Experiment—Interval Tree Search

PB18010496 杨乐园

Introduction

设计算法，实现寻找平面区域上$n \geq 2$个点的集合S中最近点对的问题。其中“最近”为通常意义下的欧氏距离，即：点$p=(x_1,y_1),q=(x_2,y_2)$之间的欧式距离为$dist(p,q)=\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$。（不可以采取暴力求解）

Purpose

实验目的：熟悉并掌握针对红黑树数据结构的扩张，学习区间重叠的三分律，并编程掌握区间树的查找算法。

Data structure

平面区域点的数据结构为：

$$
\begin{aligned}
point = \{ \\
&num : char; \\
&x : int; \\
&y : int; \\
\}
\end{aligned}
$$

其中num为该点的编号。

最近点对的数据结构为：

$$
\begin{aligned}
closestpoint = \{ \\
&left : point; \\
&right : point; \\
&distance : double; \\
\}
\end{aligned}
$$

其中，$left$记录最近点对的x坐标较小的点，$right$记录最近点对的x坐标较大的点，$distance$为最近点对的距离。

Idea

对于有限个点的集合，我们可以通过将n个点依据x坐标的中位数进行划分，这样最近点对只可能有三种情况：其一，最近点对的x坐标均在中位数的左侧；其二，最近点对的x坐标均在中位数的右侧；其三，最近点对的两个点左右两侧各一个。而其一与其二只需递归调用即可，重点即落在了其三的求解上面。基于此，我们便有如下算法设计：

1.将点集以x坐标的大小为排序依据对点集进行排序，并取x坐标的中位数为划分依据将点集划分为S_1,S_2。

2.递归调用，分别求解点集S_1,S_2的最小点对，得到相应的最小距离为δ_1,δ_2，记$\delta=min\{\delta_1,\delta_2\}$。

3.直接利用循环比较的方法求解中间过渡段点集的最小点对，得到距离为δ_3，故$min\{\delta_3,\delta\}$即为最小距离。

而对于其三的求解，我们可以知道，若最小点对左右两侧各一个，则其x坐标必在以上面划分所用的x坐标中位数的左右两侧至多δ距离范围内。接下来，将这一过渡段点集依据y坐标的大小进行升序排序，我们可以看到，基于左右两侧点集的最小距离δ的缘故，在$\delta \times 2\delta$的的矩形区域内，至多只有8个点，从而直接遍历这8个点求得最小距离即可。

Algorithm

首先我们先将求两点的欧氏距离算法写好：

```
//距离公式；
double dist(int i, int j)
{
    return sqrt((points[i].x - points[j].x) * (points[i].x - points[j].x) +
(points[i].y - points[j].y) * (points[i].y - points[j].y));
}
```

其次我们将构建相应的排序算法：

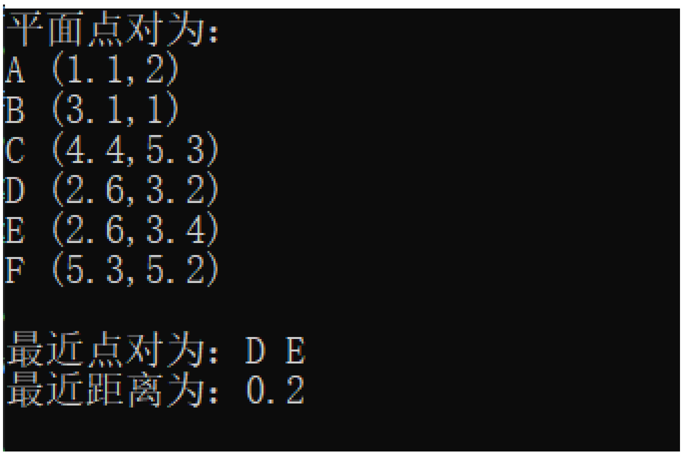
```
//依据横坐标排序，用于初始时排序；
void insertsort(int sort)
{
    point key = { 0,0,'o' };
    int i = 0;
    for (int j = 1; j < size(points); j++)
    {
        key = points[j];
        i = j - 1;
        while (i >= 0 && points[i].x > key.x)
        {
            points[i + 1] = points[i];
            i = i - 1;
        }
        points[i + 1] = key;
    }
}

//依据纵坐标排序，用于递归中排序；
void sorty(int end)
{
    point key = { 0,0,'o' };
    int i = 0;
    for (int j = 1; j < end; j++)
    {
        key = midpoint[j];
        i = j - 1;
        while (i >= 0 && midpoint[i].y > key.y)
        {
            midpoint[i + 1] = midpoint[i];
            i = i - 1;
        }
        midpoint[i + 1] = key;
    }
}
```

再者，我们直接实现求解最近点对代码：

```
//求最近点对;
closestpoint nearestpoints(int low, int high) //输入点集的起始与终止范围。
{
    closestpoint temp = { points[low],points[high],dist(low,high) };
    switch (high - low + 1)
    {
    case 2: return temp; //两个点就返回这两个点即可。
    case 3:              //三个点直接暴力求解。
    {
        if (dist(low, low + 1) < temp.distance)
            temp = { points[low],points[low + 1],dist(low,low + 1) };
        if (dist(low + 1, high) < temp.distance)
            temp = { points[low + 1],points[high],dist(low + 1,high) };
        return temp;
    }
    default: //>3的情况
    {
        closestpoint left = nearestpoints(low, floor((low + high) / 2));
        closestpoint right = nearestpoints(floor((low + high) / 2) + 1, high);
        temp = (left.distance <= right.distance) ? left : right;//取得左右最小中更小
的点对。

        int length = 0;
        int m = floor((low + high) / 2);
        for (int i = 0; i < size(points); i++)
            if (abs(points[i].x - points[m].x) <= temp.distance)
            {
                midpoint[length] = points[i];
                length++;
            }          //所以处在中间待判断的共length个。

        sorty(length); //对过渡段依据y坐标进行排序。

        for(int i=0;i<length;i++)//最多到某点的后五个点比较即可。
            for (int j = i + 1; j < ((i + 6 < length) ? (i + 6) : length); j++)
            {
                if (dist(i, j) >= temp.distance)
                    break;
                else
                    temp = { points[i],points[j],dist(i,j) };
            }

        return temp;
    }
    }
}
```

Results

通过运行程序与测试数据，我们有如下输出结果：



我们可以看到，输出结果正确。

Code

具体完整代码，参看附件文件$Nearestpoints$。