【动态规划】矩阵链乘法

矩阵链乘法

求解矩阵链相乘问题时动态规划算法的另一个例子。给定一个n个矩阵的序列（矩阵链）<A1,A2,...,An>，我们希望计算它们的乘积 A1A2...An

为了计算表达式，我们可以先用括号明确计算次序，然后利用标准的矩阵相乘算法进行计算。**完全括号化**(fully parenthesized)：它是单一矩阵，或者是两个完全括号化的矩阵乘积链的积。

例如如果有矩阵链为<A1,A2,A3,A4>,则共有5种完全括号化的矩阵乘积链。

(A1(A2(A3A4)))、(A1((A2A3)A4))、((A1A2)(A3A4))、((A1(A2A3))A4)、((A1(A2A3))A4)

对矩阵链加括号的方式会对**乘积运算的代价产生巨大影响**。我们先来分析两个矩阵相乘的代价。下面的伪代码的给出了两个矩阵相乘的标准算法，属性rows和columns是矩阵的行数和列数。

```
MATRIX-MULTIPKLY(A,B)
if A.columns≠B.rows
        error "incompatible dimensions"
else let C be a new A.rows×B.columns matrix
        for i = 1 to A.rows
             for j = 1 to B.columns
                  c(ij)=0
                   for k = 1 to A.columns
                         c(ij)=c(ij)+a(ik)*b(kj)
return C
```

两个矩阵A和B只有**相容(compatible)**，即A的列数等于B的行数时，才能相乘。如果A是p×q的矩阵，B是q×r的矩阵，那么乘积C是p×r的矩阵。计算C所需要时间由第8行的标量乘法的次数决定的，即pqr。

以矩阵链<A1,A2,A3>为例，来说明不同的加括号方式会导致不同的计算代价。假设三个矩阵的规模分别为10×100、100×5和5×50。

如果按照((A1A2)A3)的顺序计算，为计算A1A2(规模10×5)，需要做10*100*5=5000次标量乘法，再与A3相乘又需要做10*5*50=2500次标量乘法，共需**7500**次标量乘法。

如果按照(A1(A2A3))的顺序计算，为计算A2A3(规模100×50)，需

100*5*50=25000次标量乘法，再与A1相乘又需10*100*50=50000次标量乘法，共需**75000**次标量乘法。因此第一种顺序计算要比第二种顺序计算**快10倍**。

矩阵链乘法问题(matrix-chain multiplication problem)可描述如下：给定n个矩阵的链<A1,A2,...,An>，矩阵Ai的规模为p(i-1)×p(i) (1<=i<=n)，求完全括号化方案，使得计算乘积A1A2...An所需标量乘法**次数最少**。

因为括号方案的数量与n呈指数关系，所以通过暴力搜索穷尽所有可能的括号化方案来寻找最优方案是一个糟糕策略。

应用动态规划方法

下面用动态规划方法来求解矩阵链的最优括号方案，我们还是按照之前提出的4个步骤进行：

1.刻画一个最优解的结构特征

2.递归地定义最优解的值

3.计算最优解的值，通常采用自底向上的方法

4.利用计算出的信息构造一个最优解

接下来按顺序进行这几个步骤，清楚地展示针对本问题每个步骤应如何做。

步骤1：最优括号化方案的结构特征

动态规划的第一步是寻找最优子结构，然后就可以利用这种子结构从子问题的最优解构造出原问题的最优解。在矩阵链乘法问题中，我们假设A(i)A(i+1)...A(j)的最优括号方案的分割点在A(k)和A(k+1)之间。那么，继续对“前缀”子链A(i)A(i+1)..A(k)进行括号化时，我们应该直接采用**独立求解**它时所得的最优方案。

我们已经看到，一个非平凡(i≠j)的矩阵链乘法问题实例的任何解都需要划分链，而任何最优解都是由子问题实例的最优解构成的。为了构造一个矩阵链乘法问题实例的最优解，我们可以将问题**划分为两个子问题**(A(i)A(i+1)...A(k)和A(k+1)A(k+2)..A(j)的最优括号化问题)，**求出子问题实例的最优解，然后将子问题的最优解组合起来**。我们必须保证在确定分割点时，已经考察了所有可能的划分点，这样就可以保证不会遗漏最优解。

步骤2：一个递归求解方案

下面用子问题的最优解来递归地定义原问题最优解的代价。对于矩阵链乘法问题，我们可以将对所有1<=i<=j<=n确定A(i)A(i+1)...A(j)的最小代价

括号化方案作为子问题。令m[i,j]表示计算矩阵A(i..j)所需标量乘法次数的最小值，那么，原问题的最优解—计算A(1..n)所需的最低代价就是m[1,n]。

我们可以递归定义m[i,j]如下。对于i=j时的平凡问题，矩阵链只包含唯一的矩阵A(i..j)=A(i)，因此不需要做任何标量乘法运算。所以，对所有i=1,2,...,n，m[i,i]=0。若i<j，我们利用步骤1中得到的最优子结构来计算m[i,j]。我们假设A(i)A(i+1)...A(j)的最优括号化方案的**分割点**在矩阵A(k)和A(k+1)之间，其中i<=k<j。那么，**m[i,j]就等于计算A(i..k)和A(k+1..j)的代价加上两者相乘的代价的最小值**。由于矩阵**Ai的大小为p(i-1)*pi**，易知A(i..k)和A(k+1..j)相乘的代价为p(i-1)p(k)p(j)次标量乘法运算。因此，我们得到

m[i,j]=m[i,k]+m[k+1,j]+ p(i-1)p(k)p(j)

此递归公式假定最优分割点k是已知的，但实际上我们是不知道。不过，k只有j-i种可能的取值，即k=i,i+1,...,j-1。由于最优分割点必在其中，我们只需检查所有可能情况，找到最优者即可。

因此，A(i)A(i+1)...A(j)的最小代价括号化方案的递归求解公式变为：

①如果i=j，m[i,j]=0

②如果i<j，m[i,j]=min{m[i,k]+m[k+1,j]+p(i-1)p(k)p(j)} i<=k<j

m[i,j]的值给出了子问题最优解的代价，但它并未提供足够的信息来构造最优解。为此，我们用**s[i,j]保存最优括号化方案的分割点位置k**，即使得m[i,j]=m[i,k]+[k+1,j]+p(i-1)p(k)p(j)成立的k值。

步骤3：计算最优代价

现在，我们可以很容易地基于递归公式写出一个递归算法，但递归算法是指数时间的，并不必检查若有括号化方案的暴力搜索方法更好。注意到，我们需要求解的不同子问题的数目是相对较少的：每对满足1<=i<=j<=n 的i和j对应一个唯一的子问题，共有n^2(最少)。递归算法会在递归调用树的不同分支中多次遇到同一个子问题。这种**子问题重叠**的性质是应用动态规划的另一标识(第一个标识是最优子结构)。

我们采用自底向上表格法代替递归算法来计算最优代价。此过程假定矩阵Ai的规模为p(i-1)×pi(i=1,2,...,n)。它的输入是一个序列p=<p0,p1,...,pn>，其长度为p.length=n+1。过程用一个辅助表m[1..n,1..n]来

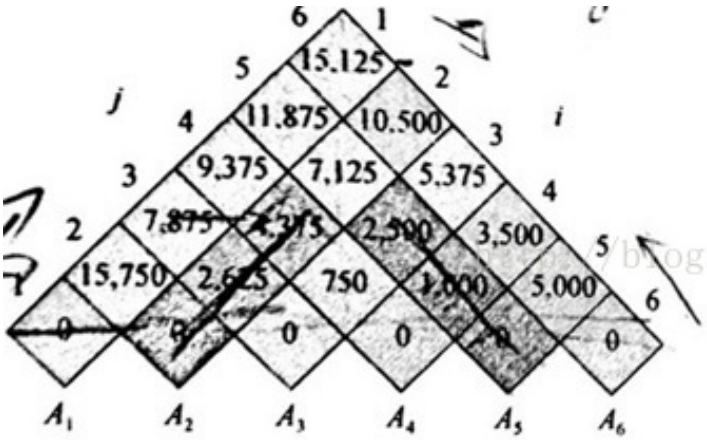
保存代价m[i,j]，用另一个辅助表s[1..n-1,2..n](s[1,2]..s[n-1,n]这里i<j)记录最优值m[i,j]对应的分割点k。我们就可以利用表s构造最优解。

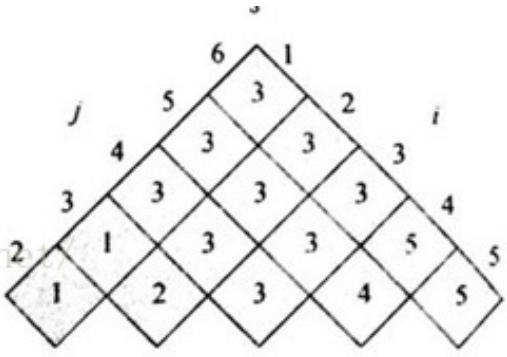
对于矩阵A(i)A(i+1)...A(j)最优括号化的子问题，我们认为其规模为链的**长度j-i+1**。因为j-i+1个矩阵链相乘的最优计算代价m[i,j]只依赖于那么少于j-i+1个矩阵链相乘的最优计算代价。因此，算法应该**按长度递增**的顺序求解矩阵链括号化问题，并按对应的顺序填写表m。(C++实现)

```
void MATRIX_CHAIN_ORDER(int *p,int Length,int m[][M],int s[][M])

        for(int i=1;i<=n;i++) m[i][i]=0;

        for(int l=2;l<=n;l++)    /* 矩阵链的长度 */

                for(int i=1;i<=n-l+1;i++)

                        int j=i+l-1;          /* 等价于 l=j-i+1 */

                        for(int k=i;k<=j-1;k++)

                                q=m[i][k]+m[k+1][j]+p[i-1]*p[k]*p[j];
```

算法5~17行for循环的第一个循环步中，利用公式对所有i=1,2,...,n-1计算m[i,i+1](长度l=2的链的最小计算代价)。第二个循环步中，算法对所有i=1,2,...,n-2计算m[i,i+2](长度l=3的链的最小计算代价)，依次类推。

下图展示了对一个长度为6的矩阵链执行此算法的过程。由于定义m[i,j]仅在i<=j时有意义，因此表m只使用主对角线之上的部分，图中的表是经过旋转的，主对角线已经旋转到水平方向，MATRIX_CHAIN_ORDER**按自下而上、自左至右的顺序**计算所有行。当计算表m[i,j]时，会用到乘积p(i-1)p(k)p(j)(k=i,i+1,...j-1)，语句m[i,j]**西南方向和东南方向**上所有表项。





n=6和矩阵规模如下表时，MATRIX_CHAIN_ORDER计算出m表和s表

矩阵	A1	A2	A3	A4	A5	A6
规模	30×35	35×15	15×5	5×10	10×20	20×25

6个矩阵相乘所需的最少标量乘法运算次数为m[1,6]=15125。表中有些表项被标记了深色阴影，相同的阴影表示在第13行中计算m[2,5]时同时访问了这些表项：

$$m[2,5] = \min\begin{cases} m[2,2]+m[3,5]+p_1p_2p_5 = 0+2\,500+35\cdot15\cdot20 = 13\,000 \\ m[2,3]+m[4,5]+p_1p_3p_5 = 2\,625+1\,000+35\cdot5\cdot20 = 7\,125 \\ m[2,4]+m[5,5]+p_1p_4p_5 = 4\,375+0+35\cdot10\cdot20 = 11\,375 \end{cases}$$
$$= 7\,125$$

简单分析MATRIX_CHAIN_ORDER的嵌套循环结构，可以看到算法的运行时间为O(n^3)。

步骤4：构造最优解

虽然MATRIX_CHAIN_ORDER求出了计算矩阵链乘积所需的最少标量乘法运算次数，但它并未直接指出如何进行这种最优代价的矩阵链乘法计算。表s[i,j]记录了一个k值，指出A(i)A(i+1)...A(j)的最优括号化方案的分割点应在A(k)和A(k+1)之间。

因此，我们A(1..n)的最优计算方案中最后一次矩阵乘法运算应该是以**s[1,n]为分界**的A(1..s[1,n])*A(s[1,n]+1..n)。我们可以用相同的方法递归地求出更早的矩阵乘法的具体计算过程，因为s[1,s[1,n]]指出了计算A(1..s[1,n])时应进行的最后一次矩阵乘法运行；s[s[1,n]+1,n]指出了计算A(s[1,n]+1..n)时应进行的最后一次矩阵乘法运算。下面给出的递归过程可以输出<A(i),A(i+1),...,A(j)>的最优括号化方案。

```
void PRINT_OPTIMAL_PARENS(int s[][M],int i,int j)

        if(i == j) cout<<"A"<<i;
```

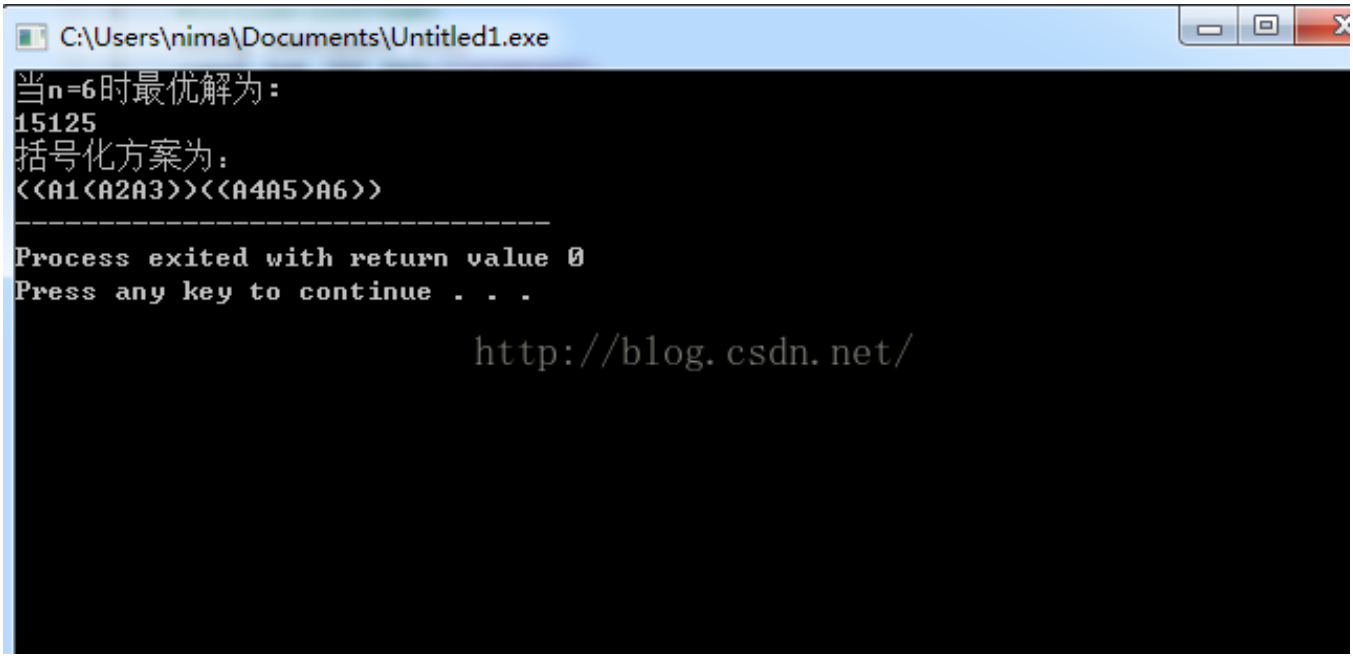
```
                PRINT_OPTIMAL_PARENS(s,i,s[i][j]);
                PRINT_OPTIMAL_PARENS(s,s[i][j]+1,j);
```

对上图中，n=6时，调用PRINT_OPTIMAL_PARENS(s,1,6)输出括号化方案 ((A1(A2A3))((A4A5)A6))

下面给出完整的代码，当n=6时的最优解和括号方案。

```
const int INT_MAX=2147483647;
void MATRIX_CHAIN_ORDER(int *p,int Length,int m[][M],int s[][M])
        for(int i=1;i<=n;i++) m[i][i]=0;
        for(int l=2;l<=n;l++)   /* 矩阵链的长度 */
                for(int i=1;i<=n-l+1;i++)
                        int j=i+l-1;          /* 等价于 l=j-i+1 */
                        for(int k=i;k<=j-1;k++)
                                q=m[i][k]+m[k+1][j]+p[i-1]*p[k]*p[j];
void PRINT_OPTIMAL_PARENS(int s[][M],int i,int j)
        if(i == j) cout<<"A"<<i;
                PRINT_OPTIMAL_PARENS(s,i,s[i][j]);
                PRINT_OPTIMAL_PARENS(s,s[i][j]+1,j);
   int p[M]={30,35,15,5,10,20,25};
   MATRIX_CHAIN_ORDER(p,M,m,s);
   cout<<"当n=6时最优解为: \n"<<m[1][6];
   PRINT_OPTIMAL_PARENS(s,1,6);
```

运行结果


C:\Users\nima\Documents\Untitled1.exe
当n=6时最优解为:
15125
括号化方案为:
((A1(A2A3))((A4A5)A6))
Process exited with return value 0
Press any key to continue . . .
http://blog.csdn.net/