北 京 林 业 大 学

2019 学年—2020 学年第 2 学期计算机算法设计与实践实验报告书

专业：计算机科学与技术(创新实验班)　班级：计创 18

姓　　名：连月菡　　学　　号：181002222

实验地点：家　　任课教师：王春玲

实验题目：实验 3 最近对问题

实验环境：Visual Studio 2019 Community

一、 实验目的

（1）进一步掌握递归算法的设计思想以及递归程序的调试技术；
（2）理解这样一个观点：**分治与递归经常同时应用在算法设计之中。**

二、 实验内容

设 $p_1=(x_1,y_1)$, $p_2=(x_2,y_2)$, …, $p_n=(x_n,y_n)$，是平面上 n 个点构成的集合 S，设计算法找出集合 S 中距离最近的点对。

三、 实验结果

距离计算函数 **double dist(point p,point q):**利用勾股定理

$$a^2+b^2=c^2$$
$$\text{距离}c=\sqrt{a^2+b^2}$$

即

$$\|pq\|=\sqrt{(p_x-q_x)^2+(p_y-q_y)^2}$$

```
1. double dist(point p, point q) {//计算距离
2.     double x = fabs(p.x - q.x);//横坐标之差
3.     double y = fabs(p.y - q.y);//纵坐标之差
4.     return sqrt(x * x + y * y);
5. }
```

主函数 **int main()**

```
1. int main(){
```

```
    int n;//个数
    cout << "输入点对的个数: ";
    while (scanf("%d",&n)!=EOF&&n) {//如果为 0 ,程序结束
        cout << "输入" << n << "个点对的 x、y 坐标。
" << endl;
        for (int i = 0; i < n; ++i) {
            scanf("%lf%lf", &p[i].x, &p[i].y);//接收来自键
盘的 n 个点对坐标
        }
        LARGE_INTEGER c1;//开始计时
         LARGE_INTEGER c2;//结束计时
         LARGE_INTEGER frequency;//计时器频率
         QueryPerformanceFrequency(&frequency);
         double quadpart = (double)frequency.QuadPart;
         QueryPerformanceCounter(&c1);
         bf(n);//蛮力法,需要的时候 修改成分治法
         QueryPerformanceCounter(&c2);
         cout << "高精度计数器用时:
" << (double)((c2.QuadPart - c1.QuadPart) * 1.0 / quadpar
t * 1.0) * 1000000 << endl;
     }
     return 0;
}
```

1. 蛮力法

通过执行蛮力搜索，可以在 O（n*n）时间内计算最接近的一对点。 为此，可以计算所有 n（n－1）/ 2 对点之间的距离，然后选择距离最小的那对。

Input: 点的数量，点的坐标

Output: 最近对的坐标

1) 初始化最小距离
2) 从头开始遍历点集到倒数第二个点,计算每个点依次与该点之后出现的点的距离,如果它们的距离小于目前存储的最小距离,那么把这个距离值赋给最小距离。把这个点对保存为最近点对。
3) 因为每一对点之间的距离都计算出来了,最后得到的最小距离就是全局范围内的最近点对的距离,继而得到最近点对。

蛮力法伪代码:

```
1. 最小距离= 一个较大值
2. for i = 1 to length(P) - 1  //遍历点对数组
```

```
    for j = i + 1 to length(P)
         p = P[i], q = P[j]
        if dist(p, q) < 最小距离
            最小距离= dist(p, q)
            最近对 = (p, q)
return 最近对
```

蛮力法C++代码:

```
void bf(int n)//蛮力法
{
    int minpair[2]; minpair[0] = 0; minpair[1] = 0;
    double mindist=9999999;
    for (int i = 0; i < n - 1; ++i)
        for (int j = i + 1; j < n; ++j)
        {
            double distance = dist(p[i], p[j]);
            if (distance < mindist) {  //如果当前距离小于最小距离
                mindist = distance;
                minpair[0] = i; minpair[1] = j;  //存储当前最近点对和最小距离
             }
         }
     cout << "最小距离是: " << mindist << endl;
     cout << "最近对是:" << "(" << p[minpair[0]].x << "," << p[minpair[0]].y << ") " << "(" << p[minpair[1]].x << "," << p[minpair[1]].y << ")" << endl;
}
```

2.分治法

Input: 点的数量 n, 点的坐标 xi, yi
Output: 最近对的坐标(x1,y1) (x2,y2) 和最小距离 mindist
预处理:把输入的点集按照 x 轴坐标从小到大进行排序,得到排序后的点集。

1) 在有序点集数组(下标范围:0,n-1)中,找到出于中间位置的点 p[n/2]。
2) 把有序点集数组划分为两部分,前一部分为 p[0]~p[n/2],另一半为 p[n/2+1]~p[n-1]
3) 在两个子数组中递归查找左侧最近对 distleft 和右侧最近对 distright,比较 distleft 和 distright,得到最近对距离上限点对 d1=min(distleft,distright)。
4) 步骤 1-3 中，得到 d1。现在考虑左右子数组的点组成的点对的距离。以 P [n / 2]的 x 轴坐标为分界线，找到 x 坐标与 P [n / 2]的 x 轴坐标的距离小于等于 d 的所有点。建立所有这些点的数组 strip []。
5) 把 strip []里的点集按照 y 轴坐标从小到大进行排序,得到排序后的点集。
6) 在 strip []中找到最小的距离 d2。
7) 最后返回 min(d1,d2)中对应的最近对。

分治法 C++代码:

结构体和计算距离, 以及比较大小的函数

```
1.  struct point {
2.      double x;
3.      double y;
4.  };
5.
6.  struct point_pair {
7.      point a;
8.      point b;
9.  }minest_pair;
10.
11. double dist(point p, point q) {//计算距离
12.     double x = fabs(p.x - q.x);//横坐标之差
13.     double y = fabs(p.y - q.y);//纵坐标之差
14.     return sqrt(x * x + y * y);
15. }
16.
17. /*分治法*/
18.
```

```
bool cmpx(point a, point b) {//比较 x 轴坐标大小
    return a.x < b.x;
}
bool cmpy(point a, point b) {//比较 y 轴坐标大小
    return a.y < b.y;
}
point_pair min_pair(point_pair m, point_pair n) {//比较两个最近对哪个更近
    return dist(m.a, m.b) > dist(n.a, n.b) ? n : m;
}


point_pair bruteforce(point p[],int n)//蛮力法 2
{
    double mindist = 9999999;//初始化为最大值
    point_pair ans;
    ans.a = p[0]; ans.b = p[1];
    for (int i = 0; i < n - 1; ++i)
        for (int j = i + 1; j < n; ++j)//一一比较
        {
            double distance = dist(p[i], p[j]);
            if (distance < mindist) {
                mindist = distance;
                ans.a = p[i]; ans.b = p[j];//保存最近对
            }
        }
    return ans;//返回最近对
}
```

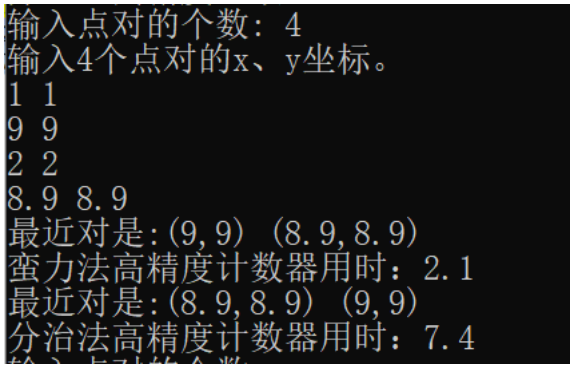
递归计算左右两边以及中间的点的最近对的函数

```
point_pair closest_strip(point strip[], int size, point_pair d) {
    double mind = dist(d.a,d.b);
    sort(strip, strip + size, cmpy);
    for (int i = 0; i < size; ++i) {
        for(int j=i+1;j<size&&(strip[j].y-strip[i].y)<mind;++j)
            if (dist(strip[i], strip[j]) < mind)
            {
                mind = dist(strip[i], strip[j]);
                d.a = strip[i];
                d.b = strip[j];
            }
    }
    return d;
```

```
}

point_pair closest_div(point p[],int n) {
    if (n <= 3) {
        return bruteforce(p,n);//剩下几个点,就直接蛮力法
    }
    int mid = n / 2;//找到中间分割点
    point mid_point = p[mid];
    point_pair disleft=closest_div(p,mid);//左半边
    point_pair disright = closest_div(p+mid,n-mid);//右半边
    point_pair d1 = min_pair(disleft, disright);//综合左右两边

    point* strip = new point[n];
    int j = 0;
    for (int i = 0; i < n; ++i) {
        if (fabs(p[i].x - mid_point.x) < dist(d1.a, d1.b))
            strip[j++] = p[i];
    }
    return min_pair(d1, closest_strip(strip, j, d1));
}

point_pair divide_conquer(point p[],int n) {//分治法总函数
    sort(p,p+n,cmpx);//按照 x 轴坐标排序
    return closest_div(p,n);//分治法 将数组对半分递归
}
```

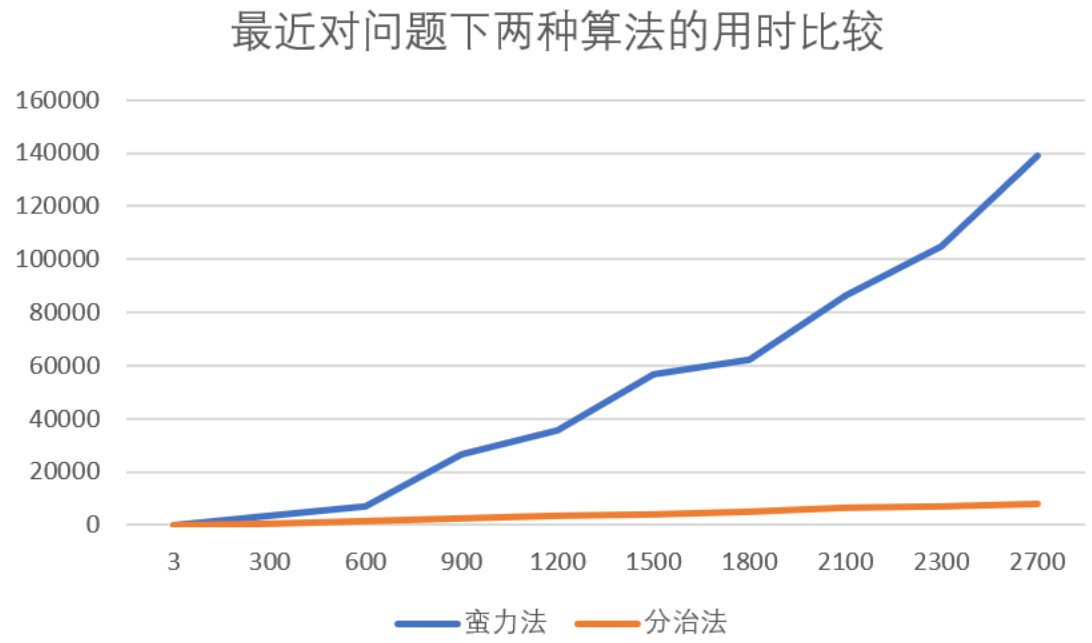
时间复杂性分析:
蛮力法: T(n) = T(n*n)
分治法: T(n) = T(nlogn)

运行程序, 可见



改变输入点对个数, 可获得如下表格数据。

	3	300	600	900	1200	1500	1800	2100	2300	2700
蛮力法	10	3512.8	6864.5	26724	35432.9	56772,9	62301	86312.4	105016	138882
分治法	1.9	658.2	1622.1	2255	3317.5	4169	4820.4	6372.4	6810.4	7857.6



测试用例: https://github.com/fafaface/Random_Set/blob/master/Input.txt
完整源代码

四、 实验中存在的问题及解决办法

Q1:如何生成大量的测试点对坐标?
A1:使用 srand()和 rand()函数即可。不过也可以在 OJ 上,提交错误答案,下载一次现有的测试用例。

通过这次实验,使我对于分治法有了更深刻的理解和认识,在 Excel 制作出折线表的时候,才惊讶地感受到 n*n 和 nlogn 的时间增长速度的鲜明差距。
除了这两种方法以外,还可以使用 K-Dtree 这样的数据结构对点对坐标进行保存,时间复杂度只有 $O(\sqrt{n})$ 。或者利用三角函数公式在平面上随机旋转点,来求得最近对。