《算法设计与分析》上机报告

姓名：	廖洲洲	学号：	PB17081504	日期：	2019.11.20
上机题目：	求平面上 n 个顶点的最近点对问题				

实验环境：
CPU：Intel Core i7-8550U; 内存:8G ; 操作系统：Win 10 ; 软件平台: JetBrains CLion ;

一、算法设计与分析：

题目一: 给定平面上 n 个点，寻找其中的一对点，使得在 n 个点的所有点对中，该点对的距离最小。

二、核心代码：

1. 算法思想：

1) 分解：P 中的点为平面上的点（x，y)。将平面上点集 P 线性分割为 2 个子集 P1 和 P2，选取一垂直线 l:x=m 来作为分割直线，其中 m 为 P 中各 x 坐标的中位数。由此 P 分割为 P1 和 P2，P1 中的所有点都在直线 l 上或直线 l 的左侧，P2 中的所有点都在直线 l 上或直线 l 的右侧。同时数组 X 被划分为 X1、X2，并按 x 坐标单调递增的顺序排序。类似地，Y 被划分为 Y1、Y2，并按 y 坐标单调递增地顺序排序。

2) 解决：把 P 划分为 P1 和 P2 后，再进行两次递归调用，一次找出 P1 中最近点对，一次找出 P2 中的最近点对。返回的最近点对距离为 δ1 和 δ2，并且置 δ=min(δ1, δ2);

3) 合并：最近点对要么是某次递归调用找出的距离为 δ 的点对，要么是 P1 中的一个点和 P2 中的一个点组成的点对。若是后者，则点对中的两个点与直线 l 的距离必定都在 δ 单位之内。则

   a) 建立一个数组 Y’,其保存与直线 l 的距离在 δ 单位之内的点，并且也是按 y 坐标排序的。

   b) 对 Y’中的每个点，分别考虑其后的 7 个点，计算出该点到其他点的距离，保存 Y’中所有点的最近点对及其距离 δ’。

   c) 返回 δ 和 δ’中更小的值，及其所对应的最近点对。

2. 核心代码

```
double Nearest(double *x,double *y,int n, struct XY *result){
    int mid;
    double d=99999999,tmp,tmp2;int i,j,m=0;
    struct XY result1,result2;
    if(n<=3){
        for(i=0;i<n-1;i++)
            for(j=i+1;j<=n-1;j++) {
                tmp=distance(x[i], y[i], x[j], y[j]);
                if (tmp < d){
                    d =tmp;
                    result->x1=x[i];result->y1=y[i];
                    result->x2=x[j];result->y2=y[j];
                }
            }
    }
    else{
        mid=n/2;
        tmp=Nearest(x,y,mid,&result1);//左子集
        tmp2=Nearest(&x[mid],&y[mid],n-mid,&result2);//右子集
        if(tmp<=tmp2){
            d=tmp;
            result->x1=result1.x1;result->y1=result1.y1;
            result->x2=result1.x2;result->y2=result1.y2;
        }
        else{
            d=tmp2;
            result->x1=result2.x1;result->y1=result2.y1;
            result->x2=result2.x2;result->y2=result2.y2;
        }
        for(i=1;i<=n;i++)
            if(fabs(X2[i]-x[mid])<=d){//距离中间点在 d 以内的点，保存下
来
                m++;
                xx[m]=X2[i];yy[m]=Y2[i];//由于 Y2 是有序的，故 yy 也是
按 y 递增有序的
            }
        for(i=1;i<=m;i++){//对范围内的每个点计算其与后 7 个点的距离
            for(j=i+1;j<=i+7&&j<=m;j++) {
                tmp=distance(xx[i],yy[i],xx[j],yy[j]);
                if(tmp<d){
                    d=tmp;
                    result->x1=xx[i];result->y1=yy[i];
                    result->x2=xx[j];result->y2=yy[j];
```

```
                    }
                }
            }
        }
        return d;
    }
```

三、结果与分析：

结果：

```
Please input:
n=12
Please input the x,y:
12 36 5 29 25 46 32 24 54 62 48 68 30 100 52 77 26 96 61 2 19 11 8 20
Nearest distance:d=5.656854
(26.000000,96.000000) (30.000000,100.000000)
进程已结束，退出代码 0
```

```
Please input:
n=5
Please input the x,y:
1 1 1 2 3 7 8 1 6 4
Nearest distance:d=1.000000
(1.000000,1.000000) (1.000000,2.000000)
进程已结束，退出代码 0
```

分析：

a) 代码初始化输入花费的时间是 O（n）。

b) 在算法的实现之前分别以 x 坐标和 y 坐标的顺序对坐标数组进行了预排序，花费的时间是 O（lgn）。

c) 调用 Nearest 时，输入的坐标数组都是按 x 坐标顺序递增排序的，然后两次递归调用输入规模为 n/2 的 Nearest。之后在寻找与直线 l 的距离在 δ 单位之内的点时，因为之前已经对 Y2 数组按 y 递增进行了预排序，故能在 O（n）时间内得到满足条件的按 y 坐标递增排序的点对 Y’。由此令 T（n）为每一步递归的运行时间，有

$$T(n)=\begin{cases}2T(n/2)+O(n) & n>3\\ O(1) & n\leqslant 3\end{cases}$$

因此 T（n）=O(nlgn)；

最后，我们得到总时间为 O(nlgn)+O(n)+O(lgn)=O(nlgn).

总结：

- 分治策略递归地求解一个问题，每层递归应用如下三个步骤：
  1) 分解步骤将问题划分为一些子问题，子问题地形式和原问题一样，只是规模更小。
  2) 解决步骤递归地求解出子问题，如果子问题的规模足够小，则停止递归，直接求解。
  3) 合并步骤将子问题的解合并成原问题的解。
- 在本实验中，我们不能在每次递归中都进行排序，否则运行时间的递归式变为 $T(n)=2T(n/2)+O(nlgn)$，其解为 $T(nlg^2n)$。因此我们使用了“预排序”来维持排序性质，而无需在每次递归调用中都进行排序。

附录（源代码）

算法源代码（C/C++/JAVA 描述）

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double *X,*X2,*xx;
double *Y,*Y2,*yy;//存储坐标
struct XY{
	double x1;double y1;double x2;double y2;
};//最近点对的坐标
void exchange(double *x,int i,int j){
	double t;
	t=x[i];
	x[i]=x[j];
	x[j]=t;
}
int partition(double *x,double *y,int p,int r){
	double k=x[r];
	int i=p-1,j;
	for(j=p;j<=r-1;j++){
		if(x[j]<=k){
			i=i+1;
			exchange(x,i,j);
			exchange(y,i,j);
		}
	}
	exchange(x,i+1,r);
	exchange(y,i+1,r);
	return i+1;
}
int quick_sort(double *x,double *y,int p,int r){//以 x 中元素递增的顺序对 x、y 排序
```

```
    int q;
    if(p<r){
        q=partition(x,y,p,r);
        quick_sort(x,y,p,q-1);
        quick_sort(x,y,q+1,r);
    }
    return 1;
}

double distance(double x1,double y1,double x2,double y2){
    return sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
}

double Nearest(double *x,double *y,int n, struct XY *result){
    int mid;
    double d=99999999,tmp,tmp2;int i,j,m=0;
    struct XY result1,result2;
    if(n<=3){
        for(i=0;i<n-1;i++)
            for(j=i+1;j<=n-1;j++) {
                    tmp=distance(x[i], y[i], x[j], y[j]);
                if (tmp < d){
                    d =tmp;
                    result->x1=x[i];result->y1=y[i];
                    result->x2=x[j];result->y2=y[j];

                }
            }
    }
    else{
        mid=n/2;
        tmp=Nearest(x,y,mid,&result1);//左子集
        tmp2=Nearest(&x[mid],&y[mid],n-mid,&result2);// 右 子
集
        if(tmp<=tmp2){
            d=tmp;
            result->x1=result1.x1;result->y1=result1.y1;
            result->x2=result1.x2;result->y2=result1.y2;
        }
        else{
            d=tmp2;
            result->x1=result2.x1;result->y1=result2.y1;
            result->x2=result2.x2;result->y2=result2.y2;
        }
```

```
            for(i=1;i<=n;i++)
                if(fabs(X2[i]-x[mid])<=d){//距离中间点在 d 以内的点，保存下来
                    m++;
                    xx[m]=X2[i];yy[m]=Y2[i];//由于 Y2 是有序的，故 yy 也是按 y 递增有序的
                }
            for(i=1;i<=m;i++){//对范围内的每个点计算其与后 7 个点的距离
                for(j=i+1;j<=i+7&&j<=m;j++) {
                    tmp=distance(xx[i],yy[i],xx[j],yy[j]);
                    if(tmp<d){
                        d=tmp;
                        result->x1=xx[i];result->y1=yy[i];
                        result->x2=xx[j];result->y2=yy[j];
                    }
                }
            }

        }
        return d;
}
int main(){
        int i,n;
        double d;
        struct XY *result;
        printf("Please input:\nn=");
        scanf("%d",&n);
        result=(struct XY *)malloc(sizeof(struct XY));
        X=(double *)malloc((n+1)* sizeof(double));
        X2=(double *)malloc((n+1)* sizeof(double));
        Y=(double *)malloc((n+1)*sizeof(double));
        Y2=(double *)malloc((n+1)*sizeof(double));
        xx=(double *)malloc((n+1)*sizeof(double));
        yy=(double *)malloc((n+1)*sizeof(double));
        printf("Please input the x,y:\n");
        X[0]=Y[0]=0;X2[0]=0;Y2[0]=0;
      for(i=1;i<=n;i++){
            scanf("%lf %lf",&X[i],&Y[i]);
            X2[i]=X[i];Y2[i]=Y[i];
        }
/*        for(i=1;i<=n;i++){
            printf("(%lf,%lf)",X[i],Y[i]);
        }*/
```

```
        quick_sort(X,Y,1,n);//将 X、Y 以 x 递增排序
        quick_sort(Y2,X2,1,n);//将 X2、Y2 以 y 递增排序

        d=Nearest(&X[1],&Y[1],n,result);
        printf("Nearest distance:d=%lf\n",d);
        printf("(%lf,%lf)
(%lf,%lf)",result->x1,result->y1,result->x2,result->y2);


        return 0;
}
```