## Chain matrix multiplication

* 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**）

是一个矩阵，是一个矩阵，相乘，得到的矩阵元素个数为，每个元素由次乘法得到，因此所需乘法次数为。

在计算矩阵连乘积时，加括号的方式对计算量有影响。

例如有三个矩阵连乘，它们的维数分别为  
 ,,。用第一种加括号方式计算，则所需数乘次数为。用第二种加括号方式计算，需要次数乘。

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：

* 表示矩阵的个数
* 表示第个矩阵
* 表示矩阵连乘
* 表示的列数
* 表示的行数
* 表示矩阵连乘断开的位置为，表示在和之间断开
* 表示的最少乘次，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，既是的列数，也是的行数。结合下面的示意图有助于理解：

1. Dpmatrixmul(A1, A2, .., An)  
   For i=1 to n   
    C(i, i)=0;  
   2. For s=1 to n-1 do  
    for i=1 to n-s do   
    j=i+s;  
    C(i, j)=min{C(i,k)+C(k+1, j)+mi-1.mk.mj,   
    i≤k<j};  
     
   3. Return C(1, n).

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

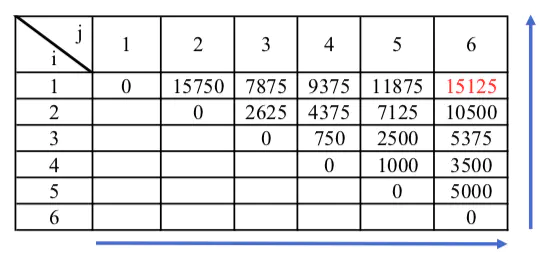
for k = begin; k < end; k++:

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

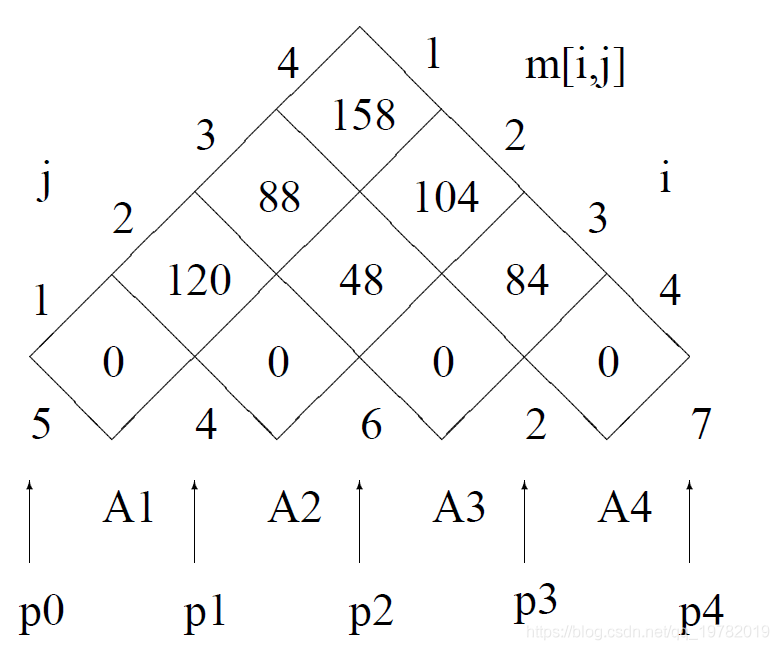
3.return dp[1][n];

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;  
}

* 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



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