Algorithms – Programming Assignment 5

Kd-Trees

B06602035 李晴妍

E-Mail : b06602035@ntu.edu.tw

Phone : 0987883537

System information

OPERATING SYSTEM: Windows

COMPILER: IntelliJ IDEA

TEXT EDITOR / IDE: IntelliJ IDEA

Describe the Node data type you used to implement the 2d-tree data structure.

DATA TYPE:

private Node lb;

the left/bottom subtree

private Node rt;

the right/top subtree

private Point2D p;

the point

private Value value;

the symbol table maps the point to this value

private RectHV rect;

the axis-aligned rectangle corresponding to this node

private boolean ver;

vertical=1, horizontal=0;

FUNCTION:

public Node(Point2D p, Value value, boolean ver, RectHV rect)

Default constructor.

rect: The rect of root is boundary.

public RectHV rect_lb()

The rectangle on the left/bottom of node

public RectHV rect_rt()

The rectangle on the right/top of node

public boolean isRT(Point2D p)

Check if the point p is on the right of node or not

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

I used *function addpoints* for range search in a kd-tree. At first, we check if the rect contains the point of root.

If yes, the queue will enqueue the point of root, and recursively do the *addpoints* for node root.leftBottom and node root.rightTop.

If the rect doesn't contain root, we will first check if the point(rect.xmin, rect.ymin) is on the right/top of root. If it is not on the right of root, then recursively do the *addpoints* for root.leftBottom.

Then we will check if the point(rect.xmax, rect.yman) is on the right/top of the root. If it is on the right of root, then recursively do the *addpoints* for root.rightTop.

```
// all points that are inside the rectangle (or on the boundary)
public Iterable<Point2D> range(RectHV rect) {
    isNull(rect);
    if (isEmpty()) return null;
    Queue<Point2D> queue = new Queue<Point2D>();

    addpoints(root, rect, queue);
    return queue;
}
```

```
private void addpoints(Node node, RectHV rect, Queue<Point2D> queue) {
    if (node == null) return;

    if (rect.contains(node.p)) {
        queue.enqueue(node.p);
        addpoints(node.lb, rect, queue);
        addpoints(node.rt, rect, queue);
        return;
    }
    if (!node.isRT(new Point2D(rect.xmin(), rect.ymin()))) {
        addpoints(node.lb, rect, queue);
    }
    if (node.isRT(new Point2D(rect.xmax(), rect.ymax()))) {
        addpoints(node.rt, rect, queue);
    }
}
```

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

When I find the nearest neighbor in a kd-tree, I first check if the distance between the target point and the closest point on this node's rectangle is less than between the target point and the closest point we found.

If so, just replacing the closest point by the node's point.

If not, the next thing we need to confirm is that if the target point is on the right/top of node or on the left/bottom of node. Based on the situation, do the DFS to check one by one node and we will find the nearest neighbor of target point.

```
// a nearest neighbor of point p; null if the symbol table is empty
public Point2D nearest(Point2D p) {
    isNull(p);
    if (isEmpty()) return null;

    return nearpoint(p, root, root.p);
}
```

```
private Point2D nearpoint(Point2D point, Node node, Point2D closest) {
    if (node == null) return closest;

    double distance = closest.distanceSquaredTo(point);

    if (node.rect.distanceTo(point) < distance) {
        double nodeDist = node.p.distanceSquaredTo(point);
        if (nodeDist < distance) closest = node.p;

        if (node.isRT(point)) {
            closest = nearpoint(point, node.rt, closest);
            closest = nearpoint(point, node.lb, closest);
        }
        else {
            closest = nearpoint(point, node.lb, closest);
            closest = nearpoint(point, node.rt, closest);
        }
    }
    return closest;
}
```

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:	100	/	12.8	=	7.8
KdTreeST:	100	/	0.00044	=	227272

Note: more calls per second indicates better performance.

Describe any serious problems you encountered.

* kdtree 的 range 和 nearest 很難寫……發現 Recursive function 是自己很弱的一塊

List any other comments here. Feel free to provide any feedback on how much you learned from doing the assignment, and whether you enjoyed doing it.

* 超級難！這學期所有演算法作業中寫最久的一次，
查了很多資料也還是不太懂，希望之後可以搞懂而且
更熟練地運用這些演算法。