算法设计与分析

第四章

姓名：张博康

学号：2014211383

班级：2014211309

一、 源代码

```
#include <iostream>
#include <sstream>
#include <fstream>
#include <cstring>
#include <cstdlib>
#include <vector>
#include <climits>
#include <cmath>
#include <queue>
#include <map>
#include <algorithm>

#define MAXSIZE 10000
#define Pi 3.141592657

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

typedef struct node
{
    struct node *left, *right;
    int weight;
    char ch;
} Node, *pNode;

typedef struct edge
{
    int u, v;
    double cost;
    bool operator < (const struct edge &m)const {
                return cost < m.cost;
        }
} Edge, *pEdge;

using namespace std;

const double R = 6378137.0; //地球半径，以 m 为单位
```

```
Station stations[MAXSIZE];
int dp[MAXSIZE][MAXSIZE];
int opr[MAXSIZE][MAXSIZE];
double weight[MAXSIZE][MAXSIZE];
int father[MAXSIZE];

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));
}

double Weight(int i, int k, int j)
{
    return weight[i][k] + weight[i][j] + weight[k][j];
}

double MinWeightTriangulation(const int &n)
{
    memset(dp, 0, sizeof(dp));
    memset(opr, 0, sizeof(opr));

    for (int i = 0; i < n; i++)
        for (int j = i; j < n; j++)
            weight[j][i] = weight[i][j] = distance(stations[i],
stations[j]);

    for (int r = 2; r <= n; r++)
        for (int i = 1; i <= n-r+1; i++)
        {
            int j = i + r - 1;
            dp[i][j] = dp[i + 1][j] + Weight(i - 1, i, j);
            opr[i][j] = i;
            for (int k = i+1; k < j; k++)
            {
                int u = dp[i][k] + dp[k + 1][j] + Weight(i - 1, k,
j);
                if (u < dp[i][j])
```

```
                {
                    dp[i][j] = u;
                    opr[i][j] = k;
                }
            }
        }

    double len = 0;
    for (int i = 0; i < n-1; i++)
    {
        len += weight[i][i+1];
    }
    len += weight[n-1][0];

    return (len + dp[1][n-1]) / 2;
}

void TraceBack(int i, int j)
{
    if (i == j)
        return;

    TraceBack(i, opr[i][j]);
    TraceBack(opr[i][j] + 1, j);

    cout << "三角剖分顶点：V" << i-1 << ",V" << j << ",V" <<
opr[i][j] << endl;
}

struct cmp{
    bool operator()(pNode node1, pNode node2){
        return node1->weight > node2->weight;
    }
};

pNode HuffmanTree(int incidence[])
{
    priority_queue<pNode, vector<pNode>, cmp > heap;

    for (int i = 0; i <= 26; i++)
    {
        pNode haffman = new Node;
        haffman->left = NULL;
        haffman->right = NULL;
```

```
        haffman->weight = incidence[i];
        if (i == 0)
            haffman->ch = '#';
        else
            haffman->ch = i - 1 + 'a';
        heap.push(haffman);
    }

    while (heap.size() != 1)
    {
        pNode node1 = heap.top();
        heap.pop();
        pNode node2 = heap.top();
        heap.pop();

        pNode haffman = new Node;
        haffman->left = node1;
        haffman->right = node2;
        haffman->weight = node1->weight + node2->weight;
        heap.push(haffman);
    }

    return heap.top();
}

int printHuffman(pNode node, string str, int incidence[])
{
    if (node->left == NULL && node->right == NULL)
    {
        cout << node->ch << ": " << str << endl;
        return incidence[(node->ch == '#') ? 0 : (node->ch - 'a' +
1)] * str.length();
    }

    int result1 = 0, result2 = 0;
    if (node->left)
        result1 = printHuffman(node->left, str + '0', incidence);
    if (node->right)
        result2 = printHuffman(node->right, str + '1', incidence);

    return result1 + result2;
}

void Dijkstra(double dist[], double map[][42], bool known[], int seq,
```

```
int path[], int scale)
{
    dist[seq] = 0;
    known[seq] = true;

    while (1) {
        for (int i = 0; i < scale; i++)
            if (!known[i] && map[seq][i] > 0 && dist[seq] +
map[seq][i] < dist[i]) {
                dist[i] = dist[seq] + map[seq][i];
                path[i] = seq;
            }
        seq = -1;
        double min = INT_MAX;
        for (int i = 0; i < scale; i++)
            if (!known[i] && min > dist[i])
            {
                min = dist[i];
                seq = i;
            }

        if (seq == -1)
            break;

        known[seq] = true;
    }
}

void Traverse(int path[], int city, int origin, map<int, int> &seq)
{
    if (city == origin)
    {
        cout << seq.find(city)->second;
        return;
    }

    Traverse(path, path[city], origin, seq);
    cout << " -> " << seq.find(city)->second ;
}

int find(int x)
{
    if (x != father[x])
        father[x] = find(father[x]);
```

```
    return father[x];
}

void unite(int x, int y)
{
    x = find(x);
    y = find(y);
    if (x != y)
        father[x] = y;
}

bool same(int x, int y)
{
    return find(x) == find(y);
}


double Kruskal(vector<Edge> &edges, bool map[][42])
{
    sort(edges.begin(), edges.end());
    double res = 0;
    int n = edges.size();

    for (int i = 0; i < n; i++) {
        Edge e = edges[i];
        if (!same(e.u, e.v)) {
            map[e.u][e.v] = true;
            unite(e.u, e.v);
            res += e.cost;
        }
    }

    return res;
}

int main(int argc, char const *argv[])
{
    int choose = 0;
    while (choose != 5)
    {
        cout << "请选择以下操作：" << endl;
        cout << "1 基于贪心法的凸多边形三角剖分" << endl;
        cout << "2 哈夫曼编码" << endl;
        cout << "3 单源最短路径" << endl;
```

```
        cout << "4 最小生成树" << 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;

        switch (choose)
        {
            case 1:
            {
                ifstream in1("附件 3-1.21 个基站凸多边形数据.txt",
ios_base::in);
                ifstream in2("附件 3-2.29 个基站凸多边形数据.txt",
ios_base::in);

                if (!in1.is_open() || !in2.is_open())
                {
                    cout << "Error opening file..." << endl;
                    exit(1);
                }
                int n = 0;
                while (in1 >> stations[n].enodedId >>
stations[n].longitude
                    >> stations[n].latitude >> stations[n].index)
                    n++;
                cout << "21 个基站凸多边形的最优三角剖分值为： " <<
MinWeightTriangulation(n) << endl;
                cout << "最优三角剖分结构为：" << endl;
                TraceBack(1, n - 1);

                n = 0;
                while (in2 >> stations[n].enodedId >>
stations[n].longitude
                    >> stations[n].latitude >> stations[n].index)
                    n++;
                cout << endl << "29 个基站凸多边形的最优三角剖分值
为： " << MinWeightTriangulation(n) << endl;
                cout << "最优三角剖分结构为：" << endl;
```

```
                TraceBack(1, n - 1);


                in1.close();
                in2.close();
                cout << "-----------------------------------------
---------------------" << endl;
                break;
            }
            case 2:
            {
                ifstream in("附件 2.哈夫曼编码输入文本.txt",
ios_base::in);
                if (!in.is_open())
                {
                    cout << "Error opening file..." << endl;
                    exit(1);
                }

                char ch;
                int incidence[MAXSIZE] = {0};
                while (in >> ch)
                {
                    if (ch == '#')
                        incidence[0]++;
                    else
                    {
                        ch = tolower(ch);
                        incidence[ch - 'a' + 1]++;
                    }
                }
                pNode Tree = HuffmanTree(incidence);
                cout << "哈夫曼编码如下：" << endl;
                int HuffmanCode = printHuffman(Tree, string(),
incidence);
                int OrdinaryCode = 0;
                for (int i = 0; i <= 26; i++)
                    OrdinaryCode += incidence[i] * 5;
                cout << "采用哈夫曼编码，输入文本需要的存储比特数: "
<< HuffmanCode << endl;
                cout << "采用定长编码，输入文本需要的存储比特数: " <<
OrdinaryCode << endl;
                in.close();
```

```
                cout << "-----------------------------------------
----------------------" << endl;
                break;
            }
            case 3:
            {
                ifstream in1("附件 1-1.22 基站图的邻接矩阵-v1.txt",
ios_base::in);
                ifstream in2("附件 1-1.42 基站图的邻接矩阵-v1.txt",
ios_base::in);
                if (!in1.is_open() || !in2.is_open())
                {
                    cout << "Error opening file..." << endl;
                    exit(1);
                }

                map<int, int> mark, seq;
                int enodedID, i = 0;
                double num;
                bool known[42];
                double map[42][42];
                double dist[42];
                int path[42];

                string line;
                getline(in1, line);
                istringstream iss(line);
                while (iss >> enodedID) {
                    mark.insert(make_pair(enodedID, i));
                    seq.insert(make_pair(i++, enodedID));
                }

                for (int i = 0; i < 22; i++) {
                    dist[i] = INT_MAX;
                    for (int j = 0; j <= 22; j++) {
                        in1 >> num;
                        if (j != 0)
                            map[i][j - 1] = num;
                    }
                }
                memset(known, 0, sizeof(known));
                Dijkstra(dist, map, known, mark.find(567443)->second,
path, 22);
```

```
                cout << "22 个基站顶点组成的图: " << endl;
                cout << "567443 到其它各点的单源最短路径：" << endl;
                for (int i = 0; i < 22; i++)
                {
                    cout << "567443->" << seq.find(i)->second << ": "
<< dist[i] << endl;
                }
                cout << "567443 到 33109 的最短路径：" << endl;
                Traverse(path, mark.find(33109)->second,
mark.find(567443)->second, seq);
                cout << endl;

                mark.clear();
                seq.clear();
                getline(in2, line);
                iss.clear();
                iss.str(line);
                i = 0;
                while (iss >> enodedID) {
                    mark.insert(make_pair(enodedID, i));
                    seq.insert(make_pair(i++, enodedID));
                }

                for (int i = 0; i < 42; i++) {
                    dist[i] = INT_MAX;
                    for (int j = 0; j <= 42; j++) {
                        in2 >> num;
                        if (j != 0)
                            map[i][j - 1] = num;
                    }
                }
                memset(known, 0, sizeof(known));
                Dijkstra(dist, map, known, mark.find(565845)->second,
path, 42);

                cout << "42 个基站顶点组成的图： " << endl;
                cout << "565845 到其他各点的单源最短路径：" << endl;
                for (int i = 0; i < 42; i++)
                {
                    cout << "565845->" << seq.find(i)->second << ": "
<< dist[i] << endl;
                }
                cout << "565845 到 565667 的最短路径：" << endl;
                Traverse(path, mark.find(565667)->second,
```

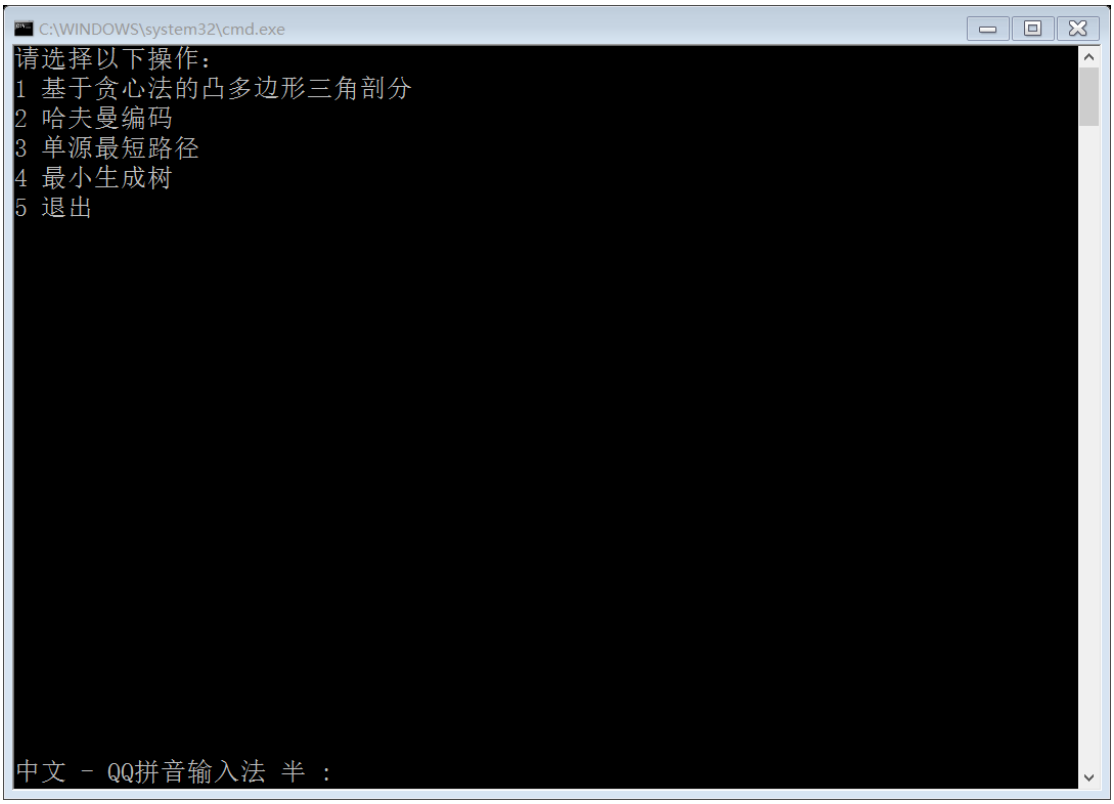
```
mark.find(565845)->second, seq);
                cout << endl;

               in1.close();
               in2.close();
                cout << "-----------------------------------------
---------------------" << endl;
                break;
             }
             case 4:
            {
                ifstream in1("附件 1-1.22 基站图的邻接矩阵-v1.txt",
ios_base::in);
                ifstream in2("附件 1-1.42 基站图的邻接矩阵-v1.txt",
ios_base::in);
                if (!in1.is_open() || !in2.is_open())
                {
                    cout << "Error opening file..." << endl;
                    exit(1);
                }

                map<int, int> mark, seq;
                int enodedID, i = 0;
                double num;
                bool map[42][42];
                vector<Edge> edges;

                string line;
                getline(in1, line);
                istringstream iss(line);
                while (iss >> enodedID) {
                    mark.insert(make_pair(enodedID, i));
                    seq.insert(make_pair(i++, enodedID));
                }

                for (int i = 0; i < 22; i++) {
                    father[i] = i;
                    for (int j = 0; j <= 22; j++) {
                        in1 >> num;
                        if (j != 0 && num > 0) {
                            Edge temp;
                            temp.u = i;
                            temp.v = j - 1;
                            temp.cost = num;
```

```
                edges.push_back(temp);
                map[i][j - 1] = false;
            }
        }
    }
    double res = Kruskal(edges, map);
    cout << "22 个基站顶点组成的图的最小生成树代价为： "
<< res << endl;
    cout << "连接的边有：" << endl;
    for (int i = 0; i < 22; i++)
        for (int j = 0; j < 22; j++)
            if (map[i][j])
                cout << i << "--" << j << '\t';
    cout << endl;

    mark.clear();
    seq.clear();
    i = 0;
    edges.clear();

    getline(in1, line);
    iss.str(line);
    while (iss >> enodedID) {
        mark.insert(make_pair(enodedID, i));
        seq.insert(make_pair(i++, enodedID));
    }

    for (int i = 0; i < 42; i++) {
        father[i] = i;
        for (int j = 0; j <= 42; j++) {
            in2 >> num;
            if (j != 0 && num > 0) {
                Edge temp;
                temp.u = i;
                temp.v = j - 1;
                temp.cost = num;
                edges.push_back(temp);
                map[i][j - 1] = false;
            }
        }
    }
    res = Kruskal(edges, map);
    cout << "42 个基站顶点组成的图的最小生成树代价为： "
<< res << endl;
```

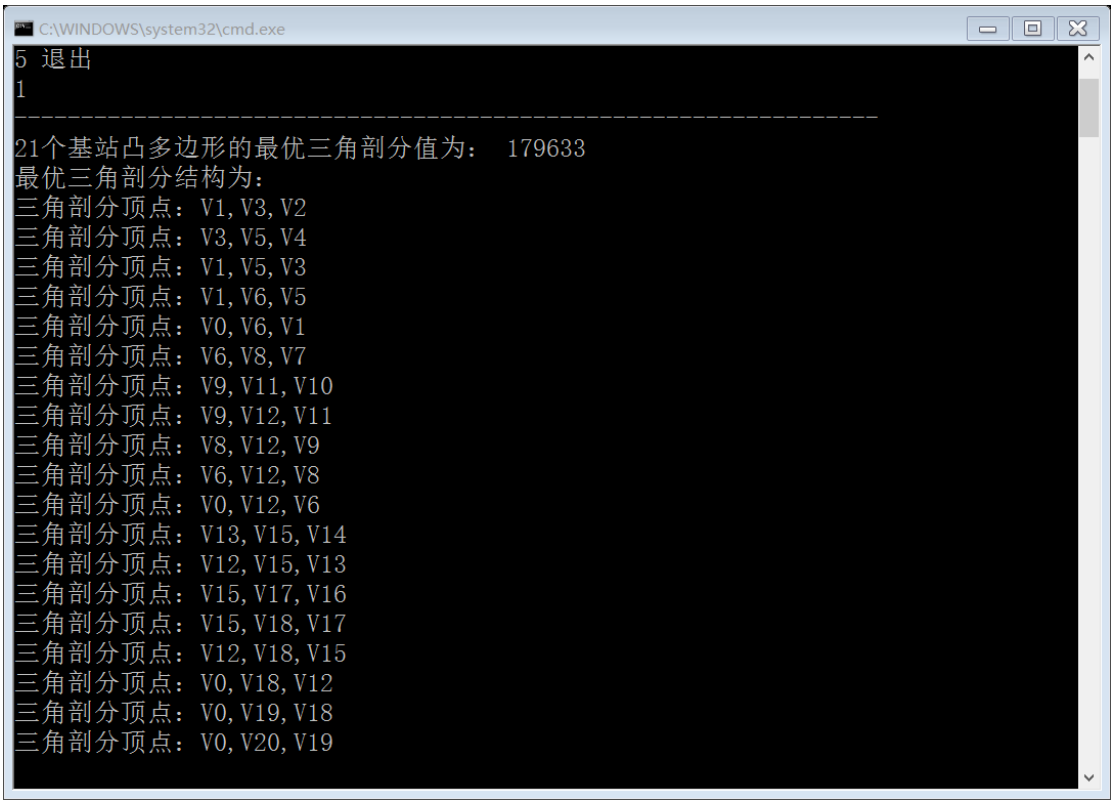
```
                cout << "连接的边有：" << endl;
                for (int i = 0; i < 42; i++)
                    for (int j = 0; j < 42; j++)
                        if (map[i][j])
                            cout << i << "--" << j << '\t';
                cout << endl;

                cout << "----------------------------------------
----------------------" << endl;
                break;
            }

            default:
                break;
        }
    }
    return 0;
}
```

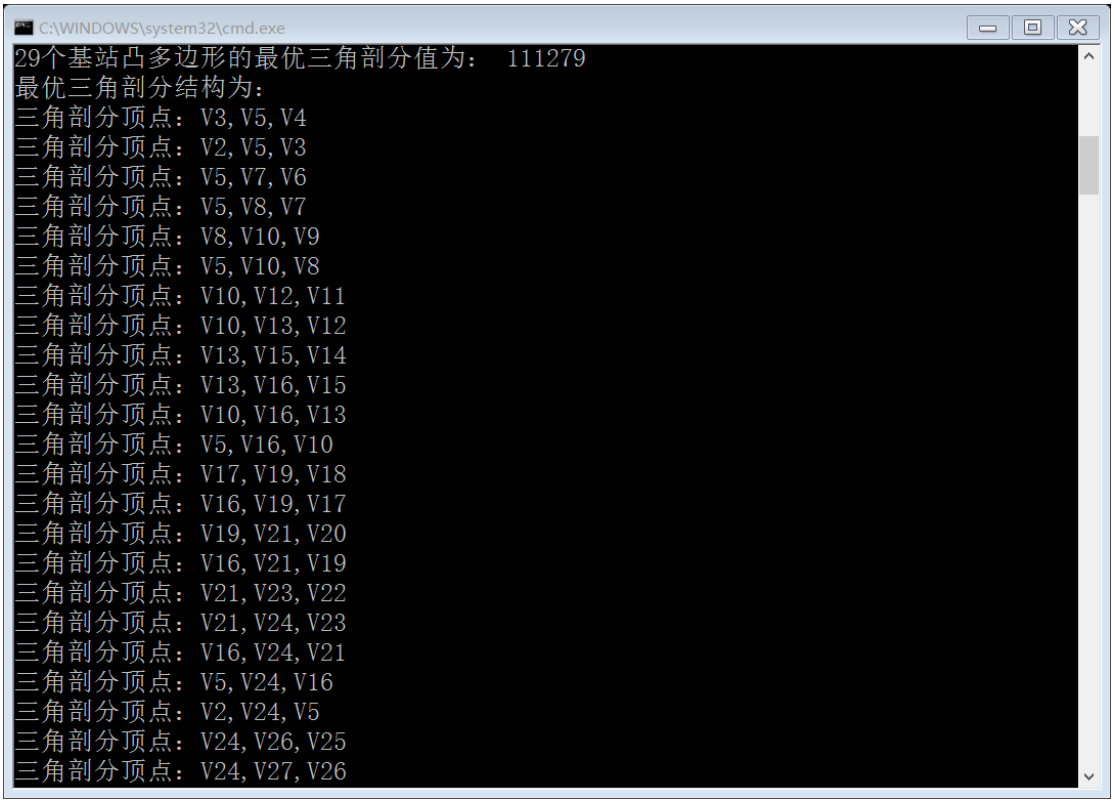
二、 运行结果

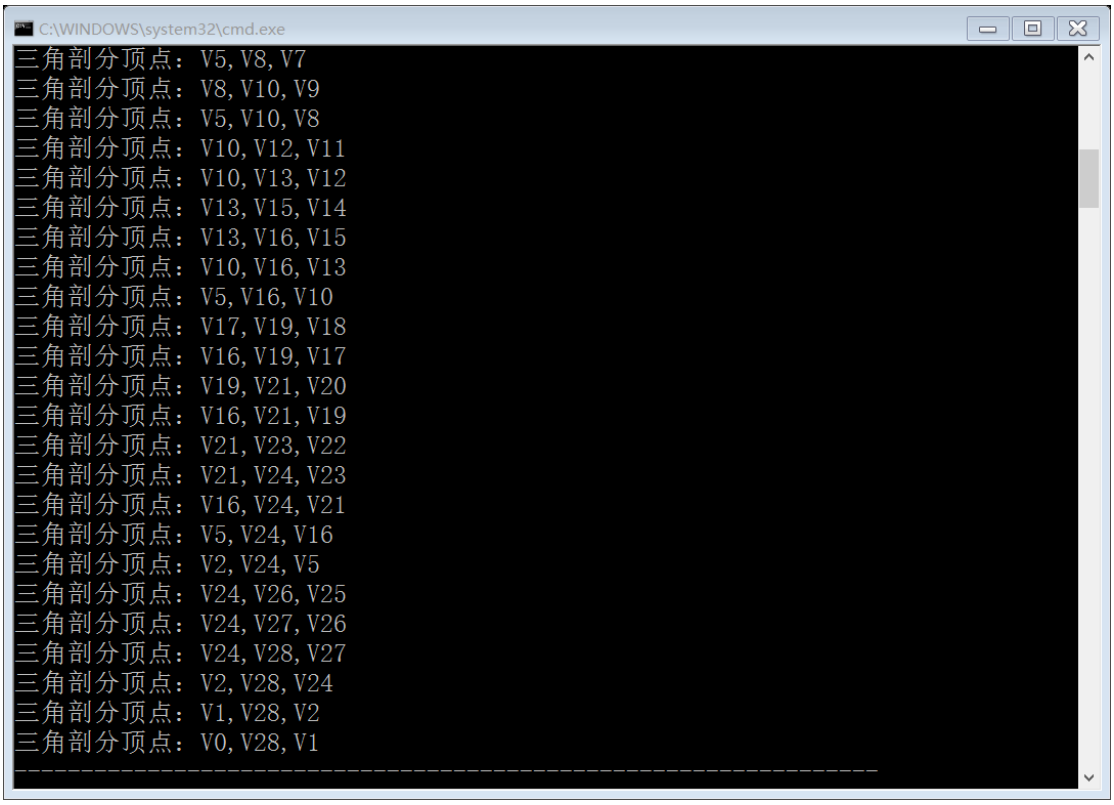
1. 开始界面（输入 1-5，选择相应操作）

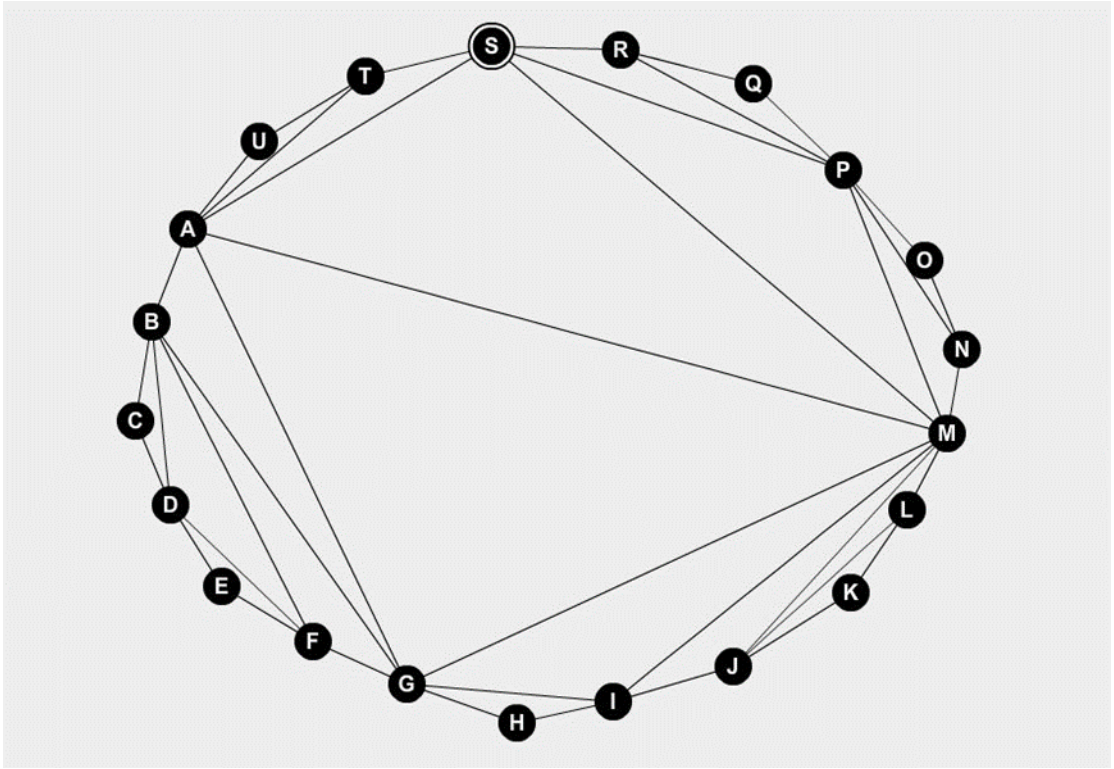


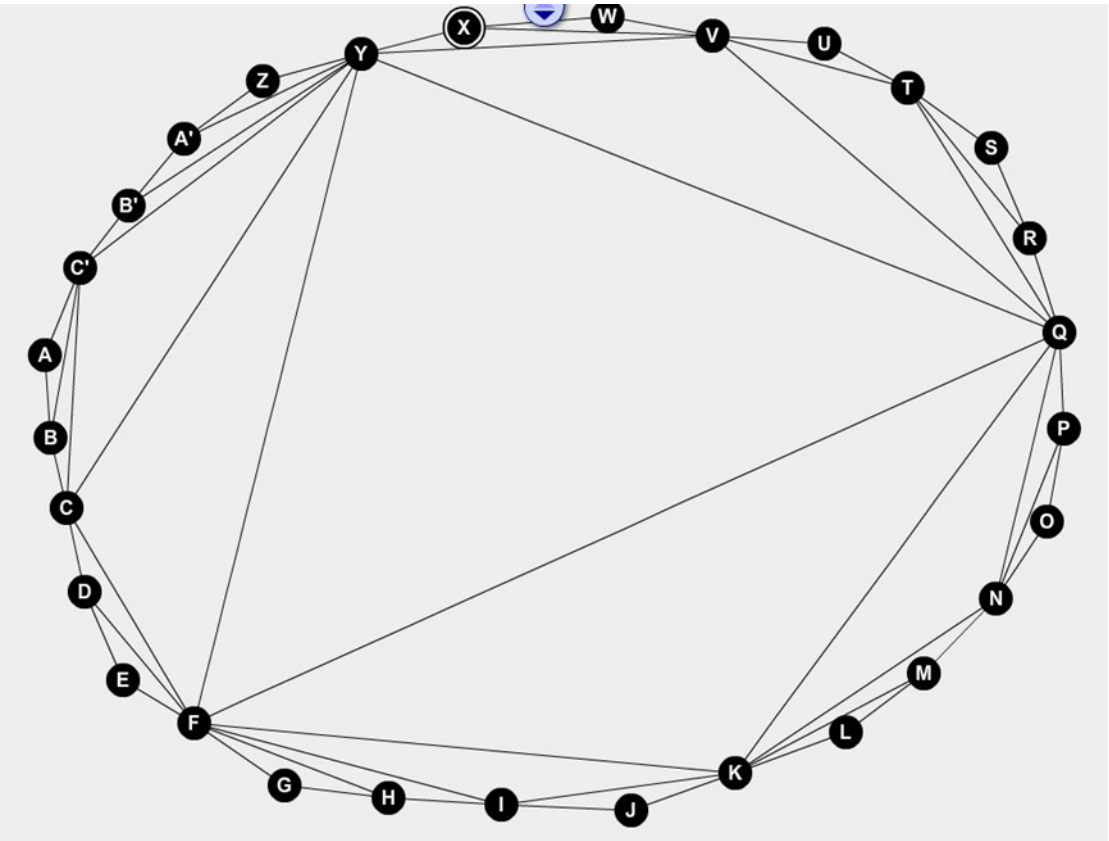
2. （输入 1）基于贪心法的凸多边形三角剖分



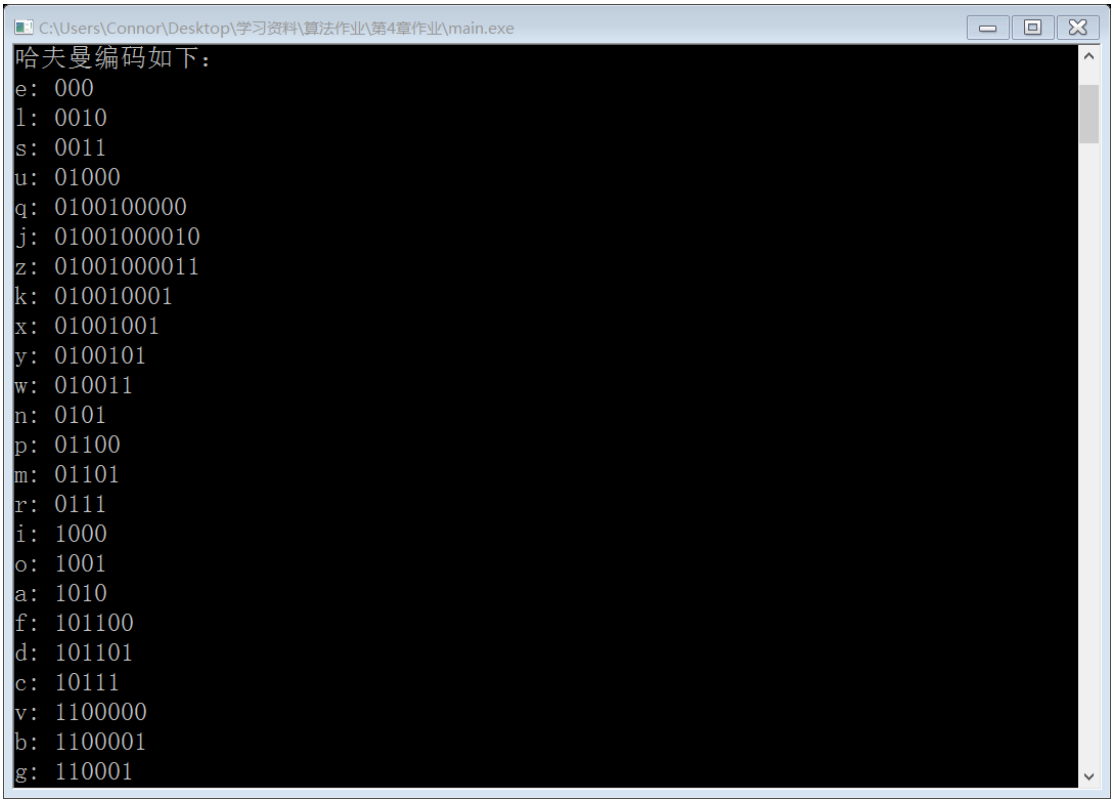


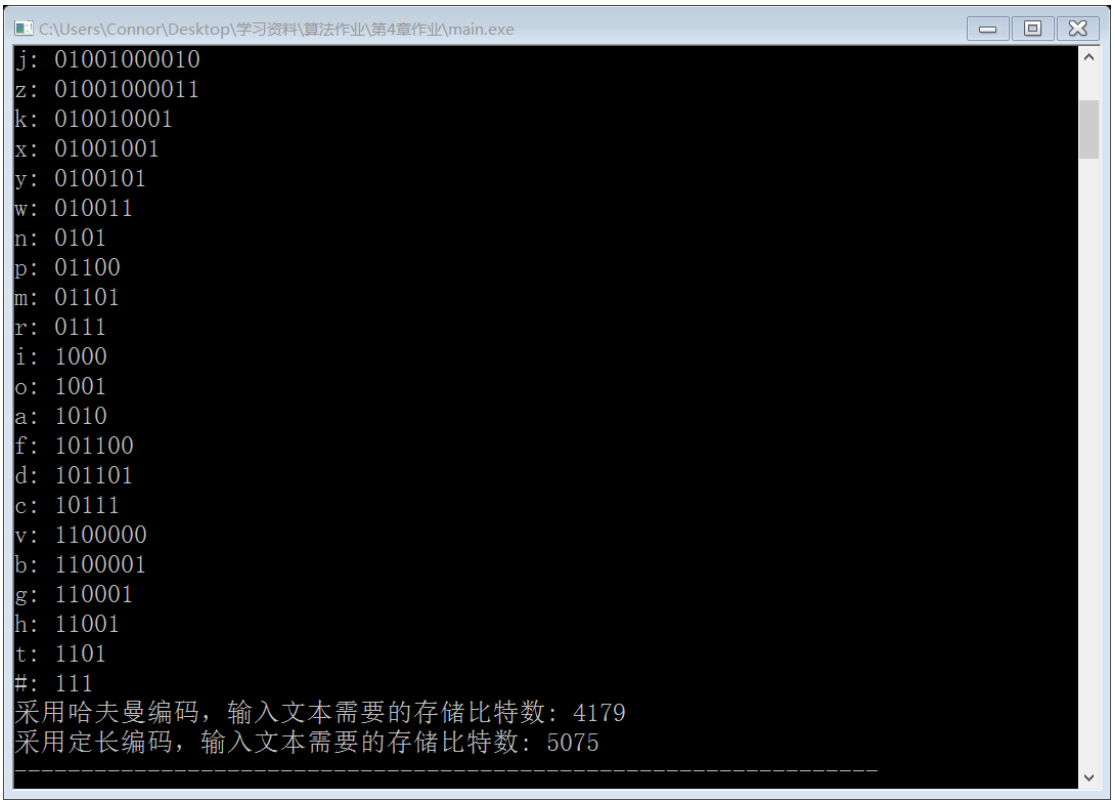

C:\WINDOWS\system32\cmd.exe
三角剖分顶点：V5, V8, V7
三角剖分顶点：V8, V10, V9
三角剖分顶点：V5, V10, V8
三角剖分顶点：V10, V12, V11
三角剖分顶点：V10, V13, V12
三角剖分顶点：V13, V15, V14
三角剖分顶点：V13, V16, V15
三角剖分顶点：V10, V16, V13
三角剖分顶点：V5, V16, V10
三角剖分顶点：V17, V19, V18
三角剖分顶点：V16, V19, V17
三角剖分顶点：V19, V21, V20
三角剖分顶点：V16, V21, V19
三角剖分顶点：V21, V23, V22
三角剖分顶点：V21, V24, V23
三角剖分顶点：V16, V24, V21
三角剖分顶点：V5, V24, V16
三角剖分顶点：V2, V24, V5
三角剖分顶点：V24, V26, V25
三角剖分顶点：V24, V27, V26
三角剖分顶点：V24, V28, V27
三角剖分顶点：V2, V28, V24
三角剖分顶点：V1, V28, V2
三角剖分顶点：V0, V28, V1


S
R
Q
P
O
N
M
L
K
J
I
H
G
F
E
D
C
B
A
U
T

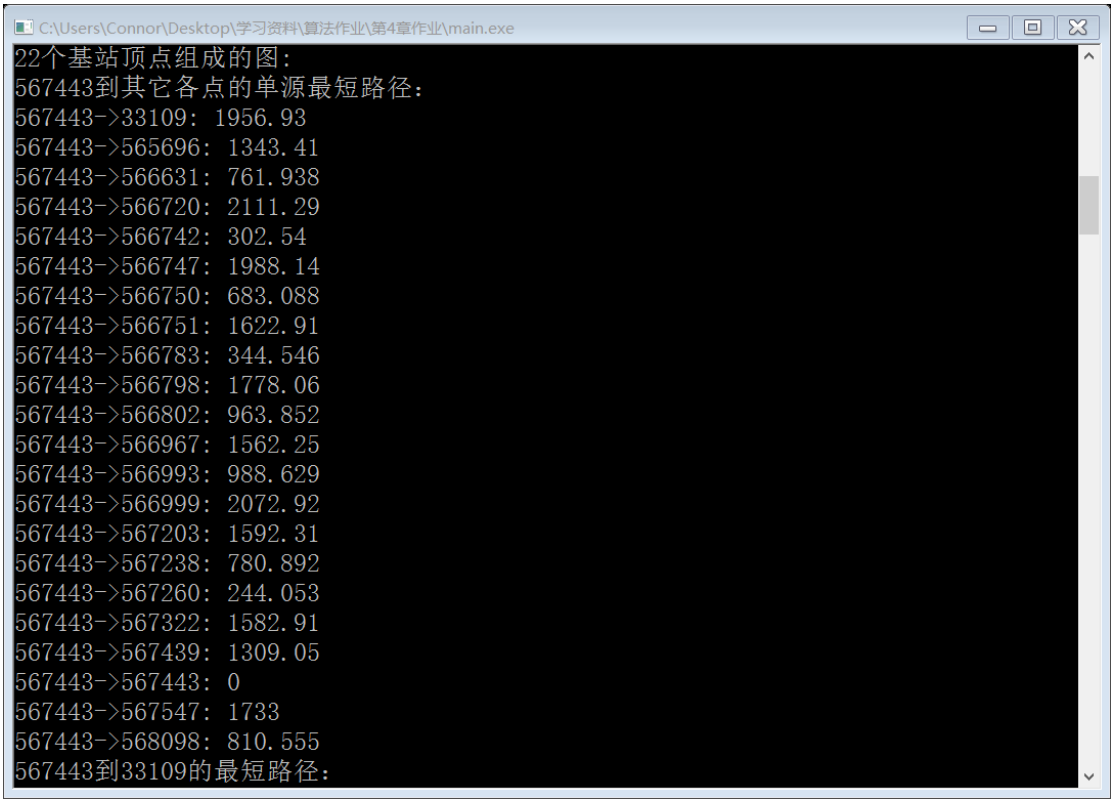

X
W
V
U
T
S
R
Q
P
O
N
M
L
K
J
I
H
G
F
E
D
C
B
A
C'
B'
A'
Z
Y

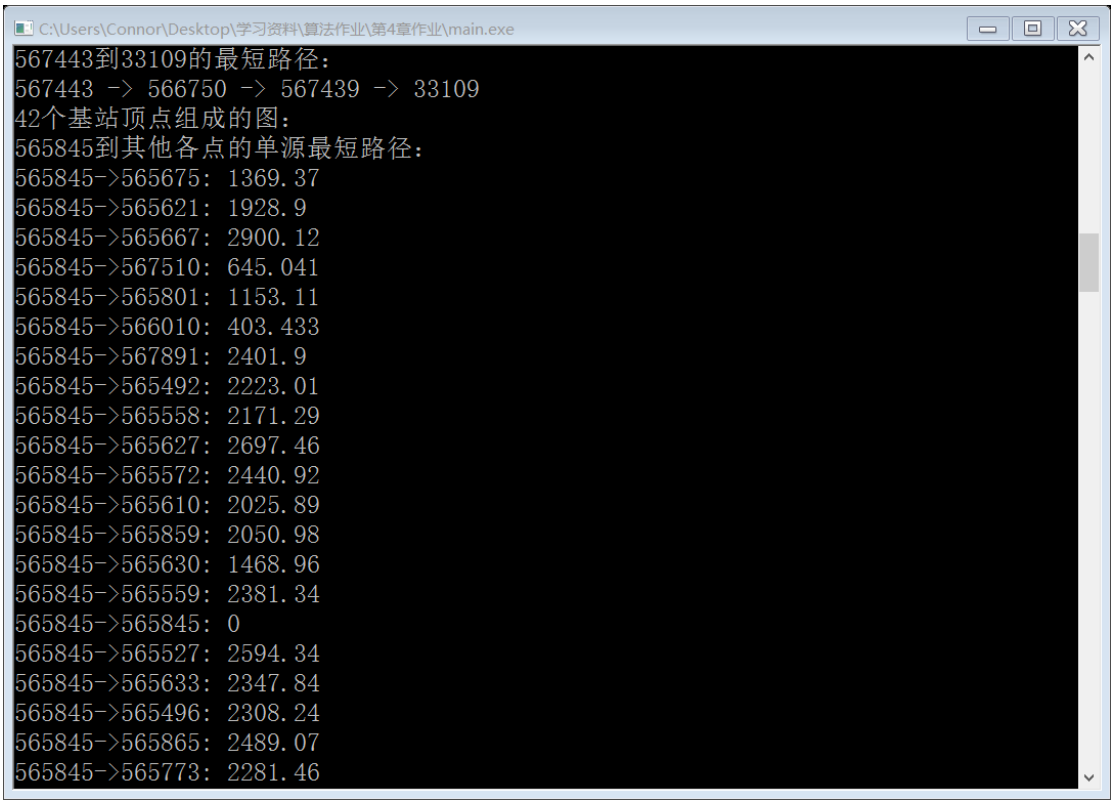
3. （输入 2）哈夫曼编码

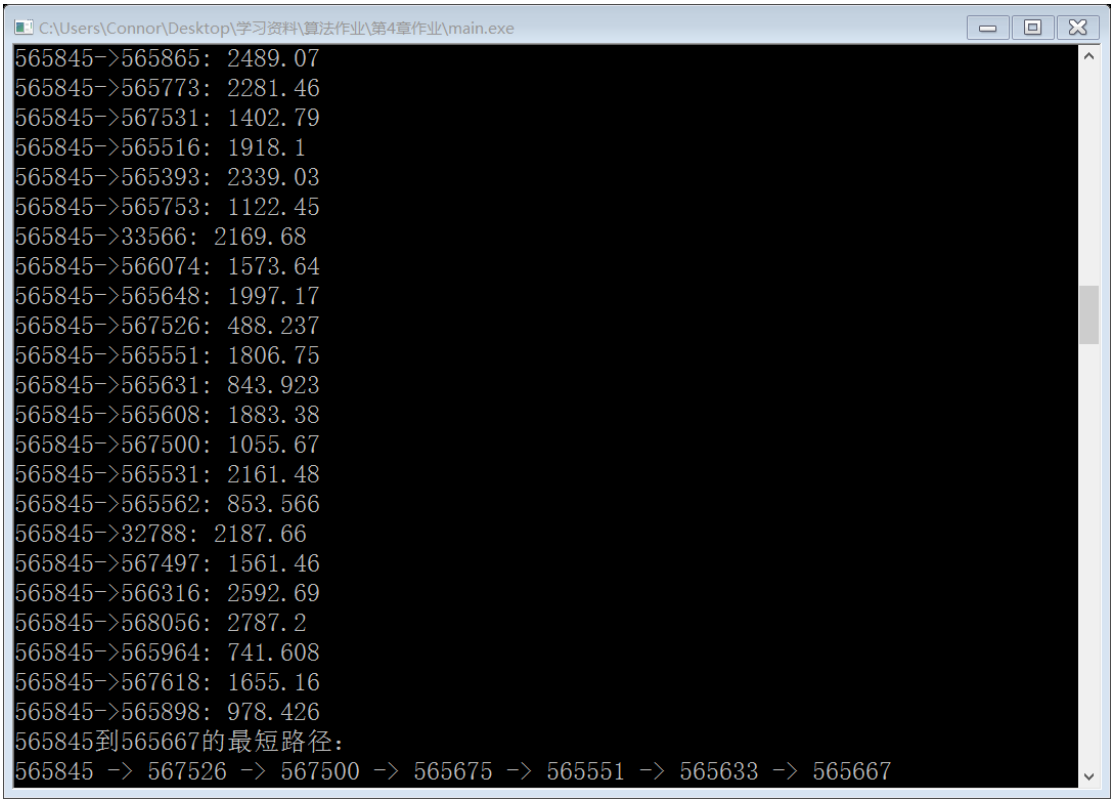

C:\Users\Connor\Desktop\学习资料\算法作业\第4章作业\main.exe
哈夫曼编码如下：
e: 000
l: 0010
s: 0011
u: 01000
q: 0100100000
j: 01001000010
z: 01001000011
k: 010010001
x: 01001001
y: 0100101
w: 010011
n: 0101
p: 01100
m: 01101
r: 0111
i: 1000
o: 1001
a: 1010
f: 101100
d: 101101
c: 10111
v: 1100000
b: 1100001
g: 110001


C:\Users\Connor\Desktop\学习资料\算法作业\第4章作业\main.exe
j: 01001000010
z: 01001000011
k: 010010001
x: 01001001
y: 0100101
w: 010011
n: 0101
p: 01100
m: 01101
r: 0111
i: 1000
o: 1001
a: 1010
f: 101100
d: 101101
c: 10111
v: 1100000
b: 1100001
g: 110001
h: 11001
t: 1101
#: 111
采用哈夫曼编码，输入文本需要的存储比特数：4179
采用定长编码，输入文本需要的存储比特数：5075

4. （输入 3）单源最短路径

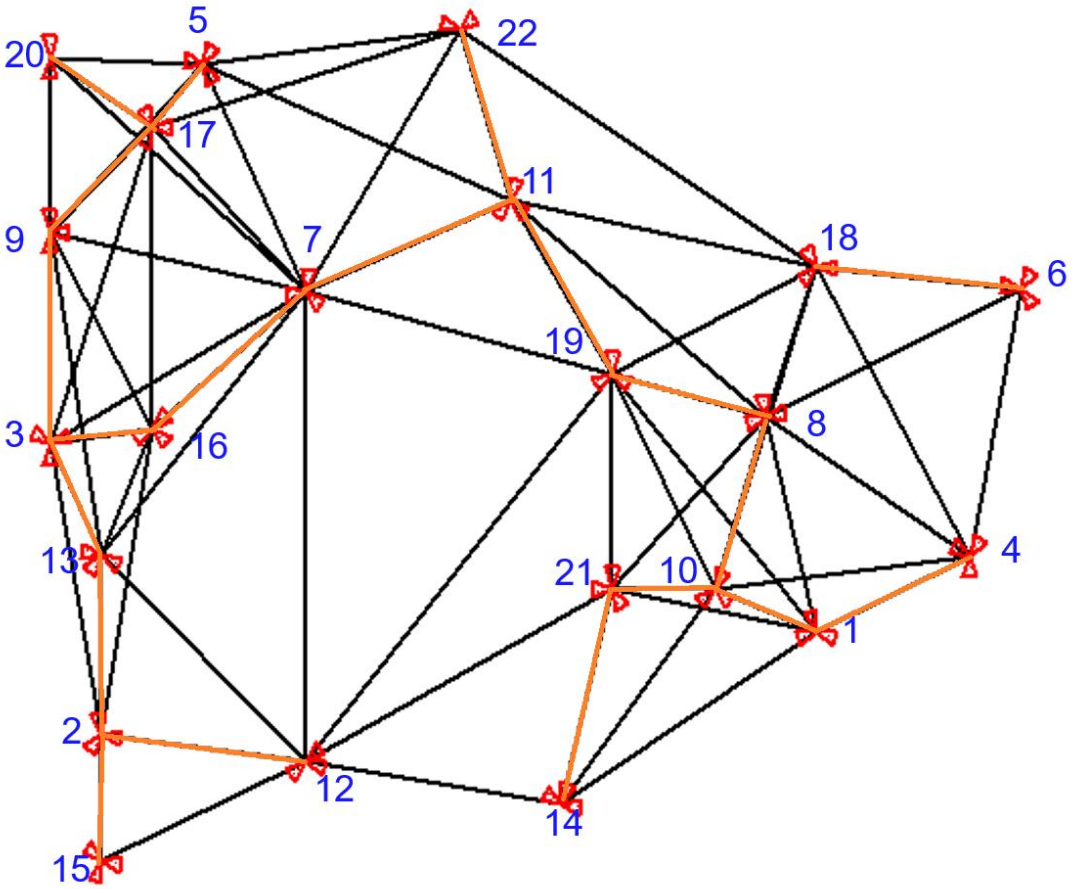

C:\Users\Connor\Desktop\学习资料\算法作业\第4章作业\main.exe
22个基站顶点组成的图：
567443到其它各点的单源最短路径：
567443->33109：1956.93
567443->565696：1343.41
567443->566631：761.938
567443->566720：2111.29
567443->566742：302.54
567443->566747：1988.14
567443->566750：683.088
567443->566751：1622.91
567443->566783：344.546
567443->566798：1778.06
567443->566802：963.852
567443->566967：1562.25
567443->566993：988.629
567443->566999：2072.92
567443->567203：1592.31
567443->567238：780.892
567443->567260：244.053
567443->567322：1582.91
567443->567439：1309.05
567443->567443：0
567443->567547：1733
567443->568098：810.555
567443到33109的最短路径：

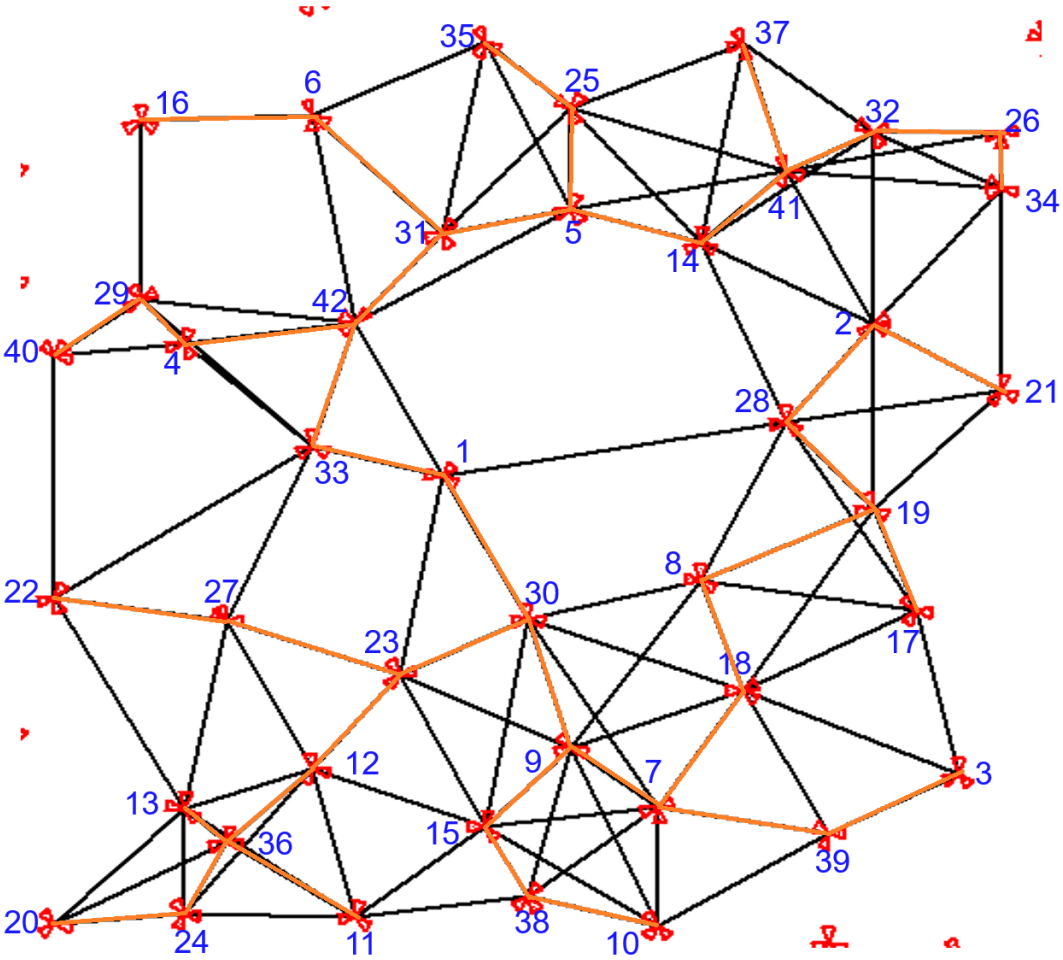

C:\Users\Connor\Desktop\学习资料\算法作业\第4章作业\main.exe
567443到33109的最短路径：
567443 -> 566750 -> 567439 -> 33109
42个基站顶点组成的图：
565845到其他各点的单源最短路径：
565845->565675: 1369.37
565845->565621: 1928.9
565845->565667: 2900.12
565845->567510: 645.041
565845->565801: 1153.11
565845->566010: 403.433
565845->567891: 2401.9
565845->565492: 2223.01
565845->565558: 2171.29
565845->565627: 2697.46
565845->565572: 2440.92
565845->565610: 2025.89
565845->565859: 2050.98
565845->565630: 1468.96
565845->565559: 2381.34
565845->565845: 0
565845->565527: 2594.34
565845->565633: 2347.84
565845->565496: 2308.24
565845->565865: 2489.07
565845->565773: 2281.46


C:\Users\Connor\Desktop\学习资料\算法作业\第4章作业\main.exe
565845->565865: 2489.07
565845->565773: 2281.46
565845->567531: 1402.79
565845->565516: 1918.1
565845->565393: 2339.03
565845->565753: 1122.45
565845->33566: 2169.68
565845->566074: 1573.64
565845->565648: 1997.17
565845->567526: 488.237
565845->565551: 1806.75
565845->565631: 843.923
565845->565608: 1883.38
565845->567500: 1055.67
565845->565531: 2161.48
565845->565562: 853.566
565845->32788: 2187.66
565845->567497: 1561.46
565845->566316: 2592.69
565845->568056: 2787.2
565845->565964: 741.608
565845->567618: 1655.16
565845->565898: 978.426
565845到565667的最短路径：
565845 -> 567526 -> 567500 -> 565675 -> 565551 -> 565633 -> 565667

5. （输入 4）最小生成树

```
C:\Users\Connor\Desktop\学习资料\算法作业\第4章作业\main.exe
3 单源最短路径
4 最小生成树
5 退出
4
------------------------------------------------------------------
22个基站顶点组成的图的最小生成树代价为： 6733.57
连接的边有：
1--4    1--10   2--12   2--13   2--15   3--9    3--13   3--16   5--17   6--18
7--11   7--16   8--10   8--18   8--19   9--17   10--21  11--19  11--22  14--21
17--20
42个基站顶点组成的图的最小生成树代价为： 13027
连接的边有：
1--30   1--33   2--21   2--28   3--39   4--29   4--42   5--14   5--25   5--31
6--16   6--31   7--9    7--18   7--39   8--18   8--19   9--15   9--30   10--38
11--36  12--23  12--36  13--36  14--41  15--38  17--19  19--28  20--24  22--27
23--27  23--30  24--36  25--35  26--32  26--34  29--40  31--42  32--41  33--42
37--41
------------------------------------------------------------------
请选择以下操作：
1 基于贪心法的凸多边形三角剖分
2 哈夫曼编码
3 单源最短路径
4 最小生成树
5 退出
```




35
37
16
6
25
32
26
41
34
31
5
14
29
42
2
40
4
21
28
1
33
19
8
22
27
30
17
23
18
12
9
7
3
13
36
15
39
20
24
11
38
10

6. （输入 5）退出