

作业一、二：合并、快速排序

合并排序、非递归合并排序、快速排序、随机选择划分快速排序

```
#`include<iostream>`

#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include <sys/time.h>
using namespace std;
#define MAXSIZE 20000//当前最大数组规模;
#define MAX 20000//当前最大最大值

typedef int KeyType,InfoType;
typedef struct RedType{
    KeyType key;//关键字
    InfoType otherinfo;
}RedType;
typedef struct SqList{//顺序存储;
    RedType r[MAXSIZE+1];//r[0]闲置或作哨兵;
    int length;//当前序列的长度;
}SqList;
class sort{
private:
    SqList L;//序列L
    SqList tmp;//临时数组;
    int DD;//无序度
    float ADD;//平均无序度
public:
    int depth;//递归深度
    void InitSqList(int n,int N);//生成序列长度为n, 最大值为N的元素相同但顺序不同的随机序列;
    int MergeSort(int low,int high);//递归归并排序, 并返回递归深度;
    void Merge(int low,int mid,int high);//归并L[low..mid]和l[mid+1...high]
    void MergeSort_not_recursion();//非递归方式的归并排序;
    int QuickSort(int low,int high);//快速排序
    int Partition(int low,int high); //对顺序表L中的子序列L.[low...high]进行非递减排序;
    int Randomized_quickSort(int low,int high);//具有随机选择划分的快排;
    int RandomizedPartition(int low,int high);//具有随机选择, 对顺序表L中的子序列L.[low...high]
    进行非递减排序;
    void ShowSqList();//显示此时的序列;
    void swap_element(int i,int j); //交换元素;
    void show_DD_ADD_level();//显示DD,ADD, 递归层次;
};
int main()
{
    srand(time(0));
```

```

sort s1,s2,s3,s4;

int n=MAXSIZE,N=MAX;
cout<<"请选择输入的方式: 1、递归归并排序; 2、非递归递归归并排序; 3、快速排序; 4、具有随机划分的快速
排序;5、终止程序"<<endl;
s1.InitSqlList(n,N);
s2=s3=s4=s1;
int key;
cin>>key;
//记录其实时间和终止时间, 毫秒级别;
struct timeval t_start,t_end;
while(key!=5)
{
    switch(key){
        case 1:
            cout<<"递归归并排序: "<<endl;
            gettimeofday(&t_start,NULL); //程序开始时间
            s1.depth=s1.MergeSort(1,MAXSIZE);
            gettimeofday(&t_end,NULL); //程序终止时间;
            s1.show_DD_ADD_level(); //输出无序度, 平均无序度和递归深度;
            break;
        case 2:
            cout<<"非递归递归归并排序: "<<endl;
            gettimeofday(&t_start,NULL); //程序开始时间
            s2.MergeSort_not_recursion();
            gettimeofday(&t_end,NULL); //程序终止时间;
            s2.show_DD_ADD_level(); //输出无序度, 平均无序度和递归深度;
            break;
        case 3:
            cout<<"快速排序: "<<endl;
            gettimeofday(&t_start,NULL); //程序开始时间
            s3.depth=s3.QuickSort(1,MAXSIZE);
            gettimeofday(&t_end,NULL); //程序终止时间;
            s3.show_DD_ADD_level(); //输出无序度, 平均无序度和递归深度;
            break;
        case 4:
            cout<<"具有随机划分的快速排序: "<<endl;
            gettimeofday(&t_start,NULL); //程序开始时间
            s4.depth=s4.Randomized_quickSort(1,MAXSIZE);
            gettimeofday(&t_end,NULL); //程序终止时间;
            s4.show_DD_ADD_level(); //输出无序度, 平均无序度和递归深度;
            break;
        default:
            break;
    }
    cout<<"运行时间: "<<t_end.tv_usec-t_start.tv_usec+1000000*(t_end.tv_sec-t_start.tv_sec)
    <<"微秒"<<endl;
    cin>>key;
}
return 0;
}

```

```

int sort::MergeSort(int low,int high)//递归归并排序;
{
    if(low==high)
    {
        tmp.r[low]=L.r[low];
        return 1;//不能继续分解时,递归深度为1;
    }
    else
    {
        int mid=(low+high)/2;
        int depth_l=MergeSort(low,mid);//左子问题求解,返回递归深度;
        int depth_r=MergeSort(mid+1,high);//右子问题求解
        Merge(low,mid,high);//将左右排好序的数组合并到另一个数组;
        return depth_l>depth_r?depth_l+1:depth_r+1;
    }
}

void sort::Merge(int low,int mid,int high)//归并L[low..mid]和l[mid+1...high]
{
    int i;//左子段的搜索比较指针的起点
    int j;//右子段的搜索比较指针的起点
    int k;//目标数组d的指针起点,符合递增顺序的tmp.r[ ]中的元素被拷贝至L.r[k]

    for(i=low,j=mid+1,k=low;i<=mid&& j<=high;k++)
        if(L.r[i].key<=L.r[j].key)
            tmp.r[k]=L.r[i++];
        else
            tmp.r[k]=L.r[j++];
    while(i<=mid)
        tmp.r[k++]=L.r[i++];
    while(j<=high)
        tmp.r[k++]=L.r[j++];
    for(int i=low;i<=high;i++)
        L.r[i]=tmp.r[i];
}

void sort::MergeSort_not_recursion()//非递归方式的归并排序;
{
    int buck=1,low,mid,high;//buck为当前子序列长度;
    while(buck<L.length)
    {
        for(low=1;low+buck<=L.length;)
        {
            mid=low+buck-1;
            high=mid+buck;
            if(high>L.length)
                high=L.length;
            //将tmp.r[]中长度为s的子序列对合并到数组L.r[];合并完后,长度buck增加,为下次合并做准备
            Merge(low,mid,high);
            low=high+1;

        }

        buck=buck*2;
    }
}

```

```

    }
}

int sort::QuickSort(int low,int high)
{
    int sorted=1;
    //从左向右扫描a[p:r]中全部元素, 判断a[p:r]是否满足非递减顺序, 满足则不进行排序;
    for(int i=low;i<=high;i++)
        if(L.r[i].key>L.r[i+1].key)
        {
            sorted=0;
            break;
        }
    if(low<high&&sorted==0)
    {
        int pivotloc=Partition(low,high);//找到枢轴点;
        int depth_l=QuickSort(low,pivotloc-1);//进行左子序排序;
        int depth_r=QuickSort(pivotloc+1,high);//进行右子序排序;
        return depth_l>depth_r?depth_l+1:depth_r+1;
    }
    else
        return 1;
}

int sort::Partition(int low,int high)//对顺序表L中的子序列L.[low...high]进行非递减排序;
{
    L.r[0]=L.r[low];
    int pivotkey=L.r[low].key;
    while(low<high)
    {
        while(high>low&&L.r[high].key>=pivotkey)//寻找小于 pivotkey的下标;
            high--;
        L.r[low]=L.r[high];
        while(high>low&&L.r[low].key<=pivotkey)
            low++;
        L.r[high]=L.r[low];
    }
    L.r[low]=L.r[0];//最后将最开始的pivotkey存放到low中, 使序列 L.[low...high]左边都小于pivotkey右边都大于pivotkey;
    return low;
}

//随机选择划分元素
int sort::Randomized_quickSort(int low,int high)
{
    int sorted=1;
    //从左向右扫描a[p:r]中全部元素, 判断a[p:r]是否满足非递减顺序, 满足则不进行排序;
    for(int i=low;i<=high;i++)
        if(L.r[i].key>L.r[i+1].key)
        {
            swap_element(i,i+1);
            sorted=0;

            break;

```

```

    }
    if(low<high&&sorted==0)
    {
        int pivotloc=RandomizedPartition(low,high);//找到枢轴点;
        int depth_l=QuickSort(low,pivotloc-1);//进行左子序排序;
        int depth_r=QuickSort(pivotloc+1,high);//进行右子序排序;
        return depth_l>depth_r?depth_l+1:depth_r+1;
    }
    else
        return 1;
}

int sort::RandomizedPartition(int low,int high)
{
    int x=rand()%(high-low+1)+low;//随机选一个L.r[x]
    swap_element(low,x);//L.r[x],与 L.r[low]交换;
    L.r[0]=L.r[low];
    int pivotkey=L.r[low].key;
    while(low<high)
    {
        while(high>low&&L.r[high].key>=pivotkey)//寻找小于 pivotkey的下标;
            high--;
        L.r[low]=L.r[high];
        while(high>low&&L.r[low].key<=pivotkey)
            low++;
        L.r[high]=L.r[low];
    }
    L.r[low]=L.r[0];//最后将最开始的pivotkey存放到low中,使序列 L.[low...high]左边都小于pivotkey右边都大于pivotkey;
    return low;
}

void sort::InitSqlList(int n,int N)
{
    ADD=0;
    DD=0;
    L.length=n;//序列的长度;
    cout<<"初始化序列"<<endl;
    //先顺序生成1-N的数;
    for(int i=1;i<=L.length;i++)
        L.r[i].key=i;
    for(int i=1;i<N;i++)
    {
        //将第i个数与r[i+1...N]中随机某个数交换;
        int j=rand()%(N-i)+i+1;
        RedType tmp=L.r[i];
        L.r[i]=L.r[j];
        L.r[j]=tmp;
    }

    //计算序列DD的值
    for(int i=1;i<=L.length;i++)
        for(int j=1;j<=L.length;j++)
            if(i<j&&L.r[i].key>L.r[j].key)

                DD++;

```

```
        ADD=DD/L.length;

    }

    void sort::swap_element(int i,int j)//交换元素
    {

        RedType tmp=L.r[i];
        L.r[i]=L.r[j];
        L.r[j]=tmp;
    }
    void sort::show_DD_ADD_level()
    {
        cout<<"DD="<<DD<<endl<<"ADD="<<ADD<<endl<<"递归深度="<<depth<<endl;
    }
}
```

编号	长度	组号	DD	ADD	快排1时间 (us)/递归层 次	快排2时间 (随 机) (us)/递归层 次	递归合并时间/ 递归层次 (us)	非递归合 并时间 (us)
1	2000	1	3050	1	0.8/1	3.9/2	973.9/12	857.2
2	2000	1	24700	12	0.8/1	0.56/5	859.5/12	800
3	2000	1	225188	112	7.5/7	9.9/7	875.5/12	811
4	2000	1	910715	455	49.8/21	69.5/22	863.6/12	845
5	2000	1	1005280	502	145.5/19	104.2/18	786.5/12	806.4
6	5000	2	11491	2	1.6/1	6.8/3	2458.4/14	2189.1
7	5000	2	167275	33	3.5/1	11.1/4	3230/14	2101.3
8	5000	2	1399400	279	2.0/1	3.0/13	2179.1/14	3100.5
9	5000	2	5637831	1127	148.5/21	228/21	2040.4/14	1942.6
10	5000	2	6181618	1236	290/22	244.2/24	2032.5/14	1961.5
11	10000	3	37749	3	3.1/1	9.5/3	4997.1/15	3852.6
12	10000	3	644699	64	3.6/1	10.6/3	4173.4/15	3954.2
13	10000	3	5906046	590	4.8/1	52.1/11	4103.9/15	4344.9
14	10000	3	2.2E+07	2240	347.7/24	338.6/22	4262.6/15	3907.2
15	10000	3	2.5E+07	2517	541.6/31	548.4/25	4550.4/15	4300.5
16	15000	4	116872	7	4.7/1	15.8/3	6672.3/15	6199
17	15000	4	1597883	106	4.8/1	27.7/6	6645.3/15	6276.7
18	15000	4	1.3E+07	840	4.7/1	63.2/9	6518.2/15	6333.2
19	15000	4	5.1E+07	3375	4.8/1	358.7/21	6026.2/15	6148.3
20	15000	4	5.7E+07	3785	833.6/28	846.9/33	6000.6/15	6008.5
21	20000	5	300896	15	6.7/1	35.1/3	8361.6/16	8347.2
22	20000	5	2563322	128	6.4/1	23.0/3	8973.9/16	8203.9
23	20000	5	2.3E+07	1132	6.4/1	90.1/12	9454.1/16	8179.8
24	20000	5	9.1E+07	4537	982.2/27	539.7/24	7934.4/16	8651.8
25	20000	5	1E+08	5014	1541.1/32	1103/30	8404.2/16	7984.2
26	30000	6	642539	21	9.8/1	24.9/3	12970.7/16	13218
27	30000	6	5760660	192	0.95/1	33.9/3	13530.1/16	12823.2
28	30000	6	5.2E+07	1725	0.95/1	111.8/13	13105.3/16	12186.9
29	30000	6	2E+08	6730	1221.4/24	1216.4/27	12330.3/16	12023.3
30	30000	6	2.2E+08	7440	2056.3/29	2059.9/29	12124.1/16	11367.1

长度	组号	avg DD	avg ADD	avg Qs (us)	avg Qs(随机) (us)	avg Qs(随机) (us)	
2000	1	2168990	216.4	40.88	38.62	871.8	
5000	2	13397600	535.4	89.12	100.62	2388.08	
10000	3	54160600	1082.8	180.16	191.84	4417.48	
15000	4	121748000	1622.6	170.52	262.46	6372.52	
20000	5	216565000	2165.2	508.56	358.18	8625.64	
30000	6	483301000	3221.6	661.3	689.38	12812.1	

划分基准元素x的选取采用2种方式：

- 1) 固定选取左端a[p]，Partition——快排算法1
- 2) 随机a[p:r]中元素，RandomizedPartition——快排算法2

比较2种划分方式下递归层次的差异：

- 选取快排算法1平均运行时间较长，但ADD比较小时递归深度更小；
- 选取快排算法2平均运行时间较短，但当数组长度较大时运行时间更，且递归深度更大；

作业三---线性时间选择

- 采用线性时间选择算法，根据基站k-dist距离，挑选出
- nk-dist值最小的基站
- nk-dist第5小的基站
- nk-dist值第50小的基站
- nk-dist值最大的基站

```
#include<iostream>
#include<time.h>
#include<stdlib.h>
#include<fstream>
#include<string.h>
using namespace std;
typedef float Type;
typedef struct data{
    int id;//基站id
    Type x,y,k_dist;//x为维度, y为经度;
}data;//基站定义
```



```

int depth=0;//递归深度;
void swap_element(Type a[],int i,int j);
void bubble_sort(Type a[],int p,int r);//当在r-p个序列中搜索时,冒泡排序
Type linear_select_3(Type a[],int p,int r,int k);//一分为三,线性时间选择第k小的数;
Type linear_select_2(Type a[],int p,int r,int k);//一分为二,线性时间选择第k小的数;
int linear_partition(Type a[],int low,int high,Type k);//按照关键字k将a[]划分成左子段关键字小于k,
右子段关键字大于k
void read_file(string file_name,data node_b[],Type dist[]);//从文件中读取1033个基站数据
data find_data(data node_b[],int num,Type key);//寻找k_dist为key的基站信息;
int main()
{
    int n=1033;
    Type num[n];
    data node_b[n];
    string file_name="data.txt";//文件名,该文件保存1033个基站数据;
    read_file(file_name,node_b,num);//从文本读取数据到内存
    int rank[4]={1,5,50,n};//d第rank[i]小的基站;
    cout<<"一分为三"<<endl;
    for(int i=0;i<4;i++)
    {
        depth=0;
        Type key=linear_select_3(num,0,n-1,rank[i]);//找到与第rank[i]小的k_dist对
        data node_key=find_data(node_b,n,key); //找到与第k小的k_dist对应的基站信息;
        cout<<"递归深度="<<depth<<endl<<"第"<<rank[i]<<"小:"<<"id:["<<node_key.id<<"]"<<"纬度="
<<node_key.x<<" 经度="<<node_key.y<<" k_dist="<<key<<" "<<endl;
    }
    cout<<"一分为二"<<endl;
    for(int i=0;i<4;i++)
    {
        depth=0;
        Type key=linear_select_2(num,0,n-1,rank[i]);//找到与第rank[i]小的k_dist
        data node_key=find_data(node_b,n,key);//找到与第rank[i]小的k_dist对应的基站信息;
        cout<<"递归深度="<<depth<<endl<<"第"<<rank[i]<<"小:"<<"id:["<<node_key.id<<"]"<<"纬度="
<<node_key.x<<" 经度="<<node_key.y<<" k_dist="<<key<<" "<<endl;
    }
    return 0;
}

void read_file(string file_name,data tmp[],Type dist[])//从文件中读取1033个基站数据
{
    ifstream fin;
    cout<<"read file"<<endl;
    fin.open(file_name.c_str(),ios_base::in);
    if(!fin)
        cout<<"文件打开失败"<<endl;
    int num;
    fin>>num;
    for(int i=0;i<num;i++)//读取每一行的数据;
    {
        fin>>tmp[i].id>>tmp[i].x>>tmp[i].y>>tmp[i].k_dist;
        dist[i]=tmp[i].k_dist;
    }
}

```

```

}
data find_data(data node_b[],int num,Type key)//寻找k_dist为key的基站信息;
{
    for(int i=0;i<num;i++)
        if(node_b[i].k_dist==key)
            return node_b[i];
}
Type linear_select_3(Type a[],int p,int r,int k)
{
    depth++;
    if(r-p<20)//a[]足够小 , 简单排序;
    {
        bubble_sort(a,p,r);
        return a[p+k-1];
    }

    int i,j;
    //寻找每一组的中点;
    for(i=0;i<=(r-p-4)/5;i++)
    {
        int s=p+5*i,t;
        t=s+4;
        bubble_sort(a,s,t);//采用冒泡排序
        swap_element(a,p+i,s+2);//将第i组的中位数 与a[p+i]交换, 移到a[p.r]的前端;
    }

    Type x=linear_select_3(a,p,p+(r-p-4)/5,(r-p+6)/10);//选取中位数的中位数作为划分基准 , (r-
    p+6)/10为 a[p,p+(r-p-4)/5]数组元素个数的中位数;

    i=linear_partition(a,p,r,x);//根据划分基准x=10, 将a[p:r]划分为2部分: 1.左子段为a[p:r], 左子段长
    度j=i-p+1;
    j=i-p+1;//左子段+中间子段的长度;

    //一分为三, 减少递归深度;
    if(k==j)//根据k与左子段长度j的比较, 采用减治法
        return a[j];
    else
    {
        if(k<j)
            return linear_select_3(a,p,i-1,k);
        else
            return linear_select_3(a,i+1,r,k-j);
    }
}
Type linear_select_2(Type a[],int p,int r,int k)
{
    depth++;
    if(r-p<20)
    {
        bubble_sort(a,p,r);
        return a[p+k-1];
    }
}

```

```

int i,j;
for(i=0;i<=(r-p-4)/5;i++)
{
    int s=p+5*i,t;
    t=s+4;
    bubble_sort(a,s,t);
    swap_element(a,p+i,s+2);//将第i组的中位数 与a[p+i]交换;
}

//选取中位数的中位数 , (r-p+6)/10为 a[p,p+(r-p-4)/5]数组元素个数的中位数;
Type x=linear_select_2(a,p,p+(r-p-4)/5,(r-p+6)/10);

i=linear_partition(a,p,r,x);//
j=i-p+1;//左子段+中间子段的长度;
//一分为三, 减少递归深度;
if(k<=j)
    return linear_select_2(a,p,i-1,k);
else
    return linear_select_2(a,i+1,r,k-j);
}

//按照关键字k将a[]划分成左子段关键字小于k, 右子段关键字大于k
int linear_partition(Type a[],int low,int high,Type k)
{
    swap_element(a,low,low+(high-low-4)/10);

    Type pivotkey=a[low];
    while(low<high)
    {
        while(high>low&& a[high]>=pivotkey)
            high--;
        a[low]=a[high];
        while(high>low&& a[low]<=pivotkey)
            low++;
        a[high]=a[low];
    }
    a[low]=pivotkey;
    return low;
}

//交换元素
void swap_element(Type a[],int i,int j)
{
    Type tmp=a[i];
    a[i]=a[j];
    a[j]=tmp;
}

//冒泡排序
void bubble_sort(Type a[],int p,int r)
{
    int i,j;
    Type tmp;
    for(i=0;i<r-p;i++)

        for(j=p;j<r-i;j++)

```

```
        if(a[j]>a[j+1])
        {
            tmp=a[j];
            a[j]=a[j+1];
            a[j+1]=tmp;
        }
    }
```

```
C:\Users\戴尔\Desktop\算法设计\第一次\线性时间选择.exe
read file
一分为三
递归深度=18
第1小:id:[568030]纬度=102.676 经度=25.0102 k_dist=103.075
递归深度=19
第5小:id:[567883]纬度=102.741 经度=25.0522 k_dist=126.096
递归深度=18
第50小:id:[568074]纬度=102.753 经度=25.0352 k_dist=208.475
递归深度=18
第1033小:id:[568313]纬度=102.863 经度=25.0983 k_dist=2735.8
一分为二
递归深度=18
第1小:id:[568030]纬度=102.676 经度=25.0102 k_dist=103.075
递归深度=19
第5小:id:[567883]纬度=102.741 经度=25.0522 k_dist=126.096
递归深度=20
第50小:id:[568074]纬度=102.753 经度=25.0352 k_dist=208.475
递归深度=18
第1033小:id:[568313]纬度=102.863 经度=25.0983 k_dist=2735.8
-----
```

一分为二和一分为三递归深度差异

一分为二递归深度大部分情况下与一分为三的递归深度相同，偶尔递归深度更深

作业四--平面最接近点对

采用平面最近点对算法，根据基站经纬度，挑选出

n距离最近的2个基站

n距离次最近的2个基站

代码中x表示纬度，y表示经度

```
#include<iostream>
#include<time.h>
#include<stdlib.h>
#include<math.h>
#include<fstream>
#include<iomanip>
using namespace std;
#define INFI 65546
#define MAXSIZE 1033//数组规模;
#define PI 3.14159265//π的值
#define R 6.371229*1e6//地球半径;
typedef double Type;

typedef struct point{
    int id;//基站id
    Type x;//纬度
    Type y;//经度
    Type k_dist;
}point;//基站信息

typedef struct Pair{
    point a;
    point b;
    Type dis;//平面两点距离
}pair2;//距离+点对

class closest_pair{
private:
    pair2 closest_p;//最近点对;
    pair2 next_closest_p;//次最接近点对;
    point xx[MAXSIZE];//X[],按x坐标排序的全部点,
    point yy[MAXSIZE];//Y[]:按Y坐标排序的全部点
    point zz[MAXSIZE];//Z[]:存放P1+P2范围内按Y坐标排序的点
    point tmp[MAXSIZE];//归并排序临时存放数组;
    ifstream fin;

public:

    void cpair();//得到最近点对和次最近点对并输出;
    pair2 closest(point xx[], point yy[],point zz[],int l,int r);//返回最接近点对,用次最接近
    点对保留被替换前的最接近点对;
    Type get_rad(Type d);//将经纬度转为弧度
    Type distance_point(point d1,point d2);//两点间的距离;
    void merge_sort_x(int low,int high);//依据x坐标归并排序
    void merge_x(int low,int mid,int high);//归并tmp[low..mid]和tmp[mid+1...high]到
    xx[low..high]
    void merge_sort_y(point a[],point b[],int low,int high);//依据y坐标归并排序
    void merge_y(point b[],point c[],int low,int mid,int high); //归并b[low..mid]和
    b[mid+1...high]到c[low..high];
    void read_file(string file_name);//从文件中读取1033个基站数据

    void show_closest_pair();//显示最近点对的经纬度、id、k_dist
```

```

};

int main()
{
    closest_pair s1;
    s1.read_file("data.txt");//该文件存储1033个基站数据，从文件中读取1033个基站数据
    s1.cpair();//得到最近点对和次最近点对并输出;
    return 0;
}

void closest_pair::cpair();//得到最近点对和次最近点对;
{
    merge_sort_x(0,MAXSIZE-1);//依据x坐标排序
    //yy[]:按Y坐标排序的全部点
    for(int i=0;i<MAXSIZE;i++)
    {
        yy[i].x=xx[i].x;
        yy[i].y=xx[i].y;
    }
    merge_sort_y(yy,yy,0,MAXSIZE-1);//依据y坐标排序
    //计算最近点对;
    closest_p=closest(xx,yy,zz,0,MAXSIZE-1);//得到最近点对;用次最接近点对保留被替换前的最接近点对;
    show_closest_pair(); //显示最近点对的经纬度、id、k_dist
}
//返回最接近点对，用次最接近点对保留被替换前的最接近点对;
pair2 closest_pair::closest(point xx[], point yy[],point zz[],int l,int r)
{
    pair2 p1;
    if(r-l==0)//只有一个点
    {
        p1.a=p1.b=xx[l];
        p1.dis=INFI;
        return p1;
    }
    if(r-l==1)//两点的情况;
    {
        p1.a=xx[l];
        p1.b=xx[l+1];
        p1.dis=distance_point(xx[l],xx[l+1]);
        return p1;
    }
    if(r-l==2)//三个点的情况;
    {
        Type d1=distance_point(xx[l],xx[l+1]);
        Type d2=distance_point(xx[l+1],xx[l+2]);
        Type d3=distance_point(xx[l],xx[l+2]);
        if(d1<d2&&d1<d3)
        {
            p1.a=xx[l];
            p1.b=xx[l+1];
            p1.dis=d1;
        }

        return p1;
    }
}

```

```

    }
    else if(d2<d3)
    {
        p1.a=xx[l+2];
        p1.b=xx[l+1];
        p1.dis=d2;
        return p1;
    }
    else
    {
        p1.a=xx[l];
        p1.b=xx[l+2];
        p1.dis=d3;
        return p1;
    }
}
//多于两个点的情况;
int m=(l+r)/2;
int f=l,g=m+1;
for(int i=l;i<=r;i++)
    if(yy[i].x>xx[m].x)
        zz[g++]=yy[i];
    else
    {
        zz[f++]=yy[i];
    }
//求左右子段p1、p2中距离最小的点对;
pair2 best,p3;
best=closest(xx,zz,yy,l,m);
p3=closest(xx,zz,yy,m+1,r);

//分别从z[L:m]中得到左、右2个子集内的最短距离，其中最小者作为最短距离d
if(best.dis>p3.dis)
{
    next_closest_p=best;//用次最近点对保留替换前的最近点对;
    best=p3;
}
else
    next_closest_p=p3;//用次最近点对保留替换前的最近点对;

merge_sort_y(xx,yy,l,r);//重构Y[]: 合并子段Z[L:m]和Z[m+1:r],结果存放在Y[],Y[]中各点按照y坐标排序;

//将Y[]中位于P1+P2区域, 即分割线l:x=m两端d=best.dist距离内的各点拷贝至zz[]
int k=l;
for(int i=l;i<=r;i++)
    if(fabs(yy[m].x-yy[i].x)<best.dis)
        zz[k++]=yy[i];
k--;
//依次搜索位于P1+P2区域中Z[L,k-1]中各点, 计算各点与P1、P2中dx2d矩形范围内的其它点间距离, 据此计算全局最近点对best

for(int i=l;i<k;i++)

```

```

        for(int j=i+1;j<=k&&zz[j].y-zz[i].y<best.dis;j++)
        {
            Type d=distance_point(zz[j],zz[i]);
            if(d<best.dis)//存在距离更小的点对, 更新最小点对best;
            {
                next_closest_p=best;
                best.a=zz[j];
                best.b=zz[i];
                best.dis=d;//用次最近点对保留替换前的最近点对;
            }
        }
    }
    return best;
}

void closest_pair::read_file(string file_name)//从文件中读取1033个基站数据
{
    ifstream fin;
    cout<<"读取文件"<<endl;
    fin.open(file_name.c_str(),ios_base::in);
    if(!fin)
        cout<<"文件打开失败"<<endl;
    int num;
    fin>>num;
    for(int i=0;i<num;i++)
    {
        fin>>xx[i].id>>setprecision(8)>>xx[i].x>>xx[i].y>>xx[i].k_dist;
    }
    fin.close();
}

void closest_pair::merge_sort_x(int low,int high)//依据x坐标排序
{
    if(low<high)
    {
        int mid=(low+high)/2;
        merge_sort_x(low,mid);
        merge_sort_x(mid+1,high);
        merge_x(low,mid,high);
    }
}
//归并tmp[low..mid]和tmp[mid+1..high]到xx[low..high]
void closest_pair::merge_x(int low,int mid,int high)
{
    int i,j,k;
    for(i=low,j=mid+1,k=low;i<=mid&&j<=high;k++)
        if(xx[i].x<=xx[j].x)
            tmp[k]=xx[i++];
        else
            tmp[k]=xx[j++];
    while(i<=mid)
        tmp[k++]=xx[i++];
    while(j<=high)
        tmp[k++]=xx[j++];
}

```



```

        for(int i=low;i<=high;i++)
            xx[i]=tmp[i];
    }

void closest_pair::merge_sort_y(point a[],point b[],int low,int high)//依据y坐标排序
{
    if(low==high)
        b[low]=a[low];
    else
    {
        int mid=(low+high)/2;
        point c[MAXSIZE];
        merge_sort_y(a,c,low,mid);
        merge_sort_y(a,c,mid+1,high);
        merge_y(c,b,low,mid,high);
    }
}
//归并b[low..mid]和b[mid+1..high]到c[low..high];
void closest_pair::merge_y(point b[],point c[],int low,int mid,int high)
{
    int i,j,k;
    for(i=low,j=mid+1,k=low;i<=mid&&j<=high;k++)
        if(b[i].y<=b[j].y)
            c[k]=b[i++];
        else
            c[k]=b[j++];
    while(i<=mid)
        c[k++]=b[i++];
    while(j<=high)
        c[k++]=b[j++];
}

Type closest_pair::distance_point(point d1,point d2)//两点间的距离;
{
    double lat1 = d1.x;
    double lat2 = d2.x;
    double lon1 = d1.y;
    double lon2 = d2.y;
    double radLat1 = get_rad(lat1);//点d1纬度对应的弧度
    double radLat2 = get_rad(lat2);//点d2纬度对应的弧度
    double a = fabs(radLat1 - radLat2);//点d1、d2纬度对应的弧度的差值
    double b = fabs(get_rad(lon1) - get_rad(lon2));//点d1、d2经度对应的弧度的差值
    //两点间经纬度距离公式
    double dist = 2 * asin(sqrt(pow(sin(a / 2), 2) + cos(radLat1)*cos(radLat2)*pow(sin(b / 2),
2)));
    dist = dist * R;
    return dist;
}

double closest_pair::get_rad(Type d) //将经纬度转为弧度
{


```

```

        return (d * PI / 180);
    }

    void closest_pair::show_closest_pair()//显示最近点对的经纬度、id、k_dist
    {
        cout<<"最近点对为: "<<endl<<"基站id="<<closest_p.a.id<<setprecision(8)<<" 纬度="
        <<closest_p.a.x<<" 经度="<<closest_p.a.y<<endl<<"基站id="<<closest_p.b.id<<"纬度="
        <<closest_p.b.x<<" 经度="<<closest_p.b.y<<" 距离为: "<<closest_p.dis<<endl;
        cout<<"最近点对为: "<<endl<<"基站id="<<next_closest_p.a.id<<" 纬度="<<setprecision(8)
        <<next_closest_p.a.x<<" 经度="<<next_closest_p.a.y<<endl<<"基站id="<<next_closest_p.b.id<<" 纬度"
        ="<<next_closest_p.b.x<<" 经度="<<next_closest_p.b.y<<" 距离为: "<<next_closest_p.dis<<endl;
    }
}

```

 C:\Users\戴尔\Desktop\算法设计\第一次\最接近点对问题2.exe

读取文件

最接近点对为:

基站id=568471 纬度=102.721 经度=25.0456

基站id=568849 纬度=102.721 经度=25.0456 距离为: 0

次最接近点对为:

基站id=566803 纬度=102.741 经度=25.05394

基站id=567389 纬度=102.741 经度=25.053888 距离为: 5.78234

Process exited after 0.4047 seconds with return value 0

请按任意键继续. . .