ADVANCE ALGORITHM
Experiment 6

Ayush Jain 60004200132 B3

Aim : To implement KD Tree.

Theory:

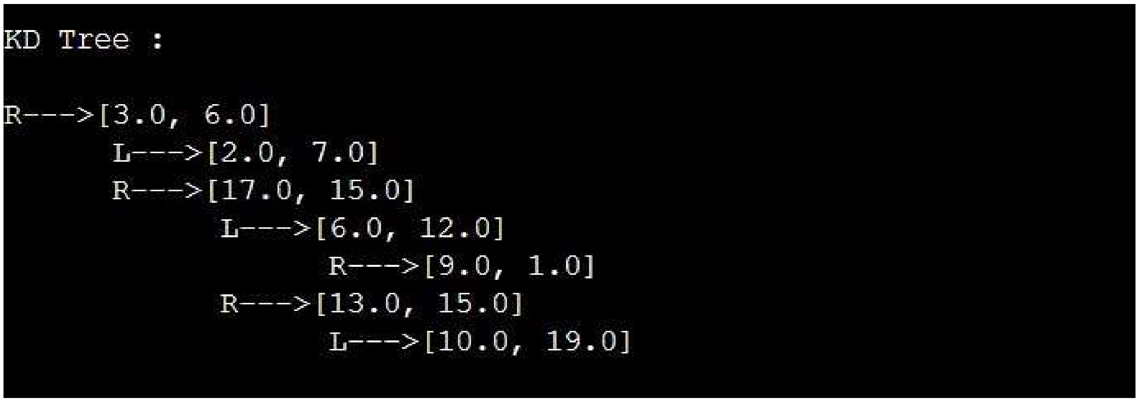
A K-D Tree(also called as K-Dimensional Tree) is a binary search tree where data in each node is a K-Dimensional point in space. In short, it is a space partitioning(details below) data structure for organizing points in a K-Dimensional space. A non-leaf node in K-D tree divides the space into two parts, called as half-spaces. Points to the left of this space are represented by the left subtree of that node and points to the right of the space are represented by the right subtree. We will soon be explaining the concept on how the space is divided and tree is formed. For the sake of simplicity, let us understand a 2-D Tree with an example. The root would have an x-aligned plane, the root's children would both have y-aligned planes, the root's grandchildren would all have x-aligned planes, and the root's great-grandchildren would all have y-aligned planes and so on.

Code:

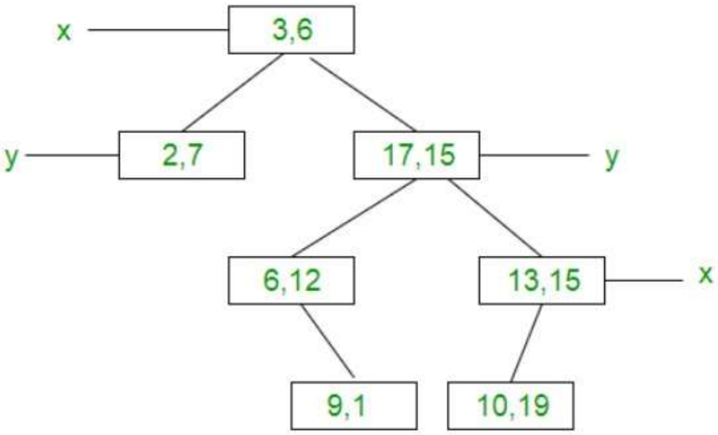
```
importjava.util.Arrays;


 public class KDTree {  private static final int K = 2; // Number

    of dimensions  private Node root;


     private static class Node {

          private double[] point; // Data point

         private Node left; // Left child  private

         Node right; // Right child


          public Node(double[] point) {
             this.point = point;
            this.left = null;
            this.right = null;
         }
    }

    public void insert(double[] point) {
        root = insert(root, point, 0);
    }
    private Node insert(Node node, double[] point, int depth) {
        if (node == null) {
            return new Node(point);
        }

        int axis = depth % K;  if (point[axis]
       < node.point[axis]) {
            node.left = insert(node.left, point, depth + 1);
        } else {  node.right = insert(node.right, point, depth +
           1);
        }
        return node;
    }

    public void printTree() {
    printTree(root, 0, "R"); // Start with root node labeled as "R"
}
```

```
private void printTree(Node node, int depth, String label) {
    if (node == null) {
        return;
    } for (int i = 0; i < depth; i++) {


         System.out.print("      "); // Indentation for tree-like format
    }
              System.out.println(label + "--->" + Arrays.toString(node.point));


    printTree(node.left, depth + 1, "L");  printTree(node.right,
   depth + 1, "R");
}


    public static void main(String[] args) {
        KDTree kdTree = new KDTree();
        double[] point1 = {3.0, 6.0};  double[]
        point2 = {17.0, 15.0};  double[] point3
        = {13.0, 15.0};  double[] point4 = {6.0,
        12.0};  double[] point5 = {9.0, 1.0};
        double[] point6 = {2.0, 7.0};  double[]
        point7 = {10.0, 19.0};
        kdTree.insert(point1);
        kdTree.insert(point2);
        kdTree.insert(point3);
        kdTree.insert(point4);
        kdTree.insert(point5);
        kdTree.insert(point6);

        kdTree.insert(point7);


         System.out.println("\nKD Tree :\n");

         // Print the k-d tree in a tree-like format  kdTree.printTree();

    }

}
```

Output:



```
KD Tree :

R--->[3.0, 6.0]
      L--->[2.0, 7.0]
      R--->[17.0, 15.0]
            L--->[6.0, 12.0]
                  R--->[9.0, 1.0]
            R--->[13.0, 15.0]
                  L--->[10.0, 19.0]
```



Conclusion: We have successfully implemented KD Tree