计算矩阵连乘积

问题描述：

在科学计算中经常要计算矩阵的乘积。矩阵A和B可乘的条件是矩阵A的列数等于矩阵B的行数。若A是一个p×q的矩阵，B是一个q×r的矩阵，则其乘积C=AB是一个p×r的矩阵。由该公式知计算C=AB总共需要pqr次的数乘。其标准计算公式为：

$$C_{ij}=\sum_{k=1}^{q}A_{ik}B_{kj} \qquad 其中 1\le i\le p，1\le j\le r$$

现在的问题是，给定n个矩阵{A1,A2,…,An}。其中Ai与Ai+1是可乘的，i=1,2,…,n-1。要求计算出这n个矩阵的连乘积A1A2…An。

问题分析：

- 运算次数=第一个矩阵的行*第一个矩阵的列*第二个矩阵的列（**一行两列**）

A是一个$p \times q$矩阵，B是一个$q \times r$矩阵，AB相乘，得到的矩阵元素个数为$p \times r$，每个元素由q次乘法得到，因此所需乘法次数为$p \times q \times r$。

在计算矩阵连乘积时，加括号的方式对计算量有影响。

例如有三个矩阵A_1, A_2, A_3连乘，它们的维数分别为10×100,100×5,5×50。用第一种加括号方式$(A_1A_2)A_3$计算，则所需数乘次数为$10 \times 100 \times 5 + 10 \times 5 \times 50 = 7,500$。用第二种加括号方式$A_1(A_2A_3)$计算，需要$100 \times 5 \times 50 + 10 \times 100 \times 50 = 75,000$次数乘。

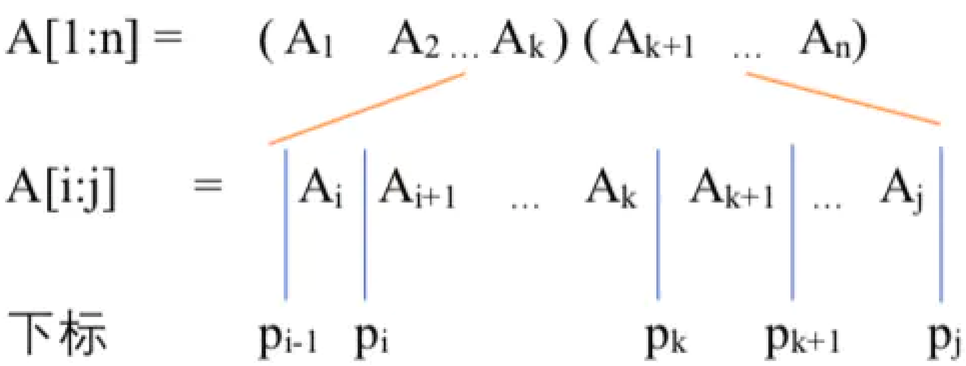
问题： 输入连乘矩阵的个数，**每个矩阵的维数**。要求输出数乘次数最少时的加括号方式，及数乘次数。

注意：矩阵的维数并非以二元组的形式输入，而是顺序行列合并后输入。例如1x2; 2x3 ; 3x11; 11x24这四个矩阵的乘积，输入维数时只需要输入1,2,3,11,24五个数字即可，假设p是维数数组，p[i-1]代表Ai的行维度，p[i]代表Ai的列维度。

引入以下符号：

- n表示矩阵的个数
- A_i表示第i个矩阵
- $A[i:j]$表示矩阵连乘$A_iA_{i+1}..A_j$
- p_i表示A_i的列数
- p_{i-1}表示A_i的行数
- k表示矩阵连乘断开的位置为k，表示在A_k和A_{k+1}之间断开
- $m[i,j]$表示$A[i:j]$的最少乘次，$m[1,n]$即问题的最优解

符号解释：由矩阵相乘的条件可知：**前一个矩阵的列数 = 后一个矩阵的行数**。因此，p_i 既是 A_i 的列数，也是 A_{i+1} 的行数。结合下面的示意图有助于理解：



伪代码

def CMU(M[]): // M是一个数组，可以表示一系列连乘的矩阵：M[i]是Ai的列数同时也是Ai+1的行数（M[i-1]是Ai的行数，M[i+1]是Ai+1的列数）

n = |M|

1.define a set dp[n+1][n+1]

for i in range[0,n]:

dp[i][i] = 0;

2.for group = 1 ; group <= n-1; group++:

for begin = 1; begin <= n-group; begin++:

end = begin + group

for k = begin; k < end; k++:

dp[begin][end] = min{dp[begin][k] + dp[k+1][end] + M[begin-1]*M[k]*M[end])

3.return dp[1][n];

代码实现

```

```

```
int CMU(int num, vector<int>& clo)
{
    // clo[i]表示第i个矩阵Ai的列数，同时也是Ai+1的行数；clo[i-1]表示第i个矩阵Ai的行数，同时也是Ai-1的列数

    // dp[i][j]: Ai乘到Aj，所需元素乘法的最少次数
    // dp[i][i] = 0,Ai和Ai无法矩阵相乘
  // dp[i][j] = dp[i][k] + dp[k+1][j] + clo[i-1]*clo[k]*[j],Ai*...*Ak的最少乘法次数 + Ak+1*...*Aj的最少乘法次数 + Ai的行*Ak的列*Aj的列
    vector<vector<int>> dp(num + 1, vector<int>(num + 1));
```

```
    // 每一组至少有一个
    for (int group = 1; group <= num - 1; group++)
    {
        // 从Ai开始乘
        for (int i = 1; i <= num - group; i++)
        {
            int j = i + group;
            int minNum = INT_MAX;
            for (int k = i; k < j; k++)
            {
                minNum = min(minNum, dp[i][k] + dp[k + 1][j] + clo[i - 1] * clo[i] * clo[j]);
                dp[i][j] = minNum;
            }
            cout << "Finall-dp[" << i << "][" << j << "] = " << dp[i][j] << endl;
        }
    }

    return dp[1][num];
}


int main()
{
    vector<int> clos = { 30,35,15,5,10,20,25 };
    cout << CMU(6, clos);
    return 0;
}
```

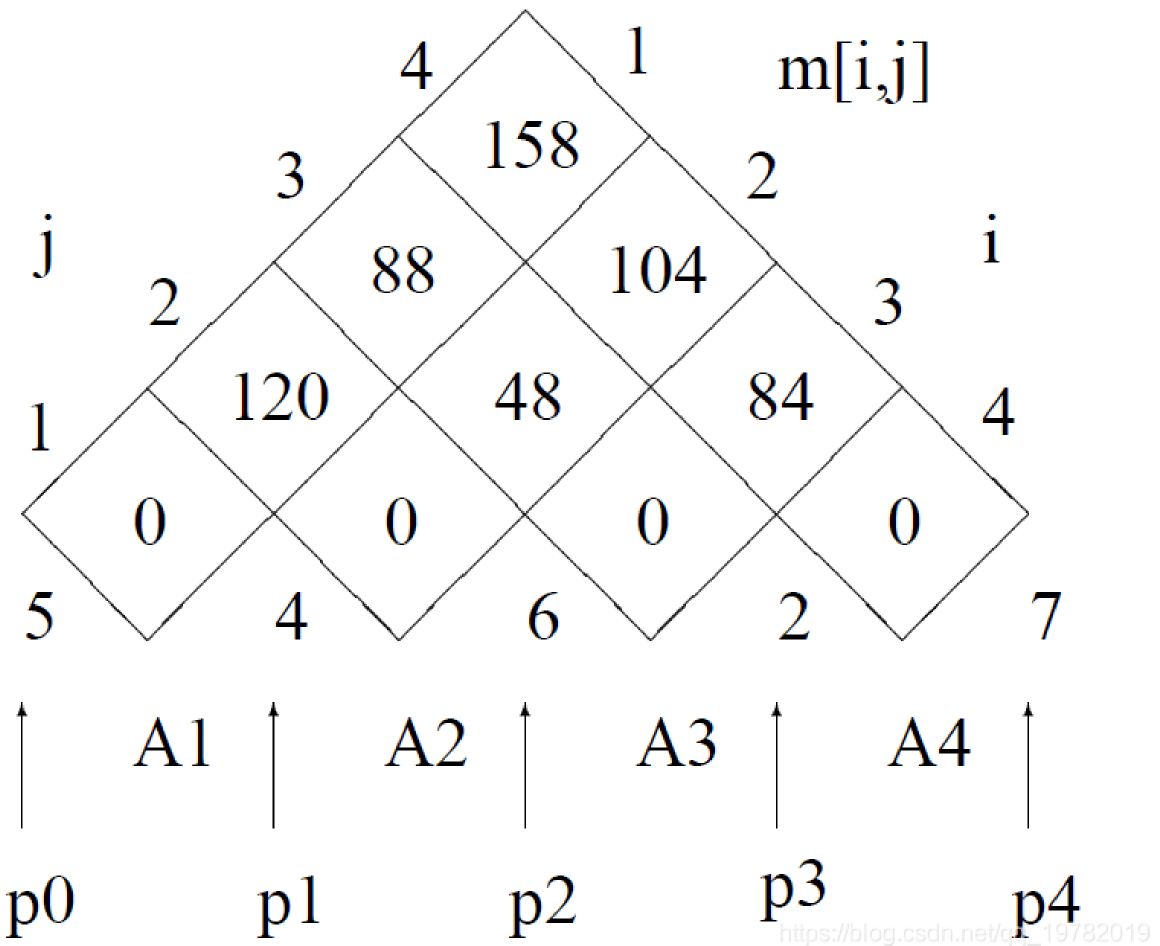
结合图示分析输出结果

- 只能是下标小的矩阵x下标大的，例如只能有A1*A2，不能有A2*A1，因此二维运算矩阵中只有对角线上半部分可以填入数据。
- 计算两两一组的情况，三三一组的情况，四四一组的情况.....沿着对角线填入数值，nn一组的情况可以通过n-k一组的情况和k一组的情况得到

i \ j	1	2	3	4	5	6
1	0	15750	7875	9375	11875	15125
2		0	2625	4375	7125	10500
3			0	750	2500	5375
4				0	1000	3500
5					0	5000
6						0

```
    vector<int> clos = { 30,35,15,5,10,20,25 };
```

```
    cout << CMU(6, clos);
temp-dp[1][2] = temp-dp[1][1] + temp - dp[2][2] = 15750
Finall-dp[1][2] = 15750
temp-dp[2][3] = temp-dp[2][2] + temp - dp[3][3] = 2625
Finall-dp[2][3] = 2625
temp-dp[3][4] = temp-dp[3][3] + temp - dp[4][4] = 750
Finall-dp[3][4] = 750
temp-dp[4][5] = temp-dp[4][4] + temp - dp[5][5] = 1000
Finall-dp[4][5] = 1000
temp-dp[5][6] = temp-dp[5][5] + temp - dp[6][6] = 5000
Finall-dp[5][6] = 5000
temp-dp[1][3] = temp-dp[1][1] + temp - dp[2][3] = 7875
temp-dp[1][3] = temp-dp[1][2] + temp - dp[3][3] = 7875
Finall-dp[1][3] = 7875
temp-dp[2][4] = temp-dp[2][2] + temp - dp[3][4] = 6000
temp-dp[2][4] = temp-dp[2][3] + temp - dp[4][4] = 4375
Finall-dp[2][4] = 4375
temp-dp[3][5] = temp-dp[3][3] + temp - dp[4][5] = 2500
temp-dp[3][5] = temp-dp[3][4] + temp - dp[5][5] = 2500
Finall-dp[3][5] = 2500
temp-dp[4][6] = temp-dp[4][4] + temp - dp[5][6] = 6250
temp-dp[4][6] = temp-dp[4][5] + temp - dp[6][6] = 3500
Finall-dp[4][6] = 3500
temp-dp[1][4] = temp-dp[1][1] + temp - dp[2][4] = 14875
temp-dp[1][4] = temp-dp[1][2] + temp - dp[3][4] = 14875
temp-dp[1][4] = temp-dp[1][3] + temp - dp[4][4] = 9375
Finall-dp[1][4] = 9375
temp-dp[2][5] = temp-dp[2][2] + temp - dp[3][5] = 13000
temp-dp[2][5] = temp-dp[2][3] + temp - dp[4][5] = 7125
temp-dp[2][5] = temp-dp[2][4] + temp - dp[5][5] = 7125
Finall-dp[2][5] = 7125
temp-dp[3][6] = temp-dp[3][3] + temp - dp[4][6] = 5375
temp-dp[3][6] = temp-dp[3][4] + temp - dp[5][6] = 5375
temp-dp[3][6] = temp-dp[3][5] + temp - dp[6][6] = 5375
Finall-dp[3][6] = 5375
temp-dp[1][5] = temp-dp[1][1] + temp - dp[2][5] = 28125
temp-dp[1][5] = temp-dp[1][2] + temp - dp[3][5] = 27250
temp-dp[1][5] = temp-dp[1][3] + temp - dp[4][5] = 11875
temp-dp[1][5] = temp-dp[1][4] + temp - dp[5][5] = 11875
Finall-dp[1][5] = 11875
temp-dp[2][6] = temp-dp[2][2] + temp - dp[3][6] = 18500
temp-dp[2][6] = temp-dp[2][3] + temp - dp[4][6] = 10500
temp-dp[2][6] = temp-dp[2][4] + temp - dp[5][6] = 10500
temp-dp[2][6] = temp-dp[2][5] + temp - dp[6][6] = 10500
Finall-dp[2][6] = 10500
temp-dp[1][6] = temp-dp[1][1] + temp - dp[2][6] = 36750
temp-dp[1][6] = temp-dp[1][2] + temp - dp[3][6] = 32375
temp-dp[1][6] = temp-dp[1][3] + temp - dp[4][6] = 15125
temp-dp[1][6] = temp-dp[1][4] + temp - dp[5][6] = 15125
temp-dp[1][6] = temp-dp[1][5] + temp - dp[6][6] = 15125
Finall-dp[1][6] = 15125
```




```
    vector<int> clos = { 5,4,6,2,7 };
    cout << CMU(4, clos);
temp-dp[1][2] = temp-dp[1][1] + temp - dp[2][2] = 120
Finall-dp[1][2] = 120
temp-dp[2][3] = temp-dp[2][2] + temp - dp[3][3] = 48
Finall-dp[2][3] = 48
temp-dp[3][4] = temp-dp[3][3] + temp - dp[4][4] = 84
Finall-dp[3][4] = 84
temp-dp[1][3] = temp-dp[1][1] + temp - dp[2][3] = 88
temp-dp[1][3] = temp-dp[1][2] + temp - dp[3][3] = 88
Finall-dp[1][3] = 88
temp-dp[2][4] = temp-dp[2][2] + temp - dp[3][4] = 252
temp-dp[2][4] = temp-dp[2][3] + temp - dp[4][4] = 104
Finall-dp[2][4] = 104
temp-dp[1][4] = temp-dp[1][1] + temp - dp[2][4] = 244
temp-dp[1][4] = temp-dp[1][2] + temp - dp[3][4] = 244
temp-dp[1][4] = temp-dp[1][3] + temp - dp[4][4] = 158
Finall-dp[1][4] = 158
```

总结

1. 输入的矩阵的元素代表的是矩阵的行和列，M[i-1]是第i个元素的行，M[i]是第i个元素的列。例如M[0]是第一个元素的行，M[1]是第二个的行，M[2]是第三个元素的行……同时，M[1]还是第1个元素的列，M[2]是第二个元素的列…..
2. 矩阵乘法中元素的运算次数A[i][j] * A[j][k] = d[i-1]*d[j]*d[k] , 即第一个矩阵的行X第一个矩阵的列X第二个矩阵的列
3. 本题的特点是dp数组是斜着填入的，最外层循环指定的是间隔gap，然后根据gap确定元组（i，j），内层循环i <= k < j用k分割ij
4. dp(i,j)的含义：Ai乘到Aj的值
5. 初始化：dp（i，i）=0，即矩阵不能自己相乘
6. 返回值：dp（1，n），即矩阵从A1乘到An