธิบายเพิ่มเติม

- โจทย์ข้อนี้ อาจารย์ได้นำมาจาก Slide ที่เรียนในห้อง ดังนั้น กระบวนการ Delete และ Rebalance จะเป็นไปตาม Slide โดยละเอียดเลย อาจารย์ขออนุญาตไม่อธิบายซ้ำในที่นี้
- โจทย์ข้อนี้ นศ จะต้องนำ ฟังก์ชัน rebalance(tree, node), Single Rotation, Double Rotation ที่พัฒนาเสร็จเรียบร้อยแล้ว จากข้อที่แล้ว ลงมาเติมในช่องว่างข้อนี้ให้ถูกต้อง
- นศ จะต้อง Copy ฟังก์ชัน delete() ทั้งสองเวอร์ชัน (static and non-static) จากการบ้านที่แล้ว มาลงในข้อนี้ตามช่องว่างให้ถูกต้องด้วย
- ที่นี้ นศ จะต้องแก้ไขฟังก์ชัน delete() ที่ copy มา เพื่อให้มีการไล่ตรวจเช็ค balance ของแต่ละโหนดและทำการซ่อมโหนดหากพบว่าโหนดไหนพัง หลังจากการลบข้อมูล
- หลังจากนั้น ให้ นศ พิจารณาว่า ควรจะเรียกใช้ฟังก์ชัน rebalance() ที่ตำแหน่งใด ในฟังก์ชัน Insert เพื่อตรวจสอบและแก้ไข Node ที่พัง
- แนวคิดของอาจารย์ คือ node ไหนโดนลบ ก็ให้คุณ rebalance node.parent ครั้งหนึ่งแล้วก็ไล่ rebalance ขึ้นไปเรื่อย ๆ จนถึง root ครับ
- สำหรับข้อนี้ ต้นไม้จะมีขนาดใหญ่มาก ดังนั้นเราจะไม่ทำการเรียกฟังก์ชัน printTree() แต่เราจะสังเกตเอาว่าเกิด Operation ทาง AVL Tree ถูกต้องหรือไม่
- หากข้อนี้ นศ ติด Bug อันมหาศาล อาจารย์ขอแนะนำให้ใช้ NetBeans ในการพัฒนา แล้วฝึกใช้ระบบ "Debug Project" เพื่อตามหา Bugs ให้เจอ แล้วกำจัดซะ
- อาจารย์ได้สร้างฟังก์ชัน Main.BFT(tree) เพื่อทำ Breadth First Traversal ต้นไม้ให้ออกมาอยู่ในรูปแบบที่สามารถตรวจสอบความถูกต้องได้ง่ายขึ้น (หวังว่าจะช่วยครับ)

C# code

```
AVLTree tree = generateTree3();

BFT(tree);

tree.delete(1);
```

BFT(tree);
Output
<pre> BFT [21 8 34 3 16 26 42 2 5 11 18 23 31 37 47 1 4 6 9 13 17 19 22 24 28 33 35 40 45 52 7 10 12 14 20 25 27 30 32 36 38 41 43 46 49 53 15 29 39 44 48 51 54 50] Perform SingleRotationFromRight(Node 3) Perform DoubleRotationFromRight(Node 8) Perform SingleRotationFromRight(Node 21) BFT [34 21 42 11 26 37 47 8 16 23 31 35 40 45 52 5 9 13 18 22 24 28 33 36 38 41 43 46 49 53 3 6 10 12 14 17 19 25 27 30 32 39 44 48 51 54 2 4 7 15 20 29 50] </pre>

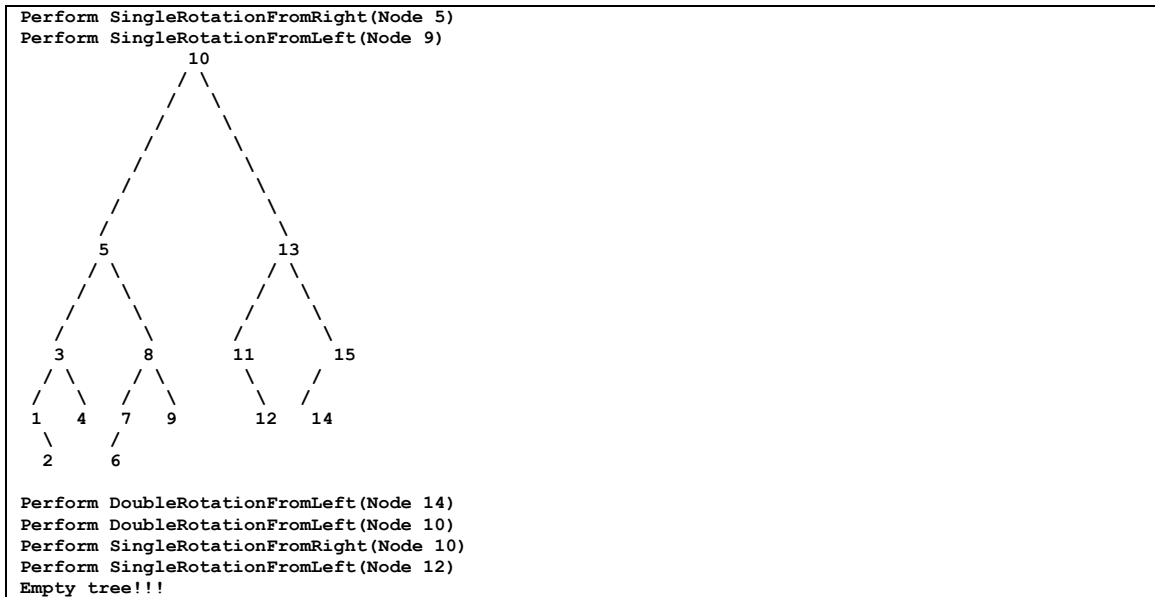
C# code
<pre> AVLTree tree = generateTree3(); BFT(tree); tree.delete(4); BFT(tree); </pre>
Output
<pre> BFT [21 8 34 3 16 26 42 2 5 11 18 23 31 37 47 1 4 6 9 13 17 19 22 24 28 33 35 40 45 52 7 10 12 14 20 25 27 30 32 36 38 41 43 46 49 53 15 29 39 44 48 51 54 50] Perform SingleRotationFromRight(Node 5) Perform DoubleRotationFromRight(Node 8) Perform SingleRotationFromRight(Node 21) BFT [34 21 42 11 26 37 47 8 16 23 31 35 40 45 52 3 9 13 18 22 24 28 33 36 38 41 43 46 49 53 2 6 10 12 14 17 19 25 27 30 32 39 44 48 51 54 1 5 7 15 20 29 50] </pre>

C# code
<pre> AVLTree tree = generateTree3(); BFT(tree); tree.delete(16); BFT(tree); </pre>
Output
<pre> BFT [21 8 34 3 16 26 42 2 5 11 18 23 31 37 47 1 4 6 9 13 17 19 22 24 28 33 35 40 45 52 7 10 12 14 20 25 27 30 32 36 38 41 43 46 49 53 15 29 39 44 48 51 54 50] Perform SingleRotationFromRight(Node 18) Perform DoubleRotationFromLeft(Node 17) Perform SingleRotationFromRight(Node 21) BFT [34 21 42 8 26 37 47 3 13 23 31 35 40 45 52 2 5 11 17 22 24 28 33 36 38 41 43 46 49 53 1 4 6 9 12 14 19 25 27 30 32 39 44 48 51 54 7 10 15 18 20 29 50] </pre>

C# code
<pre> AVLTree tree = generateTree3(); BFT(tree); tree.delete(21); BFT(tree); </pre>
Output
<pre> BFT [21 8 34 3 16 26 42 2 5 11 18 23 31 37 47 1 4 6 9 13 17 19 22 24 28 33 35 40 45 52 7 10 12 14 20 25 27 30 32 36 38 41 43 46 49 53 15 29 39 44 48 51 54 50] Perform SingleRotationFromRight(Node 23) Perform DoubleRotationFromRight(Node 26) Perform SingleRotationFromRight(Node 34) BFT [22 8 42 3 16 34 47 2 5 11 18 28 37 45 52 1 4 6 9 13 17 19 26 31 35 40 43 46 49 53 7 10 12 14 20 24 27 30 33 36 38 41 44 48 51 54 15 23 25 29 32 39 50] </pre>

C# code
<pre> AVLTree tree = generateTree3(); BFT(tree); tree.delete(35); BFT(tree); </pre>
Output
<pre> BFT [21 8 34 3 16 26 42 2 5 11 18 23 31 37 47 1 4 6 9 13 17 19 22 24 28 33 35 40 45 52 7 10 12 14 20 25 27 30 32 36 38 41 43 46 49 53 15 29 39 44 48 51 54 50] Perform DoubleRotationFromRight(Node 37) Perform SingleRotationFromRight(Node 42) BFT [21 8 34 3 16 26 47 2 5 11 18 23 31 42 52 1 4 6 9 13 17 19 22 24 28 33 38 45 49 53 7 10 12 14 20 25 27 30 32 37 40 43 46 48 51 54 15 29 36 39 41 44 50] </pre>

C# code
<pre> AVLTree tree1 = new AVLTree(); int[] keyList = { 1, 15, 3, 13, 5, 11, 9, 10, 8, 4, 12, 7, 2, 6, 14 }; for (int i = 0; i < keyList.Length; i++) tree1.insert(keyList[i]); tree1.printTree(); for (int i = 0; i < keyList.Length; i++) tree1.delete(keyList[i]); tree1.printTree(); </pre>
Output
<pre> Perform DoubleRotationFromRight(Node 1) Perform SingleRotationFromLeft(Node 15) Perform DoubleRotationFromRight(Node 3) Perform DoubleRotationFromLeft(Node 11) Perform SingleRotationFromLeft(Node 13) </pre>



5. จงปรับปรุงคลาส BSTree2 เพื่อให้สามารถดำเนินการตั้ง BSTree รุ่นแรก โดยไม่ใช่ Recursion แต่ให้ใช้ Iteration (While Loop) แทน

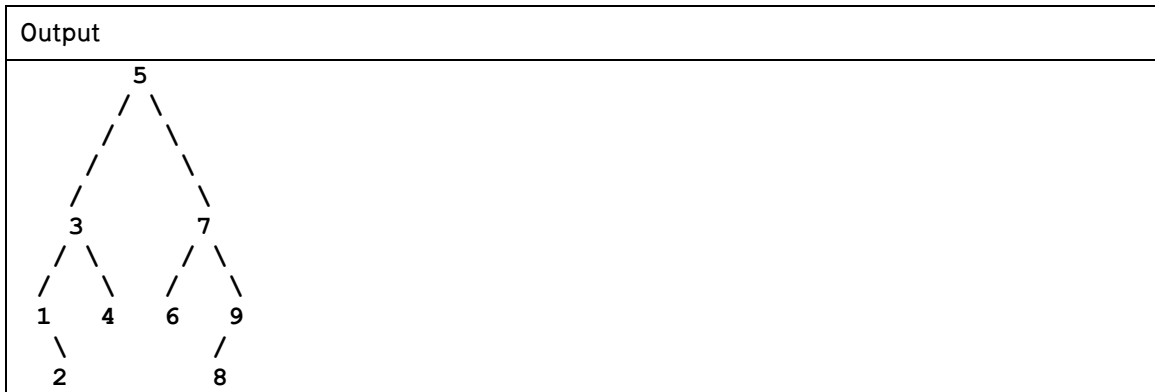
โจทย์ข้อนี้ ขอให้คุณ Copy ฟังก์ชัน find(), findMin(), findMax(), insert() จากการบ้านที่แล้วมาลง แต่คุณจะเปลี่ยนวิธีการที่ฟังก์ชันเหล่านี้เรียกตัวเอง (Recursion) เป็นการเรียกใช้วนซ้ำ (Iteration) แทน

ข้อนี้ขอให้นึกถึงตอนทำ LinkedList คือ จะเริ่มทำ Node current = head; แล้วก็วน while loop (current!=null) โดยมีการปรับปรุง current = current.next; ประมวลผลไปเรื่อย ๆ จนถึงกรณีฐาน

การประยุกต์ใช้สำหรับ BSTree ก็เช่นกัน คือทำ Node current = root; แล้วก็ทำ current = current.left (แล้วแต่กรณี) ทำไปเรื่อย ๆ จนตกสาย

ดังนั้น ข้อนี้ จะต้องไม่มีการเรียกใช้ตัวเอง (Recursion) เลย

C# code
<pre> BSTree2 tree = new BSTree2(); int[] keys = { 5, 3, 1, 2, 4, 7, 6, 9, 8 }; for (int i = 0; i < keys.Length; i++) tree.insert(keys[i]); tree.printTree(); </pre>



C# code
<pre>BSTree2 tree = new BSTree2(); int[] keys = { 5, 3, 1, 2, 4, 7, 6, 9, 8 }; for (int i = 0; i < keys.Length; i++) tree.insert(keys[i]); Node x = tree.find(2); Console.WriteLine(x.key); x = tree.find(10); Console.WriteLine(x);</pre>
Output
2

C# code
<pre>BSTree2 tree = new BSTree2(); int[] keys = { 5, 3, 1, 2, 4, 7, 6, 9, 8 }; for (int i = 0; i < keys.Length; i++) tree.insert(keys[i]); Node x = tree.findMin(); Node y = tree.findMax(); Console.WriteLine("Min = " + x.key + ", Max = " + y.key);</pre>
Output
Min = 1, Max = 9

6. จงปรับปรุงคลาส SplayTree (Splay Tree) เพื่อให้มีคุณสมบัติของ Self-Adjusting Tree และทำงานได้อย่างถูกต้อง

[1] สำหรับโจทย์ข้อนี้อาจารย์ขอแนะนำให้ นศ เริ่มทำการปรับปรุงฟังก์ชัน `zig(Node x)` เพื่อทำหน้าที่ดัน Node `x` ขึ้นไปข้างบนอีก 1 เลเวล (ทำ `zig` แค่ครั้งเดียว) ให้เสร็จสิ้นเสียก่อน

ประเด็นที่ต้องพิจารณาสำหรับฟังก์ชัน `zig(x)` จะมีประมาณ 5 กรณีดังนี้

1. กรณี `x` เป็น root: แบบนี้ไม่ต้องทำอะไร ให้แจ้งออกไปว่า Cannot perform Zig operation on the root node
2. - 3. กรณี `x` เป็นลูกของ root (อยู่ทางด้านซ้ายหรือด้านขวาแล้วแต่กรณี):
 - อาจารย์ขอแนะนำให้คุณสร้าง `x.parent` ให้เป็นตัวแปร `y` เสีย เพื่อลด Bugs
 - ตรวจสอบก่อนว่า `x` อยู่ด้านซ้ายหรือด้านขวาของ `y` (กำหนดกรณี ที่ 2-3)
 - ถ้า `x` มีลูกวงใน ก็ให้โอนลูกของ `x` ที่เป็นวงใน ไปให้ `y` (ในกรณีที่ถูกต้อง)
 - เปลี่ยน `y` มาเป็นลูกของ `x` (ในกรณีที่ถูกต้อง)
 - กำหนดให้ root ใหม่เป็น `x` (อย่าลืมตัด parent ของ root ออก)
 - เวลาเปลี่ยน parent \leftrightarrow child กับคู่ไหนก็ได้ ๆ อย่าลืมผูกทั้งขาไปและขากลับนะครับ
4. - 5. กรณี `x` ไม่ใช่ลูกของ root (เป็นหลานของไหนก็ได้ ๆ):
 - อาจารย์ขอแนะนำให้คุณสร้าง `x.parent` และ `x.parent.parent` ให้เป็นตัวแปร `y` และ `w` เสีย เพื่อลด Bugs
 - ตรวจสอบก่อนว่า `x` อยู่ด้านซ้ายหรือด้านขวาของ `y` (กำหนดกรณี ที่ 4-5)
 - ถ้า `x` มีลูกวงใน ก็ให้โอนลูกของ `x` ที่เป็นวงใน ไปให้ `y` (ในกรณีที่ถูกต้อง)
 - เปลี่ยน `y` มาเป็นลูกของ `x` (ในกรณีที่ถูกต้อง)
 - เปลี่ยน `x` มาเป็นลูกของ `w` (ในกรณีที่ถูกต้อง)
 - เวลาเปลี่ยน parent \leftrightarrow child กับคู่ไหนก็ได้ ๆ อย่าลืมผูกทั้งขาไปและขากลับนะครับ

[2] เมื่อพัฒนาฟังก์ชัน `zig()` เสร็จแล้ว คุณก็จะสามารถพัฒนาฟังก์ชัน `zigzig(Node x)` และ `zigzag(Node x)` โดยการเรียกฟังก์ชัน `zig()` อีกที เป็นจำนวน 2 ครั้ง ซึ่งนิยามและคำอธิบายฟังก์ชันทั้งสองจะอยู่ใน Slide ขอให้ นศ กลับไปดูด้วย ... ตรงนี้ ทำให้ง่ายนะครับ อย่าไปเขียนอะไรให้ซับซ้อน คำตอบมีแค่ 2 บรรทัดจบ ถ้าเขียนมากกว่านี้ แปลว่าผิดละ

[3] เมื่อพัฒนาฟังก์ชันทั้งสามเสร็จแล้ว คุณก็จะสามารถพัฒนาฟังก์ชัน `splay(Node x)` ได้ โดยคุณจะต้องพา Node `x` ไปจนถึง root โดยเรียกฟังก์ชันทั้งสามซ้ำ ๆ ไปเรื่อย ๆ โดยใช้ While loop หลักการคือ

- ถ้า Node `x` เป็นลูกของ root อยู่แล้วก็ทำ `zig(x)` เดียว
- ถ้า Node `x` เป็นหลานของใครสักคน และ `x` อยู่วงนอก ก็ทำ `zigzig(x)`
- ถ้า Node `x` เป็นหลานของใครสักคน และ `x` อยู่วงใน ก็ทำ `zigzag(x)`

- ทำไปเรื่อย ๆ จน x ไปอยู่ที่ root ก็จบ
- โจทย์ข้อนี้กำหนดให้คุณทำ splay() โดยใช้ while loop นะครับ ห้ามใช้ recursion (ฟังก์ชันเรียกตัวเอง)
- วิธีการทำก็คล้าย ๆ กับการทำ LinkedList คือ คุณเริ่มต้นจาก Node current = node แล้วคุณก็ทำ current = current.parent ไปเรื่อย ๆ จนไปถึง root
- ถ้าคุณทำ BSTree ver 2 ของปัญหาที่แล้วมา คุณจะเข้าใจว่า อาจารย์หมายถึงอะไร

[4] ถึงขั้นตอนนี้ ขอให้คุณ Copy ฟังก์ชัน insert(), find(), findMin() มาจากคลาส BSTree2 มาใส่ในคลาส SplayTree ด้วย โดยฟังก์ชันทั้งสามนี้ จะต้องถูกพัฒนาด้วยหลักการ Iteration (ไม่ใช่ Recursion) นะครับ คือ จะใช้ loop ในพัฒนาและไม่มี การเรียกตัวเอง


- ข้อดีของฟังก์ชันที่พัฒนาด้วยวิธีการนี้คือ สามารถเดินทางไบนารีต้นไม้ที่มีความลึกมาก ๆ ได้โดยไม่เกิด StackOverflow exception (ถ้าเป็น Recursion อาจจะได้)
- เมื่อคุณ Copy โค้ดมาลงในข้อนี้เสร็จเรียบร้อยแล้ว ก็ขอให้คุณหาที่ใส่ splay() ในฟังก์ชัน insert() กับ find() (ยกเว้น findMin()) ลองพิจารณาว่าคุณควรใส่ตรงไหน เพื่อให้ฟังก์ชันทั้งสองนี้ มีคุณสมบัติเป็น Splay Tree คือ โหนดล่าสุดที่ประมวลผลจะถูกพามาอยู่ที่ root
- อาจารย์ได้ปรับปรุงฟังก์ชัน find() ให้รับพารามิเตอร์ 2 ตัวคือ find(int search_key, boolean withSplay) โดยตัวแปร withSplay นี้จะเอาไว้กำหนดว่า การค้นหาโหนดครั้งนี้ จะ splay โหนดขึ้นมาที่ root ด้วยหรือไม่ ถ้ากำหนดเป็น false ก็จะไม่สplay ค้นหาเฉย ๆ ไม่ต้อง splay ก็ขอให้ นศ แกะไขฟังก์ชันให้รองรับฟีเจอร์นี้ด้วย

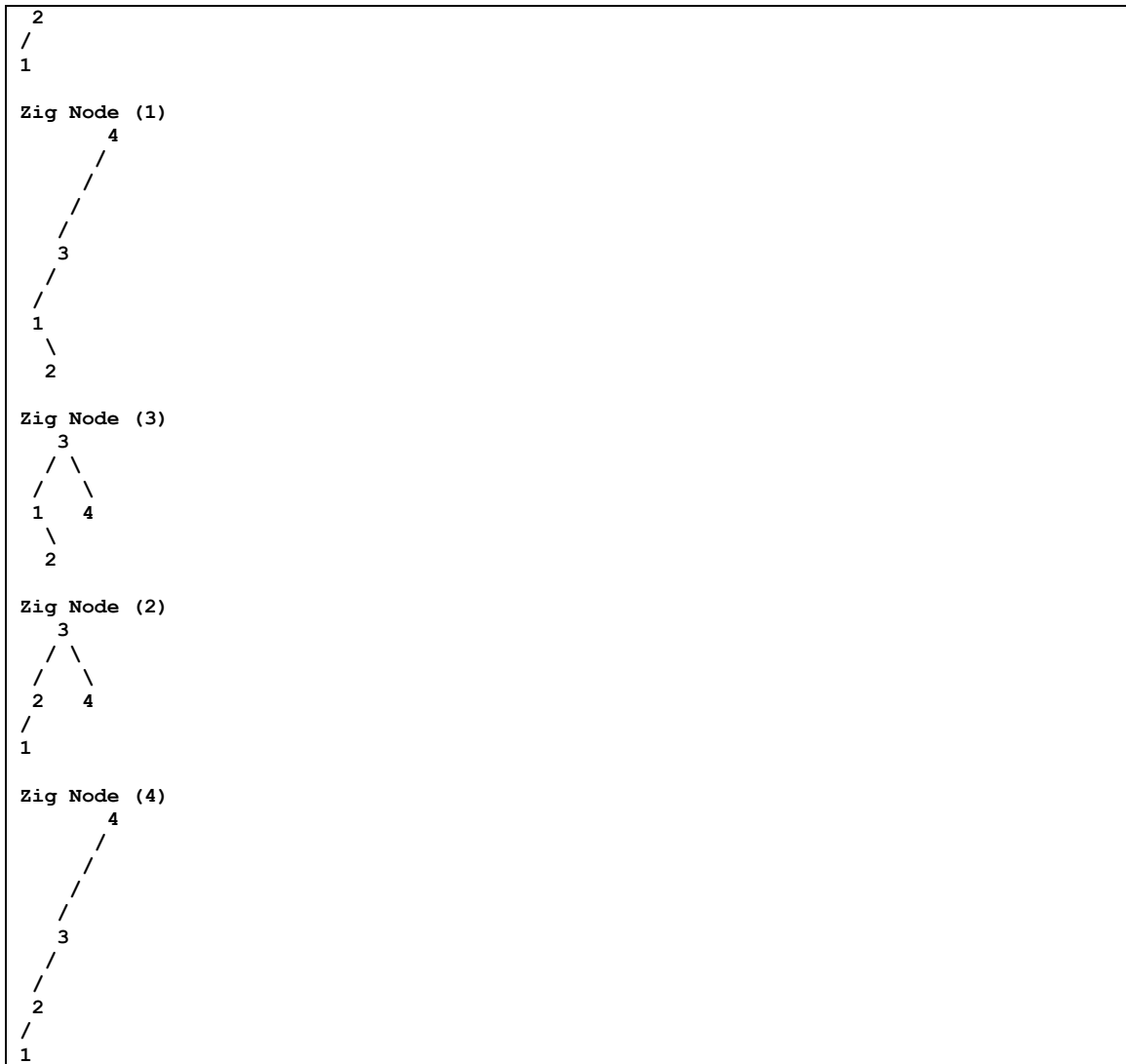
[5] พัฒนาฟังก์ชัน delete() สำหรับ Splay Tree โดยกระบวนการ delete() ที่ว่านี้ จะแตกต่างออกไปจาก Binary Search Tree ทั่วไปนะครับ อัลกอริทึมคือ

- ค้นหา Node ที่มี key ตามที่ระบุ
- Splay Node นั้นขึ้นมาที่ root (เรียก tree ต้นนี้ว่า tree1)
- เอา right sub-tree ของ root นั้นไปปลูก Splay Tree ต้นใหม่ (เรียก tree ต้นนี้ว่า tree2)
- Splay min ของ right-sub tree ต้นนั้นขึ้นมาที่ root
- โอนลูกทางด้านซ้ายของ tree1 มาให้ tree2
- กำหนด root ใหม่เป็น root ของ tree2 (แล้วลบ root เดิม)

[6] จดตัวเลขเวลาเอาไว้ ว่า BSTree กับ SplayTree ใครประมวลผลได้เร็วกว่ากัน หากคำตอบให้อาจารย์หนอยก็ถือว่าทำไม่ ทั้ง ๆ ที่โครงสร้างข้อมูลทั้งสองนี้ต่างก็ไม่มีระดับความสูงของต้นไม้ ว่าจะต้องเป็น $O(\log n)$ ตลอดเวลา ต้นไม้ทั้งสองนี้ สามารถเรียงตัวกันได้เป็น Degenerate tree ได้เลยนะเออ ถ้างั้นความเร็วก็พอ ๆ กันสิ ใช่ไหม? เตียวอาจารย์จะเอาไปถาม ในข้อสอบครับ

ตัวเลขที่ได้นี้ จะไม่เหมือนกันนะครับ เพราะอาจารย์สุ่มใหม่ทุกครั้ง ขึ้นกับประสิทธิภาพการประมวลผลของ Server ด้วย ดังนั้นการที่ตัวเลขนี้ไม่ตรงกับ คำตอบของอาจารย์จะไม่ทำให้คุณโดนหักคะแนนครับ ตัวเลขนี้ ไม่ตรวจคะแนนในการบ้านนี้ ครับ แสดงให้คุณเห็นเฉย ๆ

C# code
<pre>SplayTree tree = new SplayTree(); for (int i = 0; i < 4; i++) tree.insert(i + 1); // with splay() tree.printTree(); Node node; Console.WriteLine("Zig Node (1)"); node = tree.find(1, false); // false means no splay() tree.zig(node); tree.printTree(); Console.WriteLine("Zig Node (3)"); node = tree.find(3, false); tree.zig(node); tree.printTree(); Console.WriteLine("Zig Node (2)"); node = tree.find(2, false); tree.zig(node); tree.printTree(); Console.WriteLine("Zig Node (4)"); node = tree.find(4, false); tree.zig(node); tree.printTree();</pre>
Output




C# code

```

SplayTree tree = new SplayTree();
int[] keyList = { 5, 7, 2, 3, 1, 6, 8 };
for (int i = 0; i < keyList.Length; i++)
    tree.insert(keyList[i]); // with splay()
tree.printTree();

Node node1, node2, node3, node5, node7;
//This find(key, false) means do not splay the node
node1 = tree.find(1, false);
node2 = tree.find(2, false);

```

```
node3 = tree.find(3, false);
node5 = tree.find(5, false);
node7 = tree.find(7, false);

Console.WriteLine("ZigZig Node (1)");
tree.zigzig(node1);
tree.printTree();

Console.WriteLine("ZigZag Node (5)");
tree.zigzag(node5);
tree.printTree();

Console.WriteLine("ZigZag Node (5)");
tree.zigzag(node5);
tree.printTree();

Console.WriteLine("ZigZag Node (7)");
tree.zigzag(node7);
tree.printTree();

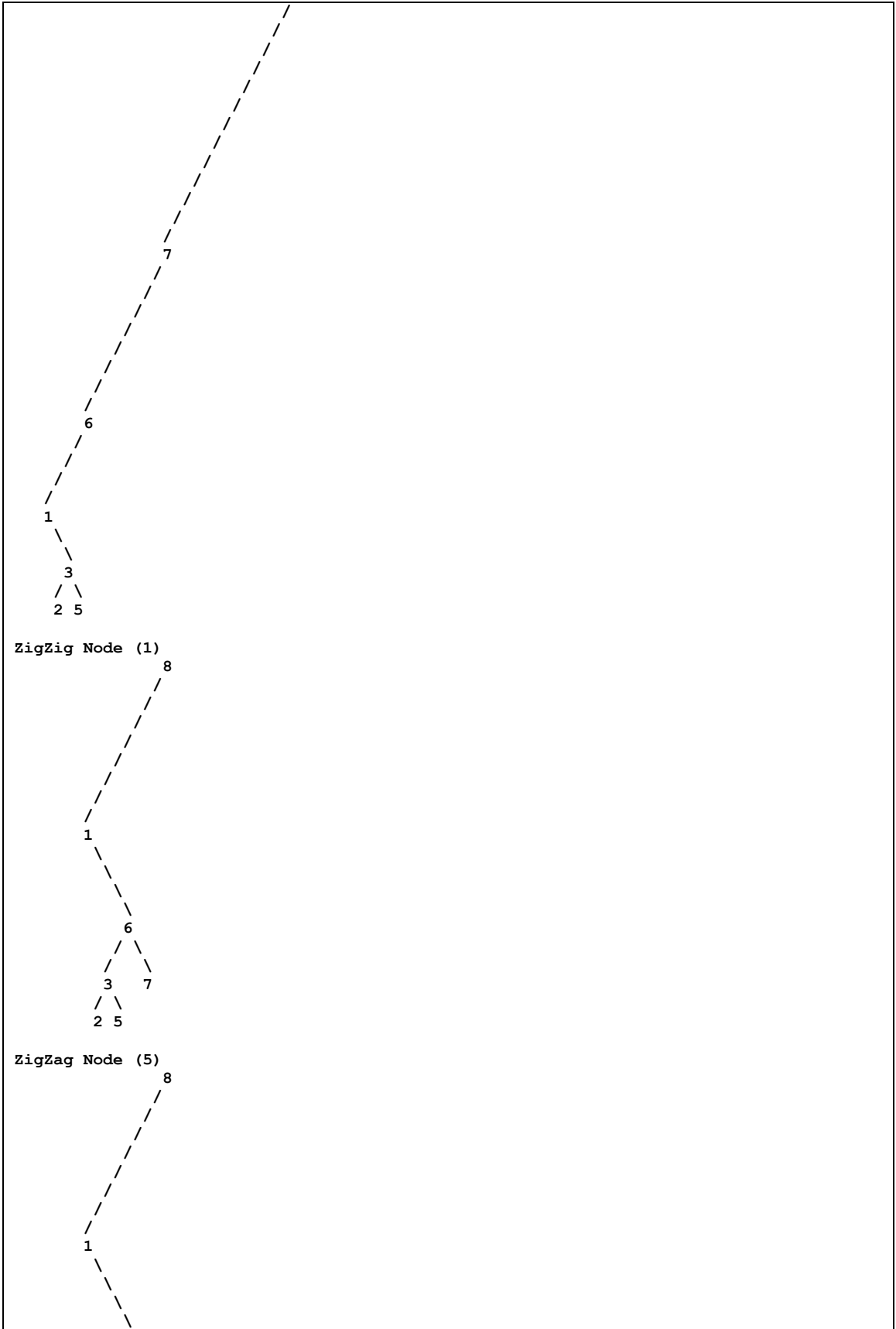
Console.WriteLine("ZigZag Node (2)");
tree.zigzag(node2);
tree.printTree();

Console.WriteLine("ZigZag Node (3)");
tree.zigzag(node3);
tree.printTree();

Console.WriteLine("ZigZig Node (7)");
tree.zigzig(node7);
tree.printTree();
```

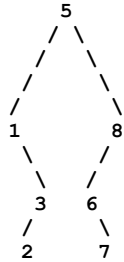
Output



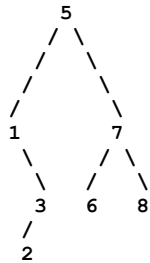




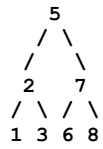
ZigZag Node (5)



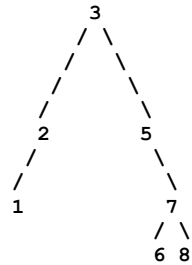
ZigZag Node (7)



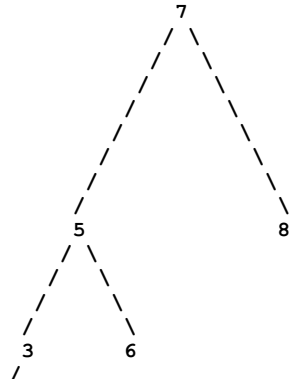
ZigZag Node (2)



ZigZag Node (3)



ZigZig Node (7)



/
2
/
1

C# code

```
Stopwatch stopwatch = new Stopwatch();
Random rnd = new Random();

BSTree2 tree1 = new BSTree2();
stopwatch.Start();
int N = 40000;
for (int i = 0; i < N; i++)
    tree1.insert(i);
stopwatch.Stop();
Console.WriteLine("Time for sequentially inserting " + N + " objects into BST = " + stopwatch.ElapsedMilliseconds + " msec");

stopwatch.Restart();
for (int i = 0; i < N; i++)
    tree1.find(rnd.Next(0, N));
stopwatch.Stop();
Console.WriteLine("Time for finding " + N + " different objects in BST = " + stopwatch.ElapsedMilliseconds + " msec");

SplayTree tree2 = new SplayTree();
stopwatch.Restart();
for (int i = 0; i < N; i++)
    tree2.insert(i);
stopwatch.Stop();
Console.WriteLine("Time for sequentially inserting " + N + " objects into SplayTree = " + stopwatch.ElapsedMilliseconds + " msec");

stopwatch.Restart();
for (int i = 0; i < N; i++)
    tree2.find(rnd.Next(0, N), true);
stopwatch.Stop();
Console.WriteLine("Time for finding " + N + " different objects in SplayTree = " + stopwatch.ElapsedMilliseconds + " msec");
```

Console.WriteLine("Which one is faster: BSTree or SplayTree?");
Output
Time for sequentially inserting 40000 objects into BST = 3217 msec Time for finding 40000 different objects in BST= 2806 msec Time for sequentially inserting 40000 objects into SplayTree = 22 msec Time for finding 40000 different objects in SplayTree = 32 msec Which one is faster: BSTree or SplayTree?

1. โปรดใช้ Starter code ที่อาจารย์แนบให้