算法分析与设计 第六讲

动态规划 基本概念及实例

主要内容

- 动态规划的基本概念
- 动态规划的基本步骤
- 动态规划问题求解实例

动态规划的求解对象

- 最优化问题
  - 工程问题中设计参数的选择
  - 有限资源的合理分配
  - 车间作业调度
  - 交通系统的规划
  - 不胜枚举…………..

动态规划（dynamic programming）

- 分治法求解回顾
  - 子问题相互独立，不包含公共子问题
- 动态规划
  - 与分治法类似，也是将问题分解为规模逐渐减小的同类型的子问题
  - 与分治法不同，分解所得的子问题很多都是重复的

动态规划求解实例-矩阵连乘问题

- 矩阵连乘问题（矩阵链乘法）
- 一般描述：
  - 对于给定的n个矩阵，M_1， M_2 ，…， M_n，其中矩阵M_i 和M_j 是可乘的，要求确定计算矩阵连乘积（ $M_1M_2 \ldots M_n$ ）的计算次序，使得按照该次数计算矩阵连乘积时需要的乘法次数最少

矩阵连乘问题

- 例，设有矩阵M_1,M_2,M_3,M_4
- 其维数分别是：$10\times20,\ 20\times50,50\times1\ ,1\times100$
- 现要求出这4个矩阵相乘的结果
- 计算次序可以通过加括号的方式确定
- 当n很大时，n个矩阵连乘的加括号的方法数是指数量级的，逐一检查不现实

矩阵连乘问题

- 目标：求出矩阵连乘$M_iM_{i+1}---M_{j-1}M_j$（$i<j$）所需的最少乘法次数
- 共有j-i+1个矩阵，可称这个矩阵连乘的规模是j-i+1
- 按照做最后一次乘法的位置进行划分，矩阵连乘一共可分为j-i种情况
- 若已知任一个规模不超过j-i的矩阵连乘所需的最少乘法次数，则可计算出$M_iM_{i+1}---M_{j-1}M_j$所需的最少乘法次数

矩阵连乘问题

- j-i种划分情况表示为通式：
  - $(M_i\text{---}M_k)\ (M_{k+1}\text{---}M_j)$（$i \le k < j$）
- 记第t个矩阵M_t的列数为r_t，并令r_0为矩阵M_1的行数
- $M_i\text{---}M_k$连乘所得是$r_{i-1} \times r_k$维矩阵
- $M_{k+1}\text{---}M_j$连乘所得是$r_k \times r_j$维矩阵
- 这两个矩阵相乘需要做$r_{i-1} \times r_k \times r_j$次乘法

矩阵连乘问题

- 由于已知$(M_i \cdots M_k)$和$(M_{k+1} \cdots M_j)$所需的最少乘法次数，记为m_{ik}和$m_{k+1,j}$
- $(M_i \ldots M_k)(M_{k+1} \ldots M_j)$的矩阵连乘所需的最少乘法次数为：$m_{ik}+m_{k+1,j}+r_{i-1}\times r_k\times r_j$
- 对满足$i \le k<j$ 的共$j-i$种情况逐一进行比较，可得：
- $m_{ij}=\min_{(i\le k<j)}\{m_{ik}+m_{k+1,j}+r_{i-1}\times r_k\times r_j\}$

矩阵连乘问题

- 对于$m_{ij}=\min_{(i\le k<j)}\{m_{ik}+m_{k+1,j}+r_{i-1}\times r_k\times r_j\}$
- $m_{ii}=0$(相当于单个矩阵的情况)
- 首先求出计算M_1M_2，M_2M_3，…，$M_{n-1}M_n$所需的最少乘法次数$m_{i,i+1}(i=1,2,\cdots,n-1)$
- 再基于以上结果，根据m_{ij}的求解公式计算$M_1M_2M_3$，$M_2M_3M_4$，…，$M_{n-2}M_{n-1}M_n$所需的最少乘法次数$m_{i,i+2}(i=1,2,\cdots,n-2)$
- 直至$m_{i,i+3}(i=1,2,\cdots,n-3)$，$m_{i,i+4}(i=1,2,\cdots,n-4)$，$m_{1,n}$

矩阵连乘问题

- 已知M_1,M_2,M_3,M_4
- 其维数分别是：$10\times20,\ 20\times50, 50\times1\ , 1\times100$
- 求m_{13}
- 求解$m_{1,n}$的算法

for i=1 to n do $m_{ii}\leftarrow 0$;

for l=1 to n-1 do

for i=1 to n-l do

$\{j\leftarrow i+l;\quad m_{ij}\leftarrow \min_{(i\le k<j)}\{m_{ik}+m_{k+1,j}+r_{i-1}\times r_k\times r_j\}\}$

矩阵连乘问题

- 该方法将时间从brute-force法的指数量级降低到了$\Theta(n^3)$

矩阵连乘问题

- 动态规划相关的重要概念
  - 子问题的高度重复性
  - 最优子结构性质
    - 问题的最优解中包含着其每一个子问题的最优解

矩阵连乘问题

for i=1 to n do $m_{ii} \leftarrow 0$;

for i=1 to n-1 do $d_{i,i+1} \leftarrow i$;

for l=1 to n-1 do

for i=1 to n-l do

{$j \leftarrow i+l$;

$m_{ij} \leftarrow \min_{(i \le k < j)} \{m_{ik}+m_{k+1,j}+r_{i-1} \times r_k \times r_j\}$

$d_{ij} \leftarrow k'$}

矩阵连乘问题

- 例$M_1M_2M_3M_4M_5M_6$，其维数分别是30×35，35×15，15×5，5×10，10×20和20×25
- 二维数组(d_{ij})

i\j	1	2	3	4	5	6
1		1	1	3	3	3
2			2	3	3	3
3				3	3	3
4					4	5
5						5

适合用动态规划方法求解的问题

- 若一个问题可以分解为若干个高度重复的子问题，且问题也具有最优子结构性质，就可以用动态规划法求解
- 具体方式：可以递推的方式逐层计算最优值并记录必要的信息，最后根据记录的信息构造最优解

动态规划方法总体思想

- 保存已解决的子问题的答案，在需要时使用，从而避免大量重复计算

Those who cannot remember the past are doomed to repeat it.

-----George Santayana,
The life of Reason,
Book I: Introduction and
Reason in Common
Sense (1905)

动态规划方法解题步骤

- 找出最优解的性质，并刻画其结构特征
- 递归地定义最优值（写出动态规划方程）
- 以自底向上的递推方式计算出最优值
- 根据计算最优值时得到的信息，以递归方法构造一个最优解

最长公共子序列问题（LCS）

- 子序列的概念
- 设X=< x_1, x_2,…, x_m>
- 若有$1 \le i_1 < i_2 < \cdots < i_k \le m$
- 使得Z=< z_1, z_2,…, z_k> = < x_{i1}, x_{i2},…, x_{ik}>则称Z是X的子序列，记为Z<X

最长公共子序列问题（LCS）

- 公共子序列的概念
- 设X，Y是两个序列，且有Z<X和Z<Y
- 则称Z是X和Y 的公共子序列

最长公共子序列问题（LCS）

- 最长公共子序列的概念
- 若Z<X，Z<Y，且不存在比Z更长的X和Y的公共子序列
- 则称Z是X和Y 的最长公共子序列，记为$Z\in LCS(X,Y)$
- 请注意，最长公共子序列往往不止一个

LCS的求解方法

- Brute-force法
- 列出X的所有长度不超过n（即│Y│）的子序列，从长到短逐一进行检查，看其是否为Y的子序列，直到找到第一个最长公共子序列
- 由于X共有2^m个子序列，故此方法对较大m没有实用价值

最长公共子序列问题（LCS）

- 记X_i=﹤x_1，…，x_i﹥即X序列的前i个字符 (1≤i≤m)
- Y_j=﹤y_1，…，y_j﹥即Y序列的前j个字符 (1≤j≤n)
- 假定Z=﹤z_1，…，z_k﹥∈LCS(X,Y)

最长公共子序列问题（LCS）

- 若$x_m=y_n$ (最后一个字符相同)，则有$z_k = x_m = y_n$ 且$Z_{k-1}\in$LCS(X_{m-1}，Y_{n-1})
- 若$x_m\neq y_n$且$z_k\neq x_m$，则有$Z\in$LCS(X_{m-1}，Y)
- 若$x_m\neq y_n$且$z_k\neq y_n$ 则有$Z\in$LCS(X，Y_{n-1})
- 求LCS(X_{m-1}，Y) 与LCS(X，Y_{n-1})的长度，不是相互独立的，具有重叠性
- 两个序列的LCS中包含了两个序列的前缀的LCS，具有最优子结构性质

最长公共子序列问题（LCS）

- 引入一个二维数组C，用C[i,j]记录X_i与Y_j的LCS的长度

- C[i,j]= $\left\{ \begin{array}{ll} 0 & \text{若} i=0 \text{或} j=0 \\ C[i-1, j-1]+1 & \text{若} i, j>0 \text{且} x_i = y_j \\ \max\{C[i-1, j], C[i, j-1]\} & \text{若} i, j>0 \text{且} x_i \neq y_j \end{array} \right.$

最长公共子序列问题（LCS）

- 为了构造出LCS，引入一个m×n的二维数组b，b[i,j]记录C[i,j]是通过哪一个子问题的值求得的，以决定搜索的方向
- 若C[i-1,j]≥C[i,j-1]，则b[i,j]中记入“↑”
- 若C[i-1,j]<C[i,j-1]，则b[i,j]中记入“←”

最长公共子序列问题（LCS）

```
LCS_L(X,Y,m,n,C)
for i=0 to m do C[i,0]←0
for j=1 to n do C[0,j]←0
for i=1 to m do {
   for j=1 to n do{
      if x[i]=y[j] then {C[i,j]←C[i-1,j-1]+1；
                     b[i,j]← “↖” ；}
}else if C[i-1,j]≥C[i,j-1] then {C[i,j]←C[i-1,j]；
                          b[i,j]← “↑” ；}
  }else{C[i,j]←C[i,j-1]；
      b[i,j]← “←” ；}
```

最长公共子序列问题（LCS）

- 输出一个LCS(X,Y)的递归算法

```
LCS_Output(b,X,i,j)
If i=0 or j=0 then return;
If b[i,j]= “↖” then        /*X[i]=Y[j]*/
  {LCS_Output(b,X,i-1,j-1)；
       输出X[i]；}
else if b[i,j]= “↑” then  /*C[i-1,j]≥C[i,j-1]*/
   LCS_Output(b,X,i-1,j)
else if b[i,j]= “←”  then  /*C[i-1,j]<C[i,j-1]*/
  LCS_Output(b,X,i,j-1)
```

最长公共子序列问题（LCS）

- 请注意，以上算法不能搜索到所有的LCS
- 没区分C[i-1,j]＞C[i,j-1] 和C[i-1,j]＝C[i,j-1]
- 当满足X[i]与Y[j]不等，且C[i-1,j]＝C[i,j-1]时，要执行b[i,j]←“←↑”，即记录两个搜索方向，才能够据此找出所有的LCS