矩阵乘法在信息学中的应用

浙江省杭州二中 俞华程

摘 要

矩阵在数学中应用很广，而在信息学中同样有很广的应用。本文将阐述矩阵的乘法这一运算在三个方面的一些应用。

关键字：快速幂 矩阵乘法 优化动态规划 图的邻接矩阵 折半递归

目 录

正文

1 预备知识

1.1 矩阵

矩阵可以看成一个$n \times m$的数表，其中n, m都是正整数。例如：

$$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \end{bmatrix} \qquad\qquad \begin{bmatrix} 1 \end{bmatrix}$$

一般可以用$\mathbf{A}\left[i, j\right]$表示第$i$行第$j$个数。

1.2 矩阵乘法

定义矩阵$\mathbf{A}$,$\mathbf{B}$。$\mathbf{A}$和$\mathbf{B}$可以作乘法操作当且仅当$\mathbf{A}$的大小是$a \times b$，$\mathbf{B}$的大小是$b \times c$，其中a, b, c皆为正整数。设矩阵$\mathbf{C} = \mathbf{AB}$，则$\mathbf{C}$的大小是$a \times c$，且有

$$\mathbf{C}\left[i, j\right] = \sum_{k=1}^{b} \mathbf{A}\left[i, k\right] \mathbf{B}\left[k, j\right] \quad (1 \le i \le a, 1 \le j \le c) \tag{1}$$

而且矩阵乘法满足结合律，即$\left(\mathbf{AB}\right)\mathbf{C} = \mathbf{A}\left(\mathbf{BC}\right)$。设$\mathbf{A}, \mathbf{B}, \mathbf{C}$的大小分别为$a \times b, b \times c, c \times d$，证明如下：

$$\begin{aligned} &\left(\left(\mathbf{AB}\right)\mathbf{C}\right)\left[i, j\right] \\ &\qquad = \sum_{l=1}^{c} \left(\sum_{k=1}^{b} \mathbf{A}\left[i, k\right] \mathbf{B}\left[k, l\right] \right) \mathbf{C}\left[l, j\right] \\ &\qquad = \sum_{k=1}^{b} \sum_{l=1}^{c} \mathbf{A}\left[i, k\right] \mathbf{B}\left[k, l\right] \mathbf{C}\left[l, j\right] \\ &\qquad = \sum_{k=1}^{b} \mathbf{A}\left[i, k\right] \left(\sum_{l=1}^{c} \mathbf{B}\left[k, l\right] \mathbf{C}\left[l, j\right] \right) \\ &\qquad = \left(\mathbf{A}\left(\mathbf{BC}\right)\right)\left[i, j\right] \end{aligned} \tag{2}$$

而$\left(\mathbf{AB}\right)\mathbf{C}$和$\mathbf{A}\left(\mathbf{BC}\right)$的大小都是$a \times d$，得证。

直接按照定义进行矩阵乘法时间复杂度是$O\left(abc\right)$，其中a, b, c分别表示两个矩阵大小是$a \times b$以及$b \times c$。将两个$N \times N$的矩阵相乘，朴素算法的时间复杂度为$O\left(N^3\right)$。其他有一些算法有更低的复杂度，例如 Strassen 算法时间复杂度约为$O\left(N^{2.81}\right)$。而现有的算法中理论复杂度最低的是 Coppersmith—Winograd 算法，时间复杂度约为$O\left(N^{2.36}\right)$。

1.3 快速幂

我们平时在计算形如a^t这样一个数的非负整数次幂的时候，通常会先递归计

算出$a^{\lfloor t/2 \rfloor}$，然后可以利用这个值$O(1)$的时间里算出a^t：

$$a^t = \begin{cases} \left(a^{\lfloor \frac{t}{2} \rfloor}\right)^2 & \text{a是偶数} \\ \left(a^{\lfloor \frac{t}{2} \rfloor}\right)^2 \cdot a & \text{a是奇数} \end{cases} \tag{3}$$

可以得出快速幂算法的复杂度是$O(\log t)$。

定义$\underbrace{\mathbf{M} \cdot \mathbf{M} \cdot \cdots \cdot \mathbf{M}}_{t} = \mathbf{M}^t$，由于矩阵乘法满足结合律，计算$\mathbf{M}^t$同样可以使用快速幂。朴素算法的时间复杂度是$O\left(N^3 \log t\right)$，其中$N$表示$\mathbf{M}$的大小是$N \times N$，因为要计算$\mathbf{M} \cdot \mathbf{M}$，$\mathbf{M}$的大小不可能是$a \times b, a \neq b$。本文中大部分算法都需要使用到矩阵的快速幂。

2 优化动态规划

2.1 生成树计数 NOI'07

2.1.1 题目大意

一张无向图$\mathbf{G}$有$N\left(N \leq 10^{15}\right)$个点，编号为$i$的点和$j$的点之间有边，当且仅当$|i-j| \leq k\,(2 \leq k \leq 5)$。给定$N, k$，求$\mathbf{G}$的生成树的个数。

2.1.2 分析

题目中提供的求行列式的值的方法并没有利用到图的特殊性。一张图的一个生成树就是这张图的一个无环子图，并且使所有点都连通。考虑到k的范围比较小，每个点都只和附近的很少的点连边。于是可以使用状态压缩动态规划，由于要判断连通性和环，需要存下当前点之前k个点的连通性以及当前点是第几个点。转移的时候，枚举当前点和前面k个点的边要选中哪些，并且判断是否出现环以及是否可能产生不连通的情况，即如果离当前最远的点不和前面其他点连通，就必须和当前点连边。设l_i为i个点的连通性的情况数。时间复杂度从原来的$O\left(N^3\right)$降到了$O\left({l_k}^2 N\right)$。

尽管这个算法已经充分利用了题目当中图的特殊性，但是对于题目的规模来说，这个复杂度还是不能够接受的，我们只能优化动态规划或者另辟蹊径了。通过观察我们发现，当前面k个点的连通性相同的时候，无论当前是哪个点，都是转移到编号比它大一的点，并且转移都是一样的。既然转移是一样的，能否使用一个自动化的工具来快速进行转移呢？能！那就是矩阵。

设$f\,[i]\,[j]$表示现在是第i个点，前k个点的连通性状态是j，达到这种状态有多少种方法，若

$$f\,[i]\,[j] = \sum_{t=1}^{l_k} a_{j,t} \cdot f\,[i-1]\,[t] \tag{4}$$

由于每一个$f\,[i]\,[j]$的值只依赖于所有的$f\,[i-1]\,[t]$。我们把所有的$f\,[i-1]\,[t]$放在一个$1 \times l_k$的矩阵$\mathbf{M}_{i-1}$里。定义矩阵$\mathbf{D}$：

$$\mathbf{D} = \begin{bmatrix} a_{1,1} & a_{2,1} & \dots & a_{l_k,1} \\ a_{1,2} & a_{2,2} & \dots & a_{l_k,2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1,l_k} & a_{2,l_k} & \dots & a_{l_k,l_k} \end{bmatrix}$$

则$\mathbf{M_i} = \mathbf{M_{i-1}D}$，因为根据矩阵乘法的定义，有：

$$\mathbf{M_i}\,[1,j] = \sum_{t=1}^{l_k} \mathbf{M_{i-1}}\,[1,t] \cdot \mathbf{D}\,[t,j]$$

即上面的(4)式。先搜索算出$a_{j,t}$以及$\mathbf{M_k}$（我还尚未研究出任何只需直接计算$\mathbf{M_0}$的优美方法），然后使用矩阵乘法的快速幂计算，可以把时间复杂度优化到$O\left({l_k}^3 \log N\right)$。通过搜索发现$l_k = 52$，这个复杂度完全可以接受。

2.2 文本生成器 SPOJ1676[1]

2.2.1 题目大意

给定$N\,(1 \le N \le 10)$个长度不超过6的单词，求由大写字母组成长度为L的包含至少一个给定单词的字符串有多少种，答案 mod 10007，$\left(1 \le L \le 10^6\right)$。

2.2.2 分析

考虑到至少包含一个单词这个条件比较难处理，我们可以先求出总的单词数即26^L，问题就转化为求不包含任何一个给定单词的字符串有多少个。以下文中的合法均指不包含任何一个给定单词。

根据题目中的条件单词长度不超过6，可以得到：若前$i-1$个字母组成的字符串不包含任何单词，那么要判断前i个字母组成的字符串是否满足要求，只需考虑第$i-5$个字母到第i即可。于是，可以得到一个简单的动态规划算法，$f\,[i]\,[a_1]\,[a_2]\,[a_3]\,[a_4]\,[a_5]$表示前$i$个字母已经满足条件，第$(i-4)\sim i$个字符分别为$a_1\sim a_5$，在这个情况下继续生成字符串，有多少合法的方案。可以得到转移方程：

$$f\,[i]\,[a_1]\,[a_2]\,[a_3]\,[a_4]\,[a_5] = \sum_{\substack{'A'\le c\le 'Z'\\ \text{legal}(a_1a_2a_3a_4a_5c)}} f\,[i+1]\,[a_2]\,[a_3]\,[a_4]\,[a_5]\,[c] \tag{5}$$

其中legal(s)表示字符串s的所有后缀均不是题目中给定的单词。则此算法状态数高达$O\left(M^5L\right)$。即使预处理了legal函数，时间复杂度还是有$O\left(M^6L\right)$（M表示字母表的大小），这个复杂度根本无法接受。

分析上一个算法的效率到底低在哪里。在$\mathrm{O}\left(M^6L\right)$当中转移占的比例很少，因此我们需要从优化状态数入手。通过观察，我们发现有许多的重复状态，比如没有单词是以$Aa_2a_3a_4a_5$开头的，也没有单词是以$Ba_2a_3a_4a_5$开头，那么

$$\begin{aligned} &f\,[i]\,['A']\,[a_2]\,[a_3]\,[a_4]\,[a_5]\\ &\qquad = \sum_{'A'\le c\le 'Z'} f\,[i+1]\,[a_2]\,[a_3]\,[a_4]\,[a_5]\,[c]\\ &\qquad = f\,[i]\,['B']\,[a_2]\,[a_3]\,[a_4]\,[a_5] \end{aligned} \tag{6}$$

这两个状态转移以后的式子完全相同，前面的算法却把它们都计算了一次，并且这种情况很普遍，我们计算了十分多的重复状态。因此，需要换一种描述状态的方式来缩减重复无用状态，不应该专注于前面具体是哪些字母，而应该专注于当前这个情况最有可能会生成哪些单词。根据这个想法，我们可以把状态改为$f\,[i]\,[j]\,[k]$，表示当前已经填了i个字母，最后k个字母恰好是单词j的前缀并且最后l个字母不是任何单词的前缀$l>k$，在这种情况下继续往下填能有多少种合法情况。转移和前一个算法很像，枚举下一个字母，判断能否匹配第j个单词的下一个字母。还需要考虑最后k个字母可能不仅仅是单词j的前缀，若下一个字母如果不匹配单词j，还可能匹配别的单词。如果加上下一个字母后无法匹配任何一个单词，那么需要找一个最长的后缀是某个单词的前缀。综合上面

[1] http://www.spoj.pl/problems/GEN/

的说法，转移可以归结为添上一个字母以后，找到最长的一个后缀是某个单词的前缀，如果有多个则找到第一个单词。

$$f\left[i\right]\left[j\right]\left[k\right]=\sum_{'A'\leq c\leq 'Z'}f\left[i+1\right]\left[\text{maxSuffix}\left(j,k,c\right)\left[0\right]\right]\left[\text{maxSuffix}\left(j,k,c\right)\left[1\right]\right] \quad (7)$$

其中maxSuffix$(j,k,c)\left[0\right]$,maxSuffix$(j,k,c)\left[1\right]$分别表示单词j的长度为k的前缀加上字母c后，最长的是某个单词前缀的后缀是哪个单词的，有多长。状态个数为$O\left(L\sum len_i\right)$，$len_i$表示第$i$个单词的长度。转移复杂度为$O\left(k^2NM\right)$，因此时间复杂度为$O\left(LMN\sum {len_i}^3\right)$。而转移中找最长的后缀是哪个单词的前缀可以进行预处理，优化后复杂度降到$O\left(MN\sum {len_i}^3+LM\sum len_i\right)$。在本题中，最坏能够达到$1.5\times 10^9$，即使平均情况也有$2.5\times 10^8$，加上需要大量的取模运算，还是无法在规定时间内出解。

再次分析效率低的地方，预处理的复杂度相比后面计算的复杂度而言几乎可以忽略了。还是和刚才一样状态数太多，状态最多可能有6×10^7。注意到和上一题一样，所有的$f\left[i\right]$都是从$f\left[i+1\right]$转移过来，并且对于不同的i转移方式完全一样。我们对状态里的后两维进行重新标号压缩为一维（状态数没有减少），将转移方程变为：

$$g\left[i\right]\left[j\right]=\sum_{k=1}^{S}a_{j,k}\cdot g\left[i+1\right]\left[k\right] \quad (8)$$

$a_{j,k}$为预处理出来的转移系数，S为合法状态数。$g\left[i\right]\left[j\right]$为重标号以后的状态。构造矩阵$\mathbf{D}$：

$$\mathbf{D}=\begin{bmatrix} a_{1,1} & a_{2,1} & \dots & a_{S,1} \\ a_{1,2} & a_{2,2} & \dots & a_{S,2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1,S} & a_{2,S} & \dots & a_{S,S} \end{bmatrix}$$

初始矩阵$\mathbf{I}$为$1\times S$大小的，每个元素值均为1，表示$g\left[L\right]\left[j\right]$。使用矩阵乘法快速幂运算计算出$\mathbf{ID}^L$即$g\left[0\right]\left[j\right]$后，答案就是$g\left[0\right]\left[0\right]$（定义0号状态为空串）。时间复杂度为$O\left(MN\sum {len_i}^3+S^3\log L\right)$，最坏情况只有不到$5\times 10^6$，稍微少用一点取模运算，就完全可以在时限内出解。

2.3 小结

通过前两个例子，我们得出，如果一个动态规划方程能够用矩阵乘法来进行优化，那么必须满足：

1. 状态必须是一维或者两维，如果状态本身有超过两维，可以通过把多维状态压缩到一维的方法降到两维（此时状态数并没有变化）。
2. 每一个状态$f\left[i\right]\left[j\right]$必须满足只和$f\left[i-1\right]\left[k\right]$有关，并且只能是线性关系。
3. 满足前两条的情况下，可以使用矩阵乘法，但是可能并不能达到优化复杂度的目的。如果对于不同的i，转移矩阵没有规律，那么是无法使用快速幂来加速矩阵乘法的。通常情况下，都有转移矩阵都相同或者至少是循环出现。

4. 假设第一维的大小是N，第二维的大小是M，则矩阵乘法的时间复杂度是$O\left(M^3 \log N\right)$，而直接转移的复杂度至多是$O\left(M^2 N\right)$，有时候甚至更小。通常只有M很小而N相当大的时候才会使用矩阵乘法，否则得不偿失。

3 图邻接矩阵上的乘法

图的邻接矩阵可以唯一地表示一张图，并且有很多神奇的性质。接下来我们将研究邻接矩阵上的矩阵乘法。

首先，我们来看一下最简单的情况，一张N个点的无向无权图。如果点a和点b连边，那么邻接矩阵$\mathbf{G}[a,b]=\mathbf{G}[b,a]=1$，否则都等于0。考虑邻接矩阵自乘，即$\mathbf{G}^2$：

$$\mathbf{G}^2[a,b]=\sum_{i=1}^{N}\mathbf{G}[a,i]\,\mathbf{G}[i,b] \tag{9}$$

上式中，$\mathbf{G}[a,i],\mathbf{G}[i,b]$的值或者等于0或者等于1。易知当且仅当$\mathbf{G}[a,i]$, $\mathbf{G}[i,b]$都是1的时候值为1，即如果从点a到点i、点i到b都有边，那么值为1。而$\mathbf{G}^2[a,b]$的值就等于值为1的项数，也就是从点a到点b经过1个中间点的路径条数。我们继续考虑$\mathbf{G}^3$：

$$\mathbf{G}^3[a,b]=\sum_{i=1}^{N}\sum_{j=1}^{N}\mathbf{G}[a,i]\,\mathbf{G}[i,j]\,\mathbf{G}[j,b] \tag{10}$$

和$\mathbf{G}^2$很相似，$\mathbf{G}[a,i]\,\mathbf{G}[i,j]\,\mathbf{G}[j,b]=1$当且仅当$\mathbf{G}[a,i]=\mathbf{G}[i,j]=\mathbf{G}[j,b]=1$，也就是说$a-i-j-b$组成一条路经。那么$\mathbf{G}^3[a,b]$的值就是从点$a$到点$b$经过2个中间点的路径条数。从另一个方面来说，$\mathbf{G}^3=\mathbf{G}^2\mathbf{G}$，从点$a$到点$b$经过2个中间点的路径中，第二个中间点是$i$的路径条数恰好为$\mathbf{G}^2[a,i]\,\mathbf{G}[i,b]$，从这里也可以说明上一个结论。那么更一般的，$\mathbf{G}^k[a,b]$是否就等就$a$到$b$经过$k-1$个中间点也就是长度为$k$的路径条数呢？答案是肯定的。和前面的推导很相似，可以使用归纳法证明，$\mathbf{G}^k=\mathbf{G}^{k-1}\mathbf{G}$，那么

$$\mathbf{G}^k[a,b]=\sum_{i=1}^{N}\mathbf{G}^{k-1}[a,i]\,\mathbf{G}[i,b] \tag{11}$$

其中$\mathbf{G}^{k-1}[a,i]$就是从a到i长度为$k-1$的路径条数，$\mathbf{G}[i,b]$则表示了i到b是否有路径，两者相乘恰好就是从a到b长度为k的路径上倒数第二个点为i的路径条数。当$k=0$时，$\mathbf{G}^0$为单位矩阵，$\mathbf{G}^0[a,a]=1$也符合a到a有一条长度为0的路径。前面的证明当中并没有用到任何无向图的信息，因此这个结论也适用于有向图。

那么对于有重边的图，我们只需要把$\mathbf{G}[a,b]$改为表示点a,b之间重边的条数即可。证明和前面的几乎一样，此处就不在赘述。而对于有向图的自环，不需要特殊考虑。对于无向图的自环，我们通常是把$\mathbf{G}[a,a]$加上2，这样在计算$\mathbf{G}^k$的时候这条边就被算成2条不同的边，如果只加上1，又会不满足矩阵里所有元素和等于边数两倍的性质。需要怎样应具体情况具体分析。

3.1 沼泽鳄鱼 ZJTSC'05

3.1.1 题目大意

池塘当中有N个石墩，有些石墩之间用石桥连接。有$NFish$条食人鱼周期性地在一些石墩之间运动，如果周期为T，那么时间i与时间$i+T$食人鱼位于同

一个石墩。某人开始时刻在石墩s，要在时刻K到达石墩e，每一个单位时间必须沿着石桥移动一次，并且每个时刻都不能与食人鱼在同一个石墩上。给定所有的石桥以及每条食人鱼的运动周期以及运动规律，求共有多少不同的合法路线，输出答案 $\bmod 10000$。$1 \leq N \leq 50, 1 \leq K \leq 2 \times 10^9, 1 \leq NFish \leq 20$。保证初始时刻$s$是可以停留的位置。

3.1.2 分析

首先，我们很容易设计出一个使用动态规划的算法。用$f[i][j]$表示当前是时刻i，正停留在石墩j上，在这种情况下继续走下去有多少种不同的合法的方案能够恰好在时刻K走到点e。可以得到状态转移方程：

$$f[i][j] = \begin{cases} 0 & \text{if Not Can}[i][j] \\ \sum_{1 \leq k \leq N, \mathbf{G}[j][k]} f[i+1][k] & \text{if Can}[i][j] \end{cases} \tag{12}$$

初始条件为：

$$f[K][e] = \begin{cases} 0 & \text{if Not Can}[K][e] \\ 1 & \text{if Can}[K][e] \end{cases} \tag{13}$$

两式中的$\text{Can}[i][j]$表示第i个时刻石墩j上是否可以停留，$\mathbf{G}[i][j]$则表示点i与j之间是否连边。我们要求的答案就是$f[0][s]$。状态一个有$O(NK)$个，而状态转移需要的时间是$O(N)$，总的复杂度高达$O\left(N^2K\right)$。对于题目的数据范围来说根本无法出解。

分析上一个算法，主要问题就出在K的范围非常大。而状态数就和K相关，使得即使再怎么优化转移也无济于事，无法达到可行的复杂度。我们重新考虑这个问题，如果没有食人鱼，任何时刻任何点都可以走，那么这个问题就变成了我们前面讨论过的求任意一张图上两点间长度恰好为K的路径条数。而这个算法的复杂度是$O\left(N^3 \log K\right)$，对于这题的数据范围绰绰有余。

既然矩阵乘法对于这题的某个特殊情况如此高效轻松地解决了，那么能否拓展到这题的所有情况呢？有食人鱼和没有食人鱼的区别就在于，前者的图是不断在变化，而后者的图始终是不变的。设$\mathbf{G}_i$表示把与时刻i不能经过的点相关的边都去掉以后的图。同样可以得到最终时刻的答案矩阵：

$$\prod_{i=1}^{K} \mathbf{G}_i \tag{14}$$

这个问题好像已经有些眉目了，只是如何快速地乘出这些矩阵。重新分析题目，发现有个重要的条件还没有利用到，$2 \leq T \leq 4$。2、3、4的最小公倍数为12，也就是说每12个单位时间，图一定变回原来的样子！即$\mathbf{G}_i = \mathbf{G}_{i+12}$。利用这个性质可以得到：

$$\prod_{i=1}^{K} \mathbf{G}_i = \left(\prod_{i=1}^{12} \mathbf{G}_i\right)^{\lfloor \frac{K}{12} \rfloor} \cdot \prod_{i=1}^{K \bmod 12} \mathbf{G}_i \tag{15}$$

预处理出$\prod_{i=1}^{12} \mathbf{G}_i$后，用矩阵乘法快速幂计算出$\left(\prod_{i=1}^{12} \mathbf{G}_i\right)^{\lfloor K/12 \rfloor}$，而后半部分至多12个矩阵相乘可以直接计算。因此，时间复杂度为$O\left(N^3 \log K\right)$，非常优秀的算法！

3.1.3 小结

通过本题，我们得到一个结论：无论图怎样变化，求从某一点走到另一个点的确定边数的路径数，只需将每个时刻的图邻接矩阵直接乘起来就得到了答案矩阵。

3.2 奶牛接力 USACO'NOV07

3.2.1 题目大意

给定一张M条边的无向带权图，求从起点S到终点E恰好经过K条边的最短路径。$2 \leq M \leq 100, 2 \leq K \leq 1000000$。保证每个连边的点度至少为2。

3.2.2 分析

这个问题和前面讲的求两点间恰好经过K条边的路径数非常相似。于是，我们也希望能够找到一个类似的矩阵乘法的方法。可是通过尝试发现，最短这个看似简单的条件，居然无法用加减乘来实现。

只能先考虑使用其他的方法，很容易想到一个动态规划的方法。$f[i][j]$表示通过i条边的路径到达某个点j，最短距离是多少，则：

$$f[i][j] = \min_{k=1}^{N} (f[i-1][k] + \mathbf{G}[k,j]) \tag{16}$$

其中，N为顶点数，若k,j间有边，$\mathbf{G}[k,j]$为边权，否则为∞。动态规划状态数是$O(KN)$，转移需要枚举上一个顶点复杂度是$O(N)$，因此总的复杂度$O\left(N^2K\right)$。

虽然这个动态规划无法在题目给定的规模下快速出解，但是却给我们很多灵感。并且与(11)在形式上非常相像，不得不使我们重新考虑前面讲到的问题。我们可以设计出一个相似的动态规划的算法，$g[i][j]$表示有多少通过i条边的路径能够到达点j，则：

$$g[i][j] = \sum_{k=1}^{N} g[i-1][k] \cdot \mathbf{G}[k,j] \tag{17}$$

惊人的相似！不同的地方就是把乘法改成了加法，加法改成了取最小值的min。这使我们产生了一个大胆的想法：能否修改矩阵乘法，使得不仅能够使得能够计算(16)，也同时满足结合律。答案是肯定的！我们重新定义矩阵乘法为：

$$\mathbf{C}[i,j] = \min_{k=1}^{b} (\mathbf{A}[i,k]\,\mathbf{B}[k,j]) \tag{18}$$

其中$\mathbf{A},\mathbf{B}$分别是大小为$a \times b, b \times c$的矩阵，$\mathbf{C} = \mathbf{AB}$大小为$a \times c$。易证仍然

满结合律：

$$
\begin{aligned}
&((\mathbf{AB})\,\mathbf{C})\,[i,j] \\
&\qquad = \min_{l=1}^{c}\left(\min_{k=1}^{b}(\mathbf{A}\,[i,k]+\mathbf{B}\,[k,l])+\mathbf{C}\,[l,j]\right) \\
&\qquad = \min_{k=1}^{b}\min_{l=1}^{c}(\mathbf{A}\,[i,k]+\mathbf{B}\,[k,l]+\mathbf{C}\,[l,j]) \\
&\qquad = \min_{k=1}^{b}\left(\mathbf{A}\,[i,k]+\min_{l=1}^{c}(\mathbf{B}\,[k,l]+\mathbf{C}\,[l,j])\right) \\
&\qquad = (\mathbf{A}\,(\mathbf{BC}))\,[i,j]
\end{aligned}
\tag{19}
$$

将修改过的矩阵乘法应用于此题，时间复杂度为$O\left(N^3\log K\right)$。题目虽然没有讲N的限制，但是由于$M\leq 100$，每个点的度至少为2。因此涉及到的点数不会超过100，效率非常高！

通过这个例子，我们隐约感到矩阵乘法在图邻接矩阵上的“神奇功能”远不止此。接下来，我们将继续发掘这些“功能”。

3.3 最小最大边路径问题

3.3.1 描述

给定一张有向图，求某两点间通过边数恰好为K的路径，使得最大边最小。

3.3.2 分析

在处理这类要求最大最小或者最小最大的时候，我们通常可以使用二分答案，然后检验的方法。对于此题，同样也适用。先二分答案，设当前要判断的答案为x，那么所有边权不超过x的边是可以在路径上的，而其他边都是不允许经过的。问题就转化为，给定一张有向图，判断某两点间是否存在恰好经过K条边的路径。便可以直接使用使用求恰好经过K条边的路径条数的算法。因为答案肯定恰好是某条边的长度，可以事先把所有的边排序，然后在排序表里二分，二分次数是$O\left(\log N\right)$。总的复杂度便是$O\left(N^3\log N\log K\right)$。

能否像前面的问题一样，把复杂度优化到$O\left(N^3\log K\right)$呢？我们参照前面的方法，得出一个动态规划的算法，$f\,[i]\,[j]$表示从起点出发，经过恰好i条边到达j，最大边最小是多少。有转移方程：

$$
f\,[i]\,[j]=\min_{k=1}^{N}\max\left\{f\,[i-1]\,[k]\,,\mathbf{G}\,[k,j]\right\} \tag{20}
$$

对比(17)，可以修改矩阵乘法为：

$$
(\mathbf{AB})\,[i,j]=\min_{k=1}^{N}\max\left\{\mathbf{A}\,[i,k]\,\mathbf{B}\,[k,j]\right\} \tag{21}
$$

由于min, max操作满足：

$$\begin{aligned}\max\left\{\min_i \max\left\{a_i, b_i\right\}, c\right\} & \\ &= \min_i \max\left\{\max\left\{a_i, b_i\right\}, c\right\} \\ &= \min_i \max\left\{a_i, b_i, c\right\}\end{aligned} \tag{22}$$

得到了和前面很相似的结合律的证明：

$$\begin{aligned}&\left(\left(\mathbf{AB}\right)\mathbf{C}\right)[i,j] \\ &= \min_{l=1}^{c} \max\left\{\min_{k=1}^{b} \max\left\{\mathbf{A}\left[i,k\right], \mathbf{B}\left[k,l\right]\right\}, \mathbf{C}\left[l,j\right]\right\} \\ &= \min_{k=1}^{b} \min_{l=1}^{c} \max\left\{\mathbf{A}\left[i,k\right], \mathbf{B}\left[k,l\right], \mathbf{C}\left[l,j\right]\right\} \\ &= \min_{k=1}^{b} \max\left\{\mathbf{A}\left[i,k\right], \min_{l=1}^{c} \max\left\{\mathbf{B}\left[k,l\right], \mathbf{C}\left[l,j\right]\right\}\right\} \\ &= \left(\mathbf{A}\left(\mathbf{BC}\right)\right)[i,j]\end{aligned} \tag{23}$$

时间复杂度降为$O\left(N^3 \log K\right)$。

3.3.3 一些拓展

拓展**1** 最小最小边问题：给定一张有向图，求边数恰好为K的路径上最小边最小是多少。

这个问题和与刚才的问题比较相似，可以使用类似的方法解决。只需把矩阵乘法中的加法和乘法均改为min运算，分配律和前面基本一样，此处不再赘述。

拓展**2** 给定一张有向图，求边数不超过K的路径上最大边最小是多少。

比较方便的方法是把原图每个点i，都增加一条i到i的边权值为$-\infty$，然后直接套用本题的算法即可，时间复杂度不变。

拓展**3** 给定一张有向图，求边数在范围K_1到K_2的路径上最大边最小是多少。

尽管区间还是可以转化为两个区间的差。但是取最小值操作无法进行做逆运算，换句话说就是知道$[0, K_2], [0, K_1 - 1]$的答案后无法求出$[K_1, K_2]$。我们重新看拓展2，增加一条i到i的权为$-\infty$的边，等价于将图矩阵的主对角线上的值都变成$-\infty$。从另一个方面来考虑，可以说是把前一个答案保存下来了，因为根据修改过的矩阵乘法的定义：

$$\left(\mathbf{AG}\right)[i,j] = \min_{k=1}^{M} \max\left\{\mathbf{A}\left[i,k\right], \mathbf{G}\left[k,j\right]\right\} \tag{24}$$

其中当$k = j$时，$\max\left\{\mathbf{A}\left[i,k\right], \mathbf{G}\left[k,j\right]\right\} = \mathbf{A}\left[i,j\right]$。即原来的答案，而其他项运算出来的结果就是新的答案。设原矩阵为$\mathbf{G_0}$，可以先乘K_1个$\mathbf{G_0}$，然后再乘$K_2 - K_1$个可以保存答案的$\mathbf{G}$，答案也就是$\mathbf{G_0}^{K_1}\mathbf{G}^{K_2-K_1}$。直接使用快速幂，时间复杂度为$O\left(N^3 \log K_2\right)$。

4 矩阵乘法与折半递归

4.1 外星语言(TopCoder Algorithm SRM 377, Div 1, 1000)[2]

4.1.1 题目大意

一种字母表包含了P个元音字母和Q个辅音字母，每个单词不超过N个元音和N个辅音。每个单词一定是一段元音字母接一段辅音字母，即所有的元音字母都在辅音前面。元音字母和辅音字母的个数都可以为0，但是整个单词长度不能为0。每个单词可以有重音，重音标在字母上。有三种可能，单词没有重音，有一个重音（可以在任何一个字母上），有两个重音（一个在元音上一个在辅音上）。不同重音拼写相同的单词被认为是不相同的。给定N, P, Q, M $\left(1 \leq N, P, Q \leq 10^9, 2 \leq M \leq 10^9\right)$，求有多少种不同的单词，输出答案 $\bmod M$。

4.1.2 分析

本题数学味较浓，且数据范围达到10^9之大。因此，我们希望找到一个数学方法直接求出答案。

通过观察可以发现，元音和辅音互不干扰，只需求出各自有多少种情况即可，而且两者在描述上等价，因此我们只需求出其中一个，另一个可直接套用前者公式。现在只需考虑如下问题：一个单词有不超过N个字母，每个字母有K种情况，可能有一个字母上有重音，也可能没有，求可能的单词数 $\bmod M$的值，设答案为$S_{N,K}$。则$S_{N,P}$就是元音的总数，$S_{N,Q}$是辅音字母的总数，除去单词长度为0的情况，我们要求的答案就是$(S_{N,P} \times S_{N,Q} + M - 1) \bmod M$。对于这个问题，当有$i$个字母的时候，可能的单词数为$(i+1)\,K^i$。则总方案数即

$$\sum_{i=0}^{N} (i+1)\,K^i \tag{25}$$

设为T，则

$$KT = \sum_{i=1}^{N+1} i \cdot K^i \tag{26}$$

两式相减得，

$$\begin{aligned} &(K-1)\,T \\ &\quad = (N+1) \cdot K^{N+1} - \sum_{i=0}^{N} K^i \\ &\quad = (N+1) \cdot K^{N+1} - \frac{K^{N+1}-1}{K-1} \end{aligned} \tag{27}$$

即

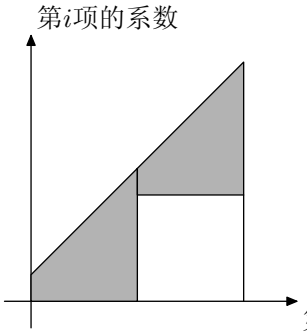
$$T = \frac{(N+1)\,K^{N+1}}{K-1} - \frac{K^{N+1}-1}{(K-1)^2} \tag{28}$$

[2]题目描述由TopCoder(中国)授权使用。

通过上面的推导，我们利用数学方法轻松地直接求出了答案。但本题就这样解决了么？不！我们要求的$S_{N,K}$是$T \bmod M$的值。但是，T的表达式中有除法运算，且分母$K-1$不一定和M互质，因此也不能采用解模线性方程的方法来解决这个除法问题。K^{N+1}的位数可能达到几十亿位，也不能用高精度直接算出答案，然后再$\bmod M$。我们不得不回到上面没有除法的(25)式。

4.1.3 折半递归

观察发现$S_{N,K}$的表达式是类似于阶梯形的，图中两块灰色的图形是完全一样的，而剩下的矩形部分就是我们熟悉的等比数列。而灰色部分就是$S_{N/2,K}$。



由此我们得到了一个基于折半的算法：

$$\begin{aligned} & S_{2N+1,K} \\ & \quad = \sum_{i=0}^{2N+1} (i+1)\, K^i \\ & \quad = \left(K^{N+1}+1\right) S_{N,K} + K^{N+1}\,(N+1) \sum_{i=0}^{N} K^i \end{aligned} \tag{29}$$

其中的Σ可以继续使用折半的方法求出：

$$\sum_{i=0}^{2N+1} K^i = \left(K^{N+1}+1\right) \sum_{i=0}^{N} K^i \tag{30}$$

上两式中，所有的K^N均可以使用快速幂来算，整个算法的时间复杂度是$O\left(\log^3 N\right)$，通过维护$\sum_{i=0}^{N} K^i$以及K^N可以做到$O\left(\log N\right)$。

4.1.4 矩阵乘法

直接求答案的复杂度是$O\left(N\right)$的，对于题目的数据范围来说，显然太高了。我们想到了递推自动化工具——矩阵乘法。怎么构造矩阵呢？我们虽然可以维护$i+1$和K^i，却没有办法把它们相乘。再次观察求答案的步骤，(26)式给了我们灵感：

$$S_{N+1,K} \equiv K \cdot S_{N,K} + \sum_{i=0}^{N+1} K^i \pmod{M} \tag{31}$$

(31)中的Σ可以用同样的方法维护：

$$\sum_{i=0}^{N+1} K^i \equiv K \cdot \sum_{i=0}^{N} K^i + 1 \pmod{M} \tag{32}$$

令初始矩阵为

$$\mathbf{I} = \begin{bmatrix} 1 & K+1 & 1 \end{bmatrix}$$

每次用作递推的矩阵

$$\mathbf{D} = \begin{bmatrix} K & 0 & 0 \\ 1 & K & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

在递推时，矩阵大小始终是1×3的，我们保持这三项分别是$S_{t,K}$、$\sum_{i=0}^{t+1} K^i$和1。由于矩阵乘法中只涉及到数的加法和乘法。因此可以边算边 mod M。这样，我们只需算出$\mathbf{ID}^N$就解决了这个问题。算法的复杂度为$O(\log N)$。

4.1.5 拓展

注意到本题要求的等价于$\sum_{i=1}^{N} i \cdot x^i$，而我们为了求得答案还维护了$\sum_{i=1}^{N} x^i$，即$\sum_{i=1}^{N} i^0 \cdot x^i$。那是否对于更大的$a$，我们可以用同样的方法来求$\sum_{i=1}^{N} i^a \cdot x^i$呢？为了表述方便，我们设$f_{N,a} = \sum_{i=1}^{N} i^a \cdot x^i$。当$a > 0$时，用前面的方法可以得到

$$\begin{aligned} & f_{N,a} - x \cdot f_{N-1,a} \\ &= \sum_{k=1}^{N} \left(k^a - (k-1)^a\right) \cdot x^k \\ &= \sum_{k=1}^{N} \left(\sum_{j=0}^{a-1} k^j \left(-1\right)^{a-j+1} \binom{a}{j} \right) \cdot x^k \\ &= \sum_{j=0}^{a-1} \binom{a}{j} \left(-1\right)^{a-j+1} \left(\sum_{k=1}^{N} k^j \cdot x^k \right) \\ &= \sum_{j=0}^{a-1} \binom{a}{j} \left(-1\right)^{a-j+1} f_{N,j} \end{aligned} \tag{33}$$

即

$$f_{N,a} = x \cdot f_{N-1,a} + \sum_{j=0}^{a-1} \binom{a}{j} \left(-1\right)^{a-j+1} f_{N,j} \tag{34}$$

并且当$a = 0$时，有

$$f_{N,0} = x \cdot f_{N-1,a} + x \tag{35}$$

由于$f_{N,i}$的值依赖于$f_{N,j}\,(j < i)$的值，可以按照a从小到大的顺序，可以依次将$f_{N,i}$用$f_{N-1,j}\,(0 \le j \le i)$线性表示出来。在$1 \times (a+2)$的矩阵中，依次维护$f_{N,0}, f_{N,1}, \cdots, f_{N,a}, 1$。于是使用类似于原题的方法，就可以求得答案。时间复杂度为$O\left(a^3 \log N\right)$。

4.2 Dice Contest CEPC'03[3]

4.2.1 题目大意

在一个宽度为四长度无限的棋盘上，有一个骰子放在(x_0, y_0)。骰子每一面上有一个权值，要将这个骰子滚到(x_1, y_1)，每次滚动的费用为滚好以后朝上面的权值，求最少的花费。$|x_0|, |x_1| \leq 10^9, 1 \leq y_0, y_1 \leq 4$，骰子每一面的权值为不超过50的正整数。

4.2.2 分析

本题的数据范围非常之大，普通的最短路算法根本无法胜任。但是，考虑到图的特殊性以及数据范围中只有横坐标的范围很大，而纵坐标的范围只有4，还是可以得到一些很优秀的算法。

由于棋盘无限长，若$x_0 \neq 0$，可以将起始位置和目标位置一起平移，使得$x_0 = 0$。如果此时$x_1 < 0$，可以将整个棋盘以及骰子作镜面反射。因此可以假设$x_0 = 0, x_1 \geq 0$。

4.2.3 折半递归

要到达$x = x_1$的格子，就必须经过$x = \lfloor \frac{x_1}{2} \rfloor$。只需求出$x = 0$的所有状态到$x = \lfloor \frac{x_1}{2} \rfloor$的所有状态的最短距离以及$x = \lfloor \frac{x_1}{2} \rfloor$到$x = x_1$的所有状态的最短距离即可。若$x_1$是偶数，由于棋盘无限长，那么从$x = 0$到中间的所有最短路就等于中间到$x = x_1$对应状态之间的最短路，只需计算一次。如果$x_1$是奇数，同理可以先算出到$x = x_1 - 1$所有状态的距离，然后预处理出$x = 1$的所有答案。这样问题就递归到$x = 0$和$x = 1$的情况。对于这两种情况可以用普通的最短路算法进行预处理计算，由于骰子权值范围不大，所以涉及到的格子不会很多，并且图非常稀疏， SPFA 或者堆优化的 dijkstra 都能在很快的时间内求出所有最短距离。设N为每一列的状态总数，每次递归的复杂度是$O\left(N^3\right)$，共要递归$O\left(\log x_1\right)$层，算法的时间复杂度是$O\left(N^3 \log x_1\right)$。

4.2.4 矩阵乘法

前面讲的图上固定边数两点间最短路与这题有很多相似的地方。考虑整条路径，对于每个$i\,(0 \leq i \leq x_1)$都存在至少一个状态满足$x = i$。我们取路径当中最后一个作为$x = i$的代表状态。我们把一列当中的N个状态看作点，那么从i列的代表状态走到$i + 1$列的代表状态可以看作从一个点走到了另一个点。只需预处理出相邻列所有状态间的最短路径，即所有的边，就将此题转化为了我们已经解决了的题。由于路径上i列的代表状态之后的点都在$x > i$上，预处理时之需要计算所有$x \geq 0$的状态。点数为N，路径上边的条数为x_1，因此复杂度也是$O\left(N^3 \log x_1\right)$。由于矩阵乘法可以优化到更低的复杂度，虽然在此题上意义不大，但是在渐进意义上，可以做到更优，复杂度可以达到$O\left(N^{2.36} x_1\right)$。

[3] http://acmicpc-live-archive.uva.es/nuevoportal/data/problem.php?p=2925

4.3 小结

由于矩阵乘法快速幂的本质就是折半递归，因此有时矩阵乘法的效率与折半递归都是一样的，甚至可能连程序都是差不多的。但是，矩阵乘法相比折半递归在很多情况下还是有优势的，比如外星语言的扩展，如果使用折半递归会非常难以得出结论，需要十分高的数学功底以及代数式变形技巧等，而用矩阵乘法，要做的只是二项式展开、移项等基本代数式变形，就可以轻松得出结论。由于矩阵乘法已经有了一些低于$O\left(N^3\right)$的算法，因此还可以优化到比折半递归效率高。

参考文献

[1] http://wikipedia.org/ 维基百科

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, "Introduction to Algorithms, Second Edition", 2001

感谢

- 感谢胡旭红老师在我写这篇论文时对我的指导和帮助。
- 感谢刘汝佳教练提供的帮助以及例题 Dice Contest。
- 感谢其他一些同学的帮助。