Lesson 03 Demo 03

Implementing an AVL Tree

Objective: To demonstrate an AVL tree implementation in JavaScript using node insertion and self-balancing rotations to maintain height-balanced binary search tree properties

Tools required: Visual Studio Code and Node.js

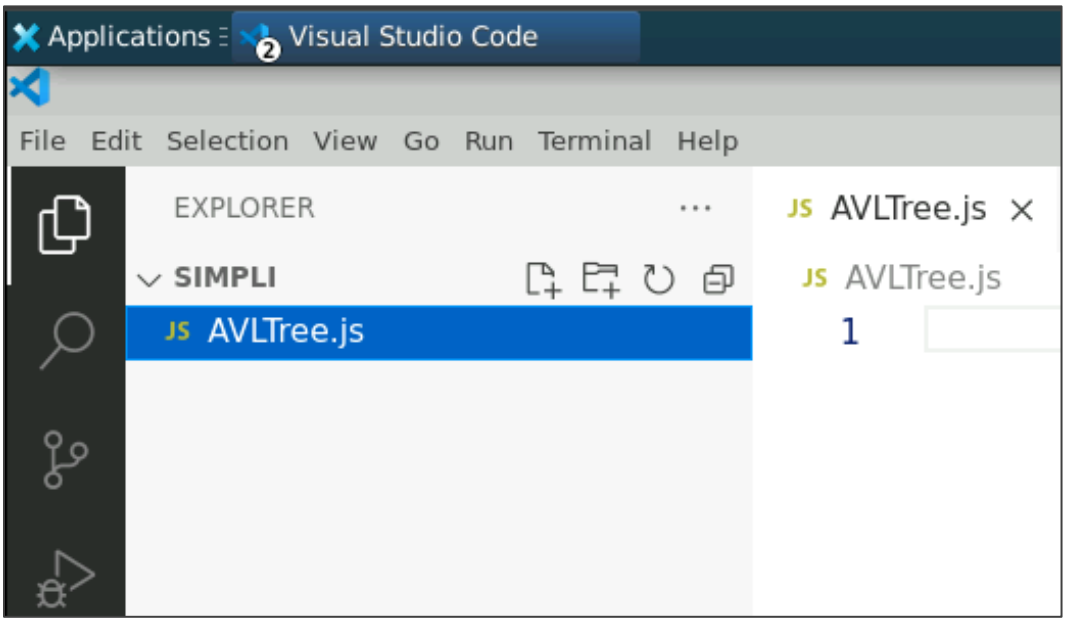
Prerequisites: A basic understanding of an AVL tree and proficiency in JavaScript

Steps to be followed:

1. Create a JavaScript file and execute it

Step 1: Create a JavaScript file and execute it

1.1 Open the Visual Studio Code editor and create a JavaScript file named **AVLTree.js**



1.2 Add the following code to the file:

```
// AVL tree node definition
class AVLNode {
  constructor(data) {
    this.data = data;
    this.left = null;
    this.right = null;
    this.height = 1;
  }
}

// AVL tree implementation
class AVLTree {
  constructor() {
    this.root = null;
  }

  // Function to get the height of a node
  getHeight(node) {
    return node ? node.height : 0;
  }

  // Function to update the height of a node
  updateHeight(node) {
    if (node) {
      node.height = Math.max(this.getHeight(node.left), this.getHeight(node.right)) +
1;
    }
  }

  // Function to perform right rotation
  rotateRight(y) {
    const x = y.left;
    const T2 = x.right;

    x.right = y;
    y.left = T2;

    this.updateHeight(y);
    this.updateHeight(x);
    return x;
  }
```

```
// Function to perform left rotation
rotateLeft(x) {
   const y = x.right;
   const T2 = y.left;

   y.left = x;
   x.right = T2;

   this.updateHeight(x);
   this.updateHeight(y);

   return y;
}

// Function to get the balance factor of a node
getBalanceFactor(node) {
   return node ? this.getHeight(node.left) - this.getHeight(node.right) : 0;
}

// Function to insert a node into the AVL tree
insert(data) {
   this.root = this._insert(this.root, data);
}

_insert(node, data) {
   // Perform standard BST insert
   if (!node) {
      return new AVLNode(data);
   }

   if (data < node.data) {
      node.left = this._insert(node.left, data);
   } else if (data > node.data) {
      node.right = this._insert(node.right, data);
   } else {
      return node; // Duplicate nodes are not allowed
   }

   // Update the height of the current node
   this.updateHeight(node);

   // Get the balance factor to check if the node became unbalanced
   const balance = this.getBalanceFactor(node);
```

```
        // Left Left Case
        if (balance > 1 && data < node.left.data) {
            return this.rotateRight(node);
        }

        // Right Right Case
        if (balance < -1 && data > node.right.data) {
            return this.rotateLeft(node);
        }

        // Left Right Case
        if (balance > 1 && data > node.left.data) {
            node.left = this.rotateLeft(node.left);
            return this.rotateRight(node);
        }

        // Right Left Case
        if (balance < -1 && data < node.right.data) {
            node.right = this.rotateRight(node.right);
            return this.rotateLeft(node);
        }

        return node;
    }
}

// Example usage
const avlTree = new AVLTree();
avlTree.insert(10);
avlTree.insert(5);
avlTree.insert(15);
avlTree.insert(3);
avlTree.insert(8);

console.log('AVL Tree root:', avlTree.root.data);
```

JS AVLTree.js > ...

```
1   // AVL tree node definition
2   class AVLNode {
3       constructor(data) {
4           this: this data;
5           this.left = null;
6           this.right = null;
7           this.height = 1;
8       }
9   }
10
11  // AVL tree implementation
12  class AVLTree {
13      constructor() {
14          this.root = null;
15      }
16
17      // Function to get the height of a node
18      getHeight(node) {
19          return node ? node.height : 0;
20      }
21
```

```
22      // Function to update the height of a node
23      updateHeight(node) {
24          if (node) {
25              node.height = Math.max(this.getHeight(node.left), this.getHeight(node.right)) + 1;
26          }
27      }
28
29      // Function to perform right rotation
30      rotateRight(y) {
31          const x = y.left;
32          const T2 = x.right;
33
34          x.right = y;
35          y.left = T2;
36
37          this.updateHeight(y);
38          this.updateHeight(x);
39
40          return x;
41      }
42
```

```
43      // Function to perform left rotation
44      rotateLeft(x) {
45          const y = x.right;
46          const T2 = y.left;
47
48          y.left = x;
49          x.right = T2;
50
51          this.updateHeight(x);
52          this.updateHeight(y);
53
54          return y;
55      }
56
57      // Function to get the balance factor of a node
58      getBalanceFactor(node) {
59          return node ? this.getHeight(node.left) - this.getHeight(node.right) : 0;
60      }
```

```
62      // Function to insert a node into the AVL tree
63      insert(data) {
64          this.root = this._insert(this.root, data);
65      }
66
67      _insert(node, data) {
68          // Perform standard BST insert
69          if (!node) {
70              return new AVLNode(data);
71          }
72
73          if (data < node.data) {
74              node.left = this._insert(node.left, data);
75          } else if (data > node.data) {
76              node.right = this._insert(node.right, data);
77          } else {
78              return node; // Duplicate nodes are not allowed
79          }
80
81          // Update height of the current node
82          this.updateHeight(node);
83
84          // Get the balance factor to check if the node became unbalanced
85          const balance = this.getBalanceFactor(node);
86
```

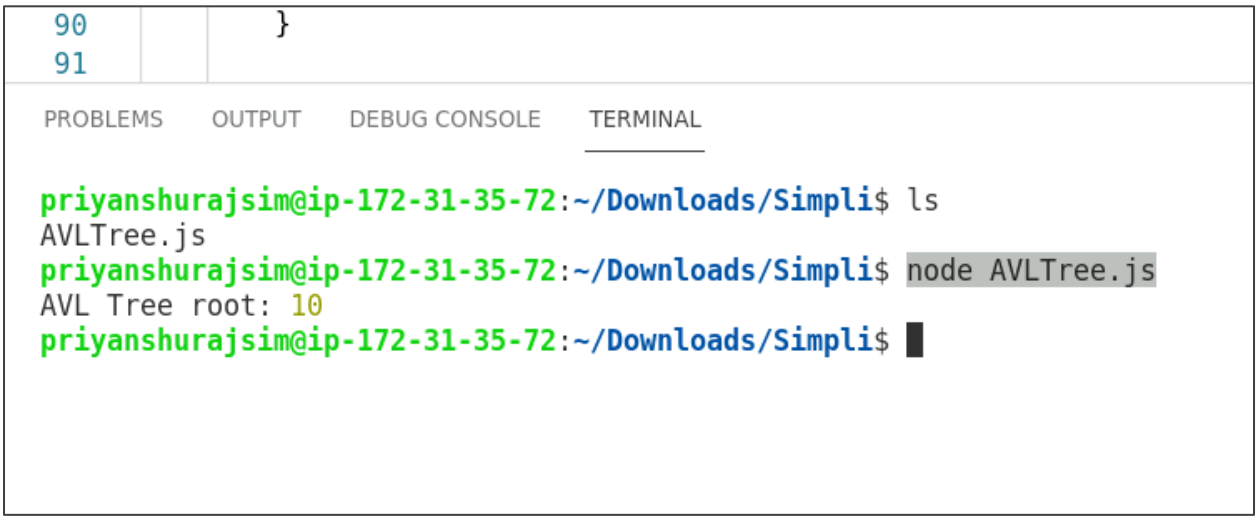
```
87          // Left Left Case
88          if (balance > 1 && data < node.left.data) {
89              return this.rotateRight(node);
90          }
91
92          // Right Right Case
93          if (balance < -1 && data > node.right.data) {
94              return this.rotateLeft(node);
95          }
96
97          // Left Right Case
98          if (balance > 1 && data > node.left.data) {
99              node.left = this.rotateLeft(node.left);
100             return this.rotateRight(node);
101         }
102
103         // Right Left Case
104         if (balance < -1 && data < node.right.data) {
105             node.right = this.rotateRight(node.right);
106             return this.rotateLeft(node);
107         }
108
109         return node;
110     }
111 }
112
```

```
113 // Example usage
114 const avlTree = new AVLTree();
115 avlTree.insert(10);
116 avlTree.insert(5);
117 avlTree.insert(15);
118 avlTree.insert(3);
119 avlTree.insert(8);
120
121 console.log('AVL Tree root:', avlTree.root.data);
122
```

1.3 Press **Ctrl** + **S** to save the file and execute it in the **TERMINAL** using the commands given below:

ls

node AVLTree.js



By following these steps, you have successfully implemented an AVL tree in JavaScript to maintain height-balanced binary search tree properties.