算法设计与分析

第二章

姓名：张博康

学号：2014211383

班级：2014211309

一、 源代码

```
#include <iostream>
#include <fstream>
#include <algorithm>
#include <utility>
#include <iomanip>
#include <cstdlib>
#include <cstring>
#include <cmath>

#define MAXSIZE 3000
#define Pi 3.141592657

using namespace std;

typedef struct station
{
    long long enodedId;
    double longitude, latitude;
    double k_dist;
} Station;

Station stations[MAXSIZE]; //储存基站数据的数组
const double R = 6378137.0; //地球半径，以 m 为单位
int n = 0;

//归并排序的合并部分
void Merge(Station a[], int l, int m, int r)
{
    int i = l, j = m + 1, k = l;
    Station temp[MAXSIZE];

    //按 k_dist 从小到大顺序将左右部分合并进 temp 数组
    while (i <= m && j <= r)
        if (a[i].k_dist < a[j].k_dist)
            temp[k++] = a[i++];
        else
            temp[k++] = a[j++];

    //总有一部分先合并完，则将另外一部分剩余元素直接合并进
temp 数组
    if (i > m)
        for (int q = j; q <= r; q++)
```

```
            temp[k++] = a[q];
        else
            for (int q = i; q <= m; q++)
                temp[k++] = a[q];

        //将 temp 数组复制给 a 数组
        for (int q = l; q <= r; q++)
            a[q] = temp[q];
    }

    //归并排序
    void MergeSort(Station a[], int left, int right)
    {
        if (left < right)
        {
            int i = (left + right) >> 1;

            //分别对左右部分进行归并排序
            MergeSort(a, left, i);
            MergeSort(a, i + 1, right);

            //合并两部分
            Merge(a, left, i, right);
        }
    }

    //根据给点值，将数组分为小于和大于两部分
    int Partition(Station a[], int left, int right, Station x)
    {
        int i = left, j = right;
        Station key = x;

        int loc = left;
        //在数组 a 中给定范围内找到 x 所在的位置，并与最左边元素交
换
        while (a[loc].enodedId != x.enodedId)
            loc++;
        swap(a[left], a[loc]);

        //将小于给定值的放在左边，大于给定值的放在右边
        while (i < j)
        {
            while (a[j].k_dist > key.k_dist && i < j)
                j--;
```

```
            if (i < j)
                a[i++] = a[j];

            while (a[i].k_dist < key.k_dist && i < j)
                i++;
            if (i < j)
                a[j--] = a[i];
        }
        a[i] = key;
        return i;
    }

    //快速排序
    void QuickSort(Station a[], int left, int right)
    {
        if (left >= right)
            return;

        int i = Partition(a, left, right, a[left]);
        QuickSort(a, left, i - 1);
        QuickSort(a, i + 1, right);
    }

    //线性时间选择
    Station Select(Station a[], int left, int right, int k)
    {
        //规模小于 75，直接进行排序
        if (right - left < 75)
        {
            QuickSort(a, left, right);
            return a[left + k - 1];
        }

        //5 个为一组，进行排序，并将中值挑选出来
        for  (int i = 0; i <= (right - left - 4)/5; ++i)
        {
            QuickSort(a, left + 5*i, left + 5*i + 4);
            swap(a[left + 5*i + 2], a[left + i]);
        }

        //在挑选的中值中寻找中值的中值
        Station x = Select(a, left, left + (right - left - 4)/5,
(right - left - 4)/10);
```

```
        //根据中值的中值划分左右两部分
        //判断 k 在左部分还是右部分，并进行递归
        int i = Partition(a, left, right, x),
            j = i - left + 1;
        if (k <= j)
            return Select(a, left, i, k);
        else
            return Select(a, i+1, right, k-j);

    }


    //计算两基站间的距离
    double distance(const Station &u, const Station &v)
    {
        double radLat1 = u.latitude * Pi / 180.0;
        double radLat2 = v.latitude * Pi / 180.0;
        double radLon1 = u.longitude * Pi / 180.0;
        double radLon2 = v.longitude * Pi / 180.0;

        return R * acos(cos(radLat1) * cos(radLat2) * cos(radLon1
- radLon2)
            + sin(radLat1) * sin(radLat2));
    }

    //寻找最近点对和次近点对
    //a1、b1 为最近点对，a2、b2 为次近点对
    pair<double, double> Closeset(Station x[], int left, int
right,
                                  int &a1, int &b1, int &a2,
int &b2)
    {
        if (right - left == 1) //2 个点的情形
        {
            a1 = left;
            b1 = right;
            a2 = -1;
            b2 = -1;
            return make_pair(distance(x[a1], x[b1]), 0x7fffffff);
        }

        if (right - left == 2) //3 个点的情形
        {
            //分别获得三种可能的最小距离
```

```
double d1 = distance(x[left], x[left + 1]);
double d2 = distance(x[left + 1], x[right]);
double d3 = distance(x[left], x[right]);

//找到最小距离和次小距离，并更新 a1、b1、a2、b2
if (d1 <= d2 && d2 <= d3)
{
    a1 = left;
    b1 = left + 1;
    a2 = left + 1;
    b2 = right;
    return make_pair(d1, d2);
}
if (d1 <= d3 && d3 <= d2)
{
    a1 = left;
    b1 = left + 1;
    a2 = left;
    b2 = right;
    return make_pair(d1, d3);
}
if (d2 <= d1 && d1 <= d3)
{
    a1 = left + 1;
    b1 = right;
    a2 = left;
    b2 = left + 1;
    return make_pair(d2, d1);
}
if (d2 <= d3 && d3 <= d1)
{
    a1 = left + 1;
    b1 = right;
    a2 = left;
    b2 = right;
    return make_pair(d2, d3);
}
if (d3 <= d1 && d1 <= d2)
{
    a1 = left;
    b1 = right;
    a2 = left;
    b2 = left + 1;
    return make_pair(d3, d1);
```

```
            }
            if (d3 <= d2 && d2 <= d1)
            {
                a1 = left;
                b1 = right;
                a2 = left + 1;
                b2 = right;
                return make_pair(d3, d2);
            }
        }

        //以中点为分割，求出左右部分的最短距离和次短距离
        int mid = (left + right) >> 1;
        int l_a1, l_b1, l_a2, l_b2;
        int r_a1, r_b1, r_a2, r_b2;
        pair<double, double> leftPair = Closeset(x, left, mid,
l_a1, l_b1, l_a2, l_b2);
        pair<double, double> rightPair = Closeset(x, mid + 1,
right, r_a1, r_b1, r_a2, r_b2);

        double d1, d2;
        //比较左右部分的结果，更新 d1、a1、b1
        if (leftPair.first <= rightPair.first)
        {
            d1 = leftPair.first;
            a1 = l_a1;
            b1 = l_b1;
        }
        else
        {
            d1 = rightPair.first;
            a1 = r_a1;
            b1 = r_b1;
        }
        //更新 d2、a2、b2
        if (max(leftPair.first, rightPair.first) <=
min(leftPair.second, rightPair.second))
        {
            if (leftPair.first < rightPair.first)
            {
                d2 = rightPair.first;
                a2 = r_a1;
                b2 = r_b1;
            }
```

```
            else
            {
                d2 = leftPair.first;
                a2 = l_a1;
                b2 = l_b1;
            }
        }
        else if (leftPair.second <= rightPair.second)
        {
            d2 = leftPair.second;
            a2 = l_a2;
            b2 = l_b2;
        }
        else
        {
            d2 = rightPair.second;
            a2 = r_a2;
            b2 = r_b2;
        }

        //找出所有跨界的最接近点对候选者，将其在 x 数组的索引记录在 t 中
        int t[MAXSIZE];
        int k = 0;
        for (int i = left; i <= right; i++)
            if (fabs(x[mid].longitude - x[i].longitude) < d1)
                t[k++] = i;
        //根据纬度对 t 进行排序
        sort(t, t + k, [x](int a, int b){return x[a].latitude < x[a].latitude; });
        //遍历所有可能最接近点对候选者
        for (int i = 0; i < k; i++)
            for (int j = i + 1; j < k && x[t[j]].latitude - x[t[i]].latitude < d1; j++)
            {
                double dp = distance(x[t[i]], x[t[j]]);
                //更新最短距离和次短距离
                if (dp <= d1)
                {
                    d2 = d1;
                    a2 = a1;
                    b2 = b1;
                    d1 = dp;
                    a1 = t[i];
```

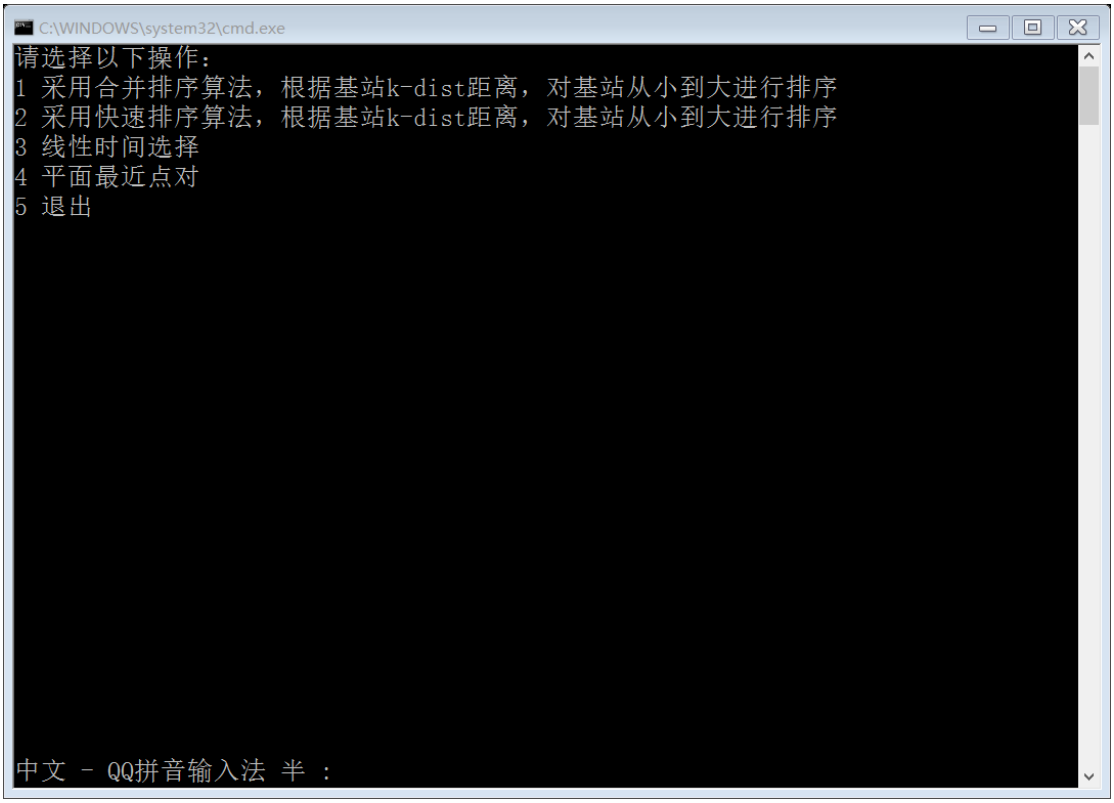
```
                    b1 = t[j];
                }
                else if (dp < d2)
                {
                    d2 = dp;
                    a2 = t[i];
                    b2 = t[j];
                }
            }
        return make_pair(d1, d2);
    }

    int main(int argc, char const *argv[])
    {
        //数据读入
        ifstream in("data.txt", ios_base::in);
        if (!in.is_open())
        {
            cout << "Error opening file..." << endl;
            exit(1);
        }
        while (!in.eof())
        {
            in >> stations[n].enodedId >>
stations[n].longitude >> stations[n].latitude >>
stations[n].k_dist;
            n++;
        }
        n -= 2;

        int choose = 0;
        while (choose != 5)
        {
            cout << "请选择以下操作：" << endl;
            cout << "1 采用合并排序算法，根据基站 k-dist 距离，对
基站从小到大进行排序" << endl;
            cout << "2 采用快速排序算法，根据基站 k-dist 距离，对
基站从小到大进行排序" << endl;
            cout << "3 线性时间选择" << endl;
            cout << "4 平面最近点对" << endl;
            cout << "5 退出" << endl;

            while (cin >> choose, !(choose >= 1 && choose <= 5))
            {
```

```
                cout << "输入不合法，请重新输入" << endl;
                cin.clear();
                cin.sync();
            }
            cout << "----------------------------------------
--------------------" << endl;

            Station OrderStations[MAXSIZE];
            memcpy(OrderStations, stations, sizeof(stations));
            switch (choose)
            {
                case 1:
                    MergeSort(OrderStations, 0, n);
                    for (int i = 0; i <= n; ++i)
                    {
                        cout << OrderStations[i].enodedId << '\t'
<< OrderStations[i].longitude << '\t' <<
OrderStations[i].latitude
                            << '\t' << OrderStations[i].k_dist <<
endl;
                    }
                    cout << "-------------------------输出完成--
-------------------------" << endl;
                    break;
                case 2:
                    QuickSort(OrderStations, 0, n);
                    for (int i = 0; i <= n; ++i)
                    {
                        cout << OrderStations[i].enodedId << '\t'
<< OrderStations[i].longitude << '\t' <<
OrderStations[i].latitude
                            << '\t' << OrderStations[i].k_dist <<
endl;
                    }
                    cout << "-------------------------输出完成--
---------------- ----------" << endl;
                    break;
                case 3:
                    Station temp;
                    temp = Select(OrderStations, 0, n, 1);
                    cout << "k_dist 值最小的基站： " << '\t' <<
temp.enodedId << '\t' << temp.longitude << '\t' << temp.latitude
<< '\t' << temp.k_dist << endl;
                    temp = Select(OrderStations, 0, n, 5);
```

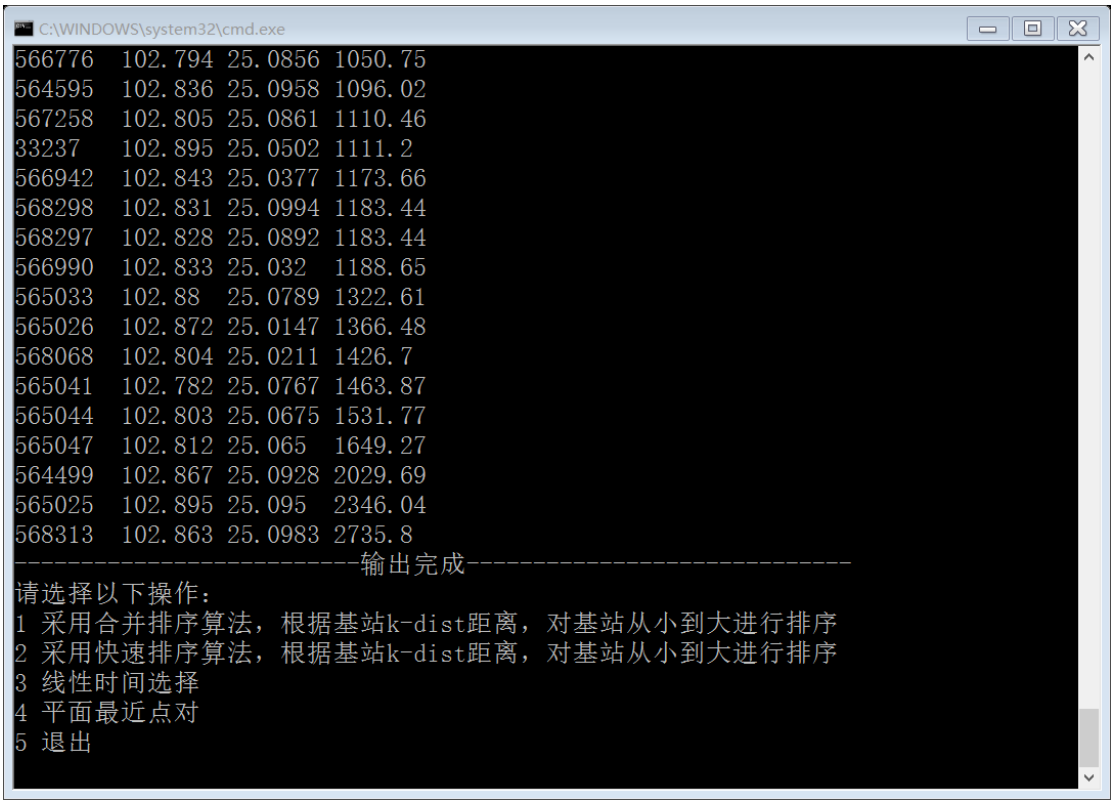
```
                    cout << "k_dist 值第 5 小的基站: " << '\t' <<
temp.enodedId << '\t' << temp.longitude << '\t' << temp.latitude
<< '\t' << temp.k_dist << endl;
                    temp = Select(OrderStations, 0, n, 50);
                    cout << "k_dist 值第 50 小的基站: " << '\t' <<
temp.enodedId << '\t' << temp.longitude << '\t' << temp.latitude
<< '\t' << temp.k_dist << endl;
                    temp = Select(OrderStations, 0, n, n + 1);
                    cout << "k_dist 值最大的基站: " << '\t' <<
temp.enodedId << '\t' << temp.longitude << '\t' << temp.latitude
<< '\t' << temp.k_dist << endl;
                    cout << "-------------------------输出完成--
-------------------------" << endl;
                    break;
                case 4:
                {
                    sort(OrderStations, OrderStations + n + 1,
[](Station a, Station b){return a.longitude < b.longitude; });
                    int a1, b1, a2, b2;
                    pair<double, double> res =
Closeset(OrderStations, 0, n, a1, b1, a2, b2);
                    cout << "距离最近的 2 个基站: " <<
OrderStations[a1].enodedId << " 和 " <<
OrderStations[b1].enodedId <<
                        "\t 距离为: " << fixed << res.first <<
endl;
                    cout << "距离次近的 2 个基站: " <<
OrderStations[a2].enodedId << " 和 " <<
OrderStations[b2].enodedId <<
                        "\t 距离为: " << fixed << res.second <<
endl;
                    cout << "-------------------------输出完成--
-------------------------" << endl;
                    break;
                }
                case 5:
                    exit(0);
                    break;
            }
        }

        return 0;
}
```

二、 运行结果

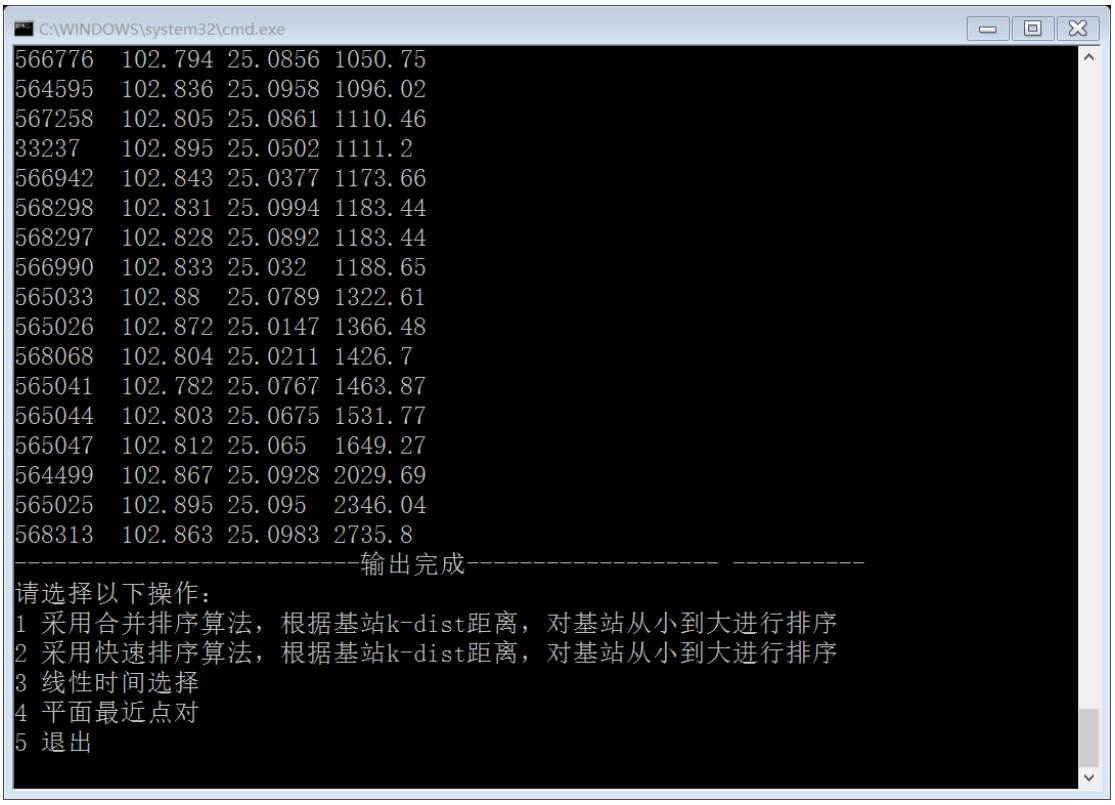
1. 开始界面（输入 1-5，选择相应操作）



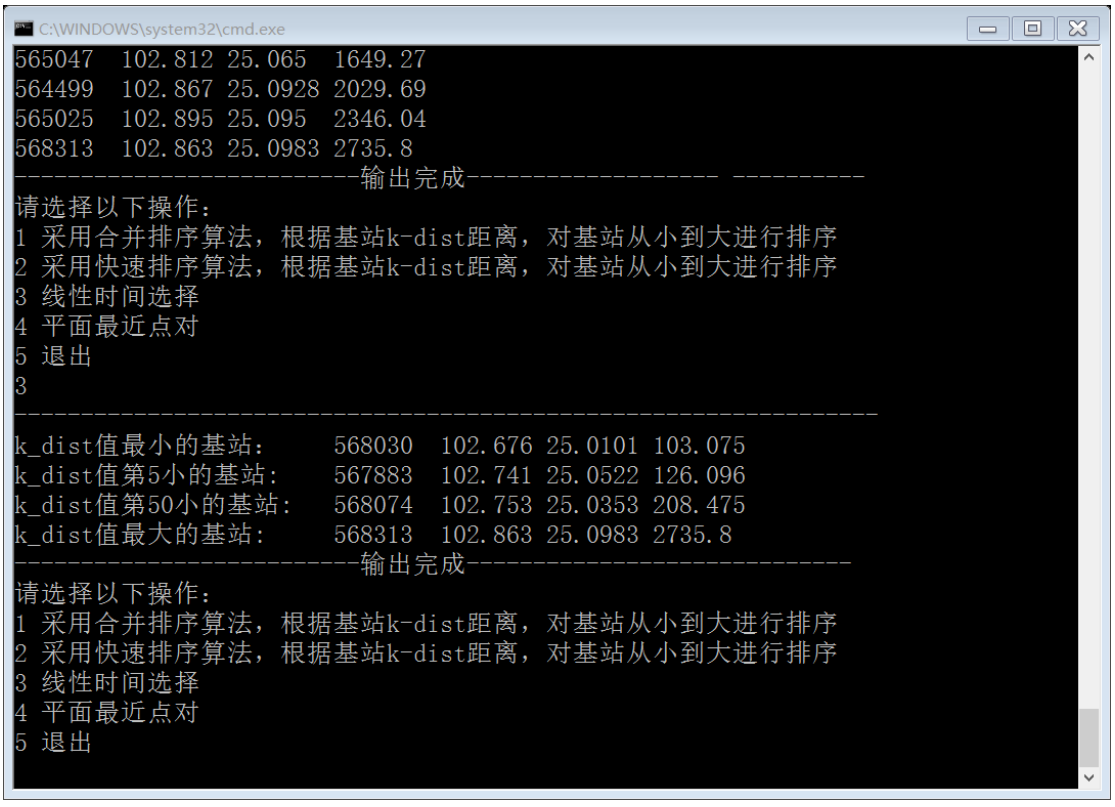
2. （输入 1）采用合并排序算法，根据基站 k-dist 距离，对基站从小到大进行排序（输出太多，只截取了最后部分）



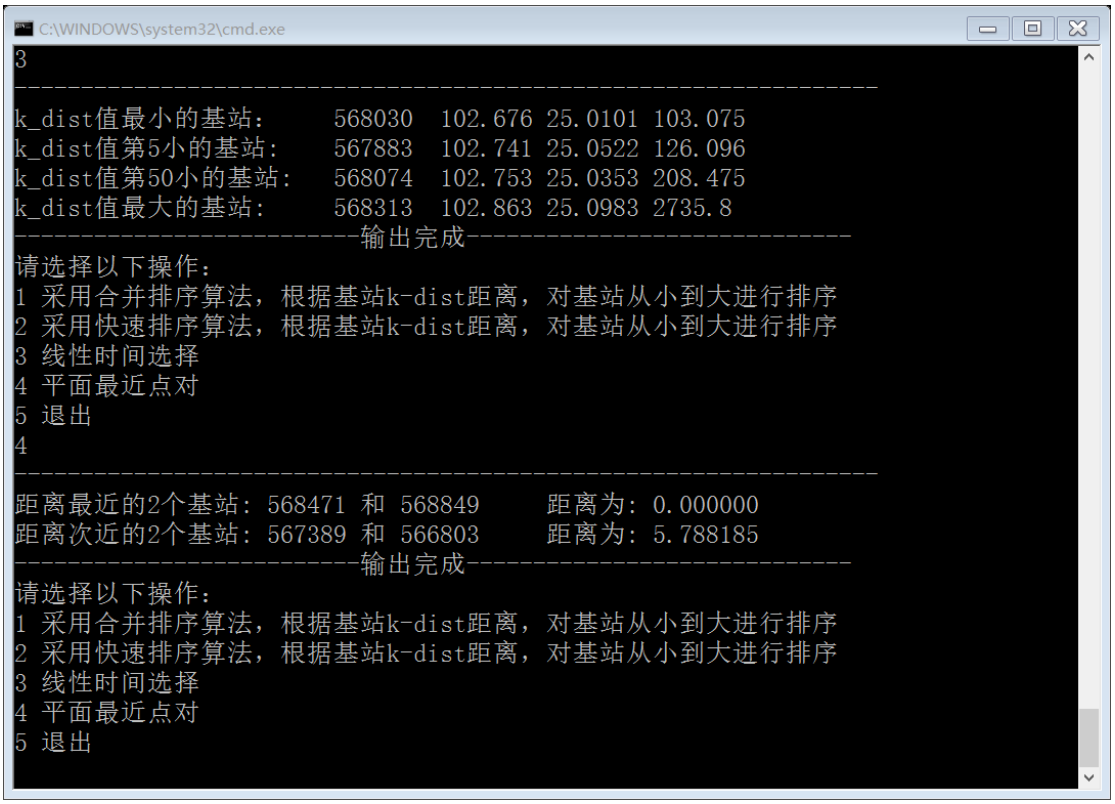
3. （输入 2）采用快速排序算法，根据基站 k-dist 距离，对基站从小到大进行排序（输出太多，只截取了最后部分）



4. （输入 3）线性时间选择



5. （输入 4）平面最近点对



6. （输入 5）退出