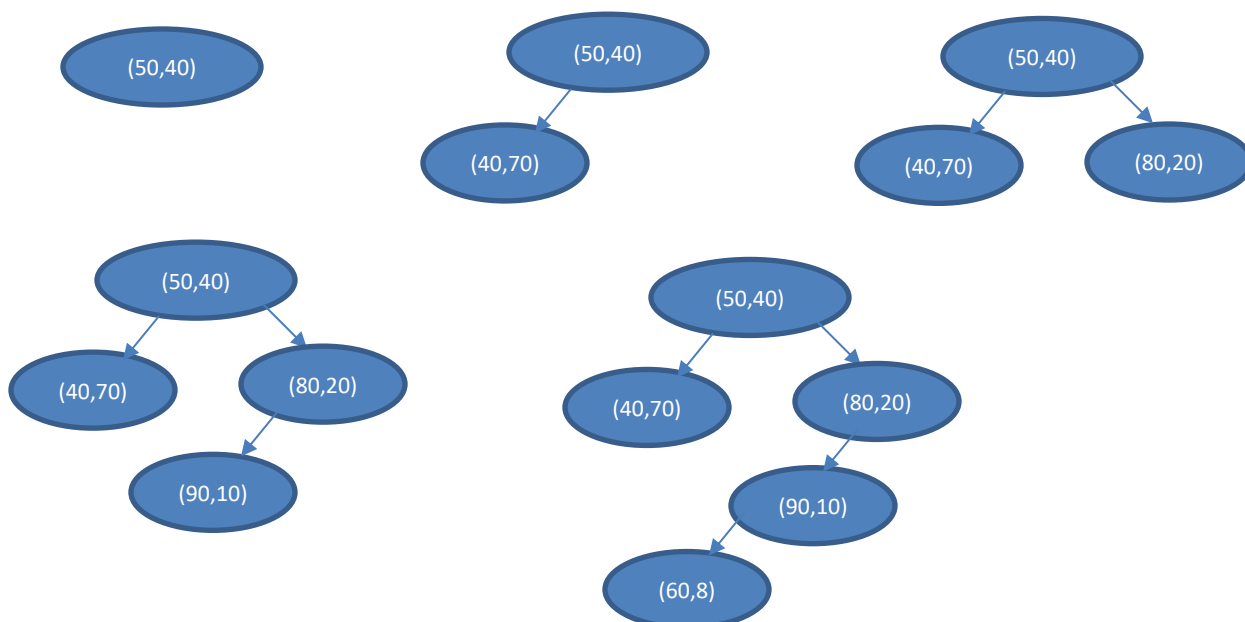


Assignment 8 (2D Binary Search Tree)

A 2D tree is a binary search tree in which each node contains a 2D point and holds the x and y coordinates of that node. 2D tree generalizes a binary search tree in that it positions each node according to either the x or y coordinates of its data point. The coordinate choice depends on the level at which the node is added into the tree. The first point you add into an empty tree is placed into a node that becomes the tree's root. If the next point to be added has an x-coordinate that is less than the x-coordinate of the point in the root, you place the new point into the left child of the root. Otherwise, you place it into root's right child. Insertions at the next level—level 3—compares y-coordinates; insertion at level 4 compares x-coordinates, and so on.

For example, suppose we want to add the points (50,40),(40,70),(80,20),(90,10), and (60,30) into an initially empty 2d tree (the process is illustrated below)

- The first point becomes the root node
- To add (40,70), you compare 40, the point's x-coordinate, with 50, the x-coordinate of the root. Since 40 is less than 50, you move to the left. The left child of root is null so the new point goes into the left child of the root.
- To add (80,20), you compare 80 with 50, the x-coordinates of the root node. Since 80 is greater than 50, you move to the right. The right child of the root is null, so new node (80,20) is placed in to the right child of the root.
- To add the next point, (90,10), you compare 90 with the x-coordinate of the root node. 90 is greater than 50, so you move to the right child and compare the y-coordinate with the right child. 10 is less than 20, the y-coordinate of the root's right child. Hence, you move to the left. The left child of (80,20) is null, hence, the new node (90,10) is added as the left child of (80,20)
- To add the next point (60,8), you compare 60 with x-coordinate of the root. 60 is greater than 50, so you move to the right. Then you compare the y coordinates of (60,8) with y-coordinate of (80,20). $8 < 20$ so you move to the left. Then you compare the x coordinate of (60,8) with the x coordinate of (90,10); $60 < 90$ so you move to the left. The left child of (90,10) is null, so you place (60,8) to the left of (90,10)..



In summary, to insert a new node into a 2d-tree you keep searching in the binary tree switching between x and y coordinates at each level until you hit a null node and you insert the new node there.

The file TwoDTree.java attached to this assignment provides a skeleton for implementation of a 2-d tree. This file contains a nested class TwoDTreeNode which represents a node in the TwoDTree and holds the x and y coordinates for the node as well as references to its left and right children.

Your job is to implement two methods in this file:

1. **Public void add(int x, int y)** : This method adds a new node with coordinates (x,y) to the 2d tree. Duplicate nodes are not allowed in the tree.
2. **public boolean contains(int x, int y)**: This method returns true if a node with given x and y coordinates exist in the TwoDTree.

I have also provided a helper method levelOrderPrint to print the level-order traversal of the tree. This method helps you test your add method and view a level order traversal of the tree after adding nodes to it. You can use any additional helper method you want.

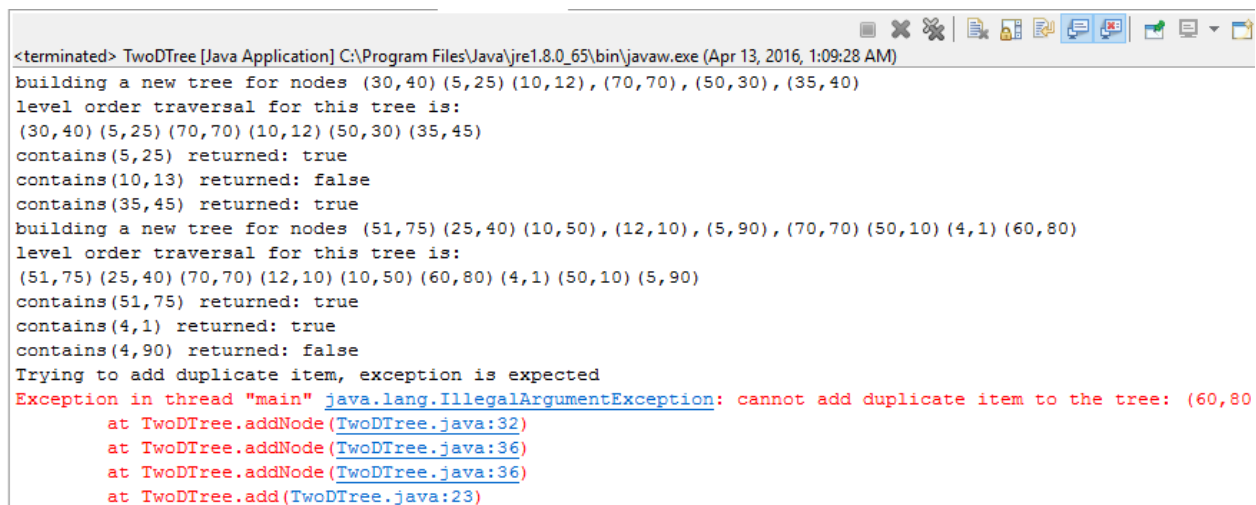
Hints:

- For implementing the add method, you can use the add(T obj) method in the GenericBinarySearchTree .java file (provided under “source code” section on blackboard) as an example and modify it to fit the specification of this problem. For example, to find the insertion point for a new node, you start from the root node and go down the tree. At each point you decide to go left or right based on comparing the current node with x or y coordinate. The choice of which coordinate to use depends on the level in which you are performing the comparison. At root level (level 0) the x coordinate is used for comparison; at next level (level 1) the y coordinate is used for comparison, at level 2, the x coordinate is used for comparison, and so on.
- Once you implement the add method, the contain method should be easy to implement. You simply follow the same process. To find a given node, you start at the root node and go down the tree to find the node. At each point you compare the x or y coordinate with the current node and decide to go left or right. The choice of coordinate depends on the level on which you are doing the comparison.
- Please make sure to test your method for various test cases. I made up a small sample main method. The main method and the result of its run is shown below:

```

public static void main(String[] args){
    System.out.println("building a new tree for nodes (30,40) (5,25) (10,12), (70,70), (50,30), (35,40)");
    TwoDTree tDTree = new TwoDTree();
    tDTree.add(30,40);
    tDTree.add(5,25);
    tDTree.add(10,12);
    tDTree.add(70,70);
    tDTree.add(50,30);
    tDTree.add(35,45);
    System.out.println("level order traversal for this tree is:");
    tDTree.levelOrderPrint();
    System.out.println("contains(5,25) returned: " + tDTree.contains(5,25) );
    System.out.println("contains(10,13) returned: " +tDTree.contains(10,13) );
    System.out.println("contains(35,45) returned: " +tDTree.contains(35,45) );
    System.out.println("building a new tree for nodes (51,75) (25,40) (10,50), (12,10), (5,90), (70,70) (50,10) (4,1) (60,80)");
    tDTree = new TwoDTree();
    tDTree.add(51,75);
    tDTree.add(25,40);
    tDTree.add(10,50);
    tDTree.add(12,10);
    tDTree.add(5,90);
    tDTree.add(70,70);
    tDTree.add(50,10);
    tDTree.add(4,1);
    tDTree.add(60,80);
    System.out.println("level order traversal for this tree is:");
    tDTree.levelOrderPrint();
    System.out.println("contains(51,75) returned: " + tDTree.contains(51,75) );
    System.out.println("contains(4,1) returned: " +tDTree.contains(4,1) );
    System.out.println("contains(4,90) returned: " +tDTree.contains(4,90) );
    System.out.println("Trying to add duplicate item, exception is expected");
    tDTree.add(60,80);}

```



```

<terminated> TwoDTree [Java Application] C:\Program Files\Java\jre1.8.0_65\bin\javaw.exe (Apr 13, 2016, 1:09:28 AM)
building a new tree for nodes (30,40) (5,25) (10,12), (70,70), (50,30), (35,40)
level order traversal for this tree is:
(30,40) (5,25) (70,70) (10,12) (50,30) (35,45)
contains(5,25) returned: true
contains(10,13) returned: false
contains(35,45) returned: true
building a new tree for nodes (51,75) (25,40) (10,50), (12,10), (5,90), (70,70) (50,10) (4,1) (60,80)
level order traversal for this tree is:
(51,75) (25,40) (70,70) (12,10) (10,50) (60,80) (4,1) (50,10) (5,90)
contains(51,75) returned: true
contains(4,1) returned: true
contains(4,90) returned: false
Trying to add duplicate item, exception is expected
Exception in thread "main" java.lang.IllegalArgumentException: cannot add duplicate item to the tree: (60,80)
    at TwoDTree.addNode(TwoDTree.java:32)
    at TwoDTree.addNode(TwoDTree.java:36)
    at TwoDTree.addNode(TwoDTree.java:36)
    at TwoDTree.add(TwoDTree.java:23)

```

What you need to turn in:

1. The modified **TwoDTree.java** class including the above two methods.
2. A document containing the Analysis of the time efficiency of your methods in two cases: case1- The 2-d binary search tree is balanced and case 2- the 2-d binary search tree is not balanced.

Grading Rubric

Add method works correctly for all test cases	8.5
Contains method works correctly for all test cases	8.5
Analysis of Time efficiency of your methods when the tree is balanced and when the tree is imbalanced. You should explain what is the average and worst case running time in each case.	3
Total	20