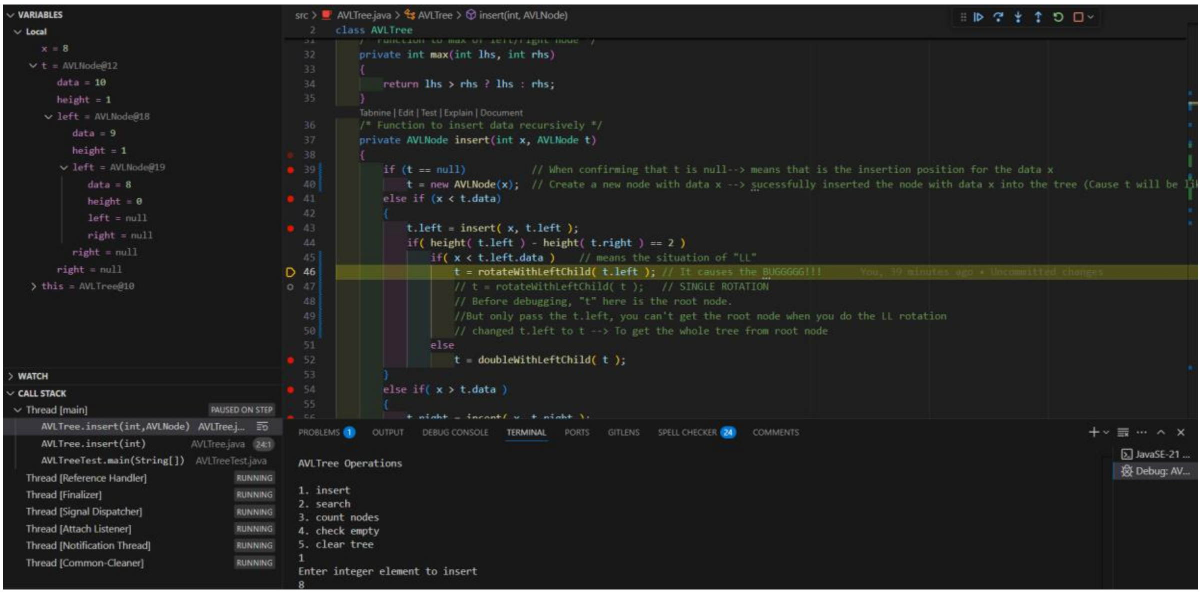
Software engineering practices
HW1 : Debug an AVL implementation

學號： 112526011　　　　姓名： 林睿瀚

A. Bug 1 : 在 AVL Tree rotation 的 旋轉邏輯(1) 及 傳入參數(2) 有誤，造成產生循環引用的樹狀結構，因此在 postorder() 遍歷整個樹的節點時，會不斷沿著這個循環遞迴，最終造成 StackOverflowError

✓ Sample test case : insert elements 10 -> 9 -> 8

因為 旋轉邏輯(1) 及 傳入參數(2) 有誤，在執行此函數後會造成循環引用的樹狀結構 8 -> 9 -> 8 -> 9 -> 8 -> 9 …………

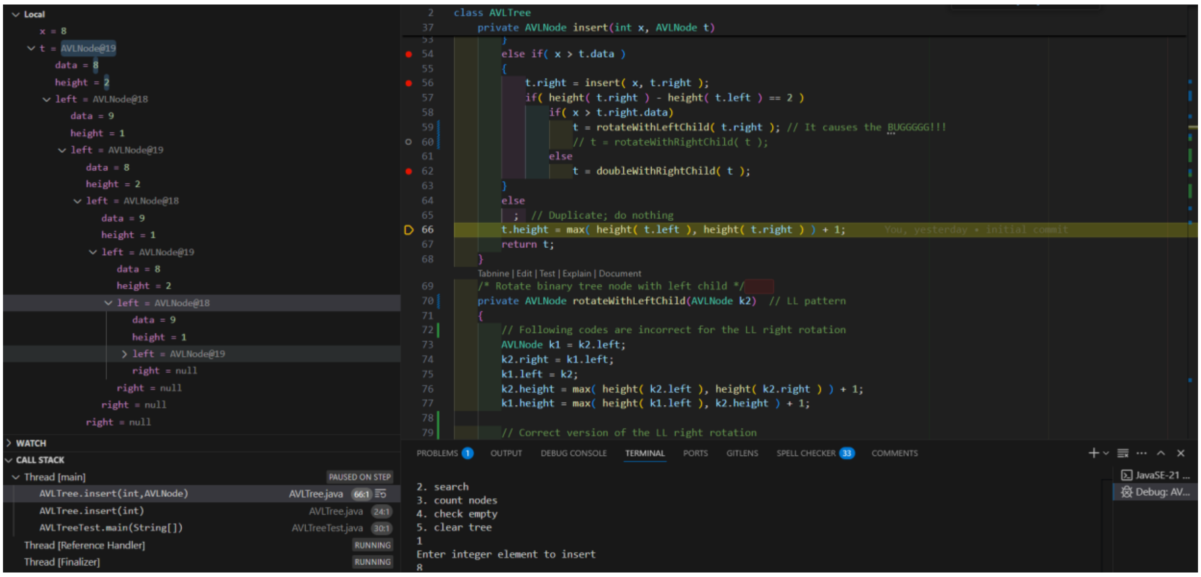


```
x = 8
t = AVLNode@12
  data = 10
  height = 1
  left = AVLNode@18
    data = 9
    height = 1
    left = AVLNode@19
      data = 8
      height = 0
      left = null
      right = null
    right = null
  right = null
this = AVLTree@10
```

```
Thread [main]                              PAUSED ON STEP
  AVLTree.insert(int,AVLNode)  AVLTree.j...
  AVLTree.insert(int)          AVLTree.java  24:1
  AVLTreeTest.main(String[])   AVLTreeTest.java
Thread [Reference Handler]                 RUNNING
Thread [Finalizer]                         RUNNING
Thread [Signal Dispatcher]                 RUNNING
Thread [Attach Listener]                   RUNNING
Thread [Notification Thread]               RUNNING
Thread [Common-Cleaner]                    RUNNING
```

```java
class AVLTree
    private int max(int lhs, int rhs)
    {
        return lhs > rhs ? lhs : rhs;
    }
    Tabnine | Edit | Test | Explain | Document
    /* Function to insert data recursively */
    private AVLNode insert(int x, AVLNode t)
    {
        if (t == null)          // When confirming that t is null--> means that is the insertion position for the data x
            t = new AVLNode(x);  // Create a new node with data x --> sucessfully inserted the node with data x into the tree (Cause t will be li
        else if (x < t.data)
        {
            t.left = insert( x, t.left );
            if( height( t.left ) - height( t.right ) == 2 )
                if( x < t.left.data )    // means the situation of "LL"
                    t = rotateWithLeftChild( t.left ); // It causes the BUGGGGG!!!      You, 39 minutes ago • Uncommitted changes
                    // t = rotateWithLeftChild( t );   // SINGLE ROTATION
                    // Before debugging, "t" here is the root node.
                    //But only pass the t.left, you can't get the root node when you do the LL rotation
                    // changed t.left to t --> To get the whole tree from root node
                else
                    t = doubleWithLeftChild( t );
        }
        else if( x > t.data )
        {
```

```
AVLTree Operations

1. insert
2. search
3. count nodes
4. check empty
5. clear tree
1
Enter integer element to insert
8
```



```
x = 8
t = AVLNode@19
  data = 8
  height = 2
  left = AVLNode@18
    data = 9
    height = 1
    left = AVLNode@19
      data = 8
      height = 2
      left = AVLNode@18
        data = 9
        height = 1
        left = AVLNode@19
          data = 8
          height = 2
          left = AVLNode@18
            data = 9
            height = 1
            > left = AVLNode@19
            right = null
          right = null
        right = null
      right = null
```

```
Thread [main]                              PAUSED ON STEP
  AVLTree.insert(int,AVLNode)  AVLTree.java  66:1
  AVLTree.insert(int)          AVLTree.java  24:1
  AVLTreeTest.main(String[])   AVLTreeTest.java  30:1
Thread [Reference Handler]                 RUNNING
Thread [Finalizer]                         RUNNING
```

```java
class AVLTree
    private AVLNode insert(int x, AVLNode t)
        else if( x > t.data )
        {
            t.right = insert( x, t.right );
            if( height( t.right ) - height( t.left ) == 2 )
                if( x > t.right.data)
                    t = rotateWithLeftChild( t.right ); // It causes the BUGGGGG!!!
                    // t = rotateWithRightChild( t );
                else
                    t = doubleWithRightChild( t );
        }
        else
            ;  // Duplicate; do nothing
        t.height = max( height( t.left ), height( t.right ) ) + 1;      You, yesterday • initial commit
        return t;
    }
    Tabnine | Edit | Test | Explain | Document
    /* Rotate binary tree node with left child */
    private AVLNode rotateWithLeftChild(AVLNode k2)  // LL pattern
    {
        // Following codes are incorrect for the LL right rotation
        AVLNode k1 = k2.left;
        k2.right = k1.left;
        k1.left = k2;
        k2.height = max( height( k2.left ), height( k2.right ) ) + 1;
        k1.height = max( height( k1.left ), k2.height ) + 1;

        // Correct version of the LL right rotation
```

```
2. search
3. count nodes
4. check empty
5. clear tree
1
Enter integer element to insert
8
```

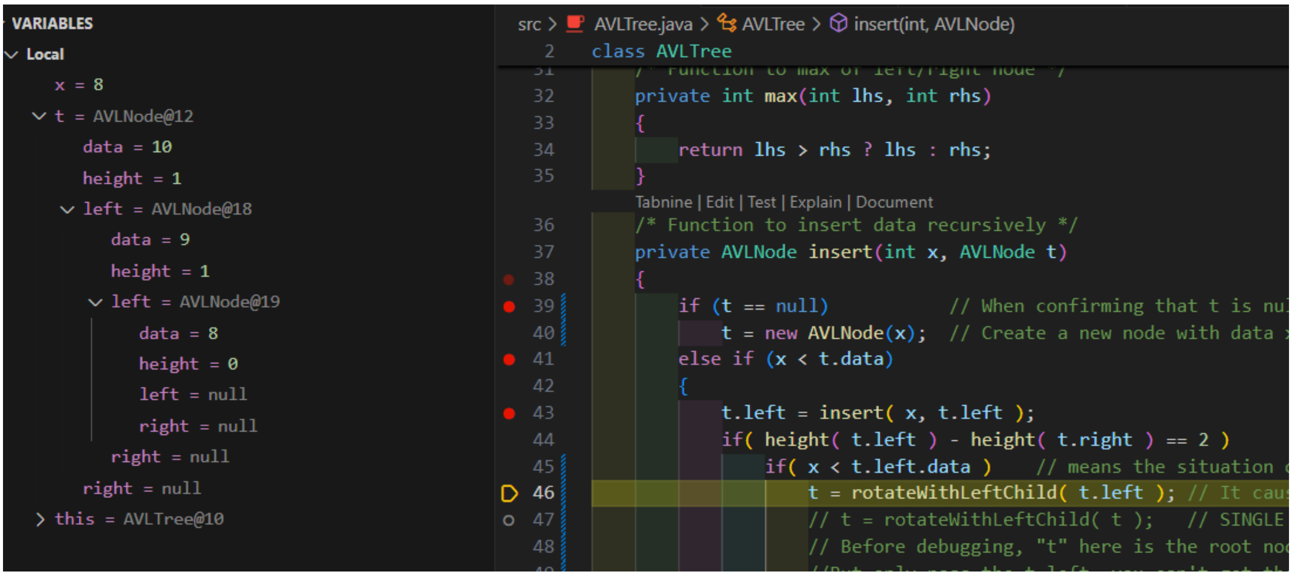
- ✓ 旋轉邏輯(1) 錯誤解釋 ： 以 LL 旋轉為例，正確的右旋轉應該是讓 k2（BF > 1 的 node）的左子指向 k1 的右子，再讓 k1 的右子指向 k2，而這裡卻將 k2 的右子指向 k1 的左子，再讓 k1 的左子指向 k2，而 RR 的旋轉邏輯亦同樣有相似於上方例子的邏輯錯誤。

  */ BF : Balance Factor

- ✓ 錯誤處及改正後的程式碼 ： *Red means the bug / Green means the corrections*

```
/* Function to insert data recursively */
    private AVLNode insert(int x, AVLNode t)
    {
        if (t == null)
            t = new AVLNode(x);
        else if (x < t.data)
        {
            t.left = insert( x, t.left );
            if( height( t.left ) - height( t.right ) == 2 )
                if( x < t.left.data )     // means the situation of "LL"
                    // t = rotateWithLeftChild( t.left ); // It causes the BUGGGGG!!!
                    t = rotateWithLeftChild( t );    // SINGLE ROTATION
                    // Before debugging, "t" here is the root node.
                    //But only pass the t.left, you can’t get the root node when you do the LL rotation
                    // changed t.left to t --> To get the whole tree from root node
                else
                    t = doubleWithLeftChild( t );
        }
        else if( x > t.data )
        {
            t.right = insert( x, t.right );
            if( height( t.right ) - height( t.left ) == 2 )
                if( x > t.right.data)
                    // t = rotateWithLeftChild( t.right ); // It causes the BUGGGGG!!!
                    t = rotateWithRightChild( t );
                else
                    t = doubleWithRightChild( t );
        }
        else
          ;  // Duplicate; do nothing
        t.height = max( height( t.left ), height( t.right ) ) + 1;
        return t;
    }

/* Rotate binary tree node with left child */
    private AVLNode rotateWithLeftChild(AVLNode k2)  // LL pattern
    {
        // Following codes are incorrect for the LL right rotation
        AVLNode k1 = k2.left;
        k2.right = k1.left;
        k1.left = k2;
        k2.height = max( height( k2.left ), height( k2.right ) ) + 1;
        k1.height = max( height( k1.left ), k2.height ) + 1;

        // Correct version of the LL right rotation
        // AVLNode k1 = k2.left;
        // k2.left = k1.right;
        // k1.right = k2;
        // k2.height = max( height( k2.left ), height( k2.right ) ) + 1;
        // k1.height = max(height(k1.left), height(k1.right)) + 1;

        return k1;
    }

    /* Rotate binary tree node with right child */
    private AVLNode rotateWithRightChild(AVLNode k1) // RR pattern
    {
        // Following codes are incorrect for the RR left rotation
        AVLNode k2 = k1.right;
        k1.left = k2.right;
        k2.right = k1;
        k1.height = max( height( k1.left ), height( k1.right ) ) + 1;
        k2.height = max( height( k2.right ), k1.height ) + 1;

        // Correct version of the RR left rotation5
        // AVLNode k2 = k1.right;
        // k1.right = k2.left;
        // k2.left = k1;
        // k1.height = max( height( k1.left ), height( k1.right ) ) + 1;
        // k2.height = max(height(k2.left), height(k2.right)) + 1;

        return k2;
    }
```

✓ 傳入參數(2) 錯誤解釋 : 以下圖為例，t 在此時指向原 root node (data=10)，而傳入參數為 t.left(data=9)，因此造成在 LL 旋轉 function 內，是讀不到原 root node(data=10) 這個節點的，因此在 function return 後，AVL Tree 失去了 AVLnode(data=10)



VARIABLES
Local

```
x = 8
t = AVLNode@12
  data = 10
  height = 1
  left = AVLNode@18
    data = 9
    height = 1
    left = AVLNode@19
      data = 8
      height = 0
      left = null
      right = null
    right = null
  right = null
this = AVLTree@10
```

src > AVLTree.java > AVLTree > insert(int, AVLNode)

```
class AVLTree
    /* Function to max of left/right node */
    private int max(int lhs, int rhs)
    {
        return lhs > rhs ? lhs : rhs;
    }
    Tabnine | Edit | Test | Explain | Document
    /* Function to insert data recursively */
    private AVLNode insert(int x, AVLNode t)
    {
        if (t == null)          // When confirming that t is nu
            t = new AVLNode(x);  // Create a new node with data x
        else if (x < t.data)
        {
            t.left = insert( x, t.left );
            if( height( t.left ) - height( t.right ) == 2 )
                if( x < t.left.data )    // means the situation o
                    t = rotateWithLeftChild( t.left ); // It caus
                    // t = rotateWithLeftChild( t );    // SINGLE
                    // Before debugging, "t" here is the root noc
```

B. Bug 2 : search function 混用 迴圈與遞迴操作，而因為使用不必要的遞迴操作，容易造成堆疊深度過大，而 StackOverflowError

✓ 錯誤處及改正後的程式碼 : *<u>Red means the bug / Green means the corrections</u>*

(單獨使用 loop or recursion operation for traversal searching to the tree)

```
private boolean search(AVLNode r, int val)
    {
        // Mixed use the operation of recursion and loop
        // --> the recursion operation is not necessary because we can only
traverse the tree by loops
        // boolean found = false;
        // while ((r != null) && !found)
        // {
        //      int rval = r.data;
        //      if (val < rval)
        //          r = r.left;
        //      else if (val > rval)
        //          r = r.right;
        //      else
        //      {
        //          found = true;
        //          break;
        //      }
        //      found = search(r, val);
        // }
        // return found;

        // Correct version by only using the loop operation
        // while (r != null) {
        //      if (val < r.data)
        //          r = r.left;
        //      else if (val > r.data)
        //          r = r.right;
        //      else
        //          return true;
        // }
        // return false;

        // Correct version by only using the recursion operation
        if (r == null)
            return false;
        if (val == r.data)
            return true;
        else if (val < r.data)
            return search(r.left, val);
        else // val > r.data
            return search(r.right, val);
    }
```