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_ar
	r,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:

```
S1:(1108.262939 -6413.291016) S2(1110.193970 -6411.252930) dis:2.807610 time:5
S1:(1108.262939 -6413.291016) S2(1110.193970 -6411.252930) dis:2.807610 time:311
```

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。