**最近点对问题**

**姓名：陈鸿绪 学号：PB21000224 日期：10.18**

**实验内容：**

读入文件data.txt中的所有数据点个数和数据点，求出两点距离最小的一对点，要求时间复杂度不大于O(nlogn)。输出最近距离点对值和距离。

**算法设计思路：**

1. 首先对点集按照x坐标进行排序，保证操作时点集x坐标的有序性。
2. 采用分治算法，将平面中的点按照x坐标的中位数mid划分成两部分，递归算法求得左右点集中的最小距离值min和点对。同时分别对左右集合递归地进行归并排序，归并排序的过程中选取x坐标处于区间[mid-min,mid+min]之中的点，设该集合为A，注意到该集合关于y坐标有序。
3. 对有序集合A，依次选取其中的点S，考察位于该点之后的8个点Ei（几何意义上这些点是x坐标区间[mid-min,mid+min]中恰好在S点上方的8个点），过程中根据S与Ei的距离更新最小距离值min和最新点对。直至A集合被遍历结束得到本层递归得到的答案。
4. 递归出口：集合中只有1或2个点情况，直接排序、求出距离即可，其中1个点的情况距离被认为无穷大。

**时间复杂度分析：**

对于规模为N的问题，该算法首先分为了等规模的两部分N/2，所以递归所占有的时间复杂度为2T(N/2)，在递归排序的过程中我们同时选取了x坐标位于区间[mid-min,mid+min]的点组成集合A，最差的情况A中的点在O(N)的数量级，此时对于A中的每个点，需要计算该点之上的8个点，所以最差情况需要6O(N)才可以完成该操作，其余的细节操作均为O(1)，所以T(N)=2T(N/2)+O(N)+O(1)为该算法的时间复杂度递推式，可以解出来T(N)=O(NlogN)。

**算法核心代码：**

float min\_dist(dot \*dot\_arr,dot \*dot\_arr\_y,int p,int r,dot &min\_dot\_l,dot &min\_dot\_r){

if(p==r) return 1e6; //递归出口，只有1个点情况

else if(r-p==1){

min\_dot\_l=dot\_arr[p],min\_dot\_r=dot\_arr[r];

if(dot\_arr\_y[p].y>dot\_arr\_y[r].y){

dot temp=dot\_arr\_y[p];

dot\_arr\_y[p]=dot\_arr\_y[r];

dot\_arr\_y[r]=temp;

}

return dist(dot\_arr[p],dot\_arr[r]);

}//递归出口，只有两个点情况

int mid=(p+r)/2;

dot min\_dot\_l\_1,min\_dot\_r\_1,min\_dot\_l\_2,min\_dot\_r\_2;

float l\_min=min\_dist(dot\_arr,dot\_arr\_y,p,mid,min\_dot\_l\_1,min\_dot\_r\_1),r\_min=min\_dist(dot\_arr,dot\_arr\_y,mid+1,r,min\_dot\_l\_2,min\_dot\_r\_2),min;

if(l\_min<=r\_min) min=l\_min,min\_dot\_l=min\_dot\_l\_1,min\_dot\_r=min\_dot\_r\_1;

else min=r\_min,min\_dot\_l=min\_dot\_l\_2,min\_dot\_r=min\_dot\_r\_2;

//以上得到左右两边不考虑跨区间的最近距离和对应点对

float x\_floor=dot\_arr[mid].x-min;

dot temp[N],temp\_right[N/2];int temp\_len=0,temp\_right\_len=0;

float r\_x\_ceil=dot\_arr[mid].x+min,r\_x\_floor=dot\_arr[mid].x-min;

int left\_p=p,right\_p=mid+1;

while((left\_p<=mid)&&(right\_p<=r)){

if(dot\_arr\_y[left\_p].y<=dot\_arr\_y[right\_p].y){

temp[temp\_len++]=dot\_arr\_y[left\_p++];

if(dot\_arr\_y[left\_p-1].x>=r\_x\_floor) temp\_right[temp\_right\_len++]=dot\_arr\_y[left\_p-1];

}

else{

temp[temp\_len++]=dot\_arr\_y[right\_p++];

if(dot\_arr\_y[right\_p-1].x<=r\_x\_ceil) temp\_right[temp\_right\_len++]=dot\_arr\_y[right\_p-1];

}

}

if(left\_p>mid){

for(int i=right\_p;i<=r;i++){

temp[temp\_len++]=dot\_arr\_y[i];

if(dot\_arr\_y[i].x<=r\_x\_ceil&&dot\_arr\_y[i].x>=r\_x\_floor) temp\_right[temp\_right\_len++]=dot\_arr\_y[i];

}

}

else{

for(int i=left\_p;i<=mid;i++){

temp[temp\_len++]=dot\_arr\_y[i];

if(dot\_arr\_y[i].x<=r\_x\_ceil&&dot\_arr\_y[i].x>=r\_x\_floor) temp\_right[temp\_right\_len++]=dot\_arr\_y[i];

}

}//以上归并排序，并同时得到[r\_x\_floor,r\_x\_ceil]区间的关于坐标y的有序点集合

for(int i=0;i<temp\_len;i++){

dot\_arr\_y[p+i]=temp[i];

}

for(int i=0;i<temp\_right\_len;i++){

for(int j=0;j<8&&i+j+1<temp\_right\_len;j++){

//选取8个点

float dist\_=dist(temp\_right[i],temp\_right[i+j+1]);

if(dist\_<min){

min\_dot\_l=temp\_right[i];

min\_dot\_r=temp\_right[i+j+1];

min=dist\_;

}

}

}

return min;

}

**实验结果分析:**



第一行是递归算法所得到的结果，第二行是朴素算法得到的结果，两者得到的结果相同，但是可以发现第一种算法对比第二种算法的所用时间大大减少。第一种用时5，第二种用时311，所以可见O(NlogN)的递归算法明显优于第二种朴素暴力O(N^2)算法。