矩阵乘法的基本应用

江苏省常州高级中学 徐毅

一、矩阵乘法用于变换

根据矩阵的定义，它可以很直接地处理一些大量重复操作的变换问题。

【例 1】Training little cats[1]

题目大意：

一个人将对 $N(1 \le N \le 100)$ 只小猫重复 $M(1 \le M \le 10^9)$ 次相同的流程：依次进行 $K(1 \le K \le 100)$ 个动作，每个动作都是以下三者中的一种：给第 i 只小猫一个花生；让第 i 只小猫把它的花生吃完；让第 i 和第 j 只小猫交换它们手上的花生。求全部动作结束后每只小猫手上的花生数。

算法分析：

大量重复操作明显提示我们使用矩阵乘法。

我们知道，一个 3×3 的矩阵乘上单位矩阵 $\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}$ 就可以得到它自己。同样，$\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}\begin{bmatrix}1\\0\\0\end{bmatrix}=\begin{bmatrix}1\\0\\0\end{bmatrix}$，通过定义可以很直观地理解为，每只猫都取它自己的。那么，对于这道题，我们以 $N\times N$ 的单位矩阵 A 为基础进行思考。很容易想到，交换花生的操作，可以表示为交换矩阵的第 i 和第 j 行；吃完花生的操作，可以表示为将第 i 行全部赋为 0。对于增加花生即增加常数的操作，似乎难以在目前的矩阵上直接实现，那么我们的一个常用技巧就是添加常数项，即给矩阵增添第 0 行第 0 列，可以理解为增加了永远拥有 1 个花生的第 0 只小猫，这样只要对 $A_{i,0}$ 加 1 即完成了增加花生的操作。由此，进行一系列修改就可以得到所需的一次流程的 $(N+1)\times(N+1)$ 的操作矩阵。用快速幂或二进制拆分的方法求得 $A^M\begin{bmatrix}1\\0\\\cdots\\0\end{bmatrix}$ 即得答案，时间复杂度 $O(N^3\log M)$。

[1] http://poj.org/problem?id=3735

似乎做到这里已经完成了，但由于这道题的时限较紧，我们必须进行一些优化。可以注意到，在我们的操作矩阵中，最多出现$(N+K)$个非0数，也就是说，这是一个特殊的稀疏矩阵。因此，在我们常用的以$O(N^3)$复杂度完成一次矩阵乘法的过程中，出现了大量的$+0$冗余操作。考虑对矩阵乘法的代码进行细微的修改，将i,j,k三重循环的顺序调换为i,k,j，在前两重循环后只有发现$A_{i,k}\neq 0$时才进入第三重循环，这样就能大大缩短时间。

二、矩阵乘法用于递推

众所周知，矩阵乘法最常见的应用是优化递推。

【例 2】Sum of products[2]

题目大意：

将正整数$N(1\le N\le 10^9)$有序拆分为两个以上的正整数，如$3=1+1+1$或$1+2$或$2+1$，求将拆分中的加号全部换为乘号后各种拆分方案的值的和，如$ans(3)=1\times1\times1+1\times2+2\times1=5$。

算法分析：

对于这种拆分类问题，若不考虑数据范围，我们很容易想到利用递推求解。

用$f(i)$表示对正整数i有序拆分为一个以上正整数得到的答案，则$ans(i)=f(i)-i$。令$f(0)=1$，枚举拆分的最后一个数，易得$f(i)=\sum_{k=1}^{i}kf(i-k)$，时间复杂度为$O(N^2)$。

我们观察到，求和公式中每一项的系数是一个关于k的一次函数（在这里函数为$y=k$），于是考虑作差，得$f(i)-f(i-1)=\sum_{k=0}^{i-1}f(i-1-k)$，令$g(i)=\sum_{k=0}^{i}f(i-k)$，$g(0)=1$，则有$f(i)=f(i-1)+g(i-1),i>1$，$g(i)=g(i-1)+f(i)$。这样利用部分和优化的递推时间复杂度为$O(N)$。

分析到这里，我们的递推式已经相对成熟了，进一步整理得到$g(i)=f(i-1)+2g(i-1)$，使得它完全适合使用矩阵乘法优化。那么易得$\begin{bmatrix}f(i)\\g(i)\end{bmatrix}=\begin{bmatrix}1&1\\1&2\end{bmatrix}\begin{bmatrix}f(i-1)\\g(i-1)\end{bmatrix}$，即

2 http://www.spoj.com/problems/SUMMUL/

$$\begin{bmatrix} f(N) \\ g(N) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}^{N-1} \begin{bmatrix} f(1)=1 \\ g(1)=2 \end{bmatrix}$$，时间复杂度为 $O(\log N)$。

三、矩阵乘法用于动态规划

虽然矩阵乘法的定义是“相乘”“求和”，但在实际应用中，可能需要改变“乘法”的含义，只要运算满足结合律即可。当矩阵乘法用于动态规划时，往往就需要用到这一点。

【例 3】Gremlins[3]

题目大意：

$T(1 \le T \le 10^{15})$ 年前 $N(1 \le N \le 100)$ 种精灵各一个出生了，第 $i(1 \le i \le N)$ 种精灵会在出生 $Y_i(1 \le Y_i \le 1000)$ 年后繁衍 $K_i(1 \le K_i \le 1000)$ 个后代，给出繁衍的每个后代的种类和孵化时间，求出现有精灵的最长家谱长度（最多上溯多少代）。

算法分析：

看到这题的初步思路应该是动态规划。

直接的想法是 $f(i, j)$ 表示过了 i 年第 j 种精灵最多繁衍的代数，但这样不仅状态数巨大，且由于与 $f(i)$ 直接相关的并不一定是 $f(i-1)$，难以使用矩阵乘法优化。

动态规划的一个常用的优化状态的技巧是下标和值的转换，在这里，即改为用 $f(i, j)$ 表示繁衍到第 i 代第 j 种精灵需要的最少年数，这样就实现了状态的连续性。显然，$f(i, j) = \min\{f(i-1, k) + Y_k + R_{k,j}\}, 1 \le i, j, k \le N$，$R_{k,j}$ 表示 k 繁衍的 j 所需的孵化时间。那么，令 $G_{i,j} = Y_i + R_{i,j}$，将运算规则进行如下改变：$C_{i,j} = \min\{A_{i,k} + B_{k,j}\}, 1 \le i, j, k \le N$，易证其满足结合律，则 $f(i) = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 \end{bmatrix} G^i$。

剩下的问题就是求答案了。在这里我使用二进制拆分的思想，首先预处理 $\min\left(G^{2^k}\right)$ 不超过 T 时所有矩阵 G^{2^k}，然后从高次到低次依次尝试，若乘 G^{2^k} 仍满足矩阵最小值不超过 T 则乘，并累加 2^k 进入答案，整体时间复杂度 $O(N^3 \log T)$。

【例 4】幸福路径[4]

题目大意：

给出一个 $N(1 \le N \le 100)$ 个带权顶点 $M(1 \le M \le 1000)$ 条边（可有自环但无重边）的有向图，从 $v_0(1 \le v_0 \le N)$ 出发，每经过一条边，所有点的权值 $w_i(0 \le w_i \le 100, 1 \le i \le N)$ 都

[3] http://www.spoj.com/problems/GREMLINS/
[4] http://www.lydsy.com/JudgeOnline/problem.php?id=2306

会乘 $\rho\left(0<\rho\le 1-10^{-6}\right)$。将一条路径从起点开始依次经过的所有点的权值和记为 H，求路径长度不限时的 $H_{\max}$。

算法分析：

这道题看似无法下手，因为路径长度不限。我们必须化“无限”为“有限”，清醒地认识到，由于计算机精度的限制，当路径长度足够大时，w_i 已无限趋向于 0，也就是说，路径长度在某个值以后继续增大意义已经不大了，这个值我们可以人为推断，在这里我取路径长度最大值 $K=2^{30}$。

因此，我们可以先尝试解决一个相对简单的问题：求一个有向图中从某个点出发经过 K 条边的最长路。我们将邻接矩阵作为矩阵乘法的初始矩阵（不存在的边设为 $-\infty$），将运算规则进行如下改变：$C_{i,j}=\max\left\{A_{i,k}+B_{k,j}\right\}, 1\le i,j,k\le N$，易证其满足结合律，则 $G_{i,j}^{d}$ 表示从 i 到 j 经过 d 条边的最长路，因此易在 $O\left(N^3\log K\right)$ 时间复杂度内求解。

进一步考虑，假如要求经过的边是任意 $d\left(0\le d\le K\right)$ 条，依然是可以解决的，我们只需要设 $G_{i,i}=0$。

接下来就是处理 ρ 的问题了。由于取 $K=2^{30}$，假如没有 ρ，我们直接从原始邻接矩阵出发，自乘 30 次即可。由于有 ρ，每次并不是直接自乘。设 G_i 表示经过 i 条边后的矩阵，$G^d=\prod_{i=1}^{d}G_i$，仔细分析可知 $G^{2r}=G^r\left(\rho^r G^r\right)$，$\rho^r G^r$ 表示将 G^r 中的每一个元素都乘 ρ^r 得到的矩阵，这样就完成了对 ρ 的处理，整体时间复杂度为 $O\left(N^3\right)$。