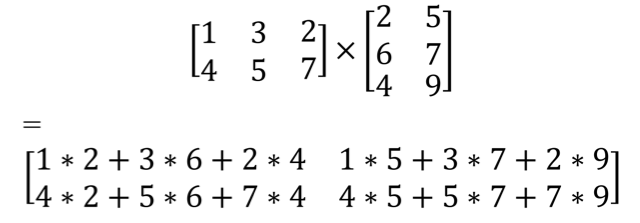
一、计算两个矩阵相乘核心算法



For i=1 to 2(行)

For j=1 to 2 列（矩阵2的列）

For k=1 to 3 (相同的矩阵1的列和矩阵2的行)

c[i][j]+=a[i][k]\*b[k][j]

结论：

设A1是一个P1行r列矩阵，A2是一个r行p2列矩阵,由此推导出两个矩阵相乘A1\*A2所用的乘积数为p1\*r\*p2

推导( A ( (BC) D ) )计算乘积次数：



BC=10\*40\*30=12000 得到矩阵：M=10\*30

MD=10\*30\*5=1500 得到矩阵：N=10\*5

AN=50\*10\*5=2500 得到矩阵50\*5

总计算乘积次数：12000+1500+2500=16000

动态规划k划分例子：

例子：(AB)(CD)：AB=50\*10\*40=20000 得到矩阵M:50\*40

CD=40\*30\*5=6000 得到矩阵N:40\*5

MN=50\*40\*5=10000 得到矩阵N:50\*5

合计：36000

A1\*A2\*A3\*A4\*A5

10\*5 5\*1 1\*10 10\*2 2\*10

P0=10 P1=5 P2=1 P3=10 P4=2 P5=10

1 2 3 4 5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 0 | 50 | 150 |  |  |
| 2 |  | 0 | 50 |  |  |
| 3 |  |  | 0 | 20 |  |
| 4 |  |  |  | 0 | 200 |
| 5 |  |  |  |  | 0 |

1 2 3 4 5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 2 |  |  |
| 2 |  | 0 | 2 | 2 |  |
| 3 |  |  | 0 | 3 |  |
| 4 |  |  |  | 0 | 4 |
| 5 |  |  |  |  | 0 |

(A1\*A2)\*A3

A2\*(A3\*A4)

M[1][3]=M[1][1]+m[2][3]+p0\*p1\*p3=0+50+

M[1][3]=m[1][2]+m[3][3]+p0\*p2\*p3=50+0+

1. i=7 j=6

x7!=y6 max[(x7,y5),(x6,y6)]

1. i=7 y=5

x7=y5

(x6,y4)+1

1. i=6,y=4

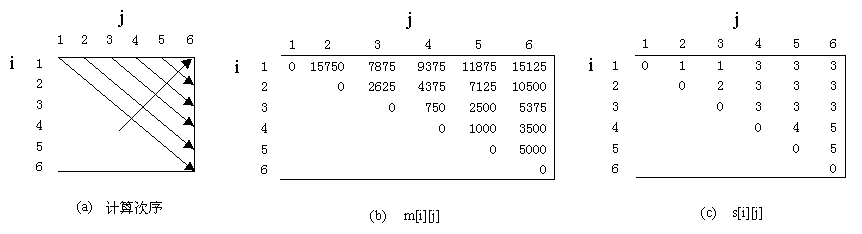
x=A,….B

Y=B,D,C,A….A

c[1,2]=MAX(C[0,2] ,C[1,1])=0

C[1,4]=C[0,3]+1=1

C[7,6]=MAX(C[6,6],C[7,5])=4



p={30,35,15,5,10,20,25}

(A1\*(A2\*A3))\*((A4\*A5)\*A6)

public static void **matrixChain**(int [] p, int [][] m, int [][] s)

{

int n=p.length-1;

**for** (int i = 1; i <= n; i++) m[i][i] = 0; /\*单矩阵乘积，都为0\*、

**for** (int r = 2; r <= n; r++) /\*控制斜列,从对角线第2列开始，到n\*/

**for** (int i = 1; i <= n - r+1; i++) { /\*i，j对应矩阵中每个元素

int j=i+r-1;

m[i][j] =m[i][i]+m[i+1][j]+ p[i-1]\*p[i]\*p[j]; /\*每一个元素求最小值，需要比较k不同位置多个值\*/

s[i][j] = i;

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

int t = m[i][k] + m[k+1][j] + p[i-1]\*p[k]\*p[j];

**if** (t < m[i][j]) {

m[i][j] = t;

s[i][j] = k;}

}

}

}

算法**matrixChain**的主要计算量取决于算法中对r，i和k的3重循环。循环体内的计算量为O(1)，而3重循环的总次数为O(n3)。因此算法的计算时间上界为O(n3)。算法所占用的空间显然为O(n2)。