算法设计与分析——第三章作业

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

- **运行结果**

一、 最长公共子序列

1. B 的结果为：

   5a22n7#85a0l56g52o7r33752t41m68#44s#7a0n1746#803e8l05l36706534605816441678 3m8p3u64t036i53on512l5#217r245242826670806522t03756#7685a12ke624704o7me0#6v4a044u2e62#0a0s71#38i2258pu5t0#415a8n5d3#100708o45u7c2e0s4#s7707m38115812577u8e5388s#77701u13p04ut

2. C-D 的结果为：

   35860525785351o710r1267447t57h00m66801is682#518h750343060656816441648660e38035663186502o51521724524282667080652203756768st325p474s30#708086t4274462030718527871303581817741547 2167#24745307n8p53u46737#40i581257783087h0e#137o3u0t37p42

3. A-D 的结果为：

   3585n7#65al771g1752067447566573102566328140634005144#6834836664073804e41186562o12152064t6708t6i52o257a2l5#6127625347437866474#2826037524768535410563 5m1107608463653781302224645077577867 1737#40458412272853o0885#2780e18734813777ut4

二、 最大字段和

1. A 的最大字段和是 2715，从第 43 个数字到第 329 个数字。

   A 的最大字段和所对应的字段如下：

   64 87 99 39 31 9 99 -2 -7 83 -46 8 16 55 -88 31 -96 51 -60 90 -13 80 50 -88 -9 -84 95 68 -23 24 53 -94 91 60 -34 -19 -53 -40 13 -31 -35 70 25 38 65 49 -99 68 -18 17 79 70 11 -93 93 -24 13 74 70 20 -2 66 97 -20 -56 89 5 -86 87 -56 53 60 73 15 -83 -73 -11 59 -85 87 -24 -81 79 70 -12 29 -4 63 -58 -48 94 20 -68 -10 76 97 72 -56 -45 -96 3 53 60 13 97 65 22 78 99 -12 68 -13 24 -73 -89 22 61 -31 73 5 2781 -85 55 68 -56 43 60 -19 -23 77 -91 -61 -57 22 -39 -64 29 41 -15 -43 -43 -4 -47 49 -21 66 0 56 45 71 -16 -35 68 60 -26 98 -22 -62 56 51 -63 -83 -62 -48 -33 911 5 57 93 35 32 -80 -54 -87 -82 -96 39 93 -89 50 29 47 7 -13 80 23 -85 -38 3 25 36 31 92 46 82 -23 -46 91 89 -40 76 -12 53 -88 -74 27 49 14 42 -60 -32 -43 -1865 -57 27 27 46 68 -29 63 84 -9 40 -42 -4 -32 -35 82 19 35 -15 84 76 -28 -42 -99 39 79 -54 -9 98 -77 95 -82 -60 -86 3 0 -85 70 -80 33 0 57 73 94 -50 -91 -46 0 42 -98 43 68 -18 -4 25 32 65 -29 -62 -76 78 12 -30 -10 61 94 92 -67 20 -51 33 95

2. B 的最大字段和是 377，从第 71 个数字到第 142 个数字。

   B 的最大字段和所对应的字段如下：

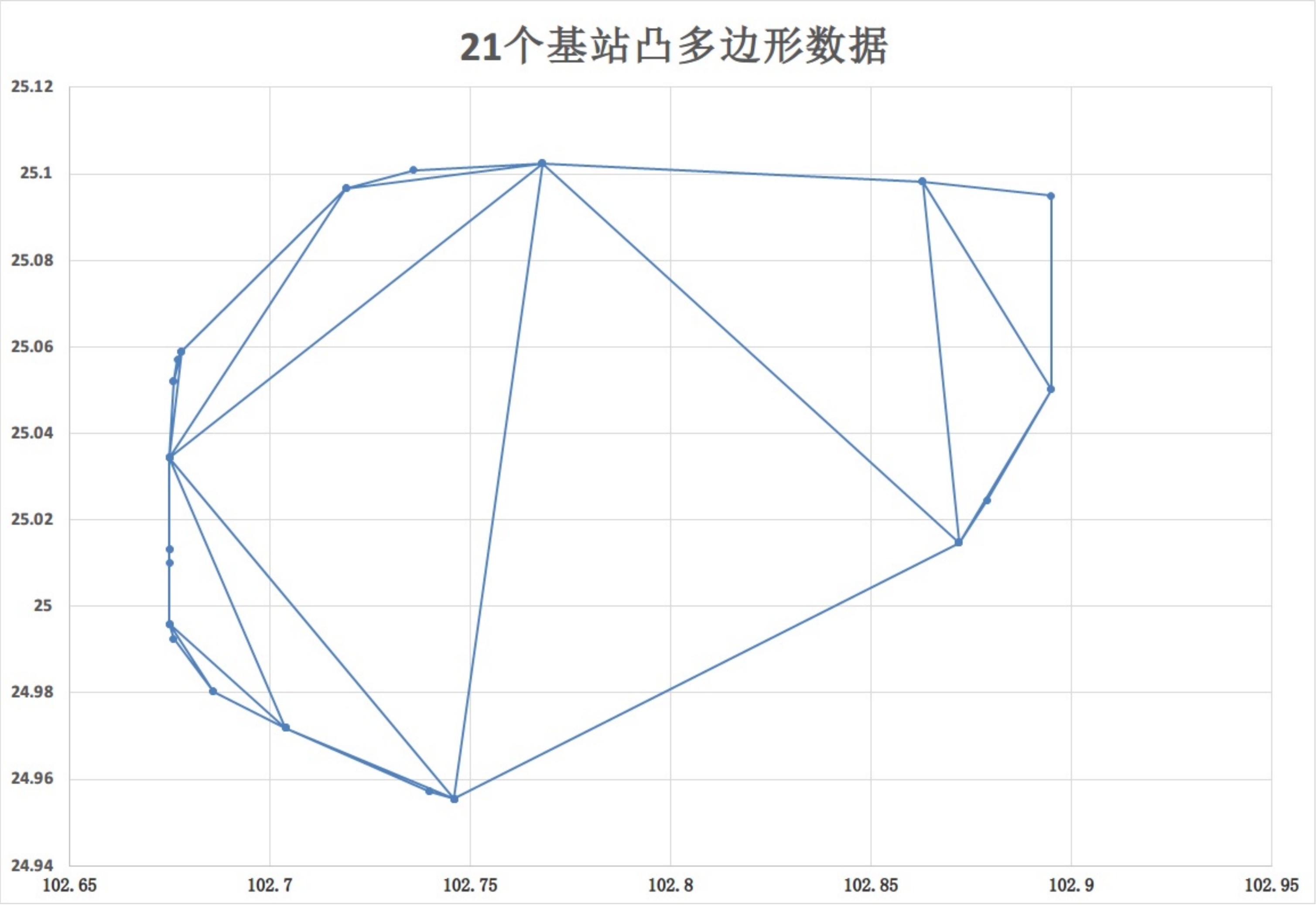
   0 34 1 -5 40 8 2 6 23 30 42 -4 45 -25 -23 -22 34 -13 -11 -12 16 44 -3 -11 -7 -30 34 49 -47 1 -21 -37 14 33 -37 28 -33 15 -36 36 27 -8 -31 24 -16 -7 38 24 34 48-27 -22 5 33 9 -26 -2 48 -20 22 38 -42 4 5 -49 10 47 -6 27 8 -10 34

三、　凸多边形最优三角剖分

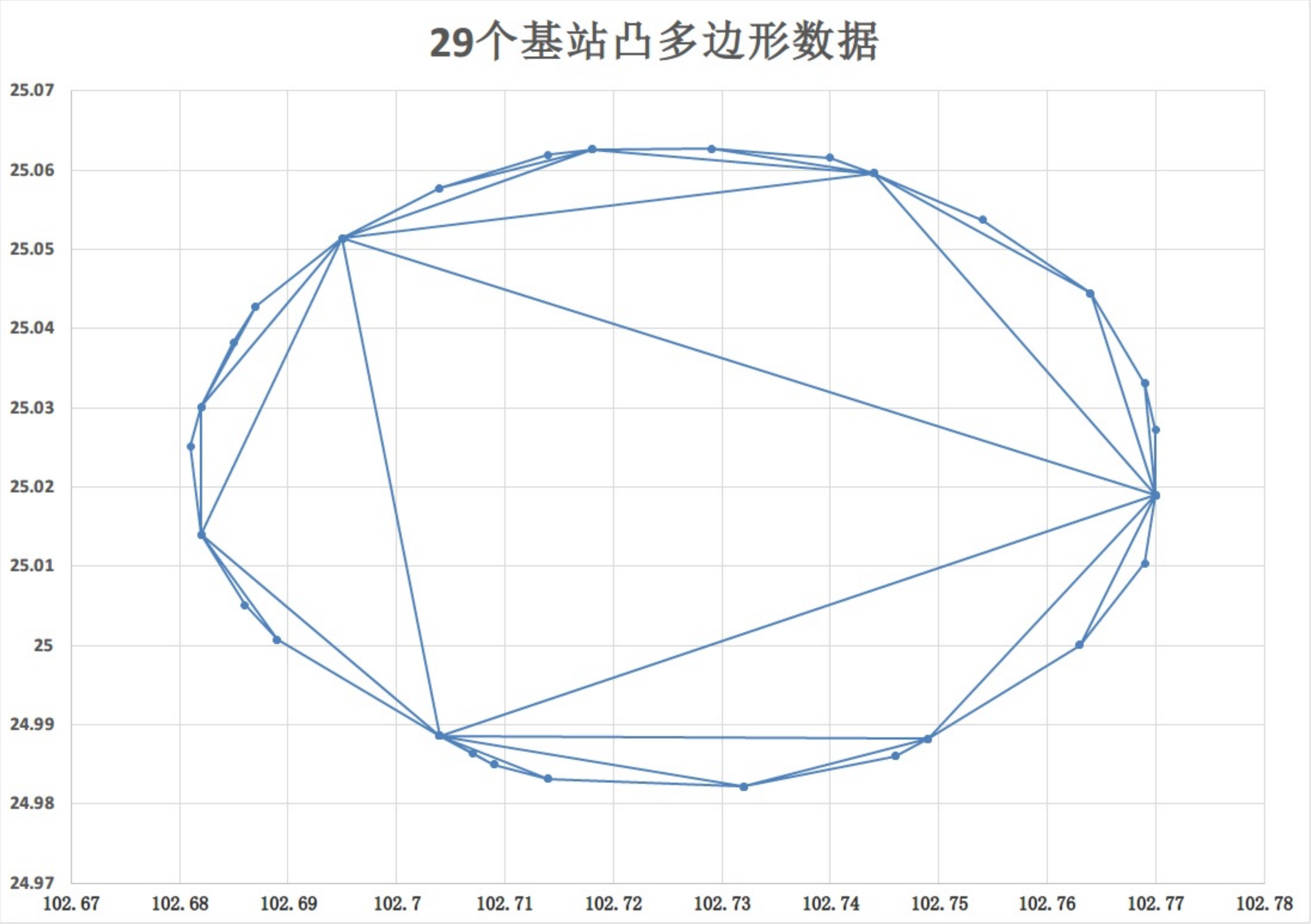
最优三角剖分结果如下：

1.　21 个基站凸多边形的最优剖分

最优三角剖分对应的最小边长弦长总和为 295847。



2. 29 个基站凸多边形的最优剖分
   最优三角剖分对应的最小边长弦长总和为 194329。



四、 0-1 背包问题

1. 对于容量为 300 的背包 A 来说，能够容纳物品的最大价值为 1085，此时重量为 298。
   所容纳的物品为：
   Item 1 Weight: 14 Value: 50
   Item 2 Weight: 11 Value: 72
   Item 4 Weight: 17 Value: 69
   Item 8 Weight: 26 Value: 59
   Item 9 Weight: 10 Value: 49
   Item 11 Weight: 16 Value: 36
   Item 18 Weight: 19 Value: 71

```
Item 20 Weight: 29 Value: 61
Item 23 Weight: 13 Value: 63
Item 24 Weight: 15 Value: 59
Item 25 Weight: 9 Value: 48
Item 26 Weight: 10 Value: 41
Item 32 Weight: 8 Value: 50
Item 33 Weight: 11 Value: 48
Item 35 Weight: 11 Value: 22
Item 38 Weight: 8 Value: 51
Item 43 Weight: 28 Value: 72
Item 44 Weight: 16 Value: 46
Item 45 Weight: 9 Value: 41
Item 50 Weight: 18 Value: 77
```

2. 对于容量为 600 的背包 B 来说，能够容纳物品的最大价值为 1652，此时重量为 600。

所容纳的物品为：

```
Item 1 Weight: 10 Value: 50
Item 4 Weight: 13 Value: 61
Item 5 Weight: 33 Value: 79
Item 7 Weight: 11 Value: 52
Item 11 Weight: 12 Value: 31
Item 14 Weight: 11 Value: 55
Item 16 Weight: 10 Value: 44
Item 22 Weight: 20 Value: 46
Item 23 Weight: 18 Value: 60
Item 25 Weight: 10 Value: 22
Item 30 Weight: 28 Value: 64
Item 34 Weight: 31 Value: 63
Item 49 Weight: 9 Value: 21
Item 51 Weight: 26 Value: 67
Item 53 Weight: 39 Value: 73
Item 58 Weight: 11 Value: 24
Item 60 Weight: 18 Value: 79
Item 61 Weight: 8 Value: 51
Item 69 Weight: 30 Value: 77
Item 72 Weight: 23 Value: 70
Item 76 Weight: 35 Value: 74
Item 77 Weight: 18 Value: 71
Item 81 Weight: 11 Value: 46
Item 82 Weight: 23 Value: 73
Item 87 Weight: 34 Value: 74
Item 88 Weight: 36 Value: 67
Item 90 Weight: 9 Value: 37
Item 93 Weight: 23 Value: 41
Item 94 Weight: 40 Value: 80
```

● 源程序代码

一、 最长公共子序列

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

using namespace std;

int c[600][600], b[600][600]; //c[i][j]存储x[i]到y[j]的最长子序列
的长度，而b[i][j]则存储c[i][j]是由哪一个子问题的解得到的
//b[][]中有三种取值，1、2、3分别对应Zk=Xm=Yn，Xm!=Yn&Zk!=Xm，
Xm!=Yn&Zk!=Yn三种情况

void lcsLength(char x[], char y[], int c[][600], int
b[][600]); //lcsLength用于求源序列X和Y的最长公共子序列的长度
void lcs(int i, int j, char x[], int b[][600]); //通过b[][]可以
构造出最长公共子序列，b[][]记录了原问题是由哪一种子问题的解得到的

int main(void){
    //求解最长公共子序列
    //从文件中读取序列a1、a2、a3、a4
    FILE * fileOne;
    fileOne = fopen("1.txt", "r");
    char a1[600], a2[600], a3[600], a4[600];
    char trash[600];
    fgets(trash, 600, fileOne);
    fgets(a1, 600, fileOne);
    fgets(trash, 600, fileOne);
    fgets(a2, 600, fileOne);
    fgets(trash, 600, fileOne);
    fgets(a3, 600, fileOne);
    fgets(trash, 600, fileOne);
    fgets(a4, 600, fileOne);

    //对序列a1、a2求最长公共子序列并构造结果
    lcsLength(a1, a2, c, b);
    cout << "The longest common substring of A and B is:" <<
```

```
endl;
    lcs(strlen(a1), strlen(a2), a, y);

    //对序列a3、a4求最长公共子序列并构造结果
    lcsLength(a3, a4, x, y);
    cout << "The longest common substring of C and D is:" <<
endl;
    lcs(strlen(a3), strlen(a4, c, y);

    //对序列a1、a4求最长公共子序列并构造结果
    lcsLength(a1, a4, x, y);
    cout << "The longest common substring of A and D is:" <<
endl;
    lcs(strlen(a1), strlen(a4), a, y);

    //end
    fclose(fileOne);
    system("PAUSE");
    return 0;
}

void lcsLength(char x[], char y[], int c[][600], int
b[][600]){
    //先求x和y的长度
    int m = strlen(x);
    int n = strlen(y) - 1;
    //定义临时变量
    int i = 0, j = 0;

    //当i=0或j=0时，c[i][j]=0
    for (i = 0; i <= m; i++)
        c[i][0] = 0;
    for (i = 0; i <= n; i++)
        c[0][i] = 0;

    for (i = 1; i <= m; i++)
        for (j = 1; j <= n; j++){
            //当x[i]=y[j]时，c[i][j] = c[i - 1][j - 1] + 1，对应第一
种情况
            if (x[i - 1] == y[j - 1]){
                c[i][j] = c[i - 1][j - 1] + 1;
                b[i][j] = 1;
            }
            //当x[i]!=y[j]时，c[i][j]=max(c[i-1][j], c[i][j-1])，分
别对应第二种情况和第三种情况
            else if (c[i - 1][j] >= c[i][j - 1]){
                c[i][j] = c[i - 1][j];
                b[i][j] = 2;
            }
            else{
                c[i][j] = c[i][j - 1];
                b[i][j] = 3;
            }
        }
}

void lcs(int i, int j, char x[], int b[][600]){ //根据b[][]来构
造最长子序列
    if (i == 0 || j == 0)
        return;
    if (b[i][j] == 1){ //第一种情况 , x[i]=y[j]，Zk = Zk-1 + z[k]
        lcs(i - 1, j - 1, x, b);
        cout << x[i - 1];
    }
    else if (b[i][j] == 2){ //第二种情况， x[i]!=y[j]，在X[i-1]和
Y[j]中继续构造
        lcs(i - 1, j, x, b);
    }
    else{ //第三种情况， x[i]!=y[j]，在X[i]和Y[j-1]中继续构造
        lcs(i, j - 1, x, b);
    }
}
```

二、 最大字段和

```
#include <iostream>
#include <cstdlib>

using namespace std;

float maxSum(int n, float a[], int &x, int &y); //求a[]中的最大
字段和

int main(void){
    //求最大字段和
    //从文件读取a和b两个字段
    FILE * fileTwo, *fileThree;
    fileTwo = fopen("2.txt", "r");
    fileThree = fopen("3.txt", "r");
    int m = 350, n = 180, i = 0;
    float a[350], b[180];
    for (i = 0; i < m; i++)
        fscanf(fileTwo, "%f", &a[i]);
    for (i = 0; i < n; i++)
        fscanf(fileThree, "%f", &b[i]);

    int aX, aY, bX, bY; //aX和aY分别对应a[]字段的最大字段和的起始位置
和终止位置

    //求出a和b的最大字段和
    float aSum = maxSum(m, a, aX, aY);
    float bSum = maxSum(n, b, bX, bY);
    cout << "The max sum of subsequence of A is " << aSum << "
from " << aX << " to " << aY << endl;
    cout << "The max sum of subsequence of B is " << bSum << "
from " << bX << " to " << bY << endl;
    system("PAUSE");
    return 0;
}

float maxSum(int n, float a[], int &x, int &y) { //求出a[]的最大
字段和
    float sum = 0;
    float b = 0;
    for (int i = 0; i < n; i++){
        if (b >= 0) //当从x到i的字段和为正时，说明上一个字a[i-1]>0
```

```
            b += a[i];
        else{ //如果从x到i的字段和为负，说明上一个字a[i-1]<0，从i开始重
新计x和sum
            x = i + 1;
            b = a[i];
        }
        if (b > sum){ //如果b>sum的话，就令sum=b，且记此时的i为最大字
段和的终止位置y
            sum = b;
            y = i + 1;
        }
    }
    return sum; //返回最大字段和
}
```

三、 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
};

//数据和文件处理
int readData(Book* book, struct baseData data[], wchar_t
loadFileName[], int n);
//求凸多边形的最优三角剖分对应的各边的权值
void minWeightTriangulation(int n, double t[][30], int
s[][30], struct baseData * data);
double dist(struct baseData a, struct baseData b); // dist用于
计算Va和Vb的距离
double w(struct baseData * data, int a, int b, int c); //w用于
计算Va,Vb和Vc构成的三角形的权值
//通过s[][]来构造最优三角剖分中的子三角形
void Traceback(int i, int j, int s[][30]);

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;
    }
    struct baseData data1[30];
    wchar_t loadFileName1[] = L"附件3-1.21个基站凸多边形数据.xls";
    if (readData(book, data1, loadFileName1, NUM1) <= 0)
        return -2;
    //定义t[][]和s[][]并初始化，t[i][j]记录了从Vi-1…Vk…到Vj组成的凸多边形的最优三角剖分的各边的权值
    //而s[i][j]则记录了与Vi-1和Vj一起构成三角形的顶点
    double t[30][30];
    int s[30][30];
    for (int i = 0; i < 30; i++)
        for (int j = 0; j < 30; j++){
            t[i][j] = 0;
            s[i][j] = 0;
        }
    //计算凸多边形的最优三角剖分所对应的权函数值
    minWeightTriangulation(NUM1 - 1, t, s, data1);
    cout << endl << t[1][NUM1 - 1] << endl;
    //根据s[][]来构造最优三角剖分的解
    Traceback(1, NUM1 - 1, s);

    //读取第二份数据“29个基站凸多边形数据”并重新初始化s[][]和t[][]
    Book* book2 = xlCreateBook();
    if (!book2){
        cout << "Error when init book." << endl;
        return -1;
    }
    struct baseData data2[30];
    wchar_t loadFileName2[] = L"附件3-2.29个基站凸多边形数据.xls";
    if (readData(book2, data2, loadFileName2, NUM2) <= 0)
        return -2;
    for (int i = 0; i < 30; i++)
        for (int j = 0; j < 30; j++){
            s[i][j] = 0;
            t[i][j] = 0;
        }


    minWeightTriangulation(NUM2 - 1, t, s, data2);
    cout << endl << t[1][NUM2 - 1] << endl;
    Traceback(1, NUM2 - 1, s);
    system("PAUSE");
    return 0;
}

void minWeightTriangulation(int n, double t[][30], int s[][30], struct baseData * data){
    //求凸多边形的最优三角剖分对应的各边的权值
```

```
  for (int i = 0; i <= n; i++) //当i=j时，t[i][j]=0
    t[i][i] = 0;
  for (int r = 2; r <= n; r++)
    for (int i = 1; i <= n - r + 1; i++){
      int j = i + r - 1;
      t[i][j] = t[i + 1][j] + w(data, i - 1, i, j); //当i<j
时t[i][j]=t[i+1][j]+w(i-1,i,j)，即凸多边形Vi...Vj=(凸多边形
Vi+1...Vj)+三角形Vi-1ViVj
      s[i][j] = i;
      for (int k = i + 1; k < i + r - 1; k++){ //k从i遍历到
j，当存在t[i][k]<t[i][j]时 更新t[i][j]和s[i][j]
        double u = t[i][k] + t[k + 1][j] + w(data, i - 1,
k, j);
        if (u < t[i][j]){
          t[i][j] = u;
          s[i][j] = k;
        }
      }
    }
}

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));
}

void Traceback(int i, int j, int s[][30]){ //根据s[][]来推得最优
三角剖分的解
  if (i == j) return;
  cout << i - 1 << "," << s[i][j] << "," << j << endl;
  Traceback(i, s[i][j], s);
  Traceback(s[i][j] + 1, j, s);
}

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

四、 0-1 背包问题

```
#include <iostream>
#include <cstdlib>

#define MAX 601

using namespace std;

void knapsack(int v[], int w[], int c, int n);//动态规划计算0-1背包问题的最优值
void traceback(int w[], int c, int x[], int n); //在得到最优值的情况下反向构造最优解

int m[MAX][MAX];//m[i][j]包括了0-1背包问题的最优值，其中m[i][j]指可选物品为i,i+1,...,n，背包容量还剩j时的0-1背包问题的最优值
//那么m[1][c]就是整个0-1背包问题的最优值

int main(void){
    //打开文件并将数据读取到w1[],w2[],v1[],v2[]中，其中wi[]是重量，vi[]是与重量对应的价值
    FILE * fileFour;
    fileFour = fopen("4.txt", "r");
    int n1 = 50, n2 = 100, i = 0; //n是物品个数，i是临时变量
    int w1[51], w2[101], v1[51], v2[101], x1[51], x2[101];
    int c1, c2;  //c是最大容量
    fscanf(fileFour, "%d", &c1);
    for (i = 1; i <= n1; i++)
        fscanf(fileFour, "%d", &w1[i]);
    for (i = 1; i <= n1; i++)
        fscanf(fileFour, "%d", &v1[i]);
    fscanf(fileFour, "%d", &c2);
    for (i = 1; i <= n2; i++)
        fscanf(fileFour, "%d", &w2[i]);
    for (i = 1; i <= n2; i++)
        fscanf(fileFour, "%d", &v2[i]);

    //计算第一个背包的解（50个物品，最大容量为300）
    knapsack(v1, w1, c1, n1); //计算最优值
    traceback(w1, c1, x1, n1); //根绝最优值构建最优解
    cout << "The best value of the pack A is " << m[1][c1] << endl;//输出结果
    cout << "The items are:" << endl;
    int weight = 0, value = 0;
    for (int i = 1; i <= n1; i++)
        if (x1[i] == 1){
            cout << "Item " << i << " Weight: " << w1[i] << "
Value: " << v1[i] << endl;
            weight += w1[i];
            value += v1[i];
        }
    cout << "The total weight is " << weight << ' ' << value <<
endl << endl;

    //计算第二个背包的解（100个物品，最大容量600）
    knapsack(v2, w2, c2, n2);
    traceback(w2, c2, x2, n2);
    cout << "The best value of the pack B is " << m[1][c2] <<
endl;
    cout << "The items are:" << endl;
    weight = 0;
    value = 0;
    for (int i = 1; i <= n2; i++)
        if (x2[i] == 1){
            cout << "Item " << i << " Weight: " << w2[i] << "
Value: " << v2[i] << endl;
            weight += w2[i];
            value += v2[i];
        }
    cout << "The total weight is " << weight << ' ' << value <<
endl;
    system("PAUSE");
    return 0;
}

void knapsack(int v[], int w[], int c, int n){
    //根据v[]和w[]来计算0-1背包问题的最优值，v[]是价值，w[]是重量，c是
最大容量，n是物品个数
    n++;
    int jMax = (w[n] - 1 > c) ? c : (w[n] - 1);
//jMax=min(w[n]-1,c)，即jmax=第n个物品重量-1和最大容量二者中的较小值

    for (int j = 0; j <= jMax; j++) //令
m[n][0],m[n][1],...,m[n][jMax]=0,即仅剩第n个物品，容量还剩
0,1,...,jMax时，不能将物品n放入
        m[n][j] = 0;
    for (int j = w[n - 1]; j <= c; j++) //令m[n][w[n-
1]],m[n][w[n]],...,m[n][c]=0,即仅剩第n个物品，容量还剩w[n-
1],w[n],...,c时，可以将物品n放入，且这个子背包的价值为v
        m[n][j] = v[n];

    for (int i = n - 1; i > 1; i--){
        jMax = (w[i] - 1 > c) ? c : (w[i] - 1); //jMax=min(w[i]-
1,c)
        for (int j = 0; j <= jMax; j++) //令
m[i][0],m[i][1],...,m[i][jMax]=m[i+1][j],即对于第i个物品，容量还剩
0,1,...,jMax时，不能将物品n放入，子背包的价值没加入第i个物品时是相同的
            m[i][j] = m[i + 1][j];
        for (int j = w[i]; j <= c; j++) //对于
m[i][w[i]],m[i][w[i]+1],...,m[i][c]来说，m[i][j]= max( m[i +
1][j] , m[i + 1][j - w[i]] + v[i] )
            m[i][j] = (m[i + 1][j] > (m[i + 1][j - w[i]] +
v[i])) ? m[i + 1][j] : (m[i + 1][j - w[i]] + v[i]);
    }
    m[1][c] = m[2][c];
    if (c >= w[1]) //如果c大于w[1]的话， m[1][c]= max( m[1][c] ,
m[2][c - w[1]] + v[1] )
        m[1][c] = (m[1][c] > (m[2][c - w[1]] + v[1])) ?
m[1][c] : (m[2][c - w[1]] + v[1]);
}

void traceback(int w[], int c, int x[], int n){
    //根据m[][]来反向构造0-1背包问题的最优解，x[i]用以存储物品i是否被放
入
    for (int i = 1; i < n; i++){
        if (m[i][c] == m[i + 1][c]) //如果m[i][c]=m[i-1][c]，说明
这个物品没被放进去，则x[i]=0
            x[i] = 0;
        else{ //否则x[i]=1，且由m[i+1][ c-w[i] ]来继续构造最优解
            x[i] = 1;
            c -= w[i];
        }
    }
    x[n] = (m[n][c] > 0) ? 1 : 0; //如果m[n]c[c]>0说明第n个物品被放
入，防止c为负数造成数组访问越界
```