动态规划复习

Hushrush.2021.04.24

1. 动态规划算法（Dynamic Programming，DP）：

* 思想是将待求解问题分成若干较小的子问题（类似于分治法），先求解子问题，再从这些子问题得到原问题的解。
* 特点是

1. 适合动态规划方法的问题分解得到的子问题往往是不独立的，而分治法的子问题是独立的。
2. 动态规划方法适合求解**最优化**问题。

* **做法是用一张表（备忘录）记录所有子问题的答案，避免重复求解。**
* **一般步骤**

1. 问题建模，**写出目标函数**；
2. 划分子问题；
3. 找出依赖关系，**写出递推方程**；
4. 验证最优子结构；
5. 构造最优解（必要时）。
6. **实例说明**
   1. **矩阵连乘问题**
      1. **题目描述**

给定n个矩阵：A1,A2,...,An，其中Ai与Ai+1是可乘的，i=1，2...，n-1。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。输入数据为矩阵个数和每个矩阵规模，输出结果为计算矩阵连乘积的计算次序和最少数乘次数。

2.1.2 问题解释

由于矩阵乘法满足结合律，故计算矩阵的连乘积可以有许多不同的计算次序。这种计算次序可以用加括号的方式来确定。若一个矩阵连乘积的计算次序完全确定，也就是说该连乘积已完全加括号，则可以依此次序反复调用2个矩阵相乘的标准算法计算出矩阵连乘积。

完全加括号的矩阵连乘积可递归地定义为：

（1）单个矩阵是完全加括号的；

（2）矩阵连乘积A是完全加括号的，则A可表示为2个完全加括号的矩阵连乘积B和C的乘积并加括号，即A=(BC)。

例如，矩阵连乘积A1A2A3A4有5种不同的完全加括号的方式：(A1(A2(A3A4)))，(A1((A2A3)A4))，((A1A2)(A3A4))，((A1(A2A3))A4)，(((A1A2)A3)A4)。每一种完全加括号的方式对应于一个矩阵连乘积的计算次序，这决定着作乘积所需要的计算量。

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)):10\*5\*50+10\*100\*50=52500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。

2.1.3问题分析

1. 最优子结构

简记矩阵连乘积为A[i:j]。设按照最优计算次序，矩阵链会在k处断开，则A[1:n]=((A[1:k])(A[k+1:n]))。即A[1:n]的计算量应该是A[1:k]+A[k+1:n]+两结果矩阵相乘次数。

问题具有**最优子结构。**如果A[1:n]的运算次数最少，则子结构A[1:k]和A[k+1:n]的计算次数也是最少的。

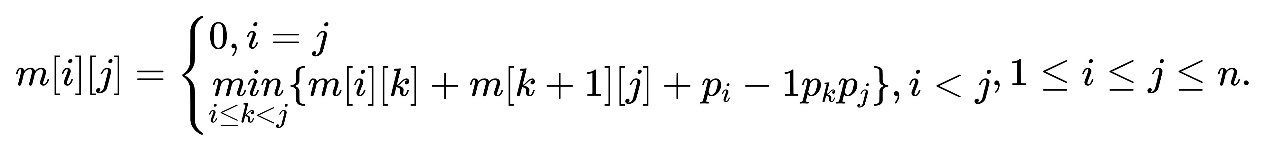
证明：设A[1:n]的运算次数最少，子链为A[1:k],A[k+1:n]。不妨设一个A[1:k]的加括号方式比原来的A[1:k]更少，则用它**代替**原来的次序可以得到更优的A[1:k]解，与A[1:n]的运算次数最少矛盾。 ◼

1. 建立递归关系

简记m[i][j]为A[i:n]的最少数乘次数；的维数为。

1. i==j时 A[i:i]无需计算。。
2. i<j时，设矩阵链在k处断开，则m[i][j]=m[i][k]+m[k+1][j]+p\_(i-1) p\_k p\_j。k的位置待定，但只有j-i种可能。

从而



1. 计算最优值
2. 递归算法（伪代码）

RecurMatrixChain(p, i, j)//返回最小乘法次数

{

if（i < j)

{

m[i][j] = inf;//全局数组。将初值设为无穷，之后优化

s[i][j] = i;//s用于存储k的位置

for k = i to j - 1 do

{

q = RecurMatrixChain(p, i, k) + RecurMatrixChain(p, k + 1, j);//对每一个k值计算最小乘法次数

if q < m[i][j]//更新k值和最小值

{

m[i][j] = q;

s[i][j] = k;

}

}

return m[i][j];

}

if (i = j)

return 0;

}

递归算法效率不高，主要是重复计算子问题，还是分治的思想。

1. 迭代实现非递归（java）

public static void matrixChain(int[]p, int[][]m, int[][]s)//p为矩阵行列参数，m为得到计算最优值，s存储k

{

int n = n.length - 1;//n为矩阵个数

for (int i = 1; i <= n; i++)//设数组从1开始

m[i][i] = 0;//不用计算

for (int r = 2; r <= n; r++)//将k的所有可能取值算一遍,r为步长

{

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

{

int j = i + r - 1;

m[i][j] = m[i + 1][j] + p[i - 1] \* p[i] \* p[j];//这一句很难。先算（取）i+1到j的，再用i与其相乘

s[i][j] = i;//上面每一步都以i为分界

for (int k = i + 1; k < j; k++)//k=i 已经算过了,k是不以i为分界的更新值

{

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;//这里m[i][j]是备忘录

}

}

}

}

}

1. 构造最优解

得到最小计算次数后若有需要可以构造最优解。根据s存储的信息可以找到全部计算次序。

public static void traceback(int[][]s, int i, int j)

{

if (i == j)return;//找到连接处

traceback(s, i, s[i][j]);

traceback(s, s[i][j] + 1, j);

System.out.println("Multiply A" + i + "." + s[i][j] + "and A" + (s[i][j] + 1) + "." + j);

}