一些说明：

1. 基于贪心法的凸多边形三角剖分使用算法的思想是每次取得边界上所形成三角形的最小，不断删除点。例如，五边形中，取 v0v1v2, v1v2v3, ⋯, v5v1v2 三角形中最小，假设是 2，则下一次删去 v3，从 v0v1v3, v1v3v4, ⋯, v5v1v3 中取最小。
2. 与上一次作业的动态规划凸多边形最优三角剖分相比较，贪心算法的弦长和不为最优（20 边形最优 178998（上次作业未去重复点，去除后结果为这个），贪心算法为 179280；28 边形最优（同前）为 109805，贪心算法为 110321），但时间上快了很多（动态规划两个多边形总耗时 0.073s，贪心算法总耗时 0.01s）。
3. 哈夫曼编码中，最后一段文字存在 Chv´atal's，即 á 占用两个 char(-95 和-28)，在进行哈夫曼编码的时候将其去掉，并且将其他符号（例如 '-'）也转为#。

注：为了方便读取，将xls文件改为csv

注：所有题目均在一个cpp文件下，主函数中依次进行各个算法

代码：

```
// Algorithm4.cpp : Designed by Xiao Yunming.
//

#include "stdafx.h" // VS projects head file

#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <map>
#include <algorithm>
#include <cmath>
#include <functional>
#include <time.h>

#define EARTH_RADIUS 6378.137
#define M_PI 3.14159265358979323846

using namespace std;

// ------------------- GREEDY_MIN_WERIGHT_TRAIAN ---------------------------

class BaseStation
{
public:
    BaseStation(int enodebid, float longitude, float latitude);
    BaseStation(string line);
    BaseStation(const BaseStation &b);

    float longitude, latitude;
    int enodebid;
};

BaseStation::BaseStation(int enodebid, float longitude, float latitude)
{
    this->enodebid = enodebid;
    this->longitude = longitude;
    this->latitude = latitude;
}

BaseStation::BaseStation(string line)
{
    size_t pos = 0;
    size_t len = line.length();
    size_t delim_len = 1;
    // to devide the line by ','
    vector<string> s;
    while (pos < len) {
        int find_pos = line.find(',', pos);
        if (find_pos < 0) {
            s.push_back(line.substr(pos, len - pos));
            break;
        }
        s.push_back(line.substr(pos, find_pos - pos));
        pos = find_pos + delim_len;
    }

    this->enodebid = stoi(s[0]);
    this->longitude = stod(s[1]);
    this->latitude = stod(s[2]);
}

BaseStation::BaseStation(const BaseStation &b)
{
    this->enodebid = b.enodebid;
    this->longitude = b.longitude;
    this->latitude = b.latitude;
}

double GetDistance(BaseStation A, BaseStation B)
{
    auto rad = [](const double& f) {return f * M_PI / 180.0; };
    double radLatA = rad(A.latitude), radLatB = rad(B.latitude);
    double radLonA = rad(A.longitude), radLonB = rad(B.longitude);
    double s = 1000 * EARTH_RADIUS * acos(cos(radLatA) * cos(radLatB) * cos(radLonA - radLonB) + 
sin(radLatA) * sin(radLatB));
    return s;
}

double Weight(vector<BaseStation> B, int a, int b, int c)
{
    double ab = GetDistance(B[a], B[b]);
    double ac = GetDistance(B[a], B[c]);
    double bc = GetDistance(B[b], B[c]);
    return ab + ac + bc;
}

void GreedyMinWerightTriangulation(double **t, int **s, vector<BaseStation> B, double &total_length)
{
    int n = B.size();
    for (int i = 0; i < n; i++)
        t[i][i] = 0;

    function<int(vector<BaseStation>&, double&)>min;
    min = [](vector<BaseStation>& b, double &total_length)
    {
        int max = b.size();
        int min_weight = Weight(b, 0, 1, 2);
        int prop = 1;
        for (int i = 1; i < max; i++) {
            int new_weight = Weight(b, i, (i + 1) % max, (i + 2) % max);
            if (new_weight < min_weight) {
                prop = (i + 1) % max;
                min_weight = new_weight;
            }
        }
        total_length += min_weight;
        return prop % max;
    };

    vector<BaseStation> S = B;
    vector<int> alive;
    for (int i = 0; i < n; i++)
        alive.push_back(1);

    while (S.size() >= 3){
        int x = min(S, total_length), y = x; // x is the relative position of the target point
                                             // where y is the absolute position
        for (int j = 0; j <= y; j++)
```

```
            if (alive[j] == 0)
                y++;
        vector<BaseStation>::iterator ss = S.begin() + x;
        S.erase(ss);

        alive[y] = 0;
        int begin = y, end = y;   // begin and end are both the absolute position starting from 0
        while (alive[begin] == 0) {
            begin = (begin - 1) % n;
            while (begin < 0)
                begin += n; // to make sure that begin is a positive number
        }
        while (alive[end] == 0)
            end = (end + 1) % n;

        int first, second, third;
        first = (x - 1) % (int)S.size();
        while (first < 0)
            first += S.size();
        second = (first + 1) % (int)S.size();
        third = (first + 2) % (int)S.size();
        t[begin][end] = Weight(S, first, second, third);
        s[begin][end] = y;
    }
}

// -------------------------- HUFFMAN_TREE ----------------------------------

#define LEAF 1
#define INTER_NODE 0

class HuffmanTree
{
public:
    double weight;
    HuffmanTree *left, *right;
    char child;

    HuffmanTree() { left = NULL; right = NULL; weight = 0.0; child = '\0'; };
    HuffmanTree(HuffmanTree left, HuffmanTree right, double weight);
    HuffmanTree(char child, double weight);
    HuffmanTree(const HuffmanTree &t);

    bool operator<(HuffmanTree &t) { return this->weight < t.weight; };
    operator int() { return left == NULL ? LEAF : INTER_NODE; };

    HuffmanTree(string s);    // the algorithm realization

    void Show(ofstream &fr);
    void ShowCode(ofstream &fr, string code);
    void HuffmanCode(map<char, string> &m, string code);
};

HuffmanTree::HuffmanTree(HuffmanTree left, HuffmanTree right, double weight)
{
    this->left = new HuffmanTree(left);
    this->right = new HuffmanTree(right);
    this->weight = weight;
    this->child = '\0';
}

HuffmanTree::HuffmanTree(char child, double weight)
{
    this->left = NULL;
    this->right = NULL;
    this->child = child;
    this->weight = weight;
}
```

```cpp
HuffmanTree::HuffmanTree(const HuffmanTree &t)
{
    this->left = t.left == NULL ? NULL : new HuffmanTree(*(t.left));
    this->right = t.right == NULL ? NULL : new HuffmanTree(*(t.right));
    this->weight = t.weight;
    this->child = t.child;
}

// the algorithm realization
HuffmanTree::HuffmanTree(string s)
{
    map<char, int>stat;
    for (int i = 0; i < s.length(); i++) {
        auto it = stat.find(s[i]);
        if (it == stat.end())
            stat.insert(map<char, int>::value_type(s[i], 1));
        else
            it->second += 1;
    }
    vector<HuffmanTree> result;
    auto it = stat.begin();
    while (it != stat.end()) {
        result.push_back(HuffmanTree(it->first, it->second));
        it++;
    }
    sort(result.begin(), result.end());

    while (result.size() > 1) {
        HuffmanTree N(result[0], result[1], result[0].weight + result[1].weight);
        result.erase(result.begin(), result.begin() + 2);
        result.push_back(N);
        sort(result.begin(), result.end());
    }

    *this = result[0];
}

void HuffmanTree::Show(ofstream &fr)
{
    if ((int)*this == INTER_NODE) {
        this->left->Show(fr);
        this->right->Show(fr);
    }
    else {
        fr << this->child << ":" << this->weight << endl;
    }
}

void HuffmanTree::ShowCode(ofstream &fr, string code)
{
    if ((int)*this == INTER_NODE) {
        this->left->ShowCode(fr, code + "0");
        this->right->ShowCode(fr, code + "1");
    }
    else {
        fr << this->child << ": weight = " << this->weight << ", code = " << code << endl;
    }
}

void HuffmanTree::HuffmanCode(map<char, string> &m, string code)
{
    if ((int)*this == INTER_NODE) {
        this->left->HuffmanCode(m, code + "0");
        this->right->HuffmanCode(m, code + "1");
    }
    else {
        m.insert(map<char, string>::value_type(this->child, code));
    }
}

// ---------------------------- GRAPH ------------------------------------

#define INF -1
#define FAIL -1

void Dijkstra(int v, vector<vector<double>> a, vector<double> &dist, vector<int> &prev)
{
    int n = a.size();
    if (v < 0 || v >= n)
        return;
    bool *s = new bool[n];

    dist.clear();
    prev.clear();
    for (int i = 0; i < n; i++) {
        dist.push_back(a[v][i]);
        s[i] = false;
        if (dist[i] == INF)
            prev.push_back(FAIL);
        else
            prev.push_back(v);
    }
    dist[v] = 0;
    s[v] = true;

    for (int i = 0; i < n; i++) {
        double temp = INF;
        int u = v;
        for (int j = 0; j < n; j++) {
            bool judge = (dist[j] > 0) && ((temp == INF) || (temp != INF && dist[j] < temp)); // that
dist[j] is smaller than temp and the infinity
            if ((s[j] == false) && (judge == true)) {
                u = j;
                temp = dist[j];
            }
        }
        s[u] = true;

        for (int j = 0; j < n; j++) {
            if ((s[j] == false) && (a[u][j] != INF)) {
                double new_dist = dist[u] + a[u][j];
                if ((dist[j] != INF && new_dist < dist[j]) || dist[j] == INF) {
                    dist[j] = new_dist;
                    prev[j] = u;
                }
            }
        }
    }
}

void Prim(vector<vector<double>> c, vector<pair<int, int>> &result)
{
    int n = c.size();
    double *low_cost = new double[n];
    int *closest = new int[n];
    bool *s = new bool[n];
    result.clear();

    s[0] = true;
    result.push_back(make_pair(0,0));
    for (int i = 1; i < n; i++) {
        low_cost[i] = c[0][i];
        closest[i] = 0;
        s[i] = false;
    }
    for (int i = 0; i < n; i++) {
        double min = INF;
        int j = 0;
        for (int k = 1; k < n; k++) {
            bool judge = (low_cost[k] > 0) && ((min == INF) || (low_cost[k] < min));
            if ((judge == true) && s[k] == false) {
                min = low_cost[k];
                j = k;
            }
        }

        s[j] = true;
        result.push_back(make_pair(closest[j], j));

        for (int k = 1; k < n; k++) {
            bool judge = (c[j][k] != INF) && ((low_cost[k] == INF) || (c[j][k] < low_cost[k]));
            if ((judge == true) && (s[k] == false)) {
                low_cost[k] = c[j][k];
                closest[k] = j;
            }
        }
    }
}

// ------------------------------ MAIN -------------------------------------

int main()
{
    ofstream fr("Result.txt");
    clock_t start, end;
    double duration;
    string temp;

    // GREEDY_MIN_WERIGHT_TRAIAN
    // init
    fstream f11("附件3-1.21个基站凸多边形数据2017.csv", ios::in | ios::out);
    fstream f12("附件3-2.29个基站凸多边形数据2017.csv", ios::in | ios::out);
    vector<BaseStation> b11, b12;
    getline(f11, temp);       getline(f12, temp); // file header
    while (getline(f11, temp) && f11.good())
        b11.push_back(BaseStation(temp));
    while (getline(f12, temp) && f12.good())
        b12.push_back(BaseStation(temp));
    f11.close();
    f12.close();

    double **result11_t = new double*[b11.size()], **result12_t = new double*[b12.size()];
    int **result11_s = new int*[b11.size()], **result12_s = new int*[b12.size()];

    for (int i = 0; i < b11.size(); i++) {
        result11_t[i] = new double[b11.size()];
        result11_s[i] = new int[b11.size()];
    }
    for (int i = 0; i < b12.size(); i++) {
        result12_t[i] = new double[b12.size()];
        result12_s[i] = new int[b12.size()];
    }
    for (int i = 0; i < b11.size(); i++) {
        for (int j = 0; j < b11.size(); j++) {
            result11_s[i][j] = -1;
            result11_t[i][j] = -1;
        }
    }
    for (int i = 0; i < b12.size(); i++) {
        for (int j = 0; j < b12.size(); j++) {
            result12_s[i][j] = -1;
            result12_t[i][j] = -1;
        }
    }
    double result11_w = 0, result12_w = 0;

    // the algorithm
    start = clock();
    GreedyMinWerightTriangulation(result11_t, result11_s, b11, result11_w);
    GreedyMinWerightTriangulation(result12_t, result12_s, b12, result12_w);
    end = clock();
    duration = (double)(end - start);

    // output
    fr << "1.凸多边形最优三角部分(贪心算法):" << endl;
    fr << "20凸多边形：(每个三维序列表示一个三角剖分的三个顶点)" << endl;
    for (int i = 0; i < b11.size(); i++) {
        for (int j = 0; j < b11.size(); j++) {
            if (result11_s[i][j] != -1) {
                fr << "(" << i << ", " << result11_s[i][j] << ", " << j << "), weight = " <<
result11_t[i][j] << endl;
                //result11_w += result11_t[i][j];
            }
        }
    }
    cout << result11_w << ", " << result12_w << endl;
    auto it1 = b11.begin();
    while ((it1 + 1) != b11.end()) {
        result11_w += GetDistance(*it1, *(it1 + 1));
        it1++;
    }
    result11_w += GetDistance(b11[0], *it1);
    result11_w = result11_w / 2;
    fr << "最小边长弦长和为：" << result11_w << endl << endl;

    fr << "28凸多边形：(每个三维序列表示一个三角剖分的三个顶点)" << endl;
    for (int i = 0; i < b12.size(); i++) {
        for (int j = 0; j < b12.size(); j++) {
            if (result12_s[i][j] != -1) {
                fr << "(" << i << ", " << result12_s[i][j] << ", " << j << "), weight = " <<
result12_t[i][j] << endl;
                //result12_w += result12_t[i][j];
            }
        }
    }
    auto it2 = b12.begin();
    while ((it2 + 1) != b12.end()) {
        result12_w += GetDistance(*it2, *(it2 + 1));
        it2++;
    }
    result12_w += GetDistance(b12[0], *it2);
    result12_w = result12_w / 2;
    fr << "最小边长弦长和为：" << result12_w << endl << endl;
    fr << "耗时" << duration / CLOCKS_PER_SEC << "秒" << endl;
    fr << endl << endl;

    // HUFFMAN_TREE
    fstream f21("附件2.哈夫曼编码输入文本(1).txt", ios::in | ios::out);
    string s21 = "";
    while (getline(f21, temp) && f21.good()) {
        s21 += temp;
    }
    // algorithm
    start = clock();
    HuffmanTree result21(s21);
    end = clock();
    duration = (double)(end - start);

    // output
    fr << "2.哈夫曼编码:" << endl;
    fr << "哈夫曼编码结果：" << endl;
    result21.ShowCode(fr, "");
    map<char, string> m21;
    result21.HuffmanCode(m21, "");

    int len_huffman21 = 0;
    fr << endl << "文本的哈夫曼编码：" << endl;
    for (int i = 0; i < s21.length(); i++) {
        fr << m21[s21[i]];
        len_huffman21 += m21[s21[i]].length();
    }
    fr << endl << endl << "哈夫曼编码长度:" << len_huffman21 << endl;
    fr << "定长编码长度:" << ((int)log2(27) + 1) * s21.length() << endl;

    fr << "耗时" << duration / CLOCKS_PER_SEC << "秒" << endl;
    fr << endl << endl;

    // GRAPH
    fstream f31("附件1-1.22基站图的邻接矩阵-v1.csv", ios::in | ios::out);
    fstream f32("附件1-1.42基站图的邻接矩阵-v1.csv", ios::in | ios::out);
    vector<vector<double>> c31, c32;
    map<int, int> m31, m32;    // the order number (starting from 0) mapping to the enodebid

    std::function<vector<double>(string, string)> StringParse; // to parse the long line to pieces with
"delim"
    StringParse = [](string s, string delim)
    {
        vector<string> str;
        vector<double>result_int;
        size_t pos = 0;
        size_t len = s.length();
        size_t delim_len = delim.length();
        while (pos < len) {
            int find_pos = s.find(delim, pos);
            if (find_pos < 0) {
                str.push_back(s.substr(pos, len - pos));
                break;
            }
            str.push_back(s.substr(pos, find_pos - pos));
            pos = find_pos + delim_len;
        }

        vector<string>::iterator it = str.begin();
        while (it != str.end()) {
            it->erase(0, it->find_first_not_of(' '));
            result_int.push_back(stod(*it));
            it++;
        }
        return result_int;
    };

    getline(f31, temp);      getline(f31, temp);
    while (getline(f31, temp) && f31.good()) {
        vector<double> x = StringParse(temp, ",");
        m31.insert(map<int, int>::value_type(((int)x[0] - 1), (int)x[1]));
        x.erase(x.begin(), x.begin() + 2);
        c31.push_back(x);
    }
    getline(f32, temp);      getline(f32, temp);
    while (getline(f32, temp) && f32.good()) {
        vector<double> x = StringParse(temp, ",");
        m32.insert(map<int, int>::value_type(((int)x[0] - 1), (int)x[1]));
        x.erase(x.begin(), x.begin() + 2);
        c32.push_back(x);
    }
```

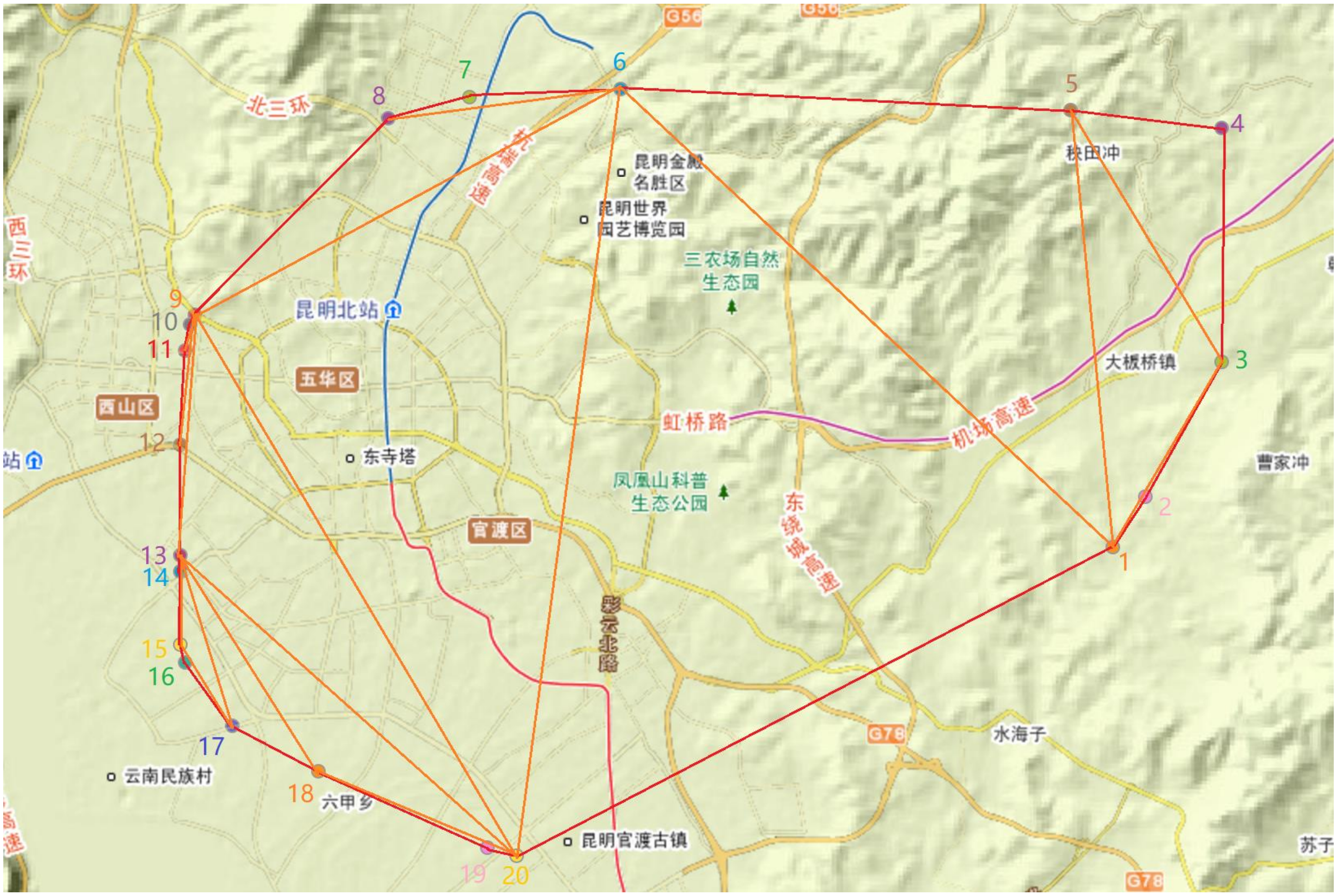
```
vector<double> dist31, dist32;
vector<int> prev31, prev32;   // result of Dijsktra
vector<pair<int,int>> result31, result32;  // result of Prim
int tar31 = 20 - 1, tar32 = 16 - 1;

// algorithm
start = clock();
Dijkstra(tar31, c31, dist31, prev31);
Dijkstra(tar32, c32, dist32, prev32);
Prim(c31, result31);
Prim(c32, result32);
end = clock();
duration = (double)(end - start);

// output
fr << "22基站单源最短路径：" << endl;
for (int i = 0; i < dist31.size(); i++) {
    double t = dist31[i];
    int s = prev31[i];
    fr << "基站" << tar31+1 << "(" << m31[tar31] << ")";
    fr << "至" << i + 1 << "(" << m31[i] << ")";
    fr << ":" << i + 1 << "(" << m31[i] << ") <- ";
    while (s != tar31 && s != FAIL) {
        fr << s + 1 << "(" << m31[s] << ") <- ";
        s = prev31[s];
    }
    fr << tar31+1 << "(" << m31[tar31] << ")" << endl;
    fr << "长度为:" << t << endl;
}
fr << endl;

fr << "42基站单源最短路径：" << endl;
for (int i = 0; i < dist32.size(); i++) {
    double t = dist32[i];
    int s = prev32[i];
    fr << "基站" << tar32+1 << "(" << m32[tar32] << ")";
    fr << "至" << i + 1 << "(" << m32[i] << ")";
    fr << ":" << i + 1 << "(" << m32[i] << ") <- ";
    while (s != tar32 && s != FAIL) {
        fr << s + 1 << "(" << m32[s] << ") <- ";
        s = prev32[s];
    }
    fr << tar32+1 << "(" << m32[tar32] << ")" << endl;
    fr << "长度为:" << t << endl;
}
fr << endl;


auto it31 = result31.begin(), it32 = result32.begin();
fr << "22基站最小生成树:" << endl;
while (++it31 != result31.end() - 1) {
    fr << (it31->first + 1) << "(" << m31[it31->first] << ")";
    fr << " -> " << (it31->second + 1) << "(" << m31[it31->second] << ")" << endl;
}
fr << endl;

fr << "42基站最小生成树:" << endl;
while (++it32 != result32.end() - 1) {
    fr << (it32->first + 1) << "(" << m32[it32->first] << ")";
    fr << " -> " << (it32->second + 1) << "(" << m32[it32->second] << ")" << endl;
}
fr << endl;
fr << "耗时" << duration / CLOCKS_PER_SEC << "秒" << endl;
fr << endl << endl;

fr.close();
```

}

结果部分：

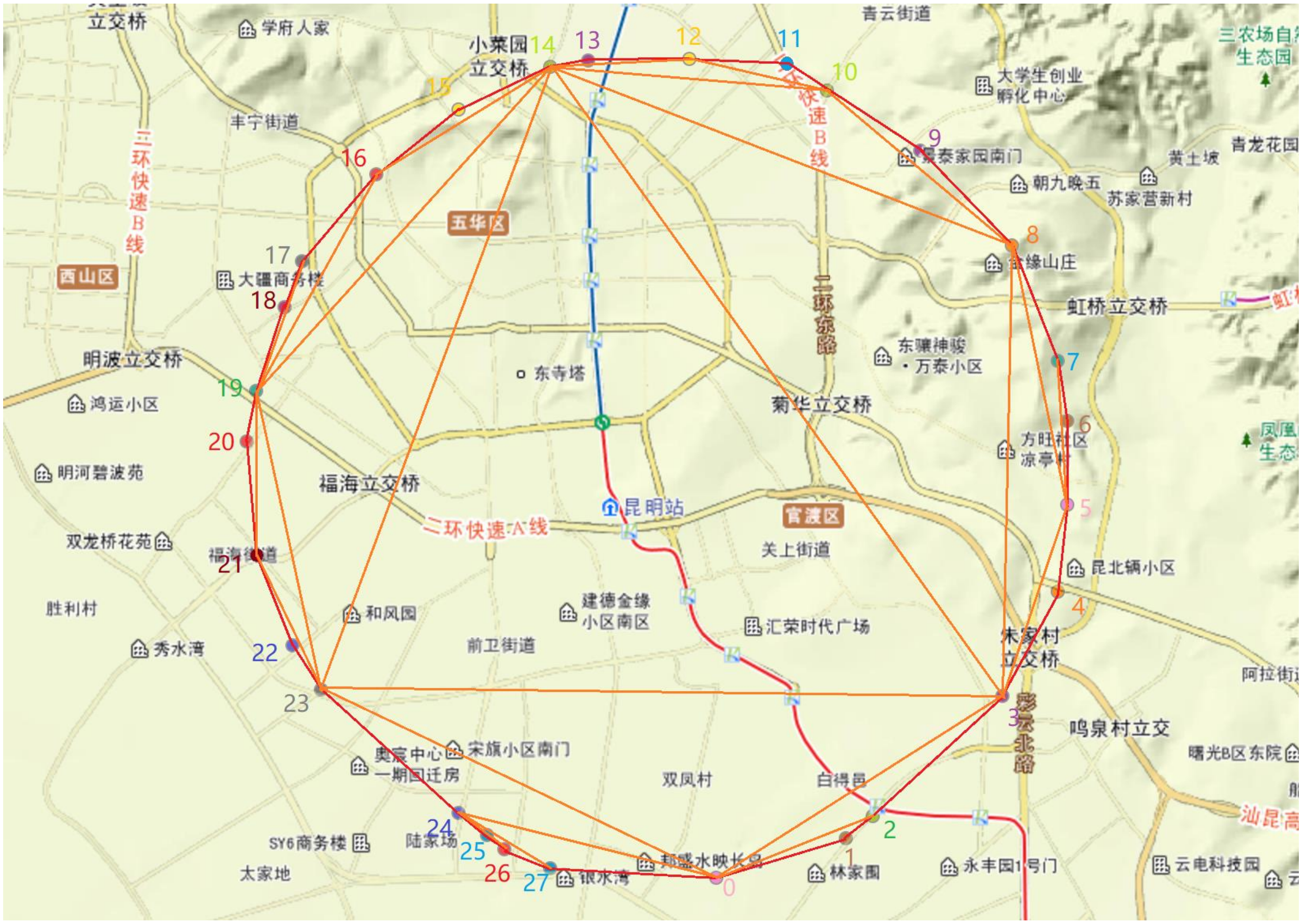
1.凸多边形最优三角部分(贪心算法):

```
20 凸多边形：(每个三维序列表示一个三角剖分的三个顶点)
(0, 1, 6), weight = 33012.1
(1, 2, 3), weight = 18812.3
(1, 3, 5), weight = 33277.8
(1, 5, 6), weight = 44794.5
(3, 4, 5), weight = 29904.4
(6, 9, 0), weight = 45158
(6, 7, 8), weight = 21147.7
(6, 8, 9), weight = 29054.2
(9, 13, 0), weight = 47909.4
(9, 10, 11), weight = 5535.27
(9, 11, 12), weight = 10231.2
(9, 12, 13), weight = 17732
(13, 18, 0), weight = 43812
(13, 14, 15), weight = 4623.8
(13, 15, 17), weight = 11323.2
(13, 17, 18), weight = 19686.1
(15, 16, 17), weight = 8084.9
(18, 19, 0), weight = 36538.3
最小边长弦长和为：179280
```



```
28 凸多边形：(每个三维序列表示一个三角剖分的三个顶点)
(0, 1, 2), weight = 7484.54
(0, 2, 3), weight = 11529.2
(3, 4, 5), weight = 7505.33
```

```
(3, 5, 8), weight = 14471.6
(3, 8, 14), weight = 23205.4
(3, 14, 23), weight = 14931.8
(5, 6, 7), weight = 5840.49
(5, 7, 8), weight = 10767.5
(8, 9, 10), weight = 8258.9
(8, 10, 14), weight = 18620.2
(10, 11, 12), weight = 6103.9
(10, 12, 14), weight = 10308.8
(12, 13, 14), weight = 5208.42
(14, 15, 16), weight = 6955.19
(14, 16, 19), weight = 15402.6
(14, 19, 23), weight = 21128.2
(16, 17, 19), weight = 8858.39
(17, 18, 19), weight = 4108.4
(19, 20, 21), weight = 6776.61
(19, 21, 23), weight = 10498.9
(21, 22, 23), weight = 7266.19
(23, 24, 0), weight = 15977.6
(23, 0, 3), weight = 21400.5
(24, 27, 0), weight = 9294.6
(24, 25, 26), weight = 2364.46
(24, 26, 27), weight = 5909.33
最小边长弦长和为：110321
```



耗时 0.01 秒

2.哈夫曼编码:

```
哈夫曼编码结果：
e: weight = 185, code = 000
n: weight = 91, code = 0010
l: weight = 98, code = 0011
y: weight = 23, code = 010000
T: weight = 5, code = 01000100
z: weight = 3, code = 010001010
C: weight = 3, code = 010001011
W: weight = 3, code = 010001100
q: weight = 3, code = 010001101
k: weight = 7, code = 01000111
p: weight = 58, code = 01001
s: weight = 107, code = 0101
r: weight = 113, code = 0110
c: weight = 62, code = 01110
f: weight = 31, code = 011110
v: weight = 16, code = 0111110
j: weight = 1, code = 0111111000
D: weight = 1, code = 01111110010
F: weight = 1, code = 01111110011
x: weight = 4, code = 011111101
I: weight = 1, code = 01111111000
L: weight = 1, code = 01111111001
M: weight = 1, code = 01111111010
O: weight = 1, code = 01111111011
A: weight = 5, code = 011111111
i: weight = 133, code = 1000
a: weight = 134, code = 1001
o: weight = 139, code = 1010
m: weight = 72, code = 10110
d: weight = 36, code = 101110
b: weight = 19, code = 1011110
w: weight = 22, code = 1011111
t: weight = 167, code = 1100
h: weight = 87, code = 11010
g: weight = 45, code = 110110
u: weight = 45, code = 110111
#: weight = 372, code = 111

文本的哈夫曼编码：
01111111100000100111101010011010110100100110011010000111111100100101111001001111011010100110100011001101010101101111
00001011111001001001000011110111111000001100111011100000111101000001000010111011101110101010110010011101111100100 1
110010001010001010010011111010010110101001110000101110110111011000011111001101010011100111110010010100011100000101
111010110101011000011101111110100100111101110001111111010011011101010001100111101001111011101111101001001111011100
0010111111110010101111110000010010011101111110011110010010101110111010010110101010111011011101110000010111101011010
101100001110111111010010011110111000111111101001101110101000110011110100111101110111110100100111101110000101111111
1001010111110101101111100010011101111100111111011111111001011110010011110110101001101000110011010101011011110000101
111110011010110111010111110011110101000010001101110111000001001110000111101001111011101110101010110010011101111110
010011100100010100010100100111110101110000000100101011111100110101001110011111000110100100100101011110101001101011
011111001101000011110000001001001110111111001111000000101100101011111100110100001111010110111110001001110111110011101
000110000011101110100100101111001001101011010111011111010000000101111111110010010111100100111101101010011010001100
110101011011110010101111100111111001010101000111110111101010011011101011010001101111101000000101101101111001111101
1111000001100111110101010010000111010000111101000000010111011101110101010110010011101111110010011100100010100010100
100111110100101101010101111000110001011011111101000100110100001110101110010011100000101100000001011001111010011110
11111001101000011101001011010101011111000110001011011101010100100001110100001111010000000010111111000001011111011000
0001000001101001001111111000000110101100101111110011010000111101110000010110000110000101110111100000100100111 0111
110011110101101111100010011101111110011101100000011100111001000101000100101110101000010011111110100010011010000111
10010011110110101001101000110011010101011011110111000001010111001101000101111000000101111100111101010100100001110100
```

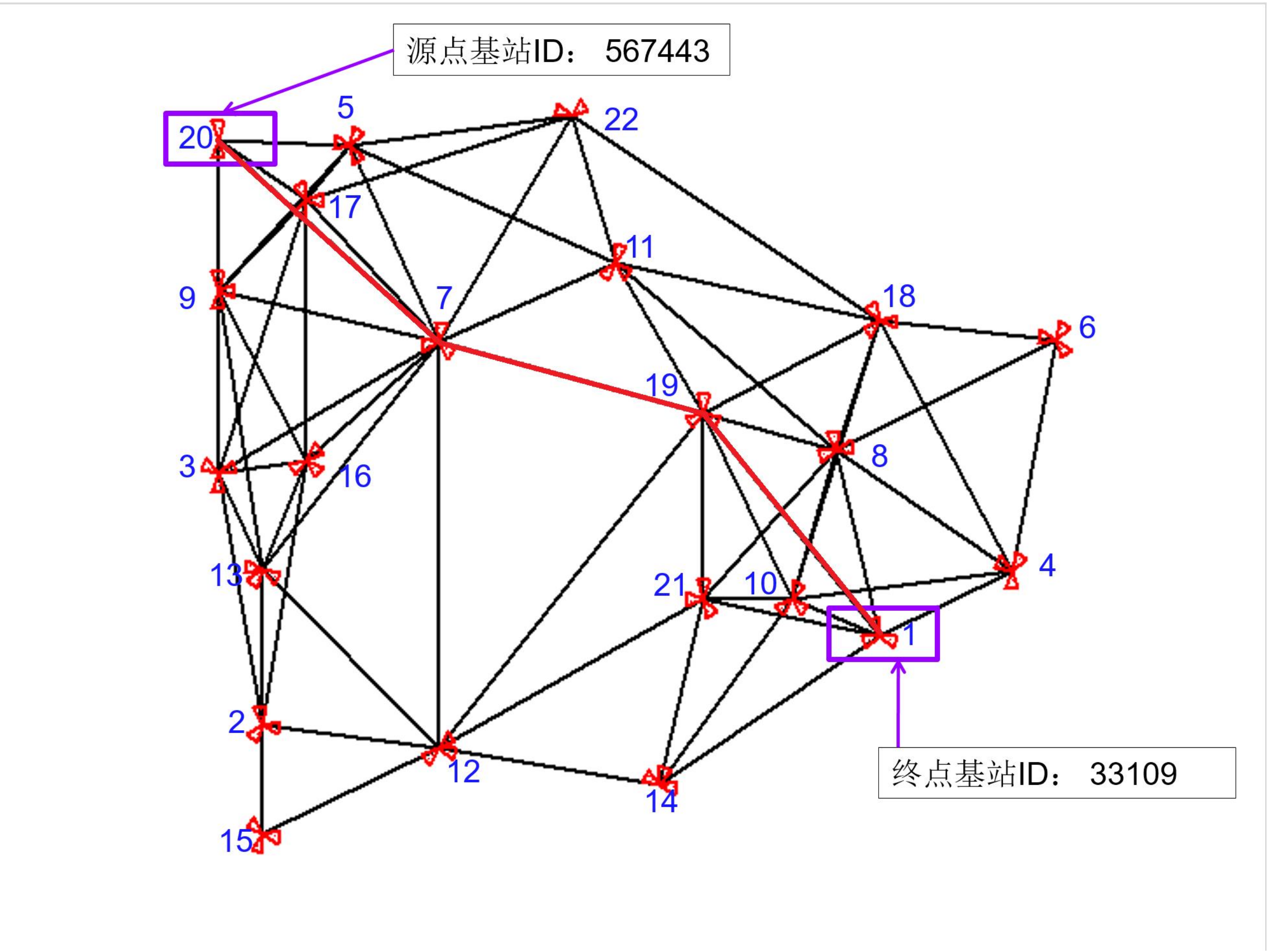
```
0011110100001110111011101010101100100111011111001001110010001010001010010011111010010110101001110000101110110111 0
1100001110111101010011011110010111011010100000001111101000001011011011111001101010011100111100000100100111011111 0
0111101011011111000100111011111001110110000001110011100100010100010010111010100001001111011111111001011110010011 1
1011010100110100011001101010110111100001011110101100110001011101111100101011110111100001110111010100110011000001 1
1011001111000011110111111011110101001101110000111110000011001000011111000000100100111011111001111000000100101110010 0
```

```
01110111000011111101100110110010010011000111101001111011111001101000011111011001100000001011100100001111011000011
0011010101010111011 1
```

```
哈夫曼编码长度:8834
定长编码长度:10475
耗时 0.025 秒
```

3.单源最短路径和最小生成树

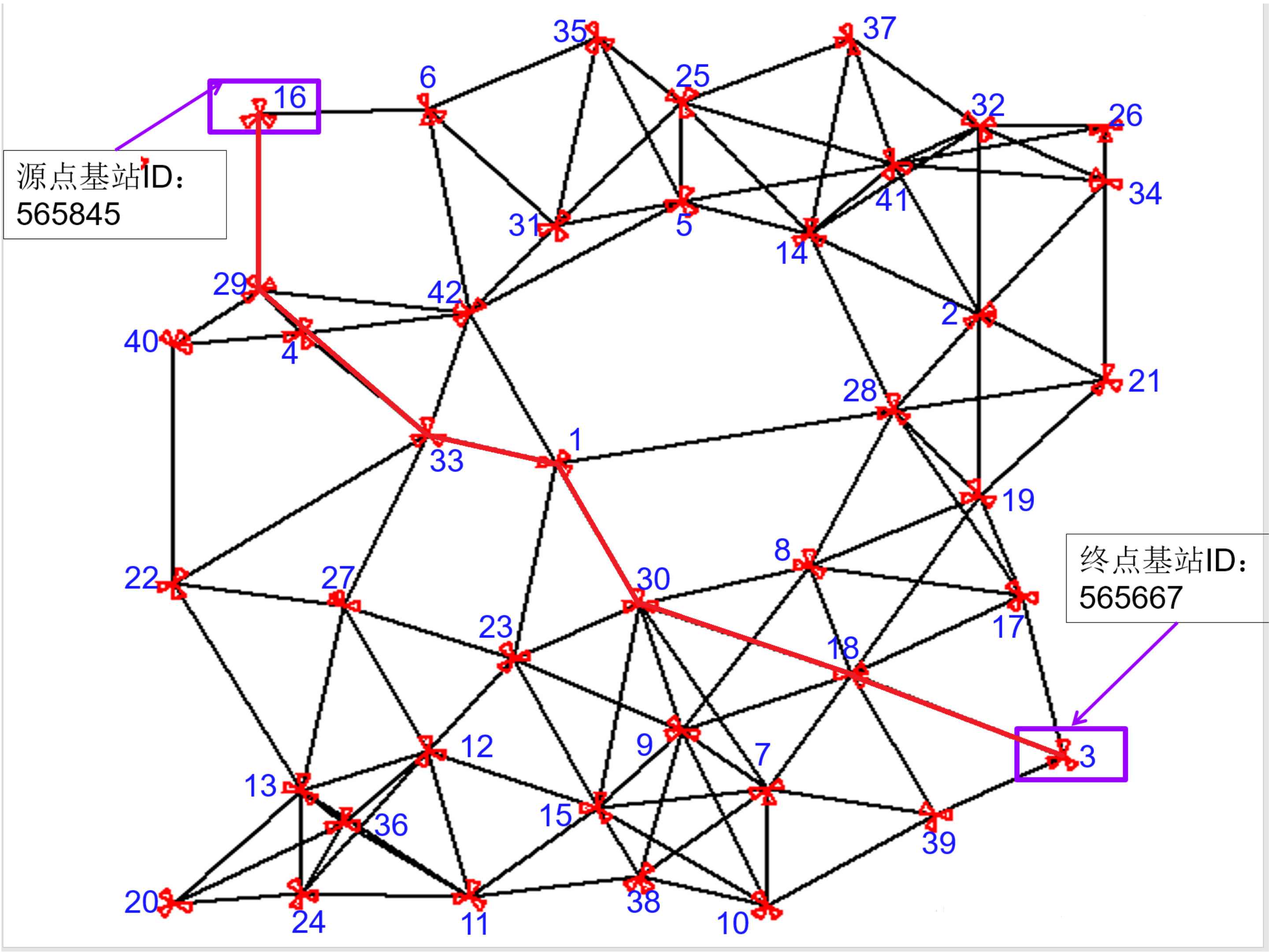
```
22 基站单源最短路径：
基站 20(567443)至 1(33109):1(33109) <- 19(567439) <- 7(566750) <- 20(567443)
长度为:1956.93
基站 20(567443)至 2(565696):2(565696) <- 13(566993) <- 9(566783) <- 20(567443)
长度为:1343.41
基站 20(567443)至 3(566631):3(566631) <- 9(566783) <- 20(567443)
长度为:761.938
基站 20(567443)至 4(566720):4(566720) <- 8(566751) <- 19(567439) <- 7(566750) <- 20(567443)
长度为:2111.29
基站 20(567443)至 5(566742):5(566742) <- 20(567443)
长度为:302.54
基站 20(567443)至 6(566747):6(566747) <- 18(567322) <- 11(566802) <- 5(566742) <- 20(567443)
长度为:1988.14
基站 20(567443)至 7(566750):7(566750) <- 20(567443)
长度为:683.088
基站 20(567443)至 8(566751):8(566751) <- 19(567439) <- 7(566750) <- 20(567443)
长度为:1622.91
基站 20(567443)至 9(566783):9(566783) <- 20(567443)
长度为:344.546
基站 20(567443)至 10(566798):10(566798) <- 19(567439) <- 7(566750) <- 20(567443)
长度为:1778.06
基站 20(567443)至 11(566802):11(566802) <- 5(566742) <- 20(567443)
长度为:963.852
基站 20(567443)至 12(566967):12(566967) <- 13(566993) <- 9(566783) <- 20(567443)
长度为:1562.25
基站 20(567443)至 13(566993):13(566993) <- 9(566783) <- 20(567443)
长度为:988.629
基站 20(567443)至 14(566999):14(566999) <- 12(566967) <- 13(566993) <- 9(566783) <- 20(567443)
长度为:2072.92
基站 20(567443)至 15(567203):15(567203) <- 13(566993) <- 9(566783) <- 20(567443)
长度为:1592.31
基站 20(567443)至 16(567238):16(567238) <- 9(566783) <- 20(567443)
长度为:780.892
基站 20(567443)至 17(567260):17(567260) <- 20(567443)
长度为:244.053
基站 20(567443)至 18(567322):18(567322) <- 11(566802) <- 5(566742) <- 20(567443)
长度为:1582.91
基站 20(567443)至 19(567439):19(567439) <- 7(566750) <- 20(567443)
长度为:1309.05
基站 20(567443)至 20(567443):20(567443) <- 20(567443)
长度为:0
基站 20(567443)至 21(567547):21(567547) <- 19(567439) <- 7(566750) <- 20(567443)
长度为:1733
基站 20(567443)至 22(568098):22(568098) <- 5(566742) <- 20(567443)
长度为:810.555
```



```
42 基站单源最短路径：
基站 16(565845)至 1(565675):1(565675) <- 33(567500) <- 29(567526) <- 16(565845)
长度为:1369.37
基站 16(565845)至 2(565621):2(565621) <- 14(565630) <- 5(565801) <- 31(565631) <- 6(566010) <- 16(565845)
长度为:1928.9
基站 16(565845)至 3(565667):3(565667) <- 18(565633) <- 30(565551) <- 1(565675) <- 33(567500) <- 29(567526) <- 16(565845)
长度为:2900.12
基站 16(565845)至 4(567510):4(567510) <- 29(567526) <- 16(565845)
长度为:645.041
基站 16(565845)至 5(565801):5(565801) <- 31(565631) <- 6(566010) <- 16(565845)
长度为:1153.11
基站 16(565845)至 6(566010):6(566010) <- 16(565845)
长度为:403.433
基站 16(565845)至 7(567891):7(567891) <- 30(565551) <- 1(565675) <- 33(567500) <- 29(567526) <- 16(565845)
长度为:2401.9
基站 16(565845)至 8(565492):8(565492) <- 30(565551) <- 1(565675) <- 33(567500) <- 29(567526) <- 16(565845)
长度为:2223.01
基站 16(565845)至 9(565558):9(565558) <- 30(565551) <- 1(565675) <- 33(567500) <- 29(567526) <- 16(565845)
长度为:2171.29
基站 16(565845)至 10(565627):10(565627) <- 9(565558) <- 30(565551) <- 1(565675) <- 33(567500) <- 29(567526) <- 16(565845)
长度为:2697.46
基站 16(565845)至 11(565572):11(565572) <- 12(565610) <- 27(566074) <- 33(567500) <- 29(567526) <- 16(565845)
长度为:2440.92
基站 16(565845)至 12(565610):12(565610) <- 27(566074) <- 33(567500) <- 29(567526) <- 16(565845)
```

```
长度为:2025.89
基站 16(565845)至 13(565859):13(565859) <- 22(567531) <- 40(565964) <- 29(567526) <- 16(565845)
长度为:2050.98
基站 16(565845)至 14(565630):14(565630) <- 5(565801) <- 31(565631) <- 6(566010) <- 16(565845)
长度为:1468.96
基站 16(565845)至 15(565559):15(565559) <- 23(565516) <- 1(565675) <- 33(567500) <- 29(567526) <- 16(565845)
长度为:2381.34
基站 16(565845)至 16(565845):16(565845) <- 16(565845)
长度为:0
基站 16(565845)至 17(565527):17(565527) <- 28(565648) <- 14(565630) <- 5(565801) <- 31(565631) <- 6(566010) <- 16(565845)
长度为:2594.34
基站 16(565845)至 18(565633):18(565633) <- 30(565551) <- 1(565675) <- 33(567500) <- 29(567526) <- 16(565845)
长度为:2347.84
基站 16(565845)至 19(565496):19(565496) <- 28(565648) <- 14(565630) <- 5(565801) <- 31(565631) <- 6(566010) <- 16(565845)
长度为:2308.24
基站 16(565845)至 20(565865):20(565865) <- 13(565859) <- 22(567531) <- 40(565964) <- 29(567526) <- 16(565845)
长度为:2489.07
基站 16(565845)至 21(565773):21(565773) <- 2(565621) <- 14(565630) <- 5(565801) <- 31(565631) <- 6(566010) <- 16(565845)
长度为:2281.46
基站 16(565845)至 22(567531):22(567531) <- 40(565964) <- 29(567526) <- 16(565845)
长度为:1402.79
基站 16(565845)至 23(565516):23(565516) <- 1(565675) <- 33(567500) <- 29(567526) <- 16(565845)
长度为:1918.1
基站 16(565845)至 24(565393):24(565393) <- 13(565859) <- 22(567531) <- 40(565964) <- 29(567526) <- 16(565845)
长度为:2339.03
基站 16(565845)至 25(565753):25(565753) <- 35(565562) <- 6(566010) <- 16(565845)
长度为:1122.45
基站 16(565845)至 26(33566):26(33566) <- 41(567618) <- 25(565753) <- 35(565562) <- 6(566010) <- 16(565845)
长度为:2169.68
基站 16(565845)至 27(566074):27(566074) <- 33(567500) <- 29(567526) <- 16(565845)
长度为:1573.64
基站 16(565845)至 28(565648):28(565648) <- 14(565630) <- 5(565801) <- 31(565631) <- 6(566010) <- 16(565845)
长度为:1997.17
基站 16(565845)至 29(567526):29(567526) <- 16(565845)
长度为:488.237
基站 16(565845)至 30(565551):30(565551) <- 1(565675) <- 33(567500) <- 29(567526) <- 16(565845)
长度为:1806.75
基站 16(565845)至 31(565631):31(565631) <- 6(566010) <- 16(565845)
长度为:843.923
基站 16(565845)至 32(565608):32(565608) <- 41(567618) <- 25(565753) <- 35(565562) <- 6(566010) <- 16(565845)
长度为:1883.38
基站 16(565845)至 33(567500):33(567500) <- 29(567526) <- 16(565845)
长度为:1055.67
基站 16(565845)至 34(565531):34(565531) <- 41(567618) <- 25(565753) <- 35(565562) <- 6(566010) <- 16(565845)
长度为:2161.48
基站 16(565845)至 35(565562):35(565562) <- 6(566010) <- 16(565845)
长度为:853.566
基站 16(565845)至 36(32788):36(32788) <- 13(565859) <- 22(567531) <- 40(565964) <- 29(567526) <- 16(565845)
长度为:2187.66
基站 16(565845)至 37(567497):37(567497) <- 25(565753) <- 35(565562) <- 6(566010) <- 16(565845)
长度为:1561.46
基站 16(565845)至 38(566316):38(566316) <- 9(565558) <- 30(565551) <- 1(565675) <- 33(567500) <- 29(567526) <- 16(565845)
长度为:2592.69
基站 16(565845)至 39(568056):39(568056) <- 18(565633) <- 30(565551) <- 1(565675) <- 33(567500) <- 29(567526) <- 16(565845)
长度为:2787.2
基站 16(565845)至 40(565964):40(565964) <- 29(567526) <- 16(565845)
长度为:741.608
基站 16(565845)至 41(567618):41(567618) <- 25(565753) <- 35(565562) <- 6(566010) <- 16(565845)
长度为:1655.16
```

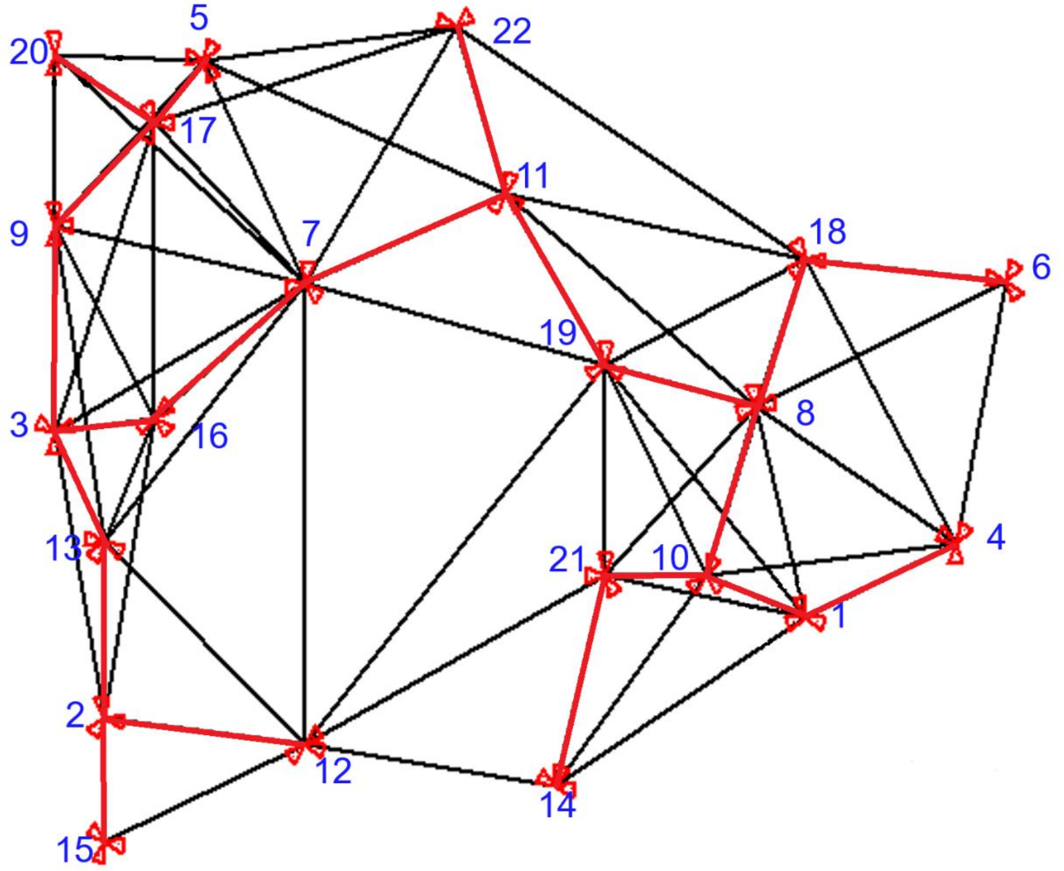
```
基站 16(565845)至 42(565898):42(565898) <- 6(566010) <- 16(565845)
长度为:978.426
```



源点基站ID：
565845

终点基站ID：
565667

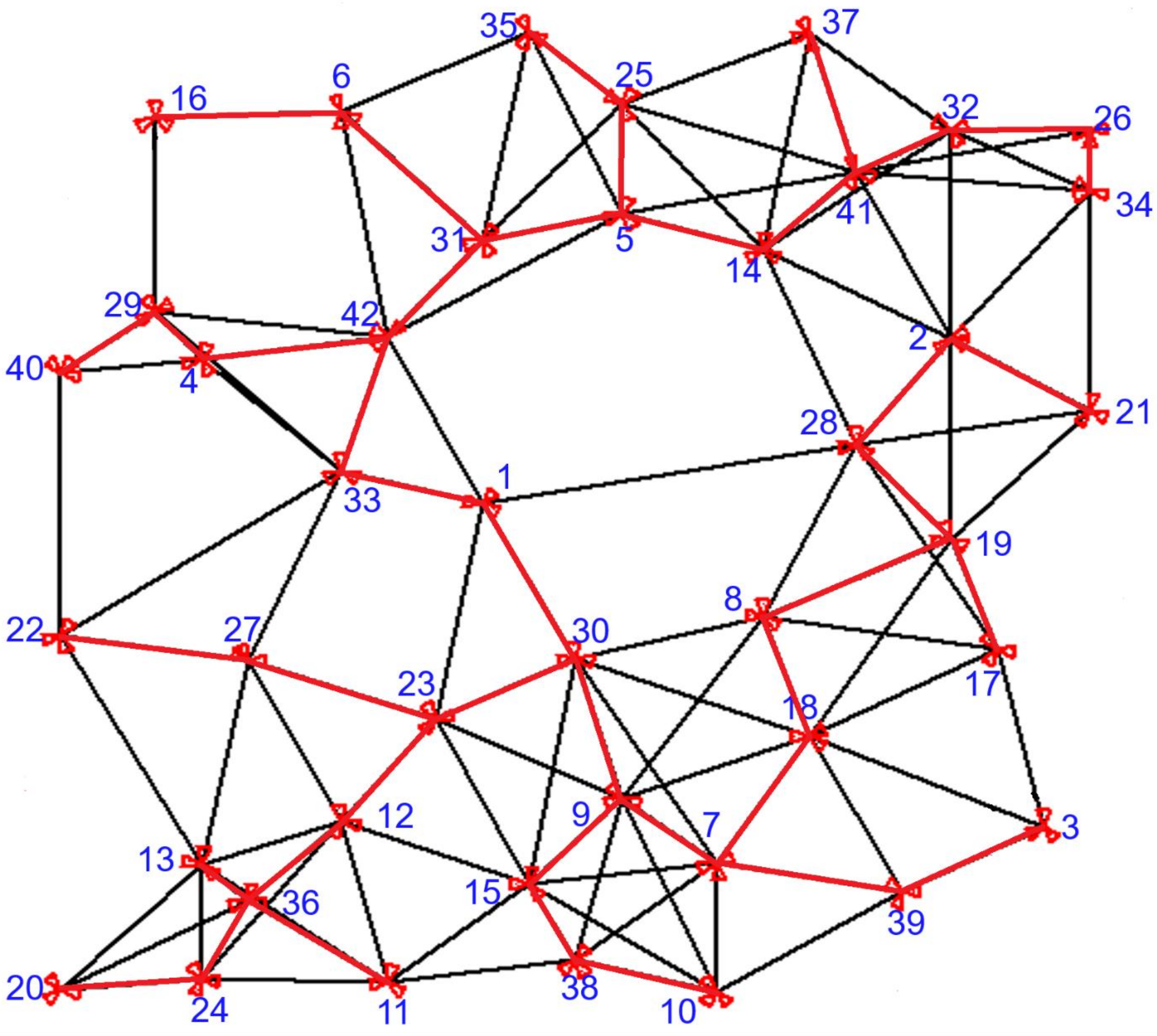
```
22 基站最小生成树:
1(33109) -> 10(566798)
10(566798) -> 21(567547)
1(33109) -> 4(566720)
10(566798) -> 8(566751)
8(566751) -> 18(567322)
8(566751) -> 19(567439)
19(567439) -> 11(566802)
11(566802) -> 22(568098)
18(567322) -> 6(566747)
21(567547) -> 14(566999)
11(566802) -> 7(566750)
7(566750) -> 16(567238)
16(567238) -> 3(566631)
3(566631) -> 13(566993)
13(566993) -> 2(565696)
2(565696) -> 15(567203)
2(565696) -> 12(566967)
3(566631) -> 9(566783)
9(566783) -> 17(567260)
17(567260) -> 5(566742)
17(567260) -> 20(567443)
```



42 基站最小生成树:

```
1(565675) -> 33(567500)
33(567500) -> 42(565898)
42(565898) -> 31(565631)
31(565631) -> 5(565801)
5(565801) -> 25(565753)
25(565753) -> 35(565562)
5(565801) -> 14(565630)
14(565630) -> 41(567618)
41(567618) -> 32(565608)
32(565608) -> 26(33566)
26(33566) -> 34(565531)
41(567618) -> 37(567497)
42(565898) -> 4(567510)
4(567510) -> 29(567526)
29(567526) -> 40(565964)
1(565675) -> 30(565551)
30(565551) -> 23(565516)
23(565516) -> 12(565610)
12(565610) -> 36(32788)
36(32788) -> 13(565859)
36(32788) -> 24(565393)
24(565393) -> 20(565865)
36(32788) -> 11(565572)
```

```
30(565551) -> 9(565558)
9(565558) -> 7(567891)
9(565558) -> 15(565559)
15(565559) -> 38(566316)
38(566316) -> 10(565627)
7(567891) -> 18(565633)
18(565633) -> 8(565492)
7(567891) -> 39(568056)
39(568056) -> 3(565667)
23(565516) -> 27(566074)
27(566074) -> 22(567531)
31(565631) -> 6(566010)
6(566010) -> 16(565845)
8(565492) -> 19(565496)
19(565496) -> 17(565527)
19(565496) -> 28(565648)
28(565648) -> 2(565621)
2(565621) -> 21(565773)
```



耗时 0.006 秒