KD_tree　B06505004 莊博翰

****

*　Describe the Node data type you used to implement the

*　2d-tree data structure.

```java
private class Node {
    // Node's position
    private Point2D point;
    // right child  (bigger one
    private Node rc;
    // left child (smaller or equal one
    private Node lc;
    // value
    private Value val;
    // horizon false   vertical true
    private boolean type;
```

如同一般的 binary search tree 會有 right child ,left child

同時要記錄這個 node 的實際位置 Point,還有儲存的值 val

最後，用一個 booean 的 type 來表示這個 node 是水平切割還是垂直切割

****

*　Describe your method for range search in a kd-tree.

****

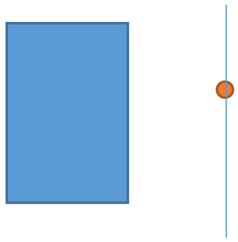
對於 range search 我先 implement 了一個 Node 的 method

```java
// recursive find all point in sub tree that inside rect ,save into ans
public void findInRectBelow(RectHV rect, Queue<Point2D> ans) {
    if (type) { // vertical
        if (rect.xmin() <= this.point.x() && lc != null) {
            lc.findInRectBelow(rect, ans);
        }
        if (rect.xmax() > this.point.x() && rc != null) {
            rc.findInRectBelow(rect, ans);
        }
    }
    else {    // horizon
        if (rect.ymin() <= this.point.y() && lc != null) {
            lc.findInRectBelow(rect, ans);
        }
        if (rect.ymax() > this.point.y() && rc != null) {
            rc.findInRectBelow(rect, ans);
        }
    }
    if (rect.contains(this.point)) ans.enqueue(this.point);
}
```

這個 method 會 recursive 去尋找這個 node 以下的 sub tree 有沒有點

在 range 中，找到了就把它放入 Queue 中，直到 right child, left child == null

當然會有一些 cut off 的條件，比如 rect 完全在 point 的左側
此時就不必 traverse Node 的 right child 了，以此類推



然後時實際在 range()中 我們只需要 call root.findRectBelow()

```java
// all points that are inside the rectangle (or on the boundary)
public Iterable<Point2D> range(RectHV rect) {
    notNull(rect);
    Queue<Point2D> re = new Queue<Point2D>();
    root.findInRectBelow(rect, re);
    return re;
}
```

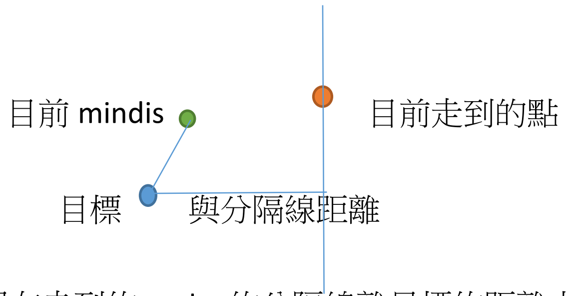
****

* Describe your method for nearest neighbor search in a kd-tree.

同樣的，先 implement 一個 Node 的 method

```java
// recursive travesal sub tree return nearest point ,
// return null when not found point that is near than nowmindis
// prune when other side will never exist a point with distance smaller than mindis
public Point2D findNearBelow(Point2D target, double nowmindis) {
    double thisdis = this.point.distanceTo(target);
    Point2D nearestPoint = null;
    Point2D tmp;
    if (thisdis < nowmindis) {
        nowmindis = thisdis;
        nearestPoint = this.point;
    }
    if (type) {
        if (target.x() - this.point.x() < nowmindis && lc != null) {
            tmp = lc.findNearBelow(target, nowmindis);
            if (tmp != null) {
                nearestPoint = tmp;
                nowmindis = nearestPoint.distanceTo(target);
            }
        }
        if (this.point.x() - target.x() < nowmindis && rc != null) {
            tmp = rc.findNearBelow(target, nowmindis);
            if (tmp != null) {
                nearestPoint = tmp;
                nowmindis = nearestPoint.distanceTo(target);
            }
        }
    }
    else {
        if (target.y() - this.point.y() < nowmindis && lc != null) {
```

首先會按照 type 分成兩種 case(vertical , horizon)，然後
recursive 去尋找這個 node 以下的 sub tree 有沒有點比現在的 mindis 還小
找不到的話就 return null，還有以些 cut off 的條件如:



如果現在走到的 node 的分隔線離目標的距離大於現在的 min distance
沒有必要 traverse right child，其他方向以此類推。
有了這個 method 後 nearest()就簡單了

```
// a nearest neighbor of point p; null if the symbol table is empty
public Point2D nearest(Point2D p) {
    notNull(p);
    return root.findNearBelow(p, p.distanceTo(root.point) + 1);
}
```

從 root 開始 findNearBelow 即可。
****
*   How many nearest-neighbor calculations can your PointST implementation
*   perform per second for input1M.txt (1 million points), where the query
*   points are random points in the unit square?
*
*   Fill in the table below, using one digit after the decimal point
*   for each entry. Use at least 1 second of CPU time.
*   (Do not count the time to read the points or to build the 2d-tree.)
*
*   Repeat the same question but with your KdTreeST implementation.
*
****

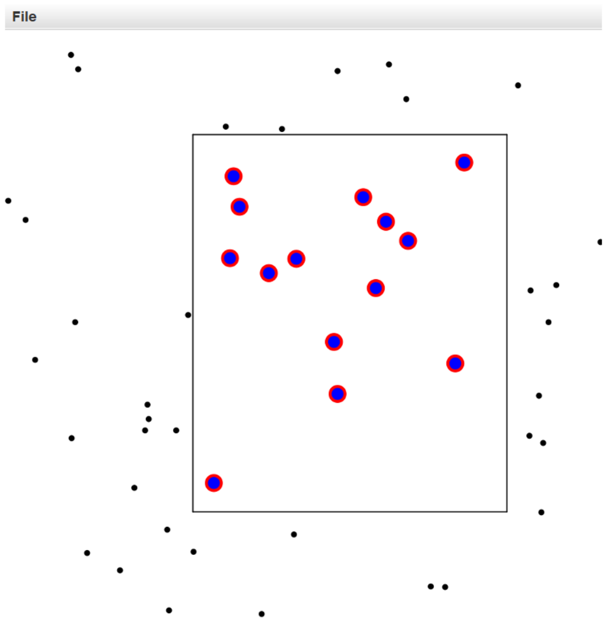
	# calls to client nearest()	/	CPU time (seconds)	=	# calls to nearest() per second
PointST:	60	/	10.72	=	5.5/sec
KdTreeST:	60000000	/	8.37=		7.1*10^6 /sec

```
60000000 times kdtree nearest :8.376 sec
60 times pointST nearest :10.724 sec

Process finished with exit code 0
```

Appendix

執行結果截圖



input50.txt