一、用动态规划方法手工求解以下问题：

有 8 万元的投资可以投给 3 个项目，每个项目在不同投资数额下（以万元为单位）的利润如下表。

投资额 盈利 项目	0	1	2	3	4	5	6	7	8
项目 1	0	5	15	40	80	90	95	98	100
项目 2	0	5	15	40	60	70	73	74	75
项目 3	0	4	26	40	45	50	51	52	53

请安排投资计划，使总的利润最大。

写出你所设的状态变量、决策变量、状态转移方程与递推关系式，和手工求解的详细步骤及结果。

答：

假设现在总共有a万元，计划分配给n个项目使用。u_i为分配给第i个项目的资金数量（万元），$g_i(u_i)$为第i个项目得到资金后提供的利润值（万元）。

那么问题为：如何确定各项目的投资金额，使得总利润最大。有以下式子：

$$\max Z = \sum_{i=1}^{n} g(u_i) \text{ 其中} \begin{cases} \sum_{i=1}^{n} u_i \le a \\ 0 \le u_i \le a, i = 1, 2, \dots, n \end{cases}$$

因此假设：$f_k(X)$表示以数量X的投资金额分配给前k个项目所得到的最大利润值。因此动态规划求解问题就是求解$f_n(a)$的问题。

当$\mathrm{k} = 1$时，$f_1(x) = g_1(x)$

当$1 \le \mathrm{k} \le \mathrm{n}$时，其递推关系式如下：

u_k为分配给第i个项目的资金数量$(0 \le u_k \le X)$, 此时有$x_{k-1} = X - u_k$的资金投资前k-1个项目，假如采用最优策略，则前k-1个项目最大利润为$f_{k-1}(X - u_k)$, 因此总利润为：

$$g_k(u_k) + f_{k-1}(X - u_k)$$

所以根据动态规划的最优化原理，有：

$$f_k(X) = \max_{0 \le u_k \le X} \{g_k(u_k) + f_{k-1}(X - u_k)\}$$

其中$\mathrm{k} = 2, 3, \dots, \mathrm{n}$表示总共有n个项目，而按照题目要求这里$u_k$为整数。

综上所述：

状态变量：x_k表示分配给前$1, \cdots, \mathrm{k}$个项目的投资总金额

决策变量：u_k表示投资给第k项目的金额

状态转移方程：$x_{k+1} = x_k + u_{k+1}$

递推关系式：

$$\begin{cases} f_k(X) = \max_{0 \le u_k \le X} \{g_k(u_k) + f_{k-1}(X - u_k)\} \quad \mathrm{k} = 2, 3, \dots, \mathrm{n} \\ f_1(x) = g_1(x) \quad k = 1 \end{cases}$$

针对本题有$\mathrm{X} = 8$，$\mathrm{n} = 3$，求$f_3(8)$的值。

手工求解的详细步骤：

1．一开始当$\mathbf{k}=\mathbf{1}$时$,\mathbf{f_1(x)}=\mathbf{g_1(x)}$

$f_1(0)=0; f_1(1)=5; f_1(2)=15; f_1(3)=40; f_1(4)=80; f_1(5)=90; f_1(6)=95; f_1(7)=98;$
$f_1(8)=100.$

x_1	0	1	2	3	4	5	6	7	8
$f_1(x_1)$	0	5	15	40	80	90	95	98	100

2．当$\mathbf{k}=\mathbf{2}$时

$f_2(0)=\max\{g_2(0)+f_1(0)\}=0+0=0;$

$$f_2(1)=\max\begin{Bmatrix} g_2(0)+f_1(1) \\ g_2(1)+f_1(0) \end{Bmatrix}=\max\begin{Bmatrix} 0+5 \\ 5+0 \end{Bmatrix}=5;$$

$$f_2(2)=\max\begin{Bmatrix} g_2(0)+f_1(2), \\ g_2(1)+f_1(1), \\ g_2(2)+f_1(0) \end{Bmatrix}=\max\begin{Bmatrix} 0+15 \\ 5+5 \\ 15+0 \end{Bmatrix}=15;$$

$$f_2(3)=\max\begin{Bmatrix} g_2(0)+f_1(3) \\ g_2(1)+f_1(2) \\ g_2(2)+f_1(1) \\ g_2(3)+f_1(0) \end{Bmatrix}=\max\begin{Bmatrix} 0+40 \\ 5+15 \\ 15+5 \\ 40+0 \end{Bmatrix}=40;$$

$$f_2(4)=\max\begin{Bmatrix} g_2(0)+f_1(4) \\ g_2(1)+f_1(3) \\ g_2(2)+f_1(2) \\ g_2(3)+f_1(1) \\ g_2(4)+f_1(0) \end{Bmatrix}=\max\begin{Bmatrix} 0+80 \\ 5+40 \\ 15+15 \\ 40+5 \\ 60+0 \end{Bmatrix}=80;$$

$$f_2(5)=\max\begin{Bmatrix} g_2(0)+f_1(5) \\ g_2(1)+f_1(4), \\ g_2(2)+f_1(3) \\ g_2(3)+f_1(2), \\ g_2(4)+f_1(1), \\ g_2(5)+f_1(0) \end{Bmatrix}=\max\begin{Bmatrix} 0+90 \\ 5+80 \\ 15+40 \\ 40+15 \\ 60+5 \\ 70+0 \end{Bmatrix}=90;$$

$$f_2(6)=\max\begin{Bmatrix} g_2(0)+f_1(6) \\ g_2(1)+f_1(5) \\ g_2(2)+f_1(4) \\ g_2(3)+f_1(3) \\ g_2(4)+f_1(2) \\ g_2(5)+f_1(1) \\ g_2(6)+f_1(0) \end{Bmatrix}=\max\begin{Bmatrix} 0+95 \\ 5+90 \\ 15+80 \\ 40+40 \\ 60+15 \\ 70+5 \\ 73+0 \end{Bmatrix}=95;$$

$$f_2(7)=\max\begin{Bmatrix} g_2(0)+f_1(7) \\ g_2(1)+f_1(6) \\ g_2(2)+f_1(5) \\ g_2(3)+f_1(4) \\ g_2(4)+f_1(3) \\ g_2(5)+f_1(2) \\ g_2(6)+f_1(1) \\ g_2(7)+f_1(0) \end{Bmatrix}=\max\begin{Bmatrix} 0+98 \\ 5+95 \\ 15+90 \\ 40+80 \\ 60+40 \\ 70+15 \\ 73+5 \\ 74+0 \end{Bmatrix}=120;$$

$$f_2(8)=\max\begin{Bmatrix} g_2(0)+f_1(8) \\ g_2(1)+f_1(7) \\ g_2(2)+f_1(6) \\ g_2(3)+f_1(5) \\ g_2(4)+f_1(4) \\ g_2(5)+f_1(3) \\ g_2(6)+f_1(2) \\ g_2(7)+f_1(1) \\ g_2(8)+f_1(0) \end{Bmatrix}=\max\begin{Bmatrix} 0+100 \\ 5+98 \\ 15+95 \\ 40+90 \\ 60+80 \\ 70+40 \\ 73+15 \\ 74+5 \\ 75+0 \end{Bmatrix}=140.$$

x_2	0	1	2	3	4	5	6	7	8
$f_2(x_2)$	0	5	15	40	80	90	95	120	140

3．当$\mathbf{k}=\mathbf{3}$时，只需要求解$f_3(8)$

$$f_3(8)=\max\begin{Bmatrix} g_3(0)+f_2(8)\\ g_3(1)+f_2(7)\\ g_3(2)+f_2(6)\\ g_3(3)+f_2(5)\\ g_3(4)+f_2(4)\\ g_3(5)+f_2(3)\\ g_3(6)+f_2(2)\\ g_3(7)+f_2(1)\\ g_3(8)+f_2(0) \end{Bmatrix}=\max\begin{Bmatrix} 0+140\\ 4+120\\ 26+95\\ 40+90\\ 45+80\\ 50+40\\ 51+15\\ 52+5\\ 53+0 \end{Bmatrix}=140$$

x_1	0	1	2	3	4	5	6	7	8
$f_1(x_1)$	0	5	26	40	80	90	106	120	**140**

最终结果为：给项目1投资4万元，项目2投资4万元，项目3不投资，将获得最大利润140万元。

二、用动态规划方法编程求解下面的问题：

一凸 8 边形P 的顶点顺时针为$\{v_1, v_2, \dots, v_8\}$，任意两顶点间的线段的权重由矩阵 D 给出。若 v_i 与 v_j 是 P 上不相邻的两个顶点，则线段 v_iv_j 称为 P 的一条弦。求 P 的一个弦的集合 T，使得 T 中所有的弦恰好将 P 分割成互不重迭的三角形，且各三角形的权重之和为最小（一个三角形的权重是其各边的权重之和）。

$$D = \begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \end{array} \begin{bmatrix} 0 & 14 & 25 & 27 & 10 & 11 & 24 & 16 \\ & 0 & 18 & 15 & 27 & 28 & 16 & 14 \\ & & 0 & 19 & 14 & 19 & 16 & 10 \\ & & & 0 & 22 & 23 & 15 & 14 \\ & & & & 0 & 14 & 13 & 20 \\ & & & & & 0 & 15 & 18 \\ & & & & & & 0 & 27 \\ & & & & & & & 0 \end{bmatrix}$$

要求：写出递推关系式、伪代码和程序相关说明，并分析时间复杂性。（请遵守第一节课提出的有关 assignment 的要求：提交的可执行程序必须能够输出结果，源代码必须可编译等等）

答：

凸多边形的最优三角剖分有最优子结构性质。可以用反证法来证明：

假如凸多边形$\{\boldsymbol{v_0}, \boldsymbol{v_1}, \dots, \boldsymbol{v_{n-1}}\}$的三角权值之和为c, 而$\{\boldsymbol{v_0}, \boldsymbol{v_1}, \dots, \boldsymbol{v_k}\}$凸多边形划分的三角权值之和为a，$\{\boldsymbol{v_k}, \boldsymbol{v_{k+1}}, \dots, \boldsymbol{v_{n-1}}\}$凸多边形划分的三角权值之和为b, 三角形$\{\boldsymbol{v_0 v_k v_{n-1}}\}$的权值之和为函数getWeight(0, k, n - 1)的结果，即返回三条边的权值之和，那么

$$\mathbf{c = a + b + getWeight(0, k, n - 1)}$$

因此只需要证明如果c是最优的，则a和b一定是最优的（即原问题包含子问题的最优子结构）。

反证法：

如果a不是最优的，那么$\{\boldsymbol{v_0}, \boldsymbol{v_1}, \dots, \boldsymbol{v_k}\}$的三角部分一定存在一个最优解$\boldsymbol{a'}$, 并且$\boldsymbol{a' \le a}$, 那么有$\boldsymbol{a' + b} + \mathbf{getWeight(0, k, n - 1) < c}$，即c不是最优解，这与假设c是最优的条件矛盾，因此如果c是最优的，则a和b一定是最优的结果。

因此做如下几个定义：

- N的值表示凸多边形$\{\boldsymbol{v_0}, \boldsymbol{v_1}, \dots, \boldsymbol{v_{n-1}}\}$的点数量。
- 二维数组$\mathbf{Weight[i][j](0 \le i < j < N)}$表示凸多边形点$\boldsymbol{v_i}$到$\boldsymbol{v_j}$的权值，该权重由题目给出。
- 定义二维数组$\mathbf{bestWeight[i][j](0 \le i \le j < N)}$表示凸多边形$\{\boldsymbol{v_0}, \boldsymbol{v_1}, \dots, \boldsymbol{v_{n-1}}\}$最优三角决策结果对应的权重之和，初始化为0。因此我们要计算的结果为凸多边形$\mathbf{bestWeight[0][N-1]}$的最优三角决策结果之和。
  注意：这里初始化的条件为i==j和i+1==j时候$\mathbf{bestWeight[i][j] = 0}$，因为这时候凸多边形变成点或者一条直线，没有三角决策存在。
- 定义二维数组$\mathbf{bestPoint[i][j] = k(0 \le i < j < N, i < k < j)}$为保存决策过程中以边$\boldsymbol{v_i v_j}$为三角一边，在点$\boldsymbol{v_i}$和$\boldsymbol{v_j}$中间的最优决策点$\boldsymbol{v_k}$的数值。
- 定义函数$\mathbf{getWeight(i, k, j)}$返回以$\boldsymbol{v_i v_k v_j}$三个点的三角形权重之和。

因此综上所述，可以得到递推关系式为：

$$\mathbf{bestWeight[i][j]} = \begin{cases} \mathbf{0} & (\boldsymbol{j - i}) \le \mathbf{1} \\ \boldsymbol{min}\{\mathbf{bestWeight[i][k] + bestWeight[k][j] + getWeight(i, j, k)}\} & \mathbf{(i < k < j)} \end{cases}$$

核心伪代码为：

MinWeightTriangulation(*int n*)

```
//首先初始化，先初始化bestWeight [n-1][n-1]因为后面循环初始化不到
bestWeight[n-1][n-1]←0;
for i←0 to n-1
    bestWeight[i][i]←0;
    bestWeight[i][i+1]←0;
//scale代表子问题的规模大小，例如子问题{V0,V1,V2}的规模为2,子问题{V0,V1...V5}的规模为5
for scale←2 to n
    //求解子问题的最后一个为n-scale-1，例如scale=2，最后一个子问题为i=6,j+8,{V6,V7,V8}
    for i←0 to n-scale
        // j 代表当前以 Vi 为起点的子问题的后边界 Vj
        j = i + scale;
        //先处理 k = i+1的情况，这是为了有一开始的初值方便对比
        bestWeight[i][i]=bestWeight[i][i+1]+bestWeight[i+1][j]+GetWeight(i,i+1,j)
        bestpoint[i][j] = i+1;
        //有了基准值之后，可以开始循环处理k=i+2的情况
        for k←i+2 to j
            temp=bestWeight[i][k]+bestWeight[k][j]+GetWeight(i,k,j)
            if temp < bestWeight[i][j]
                bestWeight[i][j] = temp
                bestpoint[i][j] = k
return bestweight[0][n - 1]
```

可运行的源代码放在后面，现在先来分析下算法的空间复杂度和时间复杂度：

算法空间复杂度为O(n^2) ,时间复杂度为O(n^3)。

分析如下：

算法有bestWeight和bestpoint两个数组分别保存决策过程最佳的权重值和中间点的结果，还有一个保存权重的weight数组，空间上复杂度3×N^2，即算法空间复杂度为O(n^2),n为点的个数。其次时间复杂度可以从伪代码中看到处理子问题规模从2→n一层循环，内部处理相同规模子问题个数从0→n-scale第二层循环，再内部处理子问题最佳中间点解，总共三层循环，时间复杂度为O(n^3)。

程序相关说明：

- 在源代码中主要有这几个函数

```
//计算Vi,Vk,Vj组成的三角形的权重之和
int GetWeight(int i, int k, int j);

//自底向上动态规划计算n变形最优三角形的权值之和
int MinWeightTriangulation(int n);

//打印凸子多边形 {Vi, ..., Vj} 的最优三角剖分结果
void Traceback(int i, int j);
```

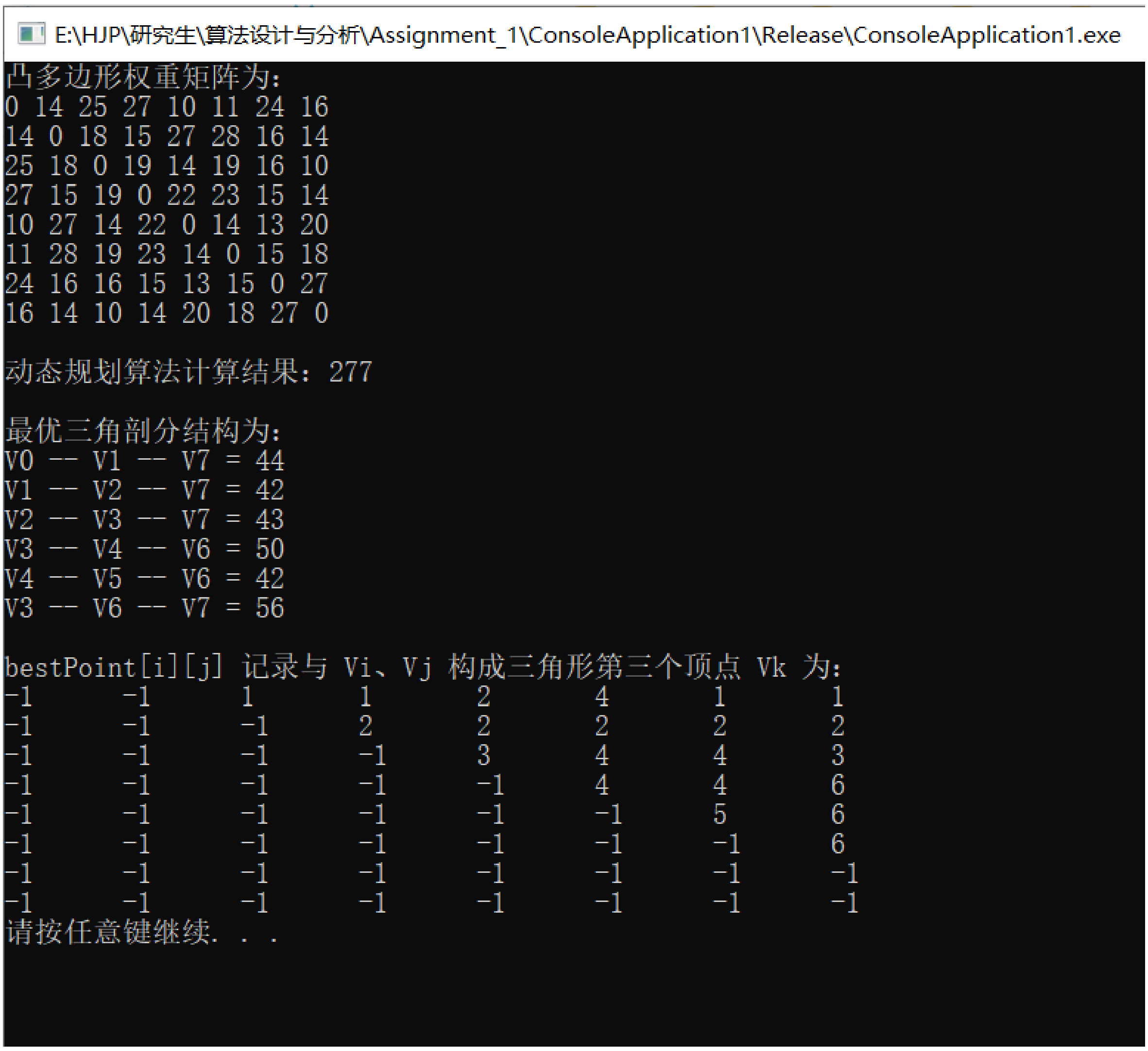
- 程序采用C++代码，在VS2015编辑环境中运行,查看运行EXE结果请查看路径 \Assignment_1\ConsoleApplication1\Release\ConsoleApplication1.exe的执行结果

  运行结果如下所示：

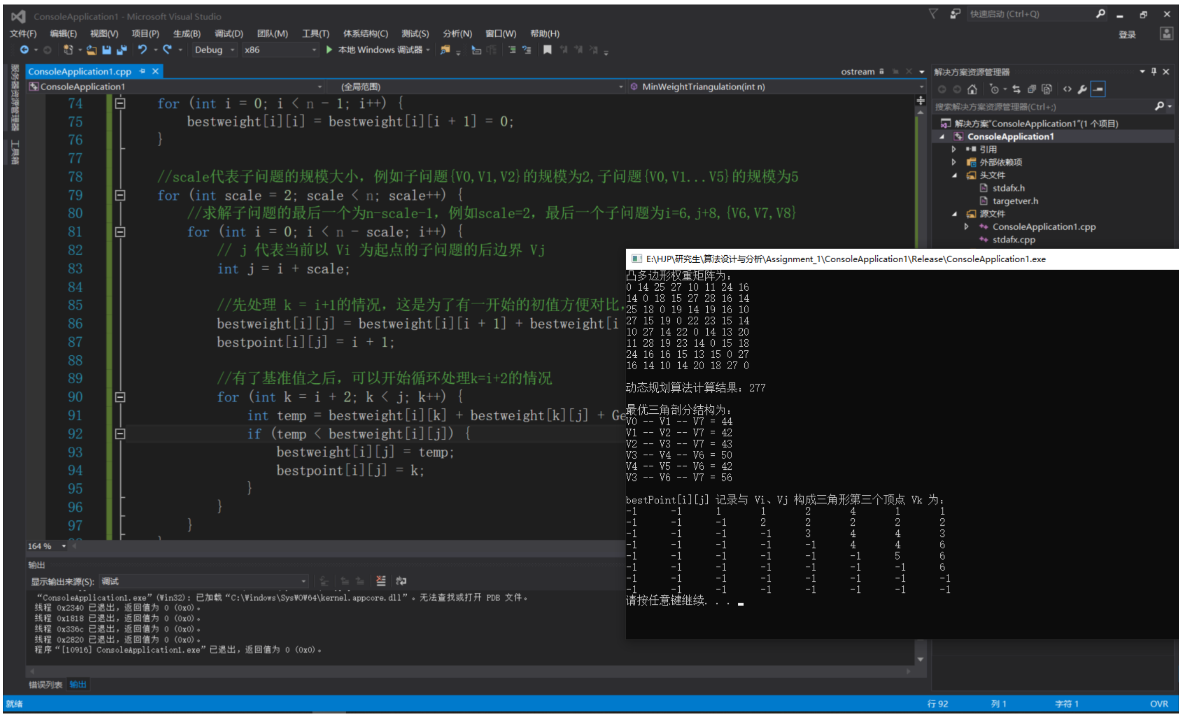
  答案最佳结果为277

  最优三角剖分结构为：

```
V0 -- V1 -- V7 = 44
V1 -- V2 -- V7 = 42
V2 -- V3 -- V7 = 43
V3 -- V4 -- V6 = 50
V4 -- V5 -- V6 = 42
V3 -- V6 -- V7 = 56
```


E:\HJP\研究生\算法设计与分析\Assignment_1\ConsoleApplication1\Release\ConsoleApplication1.exe
凸多边形权重矩阵为：
0 14 25 27 10 11 24 16
14 0 18 15 27 28 16 14
25 18 0 19 14 19 16 10
27 15 19 0 22 23 15 14
10 27 14 22 0 14 13 20
11 28 19 23 14 0 15 18
24 16 16 15 13 15 0 27
16 14 10 14 20 18 27 0
动态规划算法计算结果：277
最优三角剖分结构为：
V0 -- V1 -- V7 = 44
V1 -- V2 -- V7 = 42
V2 -- V3 -- V7 = 43
V3 -- V4 -- V6 = 50
V4 -- V5 -- V6 = 42
V3 -- V6 -- V7 = 56
bestPoint[i][j] 记录与 Vi、Vj 构成三角形第三个顶点 Vk 为：
-1 -1 1 1 2 4 1 1
-1 -1 -1 2 2 2 2 2
-1 -1 -1 -1 3 4 4 3
-1 -1 -1 -1 -1 4 4 6
-1 -1 -1 -1 -1 -1 5 6
-1 -1 -1 -1 -1 -1 -1 6
-1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1
请按任意键继续. . .

源代码运行界面：


ConsoleApplication1 - Microsoft Visual Studio
ConsoleApplication1.cpp
MinWeightTriangulation(int n)
for (int i = 0; i < n - 1; i++) {
bestweight[i][i] = bestweight[i][i + 1] = 0;
}
//scale代表子问题的规模大小，例如子问题{V0,V1,V2}的规模为2，子问题{V0,V1...V5}的规模为5
for (int scale = 2; scale < n; scale++) {
//求解子问题的最后一个为n-scale-1，例如scale=2，最后一个子问题为i=6，j+8，{V6,V7,V8}
for (int i = 0; i < n - scale; i++) {
// j 代表当前以 Vi 为起点的子问题的后边界 Vj
int j = i + scale;
//先处理 k = i+1的情况，这是为了有一开始的初值方便对比
bestweight[i][j] = bestweight[i][i + 1] + bestweight[i
bestpoint[i][j] = i + 1;
//有了基准值之后，可以开始循环处理k=i+2的情况
for (int k = i + 2; k < j; k++) {
int temp = bestweight[i][k] + bestweight[k][j] + Ge
if (temp < bestweight[i][j]) {
bestweight[i][j] = temp;
bestpoint[i][j] = k;
}
}
}
解决方案资源管理器
解决方案"ConsoleApplication1"(1 个项目)
ConsoleApplication1
引用
外部依赖项
头文件
stdafx.h
targetver.h
源文件
ConsoleApplication1.cpp
stdafx.cpp
E:\HJP\研究生\算法设计与分析\Assignment_1\ConsoleApplication1\Release\ConsoleApplication1.exe
凸多边形权重矩阵为：
动态规划算法计算结果：277
最优三角剖分结构为：
bestPoint[i][j] 记录与 Vi、Vj 构成三角形第三个顶点 Vk 为：
请按任意键继续. . .
输出
程序"[10916] ConsoleApplication1.exe"已退出，返回值为 0 (0x0)。

源代码如下：

```
#include "stdafx.h"
#include<iostream>
#include<vector>
using namespace std;

const int N = 8; //八边形

//权值函数
vector<vector<int>> weight = {
        { 0,14,25,27,10,11,24,16 },
        { 14,0,18,15,27,28,16,14 },
        { 25,18,0,19,14,19,16,10 },
        { 27,15,19,0,22,23,15,14 },
        { 10,27,14,22,0,14,13,20 },
        { 11,28,19,23,14,0,15,18 },
        { 24,16,16,15,13,15,0,27 },
        { 16,14,10,14,20,18,27,0 }};

//初始化存储数组
vector<int> tempweight(N, 0);
vector<int> temppoint(N, -1);
// bestweight[i][j] 记录凸子多边形 {Vi, ..., Vj} 三角剖分的最优权值。
vector<vector<int>> bestweight(N, tempweight);
// bestpoint[i][j] 记录与 Vi、Vj 构成三角形第三个顶点 Vk 。
vector<vector<int>> bestpoint(N, temppoint);


//计算Vi,Vk,Vj组成的三角形的权重之和
int GetWeight(int i, int k, int j);

//自底向上动态规划计算n变形最优三角形的权值之和
int MinWeightTriangulation(int n);

//打印凸子多边形 {Vi, ..., Vj} 的最优三角剖分结果
void Traceback(int i, int j);

int main() {
        cout << "凸多边形权重矩阵为：" << endl;
        for (int i = 0; i < N; i++) {
                for (int j = 0; j < N; j++) {
                        cout << weight[i][j] << " ";
                }
                cout << endl;
        }
        cout << endl;
        cout <<"动态规划算法计算结果：" << MinWeightTriangulation(N) << endl;
        cout << endl;
        cout << "最优三角剖分结构为：" << endl;
        Traceback(0, N - 1);
        cout << endl;

        cout << "bestPoint[i][j] 记录与 Vi、Vj 构成三角形第三个顶点 Vk 为：" << endl;
        for (int i = 0; i < N; i++) {
                for (int j = 0; j < N; j++) {
                        cout << bestpoint[i][j] << "\t";
                }
                cout << endl;
        }

        system("pause");
```

```
	return 0;
}


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

int MinWeightTriangulation(int n)
{
	//对动态规划数组初始化,这里初始化其实可以不做，前面已经初始化过了
	bestweight[n - 1][n - 1] = 0;//下面初始化会漏掉[n-1][n-1]点
	for (int i = 0; i < n - 1; i++) {
		bestweight[i][i] = bestweight[i][i + 1] = 0;
	}

	//scale代表子问题的规模大小，例如子问题{V0,V1,V2}的规模为2,子问题{V0,V1...V5}的规模为
5
	for (int scale = 2; scale < n; scale++) {
		//求解子问题的最后一个为n-scale-1，例如scale=2，最后一个子问题为
i=6,j+8,{V6,V7,V8}
		for (int i = 0; i < n - scale; i++) {
			// j 代表当前以 Vi 为起点的子问题的后边界 Vj
			int j = i + scale;

			//先处理 k = i+1的情况，这是为了有一开始的初值方便对比，这里也可以选择
初始化最大值9999
			bestweight[i][j] = bestweight[i][i + 1] + bestweight[i + 1][j] +
GetWeight(i, i + 1, j);
			bestpoint[i][j] = i + 1;

			//有了基准值之后，可以开始循环处理k=i+2的情况
			for (int k = i + 2; k < j; k++) {
				int temp = bestweight[i][k] + bestweight[k][j] + GetWeight(i,
k, j);
				if (temp < bestweight[i][j]) {
					bestweight[i][j] = temp;
					bestpoint[i][j] = k;
				}
			}
		}
	}

	//返回右上角的最佳数值。
	return bestweight[0][n - 1];
}

void Traceback(int i, int j)
{
	//注意回溯查找的返回条件,i+1=j表示中间没有任何点存在，bestpoint[i][j]内部的值为出始化
-1
	if (i+1 == j)
		return;
	Traceback(i,bestpoint[i][j]);
	cout << "V" << i << " -- V" << bestpoint[i][j] << " -- V" << j << " = " <<
GetWeight(i,bestpoint[i][j],j) << endl;
	Traceback(bestpoint[i][j],j);
}
```