```java
package binaryTree;

/* 实现二叉搜索树的1.数据插入
2.前序遍历
3.中序遍历
4.后序遍历
5.层次遍历
6.根据前序遍历和中序遍历求后序遍历
7.求二叉树中节点的最大距离*/

import java.util.HashMap;
import java.util.LinkedList;
import java.util.Queue;

class ImplementBinarySearchTree {

    class TreeNode {
        TreeNode leftNode;
        TreeNode rightNode;
        int val;

        TreeNode(int val) {
            this.val = val;
            this.leftNode = null;
            this.rightNode = null;
        }
    }

    private TreeNode root;

    ImplementBinarySearchTree() {
        root = null;
    }

    //插入数据
    void insertData(int val) {

        if (this.root == null) {
            this.root = new TreeNode(val);
        } else {
            insertDataHelper(val, this.root);
        }
    }

    private void insertDataHelper(int val, TreeNode root) {

        if (root.val == val) {
            return;
        } else if (root.val < val) {
            if (root.rightNode == null) {
                root.rightNode = new TreeNode(val);
                return;
            } else {
                insertDataHelper(val, root.rightNode);
            }
        } else {
            if (root.leftNode == null) {
                root.leftNode = new TreeNode(val);
                return;
            } else {
                insertDataHelper(val, root.leftNode);
            }
        }
    }


    //中序遍历（中根遍历）
    void middleOrderTraverse() {

        middleOrderTraverseHelper(this.root);
    }

    private void middleOrderTraverseHelper(TreeNode node) {
```

```
74
75          if (node == null) {
76              return;
77          }
78
79          middleOrderTraverseHelper(node.leftNode);
80          System.out.println(node.val);
81          middleOrderTraverseHelper(node.rightNode);
82      }
83
84      //前序遍历（先根遍历）
85      void preOrderTraverse() {
86
87          preOrderTraverseHelper(this.root);
88      }
89
90      private void preOrderTraverseHelper(TreeNode node) {
91
92          if (node == null) {
93              return;
94          }
95
96          System.out.println(node.val);
97          preOrderTraverseHelper(node.leftNode);
98          preOrderTraverseHelper(node.rightNode);
99      }
100
101
102     //后序遍历（后根遍历）
103     void postOrderTraverse() {
104
105         postOrderTraverseHelper(this.root);
106     }
107
108     private void postOrderTraverseHelper(TreeNode node) {
109
110         if (node == null) {
111             return;
112         }
113
114         postOrderTraverseHelper(node.leftNode);
115         postOrderTraverseHelper(node.rightNode);
116         System.out.println(node.val);
117     }
118
119
120     //层次遍历
121     void levelTraverse() {
122
123         if (root == null) {
124             return;
125         }
126
127         Queue<TreeNode> queue = new LinkedList<TreeNode>();
128         queue.offer(root);
129
130         while (!queue.isEmpty()) {
131
132             int size = queue.size();
133             for (int i = 0; i < size; i++) {
134                 TreeNode currentNode = queue.poll();
135                 System.out.println(currentNode.val);
136                 if (currentNode.leftNode != null) {
137                     queue.offer(currentNode.leftNode);
138                 }
139                 if (currentNode.rightNode != null) {
140                     queue.offer(currentNode.rightNode);
141                 }
142             }
143         }
144     }
145
146
```

```java
    void getPostTraverseAccordingToPreTraverseAndMiddleTraverse(int[] preOrderTraverse, int[] middleOrderTraverse) {

        if (preOrderTraverse == null || middleOrderTraverse == null
                || preOrderTraverse.length == 0 || middleOrderTraverse.length == 0
                || preOrderTraverse.length != middleOrderTraverse.length) {
            return;
        }

        TreeNode psudoRoot = new TreeNode(0);
        psudoRoot.leftNode = getPostTraverseAccordingToPreTraverseAndMiddleTraverseHelper(
                preOrderTraverse, 0, preOrderTraverse.length - 1,
                middleOrderTraverse, 0, middleOrderTraverse.length - 1);

        root = psudoRoot.leftNode;

        //this.postOrderTraverse();
    }

    private TreeNode getPostTraverseAccordingToPreTraverseAndMiddleTraverseHelper(
            int[] preOrderTraverse, int preStartIndex, int preEndIndex,
            int[] middleOrderTraverse, int middleStartIndex, int middleEndIndex) {

        TreeNode node = null;

        if (preStartIndex < preEndIndex) {

            node = new TreeNode(preOrderTraverse[preStartIndex]);

            int rootIndex = findRootIndex(middleOrderTraverse,
            preOrderTraverse[preStartIndex], middleStartIndex, middleEndIndex);

            node.leftNode =
            getPostTraverseAccordingToPreTraverseAndMiddleTraverseHelper(
                    preOrderTraverse, preStartIndex + 1, preStartIndex + rootIndex - middleStartIndex,
                    middleOrderTraverse, middleStartIndex, rootIndex - 1);
            node.rightNode =
            getPostTraverseAccordingToPreTraverseAndMiddleTraverseHelper(
                    preOrderTraverse, preStartIndex + rootIndex - middleStartIndex + 1, preEndIndex,
                    middleOrderTraverse, rootIndex + 1, middleEndIndex);
        } else if (preStartIndex == preEndIndex) {
            node = new TreeNode(preOrderTraverse[preStartIndex]);
        }

        if (node != null) {
            System.out.println(node.val);
        }
        return node;
    }


    private int findRootIndex(int[] middleOrderTraverse, int rootVal,
                              int startIndex, int endIndex) {

        for (int i = startIndex; i <= endIndex; i++) {

            if (middleOrderTraverse[i] == rootVal) {
                return i;
            }
        }

        return -1;
    }


    private int maxDistance = 0;
    private HashMap<TreeNode, Integer> leftMaxDistance = new HashMap<>();
    private HashMap<TreeNode, Integer> rightMaxDistance = new HashMap<>();

    int findMaxDistance() {
```

```
213
214             if (root == null) {
215                 return 0;
216             }
217
218             findMaxDistanceHelper(root);
219
220             return maxDistance;
221         }
222
223         private void findMaxDistanceHelper(TreeNode root) {
224
225             if (root.leftNode == null) {
226                 leftMaxDistance.put(root, 0);
227             } else {
228                 findMaxDistanceHelper(root.leftNode);
229             }
230
231             if (root.rightNode == null) {
232                 rightMaxDistance.put(root, 0);
233             } else {
234                 findMaxDistanceHelper(root.rightNode);
235             }
236
237             if (root.leftNode != null) {
238                 leftMaxDistance.put(root, 1 +
                    Math.max(leftMaxDistance.get(root.leftNode),
                    rightMaxDistance.get(root.leftNode)));
239             }
240             if (root.rightNode != null) {
241                 rightMaxDistance.put(root, 1 +
                    Math.max(leftMaxDistance.get(root.rightNode),
                    rightMaxDistance.get(root.rightNode)));
242             }
243
244             maxDistance = Math.max(leftMaxDistance.get(root) +
                rightMaxDistance.get(root), maxDistance);
245         }
246
247
248         // Below is for testing
249         public static void main(String[] args) {
250
251             ImplementBinarySearchTree bst = new ImplementBinarySearchTree();
252
253             int[] input = {9, 5, 6, 8, 4, 6, 2, 1};
254
255             for (int number : input) {
256                 bst.insertData(number);
257             }
258
259             System.out.println("前序遍历");
260             bst.preOrderTraverse();
261             System.out.println("中序遍历");
262             bst.middleOrderTraverse();
263             System.out.println("后序遍历");
264             bst.postOrderTraverse();
265             System.out.println("层次遍历");
266             bst.levelTraverse();
267
268             int maxDistance = bst.findMaxDistance();
269
270             ImplementBinarySearchTree bst2 = new ImplementBinarySearchTree();
271             int[] preOrder = {9, 5, 4, 2, 1, 6, 8};
272             int[] middleOrder = {1, 2, 4, 5, 6, 8, 9};
273
274             System.out.println("后序遍历");
275             bst2.getPostTraverseAccordingToPreTraverseAndMiddleTraverse(preOrder,
                middleOrder);
276
277             System.out.println("haha");
278         }
279     }
```