Traversal in level

Tuesday, February 21, 2017 9:37 PM

```
8  public class TraverseInLevelBST
9  {
10
11     public static void main (String[] args)
12     {
13         BSTNode d2x6right = new BSTNode(6, null, null);
14         BSTNode d2x5left = new BSTNode(5, null, null);
15         BSTNode d2x4right = new BSTNode(4, null, null);
16         BSTNode d2x3left = new BSTNode(3, null, null);
17         BSTNode d1x2right = new BSTNode(2, d2x5left, d2x6right);
18         BSTNode d1x1left = new BSTNode(1, d2x3left, d2x4right);
19         BSTNode root = new BSTNode(0, d1x1left, d1x2right);
20
21         List<List<BSTNode>> result = new ArrayList<>();
22         traverse(root, 1, result);
23         System.out.println(result);
24
25     }
26
27     private static void
28  traverse (BSTNode root, int level, List<List<BSTNode>> result)
29     {
30         if (root == null)           //conquer occurs when node is null
31             return;
32         if (level > result.size())
33             result.add(new ArrayList<BSTNode>());  //construct a new List for a new level.
34         result.get(level - 1).add(root);           //add node to the (level - 1)'s List as index start with 0.
35         traverse(root.getLeftNode(), level + 1, result);  //recursion on left node.
36         traverse(root.getRightNode(), level + 1, result); //recursion on right node.
37
38     }
39
40 }
41
```

Screen clipping taken: 2/21/2017 9:44 PM

Detect a subtree

Saturday, February 18, 2017 5:52 PM

```
public class DetectSubTree
{

    public static void main (String[] args)
    {

        BSTNode d2x6right = new BSTNode(6, null, null);
        BSTNode d2x5left = new BSTNode(5, null, null);
        BSTNode d2x4right = new BSTNode(4, null, null);
        BSTNode d2x3left = new BSTNode(3, null, null);
        BSTNode d1x2right = new BSTNode(2, d2x5left, d2x6right);
        BSTNode d1x1left = new BSTNode(1, d2x3left, d2x4right);
        BSTNode root = new BSTNode(0, d1x1left, d1x2right);

        BSTNode s1x3left = new BSTNode(3, null, null);
        BSTNode s1x2right = new BSTNode(2, null, null);
        BSTNode s1x1left = new BSTNode(1, s1x3left, null);
        BSTNode sample = new BSTNode(0, s1x1left, s1x2right);

        System.out.println(detectRootNode(root, sample));

    }

    private static boolean detectRootNode (BSTNode root, BSTNode sample)
    {
        boolean result = false;
        if (root == null || sample == null)
            return false;

        //result = tree1.equals(sample);
        if (root.equals(sample))                    //if root is equals, then check its subtree
            result = detectSubTree(root, sample);

        if (!result)
            result = detectRootNode(root.getLeftNode(), sample); //if the root or its subtree is not equals,
        if (!result)                                             //then have the node moved its left
            result = detectRootNode(root.getRightNode(), sample);//and even right node.
        return result;

    }

    private static boolean detectSubTree (BSTNode root, BSTNode sample)
    {
        if (sample == null) //defense programming
            return true;    //and the occurrence of recursion's conquer

        if (root == null)   //false if sample have value but root is null.
            return false;

        return root.equals(sample) && detectSubTree(root.getLeftNode(), sample.getLeftNode()) //check current node equality and then its left and right
            && detectSubTree(root.getRightNode(), sample.getRightNode());                     //node's equality
    }

```

Screen clipping taken: 2/18/2017 6:22 PM

Zigzag traversal

Saturday, February 25, 2017 10:41 AM

Given a binary tree, return the zigzag level order traversal of its nodes' values. (ie, from left to right, then right to left for the next level and alternate between).
For example: Given binary tree 3,9,20,#,#,15,7,
3
/ \
9 20
/ \
15 7
return its zigzag level order traversal as:
[
[3],
[20,9],
[15,7]
]

```
7
8  public class ZigzagTraversal
9  {
10
11     public static void main (String[] args)
12     {
13         List<List<BSTNode>> result = new ArrayList<>();
14
15         BSTNode d2x6right = new BSTNode(6, null, null);
16         BSTNode d2x5left = new BSTNode(5, null, null);
17         BSTNode d2x4right = new BSTNode(4, null, null);
18         BSTNode d2x3left = new BSTNode(3, null, null);
19         BSTNode d1x2right = new BSTNode(2, d2x5left, d2x6right);
20         BSTNode d1x1left = new BSTNode(1, d2x3left, d2x4right);
21         BSTNode root = new BSTNode(0, d1x1left, d1x2right);
22
23         traverse(root, 1, result, true);
24         System.out.println(result);
25
26     }
27
28     private static void traverse (
29         BSTNode root,
30         int level,
31         List<List<BSTNode>> result,
32         boolean leftToRight)                    //a flag to specify the direction.
33     {
34         if (root == null)
35             return;
36
37         if (level > result.size())
38             result.add(new ArrayList<BSTNode>());
39
40         if (leftToRight)
41             result.get(level - 1).add(root);
42         else
43             result.get(level - 1).add(0, root);    //insert the node at the head position.
44
45         if (root.getLeftNode() != null)
46             traverse(root.getLeftNode(), level + 1, result, !leftToRight);  //flip the boolean flag.
47         if (root.getRightNode() != null)
48             traverse(root.getRightNode(), level + 1, result, !leftToRight); //flip the boolean flag.
49     }
50
51 }
52
```

Screen clipping taken: 2/25/2017 10:44 AM

Depth Balanced Height Tree

Tuesday, February 28, 2017 9:09 PM

```
5  public class BalancedHeightBST
6  {
7
8      public static void main (String[] args)
9      {
10         BSTNode d2x6right = new BSTNode(6, null, null);
11         BSTNode d2x5left = new BSTNode(5, null, null);
12         BSTNode d2x4right = new BSTNode(4, null, null);
13         BSTNode d2x3left = new BSTNode(3, null, null);
14         BSTNode d1x2right = new BSTNode(2, d2x5left, d2x6right);
15         BSTNode d1x1left = new BSTNode(1, d2x3left, d2x4right);
16         BSTNode root = new BSTNode(0, d1x1left, null);
17
18         System.out.println(balanced(root));
19
20     }
21     // -1 is not a balanced tree
22     private static int balanced (BSTNode root)
23     {
24         if (root == null)     return 0;   //conquer when no children
25
26         int left = balanced(root.getLeftNode());       //recursion on both nodes.
27         int right = balanced(root.getRightNode());
28
29         if (left < 0 | right < 0 | Math.abs(left - right) > 1)  //depth differ of left and right
30             return -1;                                          //more than 1 will be non-balanced.
31         return Math.max(left, right) + 1;
32
33     }
34
35 }
36
```

Screen clipping taken: 2/28/2017 9:10 PM

5.4.5 Binary Tree Maximum Path Sum

Tuesday, March 21, 2017 10:23 PM

Given a binary tree, find the maximum path sum.
The path may start and end at any node in the tree. For example: Given the below binary tree,
1
/ \
2 3
Return 6.

```
5  public class MaximumPathNum
6  {
7
8      public static int result = Integer.MIN_VALUE;
9      public static void main (String[] args)
10     {
11         BSTNode d2x6right = new BSTNode(4, null, null);
12         BSTNode d2x5left = new BSTNode(5, null, null);
13         BSTNode d2x4right = new BSTNode(4, null, null);
14         BSTNode d2x3left = new BSTNode(1, null, null);
15         BSTNode d1x2right = new BSTNode(-9, d2x5left, d2x6right);
16         BSTNode d1x1left = new BSTNode(-2, d2x3left, d2x4right);
17         BSTNode root = new BSTNode(-3, d1x1left, d1x2right);
18         System.out.println(dfs(root));
19         System.out.println(result);
20
21     }
22
23     private static int dfs (BSTNode root)
24     {
25         if(root == null)
26             return 0; //return 0 when conquer occurs.
27
28         int l = dfs(root.getLeftNode()); //recursion on left node.
29         int r = dfs(root.getRightNode()); //recursion on right node.
30
31         int inter = root.getValue();
32
33         if (l > 0) //only add up if left > 0
34             inter += l;
35         if (r > 0) //only add up if right > 0
36             inter += r;
37         result = Math.max(result, inter);//global result add up only when intervals greater than 0.
38
39         return Math.max(l, r) > 0 ? Math.max(l, r) + root.getValue() : root.getValue(); //return from either r+root or l+root
40                                                                                         //when either l or r greater than 0.
41     }
42
43 }
44
```

Screen clipping taken: 3/21/2017 10:28 PM

Screen clipping taken: 3/21/2017 10:23 PM

5.2.1 Construct Binary Tree from Preorder and Inorder Traversal

Sunday, March 26, 2017 12:30 PM

```
 9
10    public static void main (String[] args)
11    {
12        BSTNode[] priorOrder = new BSTNode[] {
13            new BSTNode(1, null, null),
14            new BSTNode(
15            2,
16            null,
17 null),
18            new BSTNode(4, null, null),
19            new BSTNode(5, null, null),
20            new BSTNode(3, null, null)//,
21 //             new BSTNode(6, null, null),
22        //             new BSTNode(7, null, null)
23        };
24        BSTNode[] inOrder = new BSTNode[] {
25            new BSTNode(4, null, null),
26            new BSTNode(2, null, null),
27            new BSTNode(5, null, null),
28            new BSTNode(1, null, null),
29 //             new BSTNode(6, null, null),
30            new BSTNode(3, null, null)//,
31 //             new BSTNode(7, null, null)
32            };
33        BSTNode root = calculate(priorOrder, inOrder);
34        BSTNode.iteratePriorBST(root, true);
35
36    }
37
38    private static BSTNode calculate (BSTNode[] prior, BSTNode[] in)
39    {
40        if (prior.length == 0 || in.length == 0)   //the case in which the prior's root is the last of in.
41            return null;
42
43        if (prior.length == 1 && in.length == 1 && prior[0].equals(in[0])) //conquer occurs once array's length is 1
44            return prior[0];
45
46        BSTNode root = prior[0];
47        int distance = 0;
48        for (int i = 0; i < in.length; ++i)
49        {
50            if (root.equals(in[i]))
51                break;
52            ++distance;                          //calculate the distance of root out of in-order traversal.
53
54        }
55
56        root.setLeftNode(calculate(
57            Arrays.copyOfRange(prior, 1, distance + 1),  //calculate left tree on left of prior and in
58            Arrays.copyOfRange(in, 0, distance)));
59        root.setRightNode(calculate(
60            Arrays.copyOfRange(prior, distance + 1, prior.length), //calculate right tree on right of prior and in
61            Arrays.copyOfRange(in, distance + 1, in.length)));
62        return root;
63    }
64 }
65
```

Screen clipping taken: 3/26/2017 12:35 PM