**[动态规划——矩阵连乘的问题](http://www.cnblogs.com/liushang0419/archive/2011/04/27/2030970.html)**

《问题的引出》

看下面一个例子，计算三个矩阵连乘{A1，A2，A3}；维数分别为10\*100 , 100\*5 , 5\*50

按此顺序计算需要的次数（（A1\*A2）\*A3）:10X100X5+10X5X50=7500次

按此顺序计算需要的次数（A1\*（A2\*A3））:10X5X50+10X100X50=75000次

所以问题是：如何确定运算顺序，可以使计算量达到最小化。

枚举显然不可，如果枚举的话，相当于一个“完全加括号问题”，次数为卡特兰数,卡特兰数指数增长，必然不行。

《建立递归关系》

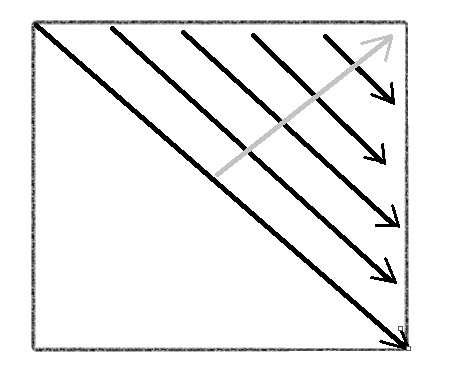
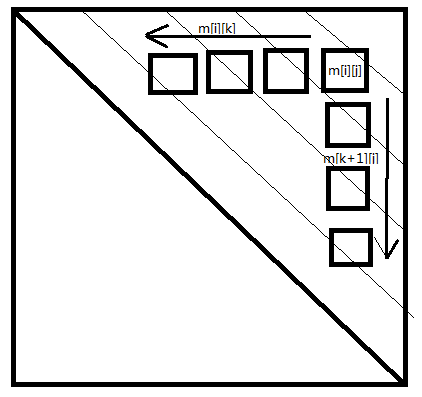
子问题状态的建模（很关键）：令m[i][j]表示第i个矩阵至第j个矩阵这段的最优解。

显然如果i=j，则m[i][j]这段中就一个矩阵，需要计算的次数为0；

     如果i>j，则m[i][j]=min{m[i][k]+m[k+1][j]+p[i-1]Xp[k]Xp[j]},其中k,在i与j之间游荡，所以i<=k<j ;

代码实现时需要注意的问题：计算顺序！！！

因为你要保证在计算m[i][j]查找m[i][k]和m[k+1][j]的时候，m[i][k]和m[k+1][j]已经计算出来了。

观察坐标的关系如图：

所以计算顺序如上右图：相应的计算顺序对应代码为13-15行

m[1][n]即为最终求解，最终的输出想为（（A1（A2 A3））（（A4 A5）A6））的形式，不过没有成功，待思考...

1 #include<iostream>  
 2  using namespace std;  
 3  const int MAX = 100;  
 4 //p用来记录矩阵的行列，main函数中有说明  
 5 //m[i][j]用来记录第i个矩阵至第j个矩阵的最优解  
 6 //s[][]用来记录从哪里断开的才可得到该最优解  
 7 int p[MAX+1],m[MAX][MAX],s[MAX][MAX];  
 8 int n;//矩阵个数  
 9   
10 void matrixChain(){  
11 for(int i=1;i<=n;i++)m[i][i]=0;  
12   
13 for(int r=2;r<=n;r++)//对角线循环  
14 for(int i=1;i<=n-r+1;i++){//行循环  
15 int j = r+i-1;//列的控制  
16 //找m[i][j]的最小值，先初始化一下，令k=i  
17 m[i][j]=m[i][i]+m[i+1][j]+p[i-1]\*p[i]\*p[j];  
18 s[i][j]=i;  
19 //k从i+1到j-1循环找m[i][j]的最小值  
20 for(int k = i+1;k<j;k++){  
21 int temp=m[i][k]+m[k+1][j]+p[i-1]\*p[k]\*p[j];  
22 if(temp<m[i][j]){  
23 m[i][j]=temp;  
24 //s[][]用来记录在子序列i-j段中，在k位置处  
25 //断开能得到最优解  
26 s[i][j]=k;  
27 }  
28 }  
29 }  
30 }  
31   
32 //根据s[][]记录的各个子段的最优解，将其输出  
33 void traceback(int i,int j){  
34 if(i==j)return ;  
35   
36 traceback(i,s[i][j]);  
37 traceback(s[i][j]+1,j);  
38 cout<<"Multiply A"<<i<<","<<s[i][j]<<"and A"<<s[i][j]+1<<","<<j<<endl;  
39 }  
40   
41 int main(){  
42 cin>>n;  
43 for(int i=0;i<=n;i++)cin>>p[i];  
44 //测试数据可以设为六个矩阵分别为  
45 //A1[30\*35],A2[35\*15],A3[15\*5],A4[5\*10],A5[10\*20],A6[20\*25]  
46 //则p[0-6]={30,35,15,5,10,20,25}  
47 //输入：6 30 35 15 5 10 20 25  
48 matrixChain();  
49   
50 traceback(1,n);  
51 //最终解值为m[1][n];  
52 cout<<m[1][n]<<endl;  
53 return 0;  
54 }

第二种：

#include<stdlib.h>  
#include<stdio.h>  
#include<limits.h>  
#include<time.h>  
#define MAX\_VALUE 100  
#define N 201 //连乘矩阵的个数(n-1)  
#define random() rand()%MAX\_VALUE   //控制矩阵的行和列的大小  
int c[N][N], s[N][N], p[N];  
int matrixchain(int n)    //3个for循环实现  
{    for(int i=0;i<n;i++)  
        c[i][i]=0;//自己根自己之间只有一个矩阵，无需计算 为0

    for(int L=2;L<=n;L++)  
        for(int i=1;i<=n-L+1;i++)//i表第几个，后面减一调整下表从0开始的问题   
        {  
            int j=i+L-1;  
            c[i-1][j-1]=INT\_MAX;//i-1因为下表从0开始   
            for(int k=i;k<=j-1;k++)//k全部断链试试，找最好的ij之间   
            {  
                int q=c[i-1][k-1]+c[k][j-1]+p[i-1]\*p[k]\*p[j];  
                if(q<c[i-1][j-1])  
                {  
                    c[i-1][j-1]=q;  
                    s[i-1][j-1]=k;  
                }  
            }  
        }  
    return c[1][n];//最终计算到1到n之间总的计算量   
}  
void Print(int s[][N],int i,int j) // 输出矩阵连乘积的计算次序  
{    if(i==j)  
        printf("A%d",i);  
    else  
    {  
        printf("(");  
        Print(s,i,s[i][j]); // 左半部子矩阵连乘  
        Print(s,s[i][j]+1,j); //左半部子矩阵连乘  
        printf(")");  
    }  
}  
int lookupchain(int i,int j) //备忘录方法  
{  
    if(c[i][j]>0)  
        return c[i][j];  
    if(i==j)  
        return 0;  
    int u=lookupchain(i,i)+lookupchain(i+1,j)+p[i-1]\*p[i]\*p[j];  
    s[i][j]=i;  
    for(int k=i+1;k<j;k++)  
    {  
        int t=lookupchain(i,k)+lookupchain(k+1,j)+p[i-1]\*p[k]\*p[j];  
        if(t<u)  
        {  
            u=t;  
            s[i][j]=k;  
        }  
    }  
    c[i][j]=u;  
    return u;  
}  
int main()  
{  
    srand((int)time(NULL));  
    for(int i=0;i<N;i++) // 随机生成数组p[]，各个元素的值的范围：1~MAX\_VALUE  
        p[i]=random()+1;  
    clock\_t start,end;     
    double elapsed;  
    start=clock();  
    //cout<<"Count: "<<matrixchain(N-1)<<endl;   //3重for循环实现  
    printf("Count: %d\n",lookupchain(1,N-1));   //备忘录方法  
    end=clock();  
    elapsed=((double)(end-start));///CLOCKS\_PER\_SEC;   
    printf("Time: %d\n",elapsed);  
    Print(s,1,N-1); //输出矩阵连乘积的计算次序  
    system("pause");  
    return 0;  
}

第三种：

矩阵连乘问题

【**问题**】：[**矩阵链乘问题**](http://www.ezloo.com/2008/05/matrixchain.html)：给定n个矩阵｛A1,A2,...,An｝，其中Ai与Ai+1是可乘的，i=1，2...，n-1。如何确定计算矩阵连乘积的计算次序，使得依此次序计算矩阵连乘积需要的数乘次数最少。  
【**解题**】：这里我采用的是动态划分算法：  
设计**动态规划算法**的步骤。  
(1)找出最优解的性质，并刻划其结构特征。  
(2)递归地定义最优值。  
(3)以自底向上的方式计算出最优值。  
(4)根据计算最优值时得到的信息，构造最优解(由子结构的最优解得到原先大问题的最优解)。  
【**解题关键**】：将一系列相乘的矩阵（Ai....Aj）划分为两部分;即（AiAi+1...Ak）(Ak+1Ak+2....Aj)，k的位置要保证左边括号和右边括号相乘的消耗最小。

【**思路**】：这里我采用两种方法来实现：  
（1）用三重for循环来实现：根据主对角线的方向及其右边与之平行的对角线方向，由上至下，从左到右分别求出C[i][j]的值，后面要求的值都可以根据前面所求的的值来求。  
C[i][j]代表矩阵链Ai..Aj相乘的最小消耗。其中：c[i][i]=0，i=1,2,....n  
求解的顺序如下：C[1][2],C[2][3],C[2][3],...,C[N-1][N],C[1][3],C[2][4]....C[N-2][N]..........C[N][N]  
最后得到的C[N][N]的值就是我们所求的。  
（2）备忘录方法（即递归算法）：将整个矩阵链分成两部分，然后在分别对两边的子矩阵链递归调用算法。  
【**程序代码**】：两种方法都在其中：

#include<iostream.h>  
#include<stdlib.h>  
#include<limits.h>  
#include<time.h>  
#define MAX\_VALUE 100  
#define N 201 //连乘矩阵的个数(n-1)  
#define random() rand()%MAX\_VALUE   //控制矩阵的行和列的大小  
int c[N][N], s[N][N], p[N];  
int matrixchain(int n)    //3个for循环实现  
{    for(int k=1;k<=n;k++)  
        c[k][k]=0;  
    for(int d=1;d<n;d++)  
        for(int i=1;i<=n-d;i++)  
        {    int j=i+d;  
            c[i][j]=INT\_MAX;  
            for(int m=i;m<j;m++)  
            {    int t=c[i][m]+c[m+1][j]+p[i-1]\*p[m]\*p[j];  
                if(t<c[i][j])  
                {  
                    c[i][j]=t;  
                    s[i][j]=m;  
                }  
            }  
        }  
    return c[1][n];  
}  
void Print(int s[][N],int i,int j) //  输出矩阵连乘积的计算次序  
{    if(i==j)  
        cout<<"A"<<i;  
    else  
    {  
        cout<<"(";  
        Print(s,i,s[i][j]);  // 左半部子矩阵连乘  
        Print(s,s[i][j]+1,j);  //左半部子矩阵连乘  
        cout<<")";  
    }  
}  
int lookupchain(int i,int j)  //备忘录方法  
{  
    if(c[i][j]>0)  
        return c[i][j];  
    if(i==j)  
        return 0;  
    int u=lookupchain(i,i)+lookupchain(i+1,j)+p[i-1]\*p[i]\*p[j];  
    s[i][j]=i;  
    for(int k=i+1;k<j;k++)  
    {  
        int t=lookupchain(i,k)+lookupchain(k+1,j)+p[i-1]\*p[k]\*p[j];  
        if(t<u)  
        {  
            u=t;  
            s[i][j]=k;  
        }  
    }  
    c[i][j]=u;  
    return u;  
}  
void main()  
{  
    srand((int)time(NULL));  
    for(int i=0;i<N;i++)  //  随机生成数组p[]，各个元素的值的范围：1~MAX\_VALUE  
        p[i]=random()+1;  
    clock\_t start,end;     
    double elapsed;  
    start=clock();  
    //cout<<"Count: "<<matrixchain(N-1)<<endl;   //3重for循环实现  
    cout<<"Count: "<<lookupchain(1,N-1)<<endl;   //备忘录方法  
    end=clock();  
    elapsed=((double)(end-start));///CLOCKS\_PER\_SEC;   
    cout<<"Time: "<<elapsed<<endl;  
    Print(s,1,N-1);  //输出矩阵连乘积的计算次序  
    cout<<endl;  
}

【**总结**】：两种算法的时间复杂度均为o(n3)，,随着数据量的增多，备忘录方法消耗的时间越长；我觉得是由于递归算法，随着数据量增大，调用函数的次数也增大，语句被执行的时间也越多，因此调用函数消耗的时间也增多。