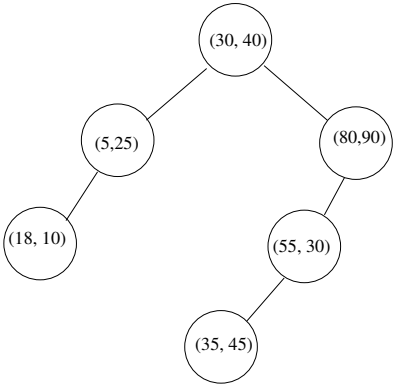
This assignment generalizes binary search trees to allow searching on 2-dimensional keys, such as Points in 2D. Such a tree is called a kd-tree for k-dimensional tree. In this assignment $k = 2$. A regular Binary Search Tree can only insert the points searching using either the x or y coordinate, but not both. The idea in a kd-tree is that, at each level, the tree compares against one dimension (called cutting dimension). The cutting dimension cycles through dimensions as you walk down the tree. For example, a 2D kd-tree first compares on the x coordinate to decide whether to go to the left or right of a node, then at the next level compares on the y coordinate. Then it's back to x again. Here is an example of some points (given in the order they are inserted into the tree) and the kd-tree associated with it. (30,40) is inserted as the root. When (5,25) is being inserted, 5 is compared with root's x coordinate, since $5 < 30$, (5,25) is inserted as the left child of the root. When (18,10) is being inserted, 18 is compared with root's x coordinate, $18 < 30$ so, we continue to the left subtree, at this level we compare 10 with (5,25)'s y coordinate, $10 < 25$, so we continue to the left subtree and insert (18,10) as the left child of (5,25). Your tree will not allow insertion of duplicate points.

(30,40) (5,25) (18,10) (80,90) (55,30) (35,45)



You will implement a KDTree2D class which supports the following methods. You will also need a Point2D class which stores x and y coordinates of a 2D point. The coordinates can be float or double.

- Constructor(s).
- public boolean insert(Point2D point): Inserts the given point into the tree if it does not already exist in the tree. Returns true if insertion is successful, returns false otherwise.

- public Point2D search(Point2D point): Searches for the given point in the tree. Returns the point if found, returns null otherwise.
- public boolean remove(Point2D point): Removes the given point from the tree if it exists in the tree and returns true. If the point does not exist in the tree returns false.
- public String toString() : Returns a String representation of the tree by doing a preorder traversal of the tree. Each node's data should appear on a separate line and be indented by dots based on its level.
- Point2D findMin(int d): Finds the point with the smallest value in the d-th dimension. Let d=0 represent the x dimension, d=1 represent the y dimension. Write this method recursively and also efficiently, in other words, do not recurse on subtrees which cannot contain the minimum. Methods checking all points in the tree will NOT get full credit for this part.
- Point2D findMax(int d): Finds the point with the largest value in the d-th dimension. Write this method recursively and also efficiently, in other words, do not recurse on subtrees which cannot contain the maximum. Methods checking all points in the tree will NOT get full credit for this part.
- Iterable <Point2D> inRectangle(Point2D ll, Point2D ur): Returns an iterable collection of points that lie in a rectangle whose lower left corner is given by ll and upper right corner is given by ur. Write this method recursively and also efficiently, in other words, do not recurse on subtrees that cannot possibly contain a point in the given rectangle. Methods checking all points in the tree for containment in the query rectangle will NOT get full credit for this part.
- Point2D nearestNeighbor(Point2D p): Returns the point in the tree that is closest to p. Write this method recursively and also efficiently, in other words, do not recurse on subtrees that cannot possibly update the current closest point. For this part research how to do the nearest neighbor searching in a kd-tree. Methods checking distance of p to all points in the tree will NOT get full credit for this part.

Write a driver program that tests your class. Do not change the signatures of the methods.

Provide a README file with your submission clearly explaining your algorithm for each method.

START EARLY!