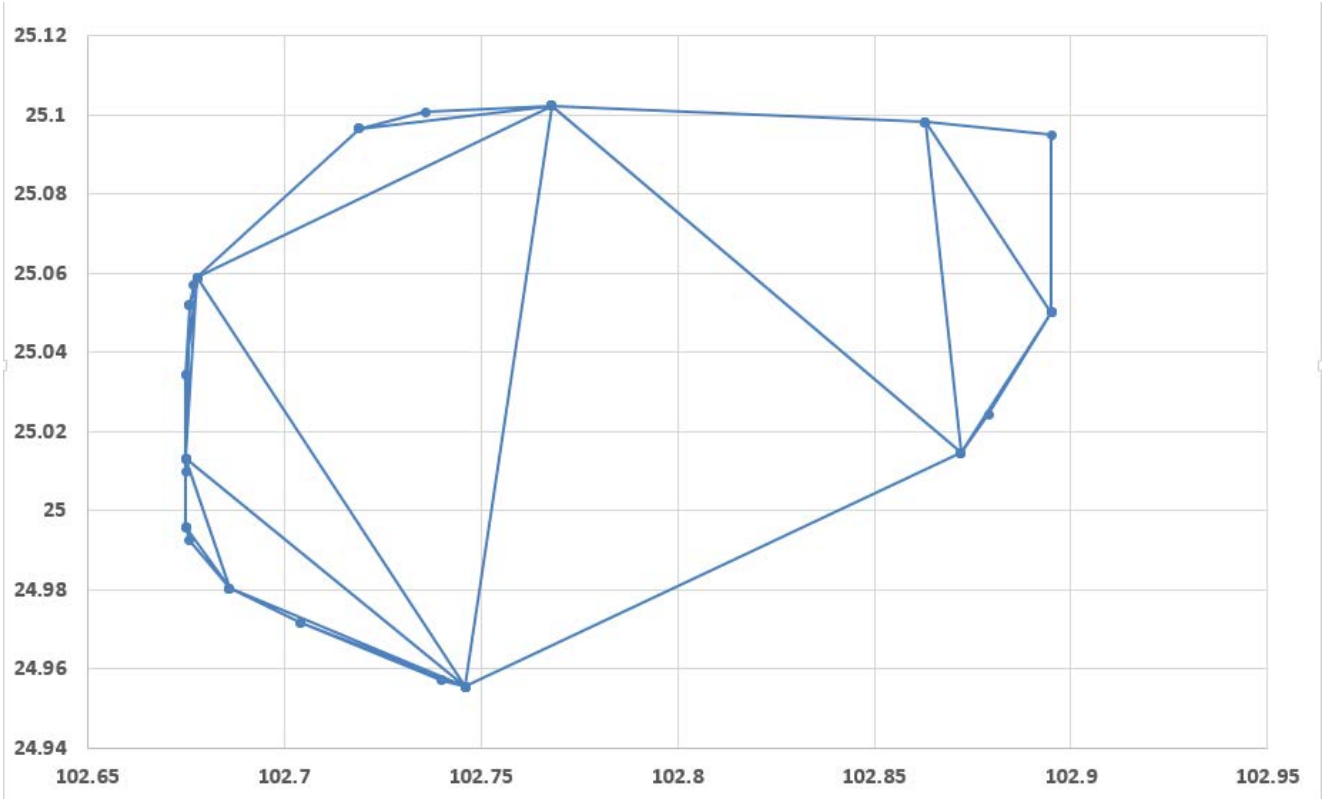
算法设计与分析——第四章作业

---学院 班级：-------- 姓名：-------- 学号：--------

● 运行结果

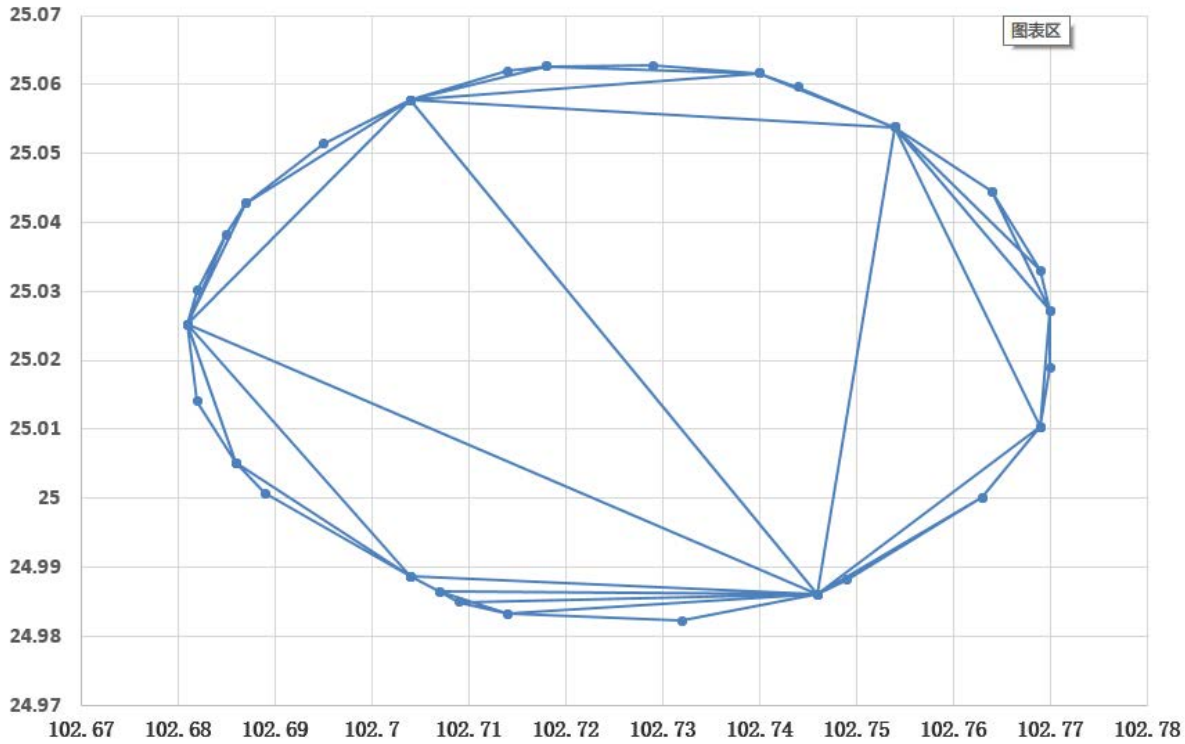
一、 基于贪心法的凸多边形次优三角剖分

1. 21 个基站凸多边形次优三角剖分的所有三角形如下：



次优三角剖分值为：297550

2. 29 个基站凸多边形次优三角剖分的所有三角形如下:



次优三角剖分值为：196426

二、 哈夫曼编码

字符	编码	编码长度
#	000	3
t	0010	4
h	00110	5
c	00111	5
d	010000	6
g	010001	6
b	0100100	7
v	0100101	7
f	010011	6
a	0101	4
o	0110	4
i	0111	4
r	1000	4
m	10010	5
p	10011	5
n	1010	4

w	101100	6
y	1011010	7
x	10110110	8
k	10110111	8
u	10111	5
s	1100	4
l	1101	4
e	111	3

编码后文字：

0111101001001101101000100100101110111011011010000010110100000101110101000101101000011100100011010010000011111000000101101010110100001011001111101110101000011101001101111010111010000000000111011010010100111011100100101001001110110101001011101000100111000011000111111010000101111000111000000100011001010010000000100101101101111111100000110001101001011100001001010101110110111111111000000101110000000111101010011101110010000001011010010000000010011100001100100001011100111111110000011000110100101110000100101010111011011111111100000010111000000011010111001010011101110010000101100111000000111010110100000101110111000110000010010101111111011000000101101000001011101010001011010000111001000110100100000101110000000101000000100110011011010000010011011010000000110001101101010010101111010010001000010100010110011111011101110010011111001110111010011011111110100000000011101101001010011101110010010100100111011010100101110100010011100001100100100110111111001000000000100011011100011000010010100101111001011110100010000011001001100000100011011100010011100001100100100110111111001000011001001111100111011101001101111111100000011110100000100011111010111100001011101000000101111000100101100000000100011011100001000011111000111100011101000000000111101010011101110010000011010111001010011101110010000100011111010101001001110110101011000011001111001100000000100011011100001011101010001011010000111001000110100100000100001111100001111000011101001001111110000000101000110010011111001110111010011011100111000001110110100101001110111001001010010011101101010010111010001001110000110001111110100001011110001110000100110110100000000101001110011001111110100101011110100100010000010001100101001000000111101010011101110010000011010111001010011101110010000100011111010101001001110110101011000011001111001100000101101000000101110101000101101000011100100011010010000011111000000110001010111010000000001001100000010010011100000011101101000100011100111001000000111010011000000010011011010000000111010010111110001011010000011110101001110111001000000111101011000010010110100011111100000000111001000000011001011101001011000000101100011100100011000000100011011100000011101101000100011100111001000001101011100101001110111001000000001011001110001100010110110100000001000110010100100000010100000111011010001000111001110010000001011101010001011010000111001000110100100001100011011010100101111110000000100011011100001000101110100101111110100000011101101001010011101110010010100100111011010100101110100010011100001100100100110111111001000000000101101000000111101000111011010001000111001110010000001011101010001011010000111001000110100100001001001110100010011000100000101001100010000000110010111010010000001010010000001011101110100001101010000011000110100101110000111101010011101110010000011110101100001001011 01

0001111111100000000011010000000111001000010010011101000100110001000000110010111010010000101100011100100011000000101101000000101101011001011001111000000011000100011011111000000001000110010110100000001000110111000010000111110001111000111010000000011010101110000000011101101010001010000101100010110100000001001100001011000011001010010000001101010111000100100111010001001100010000111101101101001111100111001000000001111010001110110100010001110011100100000101110101000101101000011100100011010010110000000111010110100001100011010010111001001111001011111000000010010011100010111110011101001110111110100000001110100110000010001101110111100000011110001000011010000000100001010010111000001110101101000000100100111000001110110101000101000011011011101111010000000000101100111000110000110010111011101000110011111100001011010000111101101100101100101001111011110000110010011000000100011001111110000001111010000000111001100101100110010111100000010110000110111101000010110011100011000010101110100001011010000001011101010001011010000111001000110100101100000001001101101000000010011011110100100000111101001000100011010101100001000111100010011100001111001011100010101011110010010010011110001100000000011010000100000111101001011000011111011011010000000001100110101100111010010111110000000001011001110001100001100101110111010000100100111000001110110101000111111100010101110100000000110101011011011010000101100011100100011000000111011010001000111001110010000010111010100010110100000111001000110100101100

最终编码后总长度：4184

三、 单源最短路径

1. 22 个基站顶点组成的图，以基站 567443（第 20 个点）为源点，

   到其余各点的最短路径长度：

33109
1956.93

565696
1343.41

566631
761.938

566720
2111.29

566742
302.54

566747
1988.14

566750
683.088

566751
1622.91

566783
344.546

566798
1778.06

566802
963.852

566967
1562.25

566993
988.629

566999
2072.92

567203
1592.31

567238
780.892

567260
244.053

567322
1582.91

567439
1309.05

567443
0

567547
1733

568098
810.555

567443 到 33109 的最短路径：567443 566750 567439 33109

2. 42 个基站顶点组成的图，以基站 565845（第 16 个点）为源点，到其余各点的最短路径长度：

565675
1369.37

565492
2223.01

565621
1928.9

565558
2171.29

565667
2900.12

565627
2697.46

567510
645.041

565572
2440.92

565801
1153.11

565610
2025.89

566010
403.433

565859
2050.98

567891
2401.9

565630

1468.96

565559
2381.34

565845
0

565527
2594.34

565633
2347.84

565496
2308.24

565865
2489.07

565773
2281.46

567531
1402.79

565516
1918.1

565393
2339.03

565753
1122.45

33566
2169.68

566074
1573.64

565648
1997.17

567526
488.237

565551
1806.75

565631
843.923

565608
1883.38

567500
1055.67

565531
2161.48

565562
853.566

32788
2187.66

567497
1561.46

566316
2592.69

568056
2787.2

565964
741.608

567618
1655.16

565898
978.426

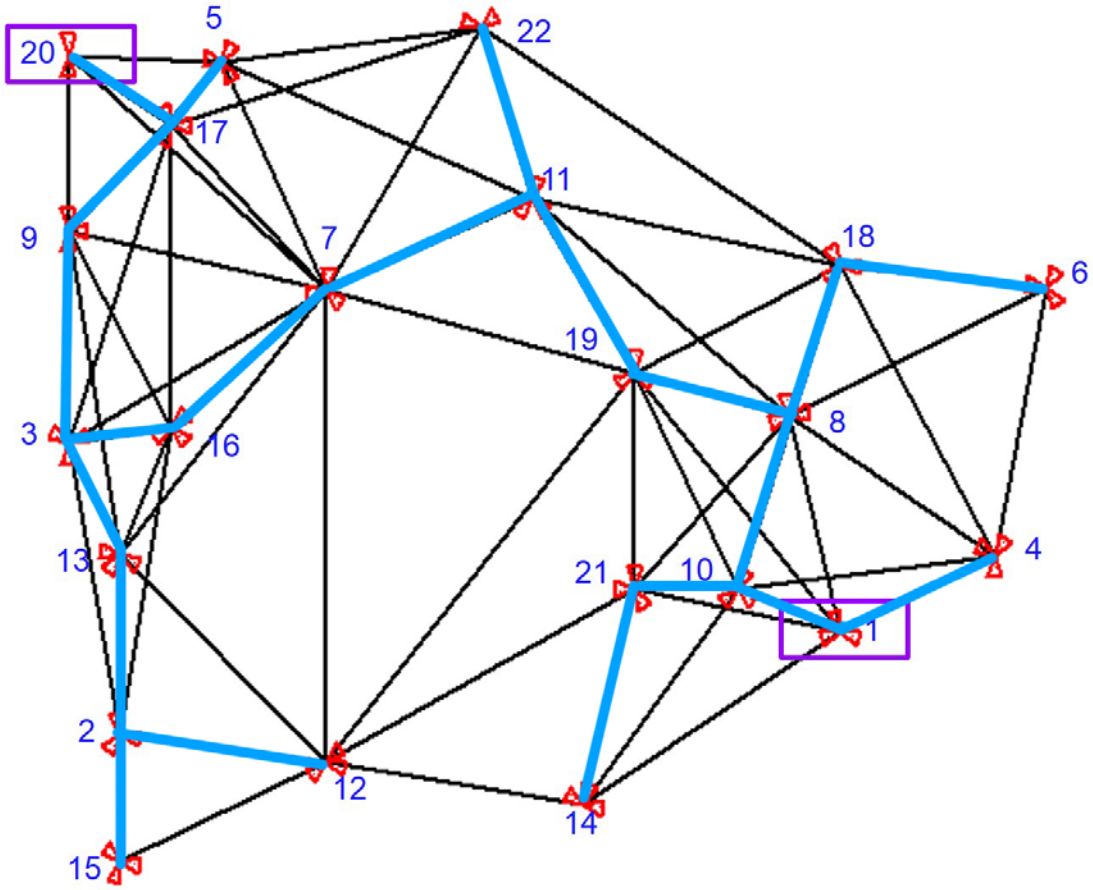
565845 到 565667 的最短路径：565845 567526 567500 565675 565551 565633 565667

四、 最小生成树

1. 22 个基站顶点组成的图

从节点 1 开始：

(1,10)
(10,21)
(1,4)
(10,8)
(8,18)
(8,19)
(19,11)
(11,22)
(18,6)
(21,14)
(11,7)
(7,16)
(16,3)
(3,13)
(13,2)
(2,15)
(2,12)
(3,9)
(9,17)
(17,5)
(17,20)

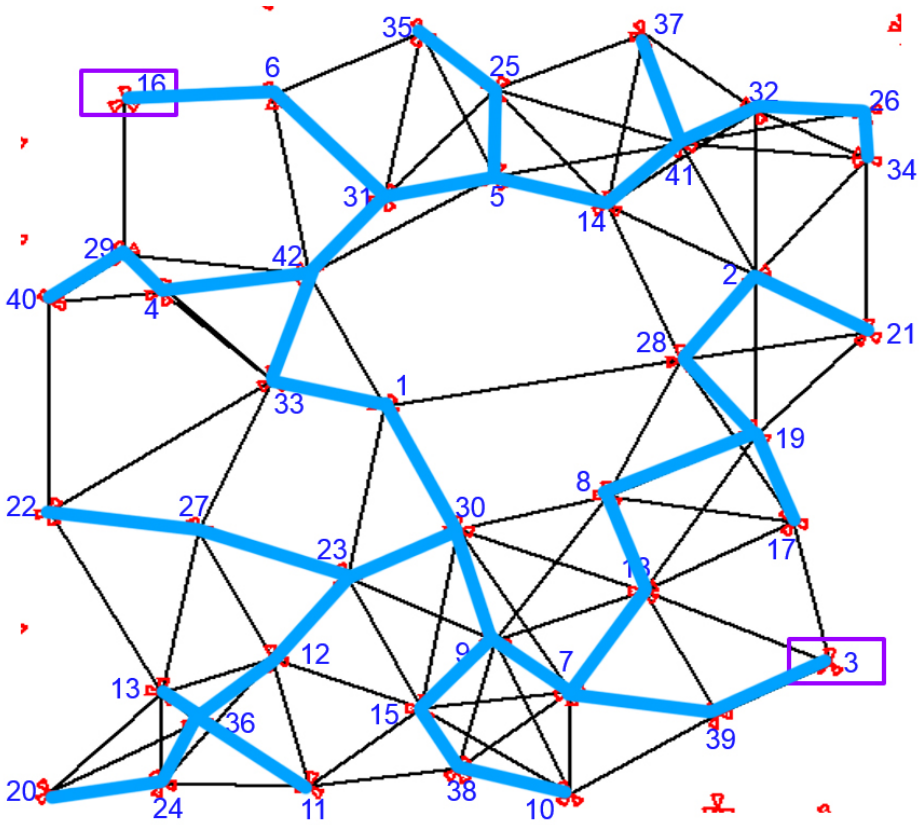


生成树的总长为：6733.57

2. 42 个基站顶点组成的图

从节点 1 开始：

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



生成树的总长为：13027

- 源程序代码

一、 基于贪心法的凸多边形最优三角剖分

```
#include <iostream>
#include <cmath>
#include "libxl.h" //用于读取excel文件

using namespace libxl; //用于读取excel文件
using namespace std;

#define NUM1 21 //第一个文件是凸21边形，即21个顶点
#define NUM2 29 //第二个文件是凸29边形，即29个顶点
#define RADIUM 6378137 //半径
const double PI = acos(-1.0); //常数PI

struct baseData{ //定义基站数据的结构
    int num; //序号
    int enodebid; //基站编号
    double longitude, latitude;  //精度和纬度
};
```

```
18
19  //数据和文件处理
20  int readData(Book* book, struct baseData data[], wchar_t
21  loadFileName[], int n);
22  //求凸多边形的最优三角剖分对应的各边的权值
23  double minWeightTriangulation(int n, struct baseData data[],
24  int &vex);
25  double dist(struct baseData a, struct baseData b); // dist用于
26  计算Va和Vb的距离
27  double w(struct baseData * data, int a, int b, int c); //w用于
28  计算Va,Vb和Vc构成的三角形的权值
29  //通过s[][]来构造最优三角剖分中的子三角形
30  void Traceback(int n, struct baseData data[], double &sum);
31
32  int main(void){
33      //从excel文件中读取数据
34      Book* book = xlCreateBook();
35      if (!book){
36          cout << "Error when init book." << endl;
37          return -1;
38      }
39      struct baseData data1[30];
40      wchar_t loadFileName1[] = L"附件3-1.21个基站凸多边形数据.xls";
41      if (readData(book, data1, loadFileName1, NUM1) <= 0)
42          return -2;
43      double sum = 0;
44      cout << "21 个基站凸多边形最优三角剖分的所有三角形如下:" << endl;
45      Traceback(20, data1, sum);//计算最优三角划分并将结果进行输出
46
47      //读取第二份数据“29个基站凸多边形数据”然后计算29个基站的凸多边形最优
48  三角剖分
49      Book* book2 = xlCreateBook();
50      if (!book2){
51          cout << "Error when init book." << endl;
52          return -1;
53      }
54      struct baseData data2[30];
55      wchar_t loadFileName2[] = L"附件3-2.29个基站凸多边形数据.xls";
56      if (readData(book2, data2, loadFileName2, NUM2) <= 0)
57          return -2;
58      cout << endl << "29 个基站凸多边形最优三角剖分的所有三角形如下:"
59  << endl;
60      Traceback(28, data2, sum);//计算最优三角划分并将结果输出
61
```

```
62      system("PAUSE");
63      return 0;
64  }
65
66
67  double minWeightTriangulation(int n, struct baseData data[],
68  int &vex){ //求出边长最小的相邻三点组成的三角形的边长并将三角形的一个顶
69  点返回
70      double min;
71      min = w(data, 0, 1, 2); //先将V0,V1,V2组成的三角形组成的三角形设
72  为最小值
73      vex = 0;
74      for (int i = 1; i <= n - 4; i++){
75          if (min > w(data, i, i + 1, i + 2))
76          {
77              min = w(data, i, i + 1, i + 2);
78              vex = i;
79          }
80      }
81      //在遍历完V0V1V2,V1V2V3,...,Vn-3Vn-2Vn-1以后
82      if (min > w(data, n - 2, n - 1, 0)){ //看Vn-2Vn-1V0是否是最小
83  值
84          min = w(data, n - 2, n - 1, 0);
85          vex = n - 2;
86      }
87      if (min > w(data, n - 1, 0, 1)){ //看Vn-1V0V1是否是最小值
88          min = w(data, n - 1, 0, 1);
89          vex = n - 1;
90      }
91      return min;
92  }
93
94  void Traceback(int n, struct baseData data[], double &sum){
95      sum = 0;
96      int i = 0;
97      for (; n > 3; n--){
98          sum += minWeightTriangulation(n, data, i); //在求出最小的
99  三角形边长以及对应顶点后
100         if (i == n - 2) //如果是第n-2个点则直接输出
101             cout << data[n - 2].num << "\t" << data[n - 1].num <<
102 "\t" << data[0].num << endl;
103         else if (i == n - 1){ //如果是第n-1个点则在输出后将data[0]删
104 除
105             cout << data[n - 1].num << "\t" << data[0].num <<
```

```
"\t" << data[1].num << endl;
            for (int j = 0; j < n - 1; j++)
                data[j] = data[j + 1];
        }
        else{//其他情况则在输出后将data[i+1]删除
            cout << data[i].num << "\t" << data[i + 1].num <<
"\t" << data[i + 2].num << endl;
            for (int j = i + 1; j < n - 1; j++)
                data[j] = data[j + 1];
        }
    }
    sum += w(data, 0, 1, 2); //将最后的V0V1V2加进去
    cout << data[0].num << "\t" << data[1].num << "\t" <<
data[2].num << endl; //输出V0V1V2
    cout << "最优三角剖分值为: " << sum << endl << endl; //输出结果
}

double w(struct baseData * data, int a, int b, int c){ //计算三
角形VaVbVc的三边之和
    return (dist(data[a], data[b]) + dist(data[a], data[c]) +
dist(data[b], data[c]));
}

double dist(struct baseData a, struct baseData b){ //已知经度和
纬度求距离
    return RADIUM*acos(cos(a.latitude*PI /
180)*cos(b.latitude*PI / 180)*cos(a.longitude*PI / 180 -
b.longitude*PI / 180) + sin(a.latitude*PI /
180)*sin(b.latitude*PI / 180));
}

//数据与文件处理
int readData(Book* book, struct baseData data[], wchar_t
loadFileName[], int

    n){ //将数据从excel文件中读出
    if (book->load(loadFileName)){ //读取book
        cout << "已读取文件。" << endl;
    }
    else{
        cout << "读取文件时错误。" << endl;
        return -2;
    }
    Sheet* sheet = book->getSheet(1);//读取excel文件中的sheet2
```

```
150     if (sheet){ //将sheet中的数据复制到结构数组中
151         for (int i = 0; i < n; i++){
152             data[i].num = i + 1;
153             data[i].enodebid = (int)sheet->readNum(i + 1, 0);
154             data[i].longitude = sheet->readNum(i + 1, 1);
155             data[i].latitude = sheet->readNum(i + 1, 2);
156         }
157     }
158     book->release();
159     return 1;
160 }
```

二、 哈夫曼编码

```
#include <iostream>
#include <cstdlib>
#include <cstring>
#include <cstdbool>

using namespace std;

struct HuffmanNode * minValue(struct HuffmanNode * freqTree);
void printTree(struct HuffmanNode * treeNode, int layer, char
a[]);

struct HuffmanNode{ //在编排哈夫曼树时所用的节点
    char text; //字符内容
    int weight; //出现频率
    bool leaf; //该节点在哈夫曼树中是否是叶子节点
    struct HuffmanNode * lchild; //左孩子
    struct HuffmanNode * rchild; //右孩子
    struct HuffmanNode * next; //在链表中指向下一个节点
};

struct HuffmanCode{ //27个字符的哈夫曼编码
    char code[20]; //编码
    int length; //编码长度
};

int value = 0; //编码总长度
struct HuffmanCode code[27]; //27个字符的哈夫曼编码（包含26个小写字
母和'#'号）

int main(void){
    //首先将编码源文字读出
    FILE * fileOne;
    fileOne = fopen("1.txt", "r");
    char text[1020];
    fgets(text, 1020, fileOne);
    printf("%s", text);

    //然后定义一个哈夫曼编码的链表并初始化
    struct HuffmanNode * freqTree = new HuffmanNode;
    freqTree->weight = INFINITY;
    freqTree->text = 0;
    freqTree->next = NULL;
```

```
43
44      struct HuffmanNode * lastNode = freqTree;
45
46      //接下来将所有的字符都读入链表中，每一种字符对应一个节点
47      for (int i = 0; text[i] != '\n'; i++){
48          while (lastNode->next != NULL){
49              if (lastNode->next->text == text[i])
50                  break;
51              lastNode = lastNode->next;
52          }
53          if (lastNode->next == NULL){
54              lastNode->next = new HuffmanNode;
55              lastNode = lastNode->next;
56              lastNode->leaf = true;
57              lastNode->text = text[i];
58              lastNode->weight = 1;
59              lastNode->lchild = NULL;
60              lastNode->rchild = NULL;
61              lastNode->next = NULL;
62          }
63          else {
64              lastNode->next->weight++;
65          }
66          lastNode = freqTree;
67      }
68      while (lastNode->next != NULL){
69          cout << lastNode->next->text << ' ' <<
70  lastNode->next->weight << endl;;
71          lastNode = lastNode->next;
72      }
73
74      //将哈夫曼链表按照频率重新组合成哈夫曼树
75      lastNode = freqTree;
76      while (lastNode->next != NULL){
77          //在链表里找到最小频率的节点
78          struct HuffmanNode * minLastNode1 = minValue(freqTree);
79          struct HuffmanNode * minValueNode1 = minLastNode1->next;
80          minLastNode1->next = minLastNode1->next->next;
81          //去掉最小以后找到次小频率的节点
82          struct HuffmanNode * minLastNode2 = minValue(freqTree);
83          struct HuffmanNode * minValueNode2 = minLastNode2->next;
84          minLastNode2->next = minLastNode2->next->next;
85          //设一个新节点，将以上节点并在左右孩子下面，然后将这个节点放回链表
86  中
```

```
87          struct HuffmanNode * treeNode = new struct HuffmanNode;
88          treeNode->leaf = false;
89          treeNode->text = 0;
90          if (minValueNode1->weight >= minValueNode2->weight){
91              treeNode->lchild = minValueNode1;
92              treeNode->rchild = minValueNode2;
93          }
94          else{
95              treeNode->lchild = minValueNode2;
96              treeNode->rchild = minValueNode1;
97          }
98          treeNode->next = NULL;
99          treeNode->weight = minValueNode1->weight +
100 minValueNode2->weight;
101         if (freqTree->next == NULL){
102             freqTree->next = treeNode;
103             break;
104         }
105         else{
106             struct HuffmanNode * temp = freqTree;
107             while (temp->next != NULL)
108                 temp = temp->next;
109             temp->next = treeNode;
110         }
111     }
112     cout << endl << endl;
113     char a[20];
114     //将编码输出
115     printTree(freqTree->next, 0, a);
116
117     //将编码后的结果输出
118     cout << endl << value << endl;
119     cout << endl << endl;
120     for (int i = 0; text[i] != '\0'; i++){
121         if (text[i] == '#'){
122             cout << code[0].code;
123         }
124         else{
125             cout << code[text[i] - 96].code;
126         }
127     }
128
129     fclose(fileOne);
130     system("PAUSE");
```

```
131     return 0;
132 }
133 
134 struct HuffmanNode * minValue(struct HuffmanNode *
135 freqTree){ //在链表中找到最小值
136     struct HuffmanNode * lastNode = freqTree;
137     struct HuffmanNode * minLastNode = freqTree;
138     while (lastNode->next != NULL){
139         if (lastNode->next->weight <=
140 minLastNode->next->weight){//!!
141             minLastNode = lastNode;
142         }
143         lastNode = lastNode->next;
144     }
145     return minLastNode;
146 }
147 
148 void printTree(struct HuffmanNode * treeNode, int layer, char
149 a[]){ //递归对哈夫曼树中的节点进行编码并输出
150     if (treeNode->leaf == true){
151         cout << treeNode->text << ' ' << treeNode->weight <<
152 endl;
153         value += treeNode->weight*layer;
154         if (treeNode->text == '#'){
155             strcpy(code[0].code, a);
156             code[0].length = layer;
157             cout << code[0].code << ' ' << code[0].length <<
158 endl;
159         }
160         else{
161             strcpy(code[treeNode->text - 96].code, a);
162             code[treeNode->text - 96].length = layer;
163             cout << code[treeNode->text - 96].code << ' ' <<
164 code[treeNode->text - 96].length << endl;
165         }
166     }
167     else{
168         if (treeNode->lchild){
169             a[layer] = '0';
170             a[layer + 1] = '\0';
171             printTree(treeNode->lchild, layer + 1, a);
172         }
173         if (treeNode->rchild){
174             a[layer] = '1';
```

```
175             a[layer + 1] = '\0';
176             printTree(treeNode->rchild, layer + 1, a);
177         }
178     }
179 }
```

三、 单源最短路径

```
#include <iostream>
#include <cstdbool>
#include <cstdio>
#include <cmath>
#include "libxl.h" //用于读取excel文件

using namespace libxl; //用于读取excel文件
using namespace std;

#define NUM1 22 //第一个文件是22个基站的邻接矩阵
#define NUM2 42 //第二个文件是42个基站的邻接矩阵

//数据和文件处理
int readData(Book* book, int id[], double arc[][50], wchar_t
loadFileName[], int sheetNum, int n);
//迪杰斯特拉算法
void shortestPath_DIJ(double arc[][50], int v0, bool p[][50],
double d[], int num, int prev[]);
//输出最短路径
void printOut(int prev[], int start, int end);

int main(void){
    //从excel文件中读取数据
    Book* book = xlCreateBook();
    if (!book){
        cout << "Error when init book." << endl;
        return -1;
    }

    int id[50];
    double arc[50][50];

    //读取数据
    wchar_t loadFileName1[] = L"附件1-1.基站图的邻接矩阵-v1.xls";
    if (readData(book, id, arc, loadFileName1, 1, NUM1) <= 0)
        return -2;

    bool p[50][50];
    double d[50];
    int prev[50];
    //迪杰斯特拉算法求22个基站中一点到其他各点的最短路径
    shortestPath_DIJ(arc, 19, p, d, NUM1, prev);
```

```
43      //输出第20个点（567443）到第1个点（33109）的最短路径
44      printOut(prev, 19, 0);
45
46      Book* book2 = xlCreateBook();
47      if (!book2){
48          cout << "Error when init book." << endl;
49          return -1;
50      }
51      if (readData(book2, id, arc, loadFileName1, 0, NUM2) <= 0)
52          return -2;
53      //迪杰斯特拉算法求42个基站中一点到其他各点的最短路径
54      shortestPath_DIJ(arc, 15, p, d, NUM2, prev);
55      //输出第16个点（565845）到第2个点（565667）的最短路径
56      printOut(prev, 15, 2);
57
58      system("PAUSE");
59      return 0;
60  }
61
62  void shortestPath_DIJ(double arc[][50], int v0, bool p[][50],
63  double d[], int num, int prev[]) {
64      //迪杰斯特拉求V0到其余各点的最短路径，arc是边长邻接矩阵，V0是源点，
65  p[v][w]为true是指V0-V的路径上有w点，d[]是最短路径的长度
66      //num是点的数量，prev[]是最短路径沿途各点，通过prev可以按序倒推最短
67  路径
68
69      //首先用V0初始化各个参数
70      bool final[50];
71      for (int i = 0; i < num; i++){
72          final[i] = false;
73          d[i] = arc[v0][i];
74          for (int j = 0; j < num; j++)
75              p[i][j] = false;
76          if (d[i] < 10000){
77              p[i][v0] = true;
78              p[i][i] = true;
79          }
80          if (d[i] == 10000)
81              prev[i] = 0;
82          else
83              prev[i] = v0;
84      }
85      d[v0] = 0;
86      final[v0] = true;
```

```
87
88      //然后求出V0到各点中路径最小值的点V，用V0-V来更新各点的最短路径，将结
89  果记录下来
90      for (int i = 1; i < num; i++){
91          double min = 10000;
92          int v = 0;
93          for (int w = 0; w < num; w++)
94              if (!final[w])
95                  if (d[w] < min){
96                      v = w;
97                      min = d[w];
98                  }
99          final[v] = true;
100         for (int w = 0; w < num; w++)
101             if (!final[w] && (min + arc[v][w] < d[w])){
102                 //如果min+arc[v][w]<d[w]，说明V0-V-W为新的较短路径，更
103 新记录并将prev[w]记录为v点
104                 d[w] = min + arc[v][w];
105                 for (int i = 0; i < num; i++)
106                     p[w][i] = p[v][i];
107                 p[w][w] = true;
108                 prev[w] = v;
109             }
110     }
111
112     //输出结果
113     for (int i = 0; i < num; i++){
114         cout << "Port " << i << endl;
115         for (int j = 0; j < num; j++)
116             if (p[i][j] == true)
117                 cout << j << " ";
118         cout << endl << d[i] << endl;
119     }
120     cout << endl;
121 }
122
123 //数据与文件处理
124 int readData(Book* book, int id[], double arc[][50], wchar_t
125 loadFileName[], int sheetNum, int n){ //将数据从excel文件中读出
126     if (book->load(loadFileName)){ //读取book
127         cout << "已读取文件。" << endl;
128     }
129     else{
130         cout << "读取文件时错误。" << endl;
```

```
131         return -2;
132     }
133     Sheet* sheet = book->getSheet(sheetNum);//读取excel文件中的
134 sheet2
135     if (sheet){ //将sheet中的数据复制到结构数组中
136         for (int i = 0; i < n; i++){
137             id[i] = (int)sheet->readNum(1, i + 2);
138         }
139         for (int i = 0; i < n; i++)
140             for (int j = 0; j < n; j++){
141                 arc[i][j] = sheet->readNum(i + 2, j + 2);
142                 if (arc[i][j] == -1)
143                     arc[i][j] = 10000;
144             }
145     }
146     book->release();
147     return 1;
148 }
149
150 void printOut(int prev[], int start, int end){ //递归倒推出最短路
151 径，从end到start
152     cout << end << ' ';
153     if (start != end){
154         printOut(prev, start, prev[end]);
155     }
156 }
```

四、 最小生成树

```
#include <iostream>
#include <cstdio>
#include <cmath>
#include "libxl.h" //用于读取excel文件

using namespace libxl; //用于读取excel文件
using namespace std;

#define NUM1 22
#define NUM2 42

struct closeEdgeNode{
    int adjvex;
    double lowcost;
};

//数据和文件处理
int readData(Book* book, int id[], double arc[][50], wchar_t
loadFileName[], int sheetNum, int n);
//普里姆算法求最小生成树
void MiniSpanTree_PRIM(double arc[][50], int id[], int num);

int main(void){
    //从excel文件中读取数据
    Book* book = xlCreateBook();
    if (!book){
        cout << "Error when init book." << endl;
        return -1;
    }
    int id[50];
    double arc[50][50];
    wchar_t loadFileName1[] = L"附件1-1.基站图的邻接矩阵-v1.xls";
    if (readData(book, id, arc, loadFileName1, 1, NUM1) <= 0)
        return -2;
    //普里姆算法求22个基站的最小生成树
    MiniSpanTree_PRIM(arc, id, NUM1);

    Book* book2 = xlCreateBook();
    if (!book2){
        cout << "Error when init book." << endl;
        return -1;
    }
```

```
    if (readData(book2, id, arc, loadFileName1, 0, NUM2) <= 0)
        return -2;
    //普里姆算法求42个基站的最小生成树
    MiniSpanTree_PRIM(arc, id, NUM2);
    system("PAUSE");
    return 0;
}

void MiniSpanTree_PRIM(double arc[][50], int id[], int num){
    //普里姆算法求最小生成树
    int u;
    double sum = 0;
    printf("Please input the starting vertex of minimum
spanning tree:");
    cin >> u;
    struct closeEdgeNode closedge[50];
    //用点u来初始化各个节点的closedge
    int k = u - 1;
    for (int j = 0; j < num; j++)
    {
        if (j != k){
            closedge[j].adjvex = k;// id[k];
            closedge[j].lowcost = arc[k][j];
        }
    }
    closedge[k].lowcost = 0;

    //求出closedge中的最小值，将此点加入最小生成树S中，然后用该点到其他点
的值更新closedge
    for (int i = 1; i < num; i++){
        int b = 1;
        k = 0;
        for (double min = closedge[0].lowcost; b < num; b++){
            if (min == 0 && closedge[b].lowcost != 0){
                min = closedge[b].lowcost;
                k = b;
            }
            else if (closedge[b].lowcost != 0 &&
closedge[b].lowcost<min){
                min = closedge[b].lowcost;
                k = b;
            }
        }
        printf("(%d,%d)\n", closedge[k].adjvex + 1, k + 1);
```

```
87          sum += arc[closedge[k].adjvex][k];
88          closedge[k].lowcost = 0;
89          for (int j = 0; j < num; j++)
90             if (arc[k][j] < closedge[j].lowcost)
91             {
92                closedge[j].adjvex = k;
93                closedge[j].lowcost = arc[k][j];
94             }
95
96       }
97       cout << "The sum is " << sum << endl;
98    }
99
100   //数据与文件处理
101   int readData(Book* book, int id[], double arc[][50], wchar_t
102   loadFileName[], int sheetNum, int n){ //将数据从excel文件中读出
103      if (book->load(loadFileName)){ //读取book
104         cout << "已读取文件。" << endl;
105      }
106      else{
107         cout << "读取文件时错误。" << endl;
108         return -2;
109      }
110      Sheet* sheet = book->getSheet(sheetNum);//读取excel文件中的
111   sheet
112      if (sheet){ //将sheet中的数据复制到结构数组中
113         for (int i = 0; i < n; i++){
114            id[i] = (int)sheet->readNum(1, i + 2);
115         }
116         for (int i = 0; i < n; i++)
117            for (int j = 0; j < n; j++){
118               arc[i][j] = sheet->readNum(i + 2, j + 2);
119               if (arc[i][j] == -1)
120                  arc[i][j] = 10000;
121            }
122      }
123      book->release();
124      return 1;
125   }
126
```