 给定n个矩阵A1，A2.......An，其中Ai与Ai+1是可乘的。

    由于矩阵的乘法满足结合律，故计算矩阵的连乘积可以有许多不同的计算次序。这中计算次序可以用加括号的方式来确定。例如，矩阵连乘积A1A2A3A4可以有一下5种不同的完全加括号方式：

    (A1(A2(A3A4)))

    (A1((A2A3)A4))

    ((A1A2)(A3A4))

    ((A1(A2A3))A4)

    (((A1A2)A3)A4)

    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，其乘机C=AB是一个p\*r的矩阵，总共需要pqr次数乘。

    为了说明在计算矩阵连乘积时，加括号方式对整个计算量的影响，我们考察计算3个矩阵A1A2A3的连乘积的例子。这3个矩阵的尺寸分别为10\*100，100\*5和5\*50。若以((A1A2)A3)这种方式计算，3个矩阵的连乘积需要的数乘次数为7500。若以(A1(A2A3))这种方式计算，所需的数乘次数为75000。显然，在即算矩阵连乘积时，加括号方式对计算量有很大影响。

    于是，人们自然会提出矩阵连乘积的最优计算次序问题。即对于给定的n个矩阵，如何确定计算矩阵连乘积的计算次序（完全加括号方式），使得以此计算次序计算矩阵连乘积所需的数乘次数最少。

    穷举搜索法不是一个有效算法，它的时间复杂度是随n的增长呈指数增长的。下面考虑用动态规划法解矩阵连乘积的最优计算次序问题。

    1.分析最优解的结构

    设计求解具体问题的动态规划算法的第一步是刻画该问题的最优解的结构特征。我们将矩阵连乘积AiAi+1....Aj简记为A[ i : j ]。考察计算A[ 1: n]的最优计算次序。设这个计算次序在矩阵Ak和Ak+1之间将矩阵链断开，1<=k<n，则其相应的完全加括号形式为((A1...Ak)(Ak+1...An))。以此次序，总的计算量为A[ 1 : k ]的计算量加上A[ k+1 : n ]的计算量， 再加上A[ 1 : k ]和A[ k+1 : n ]相称的计算量。

    这个问题的关键特征是：计算A[ 1 :n ]的最优次序所包含的计算矩阵子链a[ 1 : k ]和A[ k+1 : n ]的次序也是最优的。因此，矩阵连乘积计算次序问题的最优解包含着其子问题的最优解。这种性质称为最优子结构性质。问题的最优子结构性质是该问题可以用动态规划算法求解的显著特征。

    2.建立递归关系

    设计动态规划算法的第二步就是递归地定义最优值。对于矩阵连乘积的最有计算次序问题，设计算A[ i : j ], 1<=i<=j<=n，所需的最少数乘次数为m[ i ][ j ]，则原问题的最优值为m[ 1 ][ n]。

    当i=j时，A[ i  ; j ]=Ai，为单一矩阵，无需计算，因此m[ i ][ i ]=0。

    当i < j时，可以利用最优子结构的性质来计算m[ i ][ j ]。事实上，若计算A[ i : j ]的最优次序在Ak和Ak+1之间断开，i<=k<j，则m[ i ][ j ]=m[ i ][ k ]+m[k+1][ j ]+Pi-1\*Pk\*Pj。其中Pi表示第i个矩阵的列数，也是第i-1个矩阵的行数，P0表示第一个矩阵的行数。由于在计算时并不知道断开点k的位置，所以k还未定。不过k的位置只有j-i个可能。从而m[ i ][ j ]可以递归地定义为

                        当i=j   m[ i ][ j ] = 0

                        当i<j   m[ i ][ j ] = min{ m[ i ][ k ]+m[ k+1 ][ j ]+Pi-1\*Pk\*Pj }

    m[ i ][ j ]给出了最优值，即计算A[ i : j ]所需的最少数乘次数。同时还确定了计算A[ i : j ]的最优次序中的断开位置k，也就是说，对于这个k有

                        m[ i ][ j ]=m[ i [ k ]+m[ k+1 ][ j] + Pi-1\*Pk\*Pj

    若将对应于m[ i ][ j ]的断开位置k记为s[ i ][ j ]， 在计算最优值m[ i ][ j ]后，可以递归地有s[ i ][ j ]构造出相应的最优解。

    3.计算最优值

    根据计算m[ i ][ j ]的递归式，容易写一个递归算法计算m[ 1 ][ n ]。但是简单地递归将好费指数计算时间。在递归计算时，许多子问题被重复计算多次。这也是该问题可以用动态规划算法求解的又一显著特征。

    用动态规划算法解决此问题，可依据其递归是以自底向上的方式进行计算。在计算的过程中，保存以解决的子问题答案。每个子问题只计算一次，而在后面需要时只要简单查一下，从而避免大量的重复计算。全部代码如下：

**[cpp]** [view plain](http://blog.csdn.net/simmerlee/article/details/7731594" \o "view plain) [copy](http://blog.csdn.net/simmerlee/article/details/7731594" \o "copy)

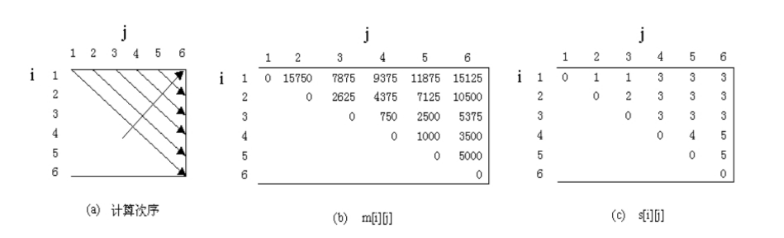
1. #include <stdio.h>
3. **void** matrix\_chain(**int** \* p, **int** n, **int**[][100], **int**[][100]);
4. **void** trace\_back(**int** i, **int** j, **int** [][100]);
6. **int** main()
7. {
8. **int** size[100], total, i=0, m[100][100], s[100][100],j;
9. printf("Please enter the number of matrix:\n");
10. scanf("%d", &total);
11. printf("Please enter a series of number.\nThe first number is the row of the first matrix, others are column of each matrix respectively:\n");
12. **while**(i<=total)
13. scanf("%d", &size[i++]);
14. matrix\_chain(size, total, m, s);
15. printf("The result is:\n");
16. trace\_back(1, 6, s);
17. printf("\n");
18. **return** 0;
19. }
21. **void** matrix\_chain(**int** \* p, **int** n, **int** m[][100], **int** s[][100])
22. {
23. **int** i,j,k,r,t;
24. **for**(i=1; i<=n; i++) m[i][i]=0;                   //对角线的元素都为0
25. **for**(r=2; r<=n; r++)
26. **for**(i=1; i<=n-r+1; i++)
27. {
28. j=i+r-1;
29. m[i][j]=m[i+1][j]+p[i-1]\*p[i]\*p[j];      //先计假设第一个点是最佳断开位置，算出其所需的数乘次数
30. s[i][j]=i;                               //记录下断开的位置
31. **for**(k=i+1; k<j; k++)
32. {
33. t=m[i][k]+m[k+1][j]+p[i-1]\*p[k]\*p[j];//如果从其他断开点断开矩阵链，所需的数乘次数比原来的小，就替换
34. **if**(t<m[i][j])
35. {m[i][j]=t; s[i][j]=k;}
36. }
37. }
38. }
40. **void** trace\_back(**int** i, **int** j, **int** s[][100])
41. {
42. **if**(i==j) {printf("A%d",i);**return**;}
43. printf("(");
44. trace\_back(i,s[i][j],s);
45. printf(",");
46. trace\_back(s[i][j]+1, j, s);
47. printf(")");
48. }

    算法matrix\_chain首先计算出m[ i ][ i ]=0，然后再根据递归式，按照矩阵链长递增的方式依次计算m[ i ][ i+1 ]，m[ i ][ i+2 ]......在计算m[ i ][ j ]时，只是用到已计算出的m[ i ][ k ]和m[ k+1 ][ j ]。

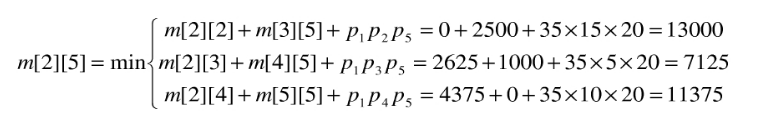
    比如，计算矩阵连乘积A1A2A3A4A5A6，其尺寸分别为：

    A1 : 30\*35        A2 : 35\*15        A3 : 15\*5        A4 : 5\*10        A5 : 10\*20        A6 : 20\*25

    动态规划算法matrix\_chain计算m[ i ][ j ]先后次序如图所示，计算结果为m[ i ][ j ]和s[ i ][ j ]，其中第0行和第0列没有使用。



    例如，在计算m[2][5]时，依递归式有



    所以m[2][5] = 7125，且k=3，因此，s[2][5]=3。

    算法matrix\_chain的主要计算量取决于程序中对r，i和k的三重循环。循环体内的计算量为O( 1 )，而三重循环的总次数为O( n^3 )。因此，概算发的计算时间的上界为O(n^3)。所占用的空间显然为O(n^2)。由此可见动态规划算法比穷举搜索法要有效的多。

    4.构造最优解

    动态规划算法的第四布是构造问题的最优解。算法matrix\_chain只是计算出了最优值，并未给出最优解。也就是说，通过matrix\_chain的计算，只是到最少数乘次数，还不知道具体应按什么次序来做矩阵乘法才能达到最少的数乘次数。

    事实上，matrix\_chain已记录了构造最优解所需要得全部信息。s[ i ][ j ]中的数表明，计算矩阵链A[ i : j ]的最佳方式应在矩阵Ak和Ak+1之间断开，级最优的加括号方式应为(A[ i : k ])(A[ k+1 : j ])。因此，从s[ 1 ][ n ]记录的信息可知计算A[ 1 : n ]的最优加括号方式为(A[ 1 : s[ 1 ][ n ]})(A[s[ 1 ][ n ]+1 : n])。同理可以最终确定A[ 1 : n ]的最优完全加括号方式，即构造出问题的一个最优解。

    算法trace\_back按算法batrix\_chain计算出的断点矩阵s只是的加括号方式输出计算A[ i : j ]的最优计算次序。要输出A[ 1 : n ]的最优计算次序只要调用trace\_back(1, n, s)即可。对上面所举的例子，有如下运行结果：

Please enter the number of matrix:  
6  
Please enter a series of number.  
The first number is the row of the first matrix, others are column of each matrix respectively:  
30 35 15 5 10 20 25  
The result is:  
((A1,(A2,A3)),((A4,A5),A6))

•矩阵连乘计算次序问题的最优解包含着其子问题的最优解。这种性质称为最优子结构性质。

•在分析问题的最优子结构性质时，所用的方法具有普遍性：首先假设由问题的最优解导出的子问题的解不是最优的，然后再设法说明在这个假设下可构造出比原问题最优解更好的解，从而导致矛盾。

•利用问题的最优子结构性质，以自底向上的方式递归地从子问题的最优解逐步构造出整个问题的最优解。最优子结构是问题能用动态规划算法求解的前提。

•递归算法求解问题时，每次产生的子问题并不总是新问题，有些子问题被反复计算多次。这种性质称为子问题的重叠性质。

•动态规划算法，对每一个子问题只解一次，而后将其解保存在一个表格中，当再次需要解此子问题时，只是简单地用常数时间查看一下结果。

•通常不同的子问题个数随问题的大小呈多项式增长。因此用动态规划算法只需要多项式时间，从而获得较高的解题效率。

#include<stdio.h>

#include<iostream>

using namespace std;

int min(int a,int b){

return a<b?a:b;

}

int main(){

int n,p[105],f[105][105];

while(scanf("%d",&n)&&n){

int i,j,k;

for(i=0;i<=n;i++)scanf("%d",&p[i]);

for(i=1;i<=n;i++){

for(j=0;j<=n;j++)f[i][j]=1<<30;

f[i][i]=0;

f[i][i+1]=p[i-1]\*p[i]\*p[i+1];

}

for(k=2;k<=n;k++)

for(i=1;i<=n-k;i++)

for(j=0;j<k;j++)

f[i][i+k]=min( (f[i][i+j]+f[i+j+1][i+k]+p[i-1]\*p[i+j]\*p[i+k]), (f[i][i+k]) );

printf("%d\n",f[1][n]);

}

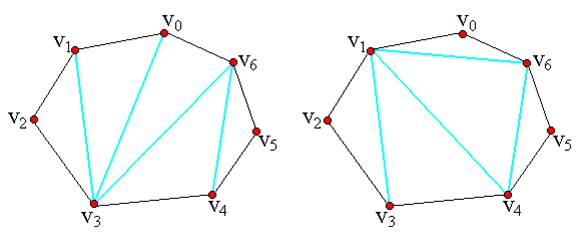
return 0;

}

**最优三角剖分**

**题目描述：**

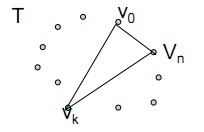
用多边形顶点的逆时针序列表示凸多边形，即P={v0,v1,…,vn-1}表示具有n条边的凸多边形。



　　给定凸多边形P，以及定义在由多边形的边和弦组成的三角形上的权函数w。要求确定该凸多边形的三角剖分，使得即该三角剖分中诸三角形上权之和为最小。

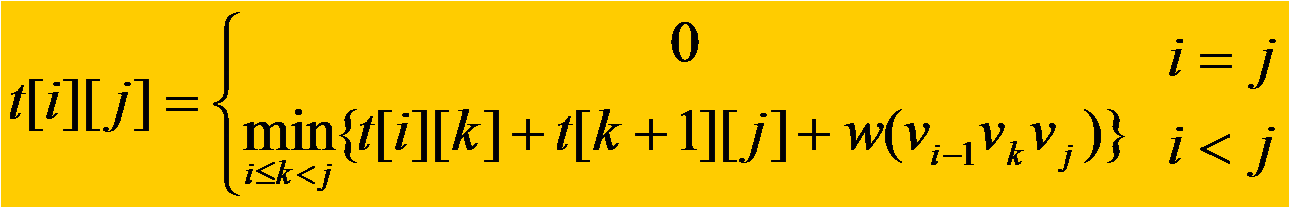
**解题思路：**

　　若凸(n+1)边形P={v0,v1,…,vn-1}的最优三角剖分T包含三角形v0vkvn，1≤k≤n-1，则T的权为3个部分权的和：三角形v0vkvn的权，子多边形{v0,v1,…,vk}和{vk,vk+1,…,vn}的权之和。可以断言，由T所确定的这2个子多边形的三角剖分也是最优的。因为若有{v0,v1,…,vk}或{vk,vk+1,…,vn}的更小权的三角剖分将导致T不是最优三角剖分的矛盾。

****

　　那么我们定义一个t[i][j]，1<=i<=j<=N,为凸子多边形{**vi-1,vi,…,vj**}的最优三角剖分所对应的权函数值，即其最优值。据此定义，要计算的凸(n+1)边形P的最优权值为t[1][n]。

　　t[i][j]的值可以利用最优子结构性质递归地计算。当j-i≥1时，凸子多边形至少有3个顶点。由最优子结构性质，t[i][j]的值应为t[i][k]的值加上t[k+1][j]的值，再加上三角形vi-1vkvj的权值，其中i≤k≤j-1。由于在计算时还不知道k的确切位置，而k的所有可能位置只有j-i个，因此可以在这j-i个位置中选出使t[i][j]值达到最小的位置。由此，t[i][j]可递归地定义为：



对于要求的t[1][n]，可以用通过由下至上的，从链长（多边形的边）为2开始计算，每次求t[i][j]的最小值，并且记录最小值所对应的K值，根据最优子结构的性质，逐步向上就可以求出t[1][n]的最小值。

类似的，求三角划分顶点的乘积的最小值问题，也是一样的。