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。为分配给第i个项目的资金数量（万元），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：**

**因此假设：表示以数量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的问题。  
当时，**

**当时，其递推关系式如下：**

**为分配给第i个项目的资金数量(,此时有的资金投资前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,因此总利润为：**

**所以根据动态规划的最优化原理，有：**

**其中表示总共有n个项目，而按照题目要求这里为整数。**

**综上所述：**

**状态变量：表示分配给前1,…,k个项目的投资总金额**

**决策变量：表示投资给第k项目的金额**

**状态转移方程：**

**递推关系式:**

**针对本题有的值。**

**手工求解的详细步骤：**

**1．一开始当**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **x1** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| **f1(x1)** | 0 | 5 | 15 | 40 | 80 | 90 | 95 | 98 | 100 |

**2．当时**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **x2** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| **f2(x2)** | 0 | 5 | 15 | 40 | 80 | 90 | 95 | 120 | 140 |

**3．当时，只需要求解**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **x1** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| **f1(x1)** | 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{*v1，v2，… ，v8*}，任意两顶点间的线段的权重由矩阵 D 给出。若 *vi* 与 *vj* 是P 上不相邻的两个顶点，则线段 *vivj* 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）。

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：**

**假如凸多边形的三角权值之和为c,而凸多边形划分的三角权值之和为a，凸多边形划分的三角权值之和为b,三角形的权值之和为函数getWeight(0, k, n - 1)的结果，即返回三条边的权值之和，那么**

**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，那么的三角部分一定存在一个最优解,并且,那么有，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的值表示凸多边形的点数量。**
* **二维数组表示凸多边形点到的权值，该权重由题目给出。**
* **定义二维数组表示凸多边形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，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。**

* **定义二维数组为保存决策过程中以边为三角一边，在点和中间的最优决策点的数值。**
* **定义函数返回以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：**

**核心伪代码为：**

//首先初始化，先初始化 [n-1][n-1]因为后面循环初始化不到

**1 bestWeight[n-1][n-1]←0;**

**2 for i←0 to n-1**

**3 bestWeight[i][i]←0;**

**4 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

**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}

**6 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

**7 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

**8 bestWeight[i][i]=bestWeight[i][i+1]+bestWeight[i+1][j]+GetWeight(i,i+1,j)**

**9 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

**10 for k←i+2 to j**

**11 temp=bestWeight[i][k]+bestWeight[k][j]+GetWeight(i,k,j)**

**12 if temp < bestWeight[i][j]**

**13 bestWeight[i][j] = temp**

**14 bestpoint[i][j] = k**

**15 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(n2) ,时间复杂度为0(n3)。**

**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×N2，即算法空间复杂度为O(n2),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0(n3)。**

**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**

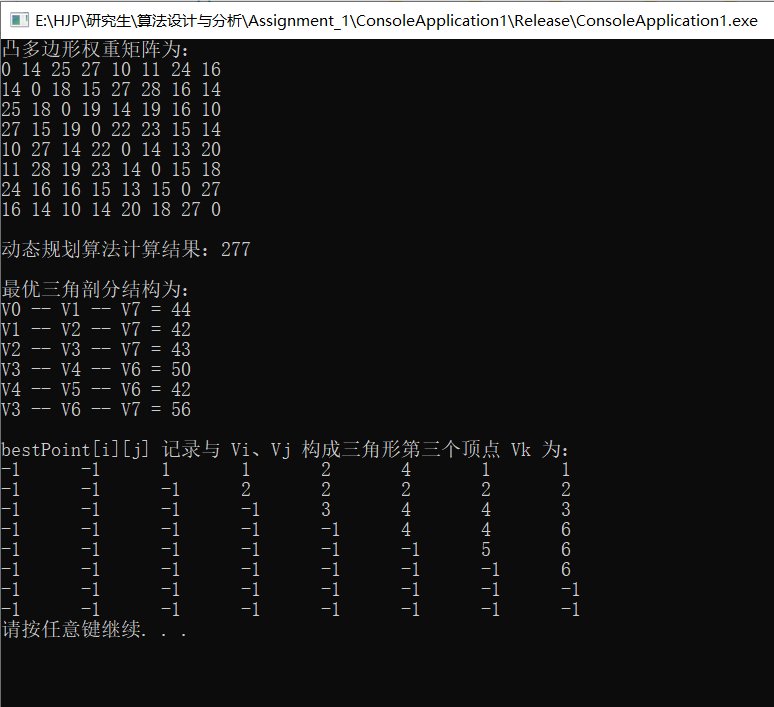
**V1 -- V2 -- V7 = 42**

**V2 -- V3 -- V7 = 43**

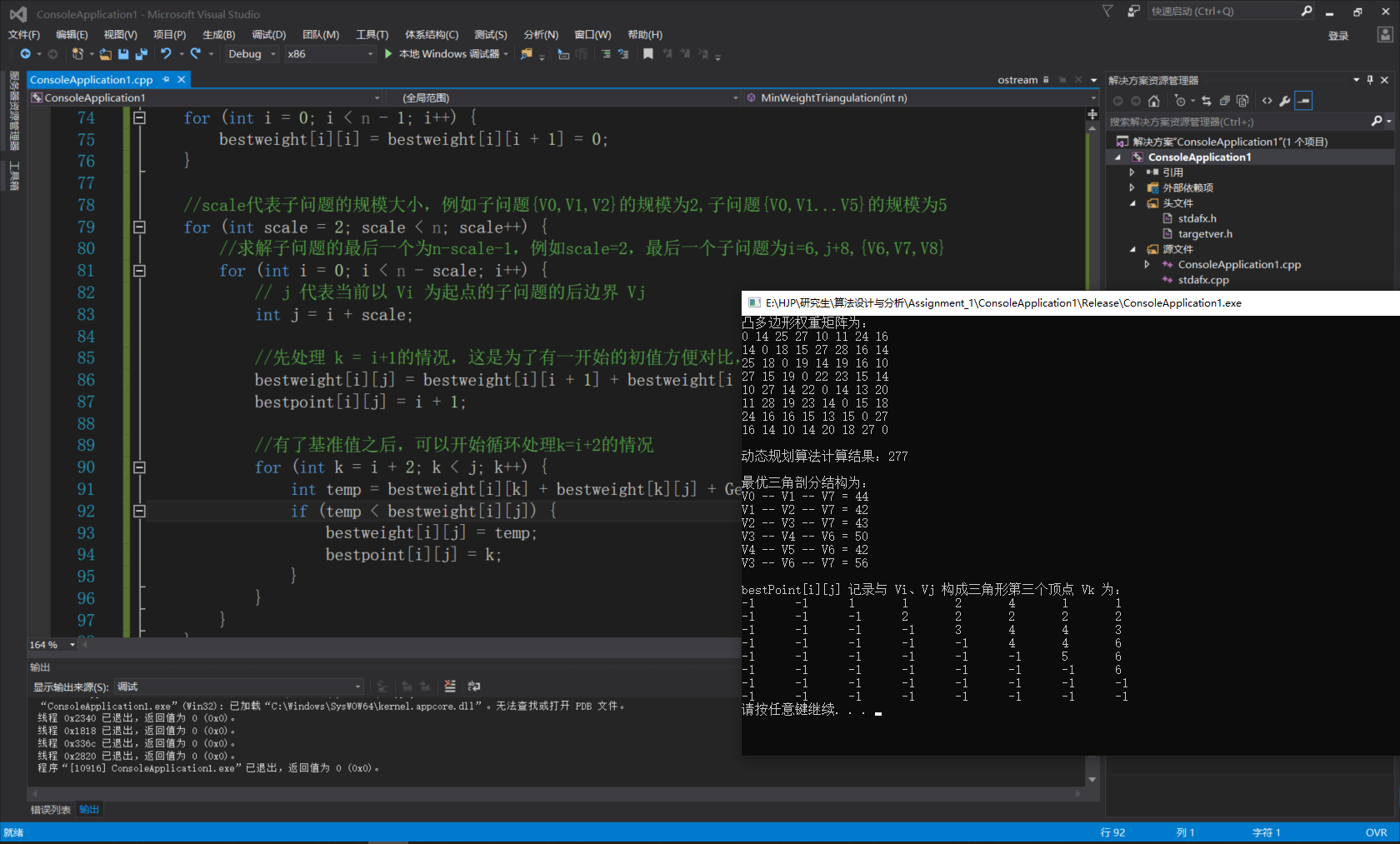
**V3 -- V4 -- V6 = 50**

**V4 -- V5 -- V6 = 42**

**V3 -- V6 -- V7 = 56**

****

**源代码运行界面：**

****

**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);

}