位运算及其对程序的优化

常州市第一中学 戴涵俊

JSOI2009 省队论文

目 录

序　言

程序中所有的数据在计算机内存中都是以二进制的形式储存的。位运算，本质上就是直接对整数在内存中的二进制位进行运算，同时，数的各个二进制位互不影响。由于位运算直接对内存数据进行操作，不需要转换成十进制，因此处理速度非常快，在信息学竞赛中往往可以优化理论时间复杂度的系数。另外，位运算还有很多特殊的技巧，能够帮助我们简化代码、美化程序等等。本文就结合自己的学习和应用经验，介绍一些位运算及其对程序的优化方法。

正 文

一. 位运算的基本操作

1. 位运算介绍

① and

X	Y	X and Y
0	0	0
1	0	0
0	1	0
1	1	1

通过上表不难发现，只有当 x 和 y 都为 1 时 and 后值才为 1。and 运算主要是用来取出某个二进制位。例如：A and (1 shl 5)就是取出 A 的二进制数从右往左第 6 位。

② or

X	Y	X and Y
0	0	0
1	0	1
0	1	1
1	1	1

or 运算通常用来强行给二进制的某一位赋值，注意 or 运算可能导致变量越界，对于有符号类型，or 可能把符号位取反，无符号类型可能直接就变成 0 了，这有可能会导致数据的丢失，使得程序崩溃。因此，执行 or 运算，应该尽量保证变量不超界，或者更保险地，是非负数。

③ xor

X	Y	X and Y
0	0	0
1	0	1
0	1	1
1	1	0

当两个位不同时得到 1，否则为 0。因此 xor 通常可以用来取反。有意思的是 xor 的逆运算是其本身。于是，我想起了一道做过的有点诡异的题目：给你 n 个数，n 大到只保证你读入不超时，其中仅有一个数出现了奇数次，要你找出这个数。方法是只要通过读一个 xor 一个，出现偶数次的肯定抵消了，剩下来的就是那个数了。

④ not

X	not (X)
0	1
1	0

not 操作就是直接把内存中的 0 和 1 全部取反。对于有符号类型，符号位也会取反，这是需要注意的，比如 x＝2147483647，x：longint；not（x）就得到了－2147483648，即 011111……1 是表示 maxlongint，100000……0 是表示的－maxlongint-1。

⑤ shl

左移，就是把二进制数整体向左移动 x 个位，并且右数 x 个单位是 0；如果移出界，那么移出部分就丢失了，而不会 runtime error。对于有符号类型，移位当然会移到符号位上去，比如 x＝2^30,x：longint；那么 x shl 1 就得到－2147483648；

⑥ shr

右移，就是把二进制数整体向右移动 x 个位，原来的最高 x 个位就变为 0；shr 相当于 div 2。

2. 位运算的优先级

not ＞ and , shl ,shr ＞ or,xor

比如下面的几个运算：

not 1 or 1 ＝ -1

not 1 and 1 ＝ 0

1 and 1 shl 1 ＝ 2

not 1 or 0 shl 1 xor 0 and 1 = -2

虽然掌握各个运算符的优先级并不困难，但是为了避免出错，增强程序的可读性，利于调试，我们还是在需要的地方添加括号来保证优先运算。

3. 位运算的口诀

清零取反要用与，某位置一可用或。
若要取反和交换，轻轻松松用异或。

4. 举例

例 1、一个文件中有 9 亿个不重复的 9 位整数，现在要求对这个文件进行排序（当然时间可以不止 1 秒，但要求出可行解）。

[问题分析]

拿到题目也许会吓一跳，这么多数，就是想快排存也存不下啊。线性时间的排序算法—桶排序？貌似是这样，但是 9 位数，这么大的哈希数组也弄不下啊。

注意到每个数都不同，那么每个数顶多出现 1 次，也就是说，要么出现，要么不出现。那么，我们可以用 0 表示未出现，1 表示出现。当然，这里一个 longint 不够，那么就 10^9 div 32+1 个 longint，对于一个数，判断它是在哪一个 longint 的哪一位，把它变成 1。输出时当作哈希表一样处理就可以了，总的时间复杂度是 0（n），比爆空间且缓慢的快排、裸的桶排序等算法要好的多。

二. 位运算的实用技巧

1. 对于 mod 运算的优化

一些特殊的模运算可以用位运算来代替，比如模 2 的整数次幂，我们可以用 and 运算来代替 mod 运算，程序段如下（以 mod 1048576 为例，y=134328497）。

mod 版本：for i:=1 to time do x:=y mod base;

and 版本：for i:=1 to time do x:=y and (1 shl 20-1);

当 time 分别取 10000000，100000000，1000000000 的时候，各自的用时如下：

	mod	and
10000000	0.22s	0.14s
100000000	1.26s	0.52s
1000000000	12.00s	4.23s

上面是在我的酷睿 T2370 笔记本上测试的，用 go32v2 编译执行。虽然不同电脑，不同编译器结果可能不同，但是，我们不难发现，改成 and 之后，效率有了较大的飞跃，而且运算次数越多，效率提升越明显。

2. 位运算的一些技术

运算要求	用位运算实现	实际应用
把右数第 k 位强行赋为 1	x or (1 shl (k-1))	通过这些操作，我们可以用 01 表示某个状态并且方便地改变这一状态，在哈希表、状态 DP、一些搜索记录状态中很实用
把右数第 k 位强行赋为 0	x and not(1 shl (k-1))	
右数第 k 位取反	x xor (1 shl (k-1))	
去掉右起第一个 1 的左边	x and (x xor (x-1))	树状数组中用到的低位技术
求 x 和 y 的平均值下取整	x and y+(x xor y) shr 1	避免 x+y 超界但结果不超界的情况
求 x 的相反数（x 基类型为有符整型）	not x＋1	更好地理解 not 以及数在计算机中的存储方式
交换变量 x 和变量 y 的值	x:=x xor y; y:=x xor y; x:=x xor y;	省去交换变量时候需要的临时变量
用位运算取绝对值（以 32 位整数为例）	x xor (not (x shr 31) + 1) + x shr 31	

三. 位运算对一些数据结构的优化

1. 循环队列

循环队列比较方便的实现可以用一个头指针 head，一个尾指针 tail，每次取出来的是 head mod base。这里不妨把 base 设置为 2 的整数次幂－1，然后用 and 来取模，既方便了代码书写，又不会降低效率，还美化了程序。

举一个例子，SPFA 是大家熟悉的求最短路径的简单、高效的算法，它需要维护一个队列，并且队列中最多会有 n 个元素（n 为顶点数）。于是，这里的队列我们可以用位运算优化的循环队列来做，代码见附录 1。

2. 树状数组

这种数据结构可以用线段树来代替，但是线段树比较烦琐，并且理论时间复杂度系数比较大，而树状数组就比较好写，且系数比较小。关于树状数组的具体原理就不赘述了，这里讲讲它的关键技术——低位技术（Lowbit），树状数组需要知道一个数的二进制表示中从右往左第一个 1 的位，用位运算实现只要执行：L[i]：= i and （i xor （i-1））。

3. 集合

用一个二进制数来表示一个集合的状态，用来替代 PASCAL 中缓慢的“集合”这种数据结构，可以大大提高程序的运行效率。因为每次位运算的操作可以看作是 O(1)的；同时，这种表示方法也利于编程，我们可以方便地用一个二进制数来表示。唯一的缺点就是，数感不好的同学不能一眼看出一个十进制数的某一位是 1 还是 0。

下面列举了一些集合的操作用位运算来实现的方法。

功能	用位运算实现
集合的并、交、差	A or B ； A and B ； A and not B
添加/删除某一个元素	A or 1 shl (k-1) ； A and not(1 shl (k-1))
判断某一元素是否存在	A and (1 shl (k-1)) 是否为 0。 为 0 就是不存在
从 A 集合中删去 B 集合（B 集合为 A 集合的子集）	A xor B

例 2、PIGS

问题描述：

尼克在一家养猪场工作，这家养猪场共有 M 间锁起来的猪舍，由于猪舍的钥匙都给了客户，所以尼克没有办法打开这些猪舍，客户们从早上开始一个接一个来购买生猪，他们到达后首先用手中的钥匙打开他所能打开的全部猪舍，然后从中选取他要买的生猪，尼克可以在此期间将打开的猪舍中的猪调整到其它开着的猪舍中，每个猪舍能存放的猪的数量是没有任何限制的。买完猪后客户会将他打开的猪舍关上。

好在尼克事先知道每位客户手中有哪些钥匙，要买多少猪，以及客户到来的先后次序。请你写一个程序，帮助尼克求出最多能卖出多少头生猪。

输入格式：

输入文件的第一行包含两个整数 M 和 N，1≤M≤1000，1≤N≤100，M 为猪舍的数量，N 为客户人数，猪舍的编号为 1 到 M，客户的编号为 1 到 N。

输入文件第二行包含 M 个空格隔开的整数，依次表示每个猪舍中的生猪数量，每个整数大于等于 0，且小于等于 1000。

接下来的 N 行每行表示一位客户的购买信息，第 I 个客户的购买信息位于第 I+2 行，其

格式如下：

A K_1 K_2……K_A B

它表示该客户共有 A 把钥匙，钥匙编号依次为 K_1 K_2……K_A，且 $K_1<K_2<……<K_A$，B 为该客户要买的生猪的头数。

输出格式

输出文件仅有一行包含一个整数，表示尼克最多能卖出的生猪的头数。

样例：

PIGS.IN

```
3 3
3 1 10
2 1 2 2
2 1 3 3
1 2 6
```

PIGS.OUT

```
7
```

[问题分析]

看到每个猪舍有数量限制、每个人有需求量上限，我们不难想到网络流的模型。首先可能会想到一个比较裸的模型，就是先一个源点向 m 个猪舍连边，上限为猪的数量，然后第一个客户向可以取到的猪舍连边，上限为正无穷，并且他向汇点连边，上限为需求量；第一个客户再伸出 m 条边重新引出 m 个猪舍，再作为新的源点向第二个客户连边，以此类推。这个图的层次、点数都相当多，显然，我们要进行优化。

因为可以把一个猪舍的猪趁打开猪舍时候赶到另外一个猪舍去，于是两个不同的客户如果有相同的猪舍可以开，就可以共享他们所能开到的所有猪舍的猪。于是我们可以直接对两个客户连边，即，如果客户 x 和 y，满足 x<y 并且 x 和 y 所能开的猪舍交集不为空，那么 x 就向 y 连边，这样 x 的猪舍的猪都能被 y 享用到了。

问题似乎到这里就完了，其实不然。由于要对任意两个人进行判集合的交是否为空，那么复杂度是 100*100，然后要对所有的猪舍进行比对，这个复杂度是 1000，于是，这个预处理的理论复杂度上限可能是 0（10^8），虽然经试验下来不太可能达到这个复杂度，但是我们还是要用一种简单易行的方法进行优化——位运算。

这里用整数来表示猪舍的有无情况，这样最大可以有 1000 个二进制位，于是还要分段处理。不妨每段用一个 64 位的 qword，把 1000 总共分成 16 段，对于每个猪舍编号 k，先判断是哪一段的（用（k-1）div 64+1，可以写成 shr 6 加速），然后再在那一段进行赋值。于是，对于两段猪舍，看有无交集，只需要 and 一下，看是不是 0 就可以了。这样，我们把预处理的时间复杂度降到了 0（100*100*16）。

虽然本题用位运算做有点显得多余了，但是这种方法还是有推广意义的。

例 3、山贼集团

问题描述：

某山贼集团在绿荫村拥有强大的势力，整个绿荫村由 N 个连通的小村落组成，并且保证对于每两个小村落有且仅有一条简单路径相连。小村落用阿拉伯数字编号为 1,2,3,4,…,n，

山贼集团的总部设在编号为 1 的小村落中。山贼集团除了老大坐镇总部以外，其他的 P 个部门希望在村落的其他地方建立分部。P 个分部可以在同一个小村落中建设，也可以分别建设在不同的小村落中。每个分部到总部的路径称为这个部门的管辖范围，于是这 P 个分部的管辖范围可能重叠，或者完全相同。在不同的村落建设不同的分部需要花费不同的费用。每个部门可能对他的管辖范围内的小村落收取保护费，但是不同的分部如果对同一小村落同时收取保护费，他们之间可能发生矛盾，从而损失一部分的利益，他们也可能相互合作，从而获取更多的利益。现在请你编写一个程序，确定 P 个分部的位置，使得山贼集团能够获得最大的收益。

输入格式：

输入文件第一行包含一个整数 N 和 P，表示绿荫村小村落的数量以及山贼集团的部门数量。

接下来 N-1 行每行包含两个整数 X 和 Y，表示编号为 X 的村落与编号为 Y 的村落之间有一条道路相连。(1<=X,Y<=N)

接下来 N 行，每行 P 个正整数，第 i 行第 j 个数表示在第 i 个村落建设第 j 个部门的分部的花费 Aij。

然后有一个正整数 T，表示下面有 T 行关于山贼集团的分部门相互影响的代价。($0<=T<=2^P$)

最后有 T 行，每行最开始有一个数 V，如果 V 为正，表示会获得额外的收益，如果 V 为负，则表示会损失一定的收益。然后有一个正整数 C，表示本描述涉及的分部的数量，接下来有 C 个数，Xi，为分部门的编号(Xi 不能相同)。表示如果 C 个分部 Xi 同时管辖某个小村落（可能同时存在其他分部也管辖这个小村落），可能获得的额外收益或者损失的收益为的|V|。T 行中可能存在一些相同的 Xi 集合，表示同时存在几种收益或者损失。

输出格式：

输出文件一行包含一个数 Ans，表示山贼集团设置所有分部后能够获得的最大收益。

样例数据：

输入样例	输出样例
2 1 1 2 2 1 1 3 1 1	5

数据规模：

对于 40%的数据，1<=P<=6。

对于 100%的数据，1<=N<=100，1<=P<=12，保证答案的绝对值不超过 10^8。

[问题分析]

本题还是有点意思的状态压缩树型动态规划，首先，我们用一个集合表示当前村落已经驻扎的山贼情况（这个可以用二进制数来表示），那么状态转移方程就可以如下表示：

f[i,j]:=max{f[i,x]+f[k,y]}＋profit[j]，其中 i 表示子树的根节点，k 表示儿子节点，j 表示根节点的状态集合，x or y＝j，profit 为对应状态下的总的盈利值。

由上，我们需要快速知道给定的一个集合，它的所有子集是什么，如果纯粹地对每个集合进行暴力穷举，这个时间复杂度是 4^n，但是，我们知道一个结论，就是 n 个元素的全集，它的所有子集的所有子集个数总和为 3^n（这个比较容易证明，在此不再赘述）。所以，我们希望不要穷举没用的集合。

一种方式比较容易想到，就是预处理每一个集合各自的子集，并用拉链存储。预处理时候再用队列优化，预处理用 n*3^n。在当时我就是这么做过的。

另外有一种更好的方式，不需要预处理，在参考了 NOI 专刊上的论文后感觉它与 lowbit 技术很好地结合了起来。比如当前的集合为 A，那么我们依次枚举出它的子集：

```
B:=A;
While A>0 do
  Begin  A:=(A-1) and B;   End;
```

A－1 的意思就是把 A 集合中最后一个 1 变成 0，最后一个 1 后面的 0 都变成 1（最后是指右边），然后再 and 一个 B，保证是原来 A 的子集。这样做就相当方便了。

4. 哈希表

哈希表最关键的莫过于哈希函数了，一个好的哈希函数可以大大降低哈希过程中的冲突，一般的哈希函数就是 mod 一个大质数。其实，一些用位运算写的哈希函数能够获得不错的效果，华丽的位运算也体现了它独特的魅力。

下面就介绍几个优秀的字符串哈希函数。

```
//DJB Hash
function DJBHash(s:string):un;
var hash:int64;
    i:longint;
begin
    hash=5381;
    for i:=1 to length(s) do hash:=hash+(hash<<5)+ord(s[i]);
    DJBHash:=hash and base
end;

//AP Hash
function APHash(s:string):un;
var hash:int64;
    i:longint;
begin
    hash:=0;
    for i:=1 to length(s) do
    if odd(i) then hash:=hash xor (hash<<7) xor ord(s[i]) xor (hash>>3)
    else hash:=hash xor not ((hash<<11) xor ord(s[i]) xor (hash>>5));
    APHash:=hash and base
end;
```

```
//SDBMHash
function SDBHash(s:string):un;
begin
    hash:=0;
    for i:=1 to length(s) do hash:=ord(str[i])+(hash<<6)+(hash<<16)-hash;
    // equivalent to : hash = 65599*hash+ord(str[i])
    SDBHash:=hash and base
end;

//JS Hash
function JSHash(s:string):un;
begin
    hash:=1315423911;
    for i:=1 to length(s) do hash:=hash xor ((hash<<5)+ord(str[i])+(hash>>2));
    JSHash:=hash and base
end;
```

上面的哈希函数原理就不再详细阐述了，重要的是运用。用上述哈希函数来优化哈希表，可以使你的程序得到进一步优化——华丽中彰显实用。

例 4、cryptcow（USACO 4.1）

问题描述：

农民 Brown 和 John 的牛们计划协同逃出它们各自的农场。它们设计了一种加密方法用来保护它们的通讯不被他人知道。 如果一头牛有信息要加密，比如"International Olympiad in Informatics"，它会随机地把 C，O，W 三个字母插到到信息中（其中 C 在 O 前面，O 在 W 前面），然后它把 C 与 O 之间的文字和 O 与 W 之间的文字的位置换过来。这里是两个例子：

International Olympiad in Informatics-> CnOIWternational Olympiad in Informatics

International Olympiad in Informatics-> International Cin InformaticsOOlympiad W

为了使解密更复杂，牛们会在一条消息里多次采用这个加密方法（把上次加密的结果再进行加密）。一天夜里，John 的牛们收到了一条经过多次加密的信息。请你写一个程序判断它是不是这条信息经过加密（或没有加密）而得到的：

Begin the Escape execution at the Break of Dawn

输入格式：

一行,不超过 75 个字符的加密过的信息。

Begin the EscCution at the BreOape execWak of Dawn

输出格式：

一行，两个整数. 如果能解密成上面那条逃跑的信息，第一个整数应当为 1，否则为 0；如果第一个数为 1，则第二个数表示此信息被加密的次数，否则第二个数为 0。

1 1

[问题分析]

本题是比较明显的搜索题，但是，如果不加一点点优化是无法在有效时间里面出解的。其中之一的优化便是哈希判重了，即，已经搜索过的状态就没有必要再搜索了。对于字符串的哈希我们不妨使用上面几个强大的哈希函数来进行。其余的就不多说了。

四. 位运算对一些算法的优化

1. 状态压缩动态规划

这里的状态是可以用集合来表示的状态，其实上面一个例子已经讲到用位运算来做状态压缩动态规划了，下面再举一个例子。

例 5、BOND

【问题描述】

所有人知道秘密特务 007，詹姆斯·邦德，但是很少人知道，许多时候他并不亲自去做任务，而是让他的表兄弟们完成。现在每当詹姆斯收到任务，他就将任务分发给大家，于是他需要你的帮助。

詹姆斯每月收到一张任务单。对于每一个任务，他都根据以往的经验，计算出他和其他几位兄弟完成的成功概率。你的程序应当找到一种分配方案，使得所有任务都被成功完成的概率最大。

注：所有任务都被成功完成的概率，等于每个任务都被成功完成的概率之积。

【输入格式】

第一行包含一个整数 N(<=20)，表示有 N 个任务，而他有 N 个兄弟来替他完成。

接下来包含 N 行，每行包含 N 个 0 到 100 之间的整数。第 i 行第 j 列的数，表示第 i 个任务被第 j 个兄弟完成的成功概率。概率以百分比的形式给出。

【输出格式】

仅一行，输出所有任务都被成功完成的最大概率，要求误差不超过 $\pm 10^{-6}$。

【测试样例】

bond.in

```
2
100 100
50 50
```

bond.out

```
50.000000
```

【问题分析】

本题还是比较明显的状态压缩动态规划，n<=20。我们可以按照任务被完成情况来划分状态，f[i,j]表示完成到第 i 个任务为止，人员使用情况为 j 的时候所能获得的最大概率。显然，这个状态划分满足最优子结构和独立性。对于状态 j，我们可以方便地用每个二进制

位上的数是 0 还是 1 来表示某一个人有没有被用。于是状态转移方程如下：

F[i,j]:=max{f[i-1,k]*a[i,r] | k or (1 shl (r-1))=j 且 k and (1 shl (r-1))=0}；

我们发现这个方程如果直接拿递推写还有点麻烦，于是想到用队列优化，这样不仅可以方便程序实现，还剔除了一些冗余状态。

实际测试下来还超时一个点，标准算法是用费用流或者 km 写的，这里就不再展开讨论。

2. 搜索

对于搜索的优化，可以有多个方面，同样，位运算可以从多个方面对搜索进行优化。

（1）状态表示

下文例题中的 TV 一题，便是通过用二进制数结合位运算来表示状态并进行状态转移，这样不仅提高了搜索效率，而且还方便了程序实现。

（2）状态判重

搜索常常需要开一个哈希表来记录状态是否达到或者该状态的最优值，有了位运算，一方面，我们将状态表示成整数，通过直接寻址或者拉链储存状态；另外一方面，加入位运算的哈希函数更加强大，减少哈希过程中的冲突。

例 6、TV

源程序名 TV.??? (PAS,BAS,C,CPP)
可执行文件名 TV.EXE
输入文件名 TV.IN
输出文件名 TV.OUT
运行时间限制 3S

问题描述：

大卫有一台旧电视机，上面的许多按钮已无法正常工作，当电视机新的时候，按下某一按钮，其他按钮都将被释放，只有被按的按钮工作。

现在按下某个按钮后，有一些按钮将被释放，而另外的一些按钮将不改变原状态。大卫知道按下每一个按钮会产生什么样的效果。

编写程序帮助大卫计算，从给定的状态到只有按钮 3 工作而其他按钮都被释放这个最终状态所需按下的按钮序列的最短长度。

输入格式：

输入文件的第一行包含一个整数 N(3≤N≤20)，表示电视机的按钮数。第二行包含用空格隔开的 N 个二进制数，表示各按钮的初始状态，0 表示相应的按钮是释放的，1 表示相应的按钮是按下的。

接下来的 N 行表示按下某个按钮时将有哪些按钮被释放。第 M+2 行由数字 K 开头，紧跟着 K 个数字(按升序排列)，表示当按下按钮 M 时被释放的按钮数及按钮号码(按钮号码用数字 1 到 M 表示)。每个按钮不能释放其本身，也可能不释放任何按钮。输入数据保证有解。

输出格式：

输出数据仅有一行，必须包含从给定的状态到只有按钮 3 工作而其他按钮都被释放这个最终状态所需按下的按钮序列的最短长度。

样例：

TV.IN

5

1 1 0 0 1

4 2 3 4 5

4 1 3 4 5

2 2 4

0

4 1 2 3 4

TV. OUT

3

[问题分析]

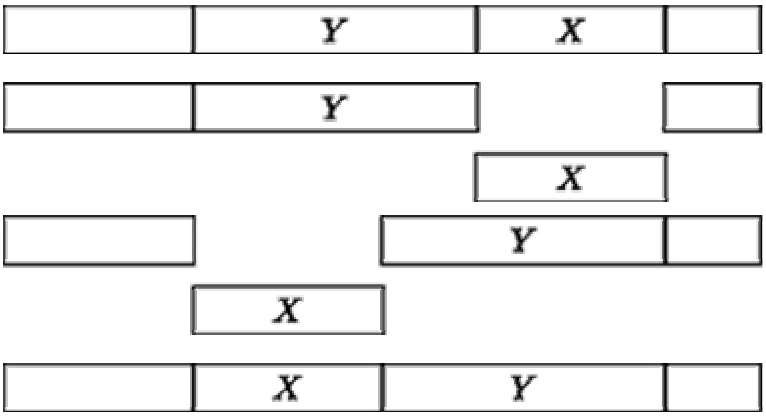
这题算法很明显，就是宽搜了。注意到n<=20，于是可以用一个 longint 来表示状态，0 表示按下，1 表示弹出，然后按照题目给定的操作规则进行搜索就可以了，在此不再赘述，详见附录。

例 7、POJ 3460 Booksort

题目大意：

给出 n 的一个全排列（n<=15），现在要你进行规定的一系列操作，使得这个全排列最终变为 1 2 3 … n，操作的规则是这样的，每次取出其中的一段，将这一段左/右面的等长一段（如果长度不够，那就全部的）与这一段交换位置。求使得最少需要的操作次数。如果次数>＝5 就输出“5 or more”。

图例：



样例输入：

3

6

1 3 4 6 2 5

5

5 4 3 2 1

10

6 8 5 3 4 7 2 9 1 10

样例输出：

2

3

5 or more

[问题分析]

本题可以用宽搜做。主要讲状态表示上面，我们发现 n<=15，所以可以用一个 qword 来表示状态，每 4 位表示原来排列中某一位上的数，这样，状态就可以表示了。转移的时候也相当方便，比如取出某一段[i,j]，那么我们只需要令 tmp＝(1 shl (j*4)-1) xor (1 shl (i*4)-1),然后拿 state and tmp 便取出了[i,j]这一段。剩下来的就是普通的移位操作了。这样比数组操作要方便得多。

五. 总结

位运算虽然表面上只是简单的几个操作，但是其中蕴含了很多的东西可以挖掘。掌握好位运算，首先，你对计算机的认识就更深了一步；其次，位运算作为一种底层操作，它所拥有的效率可以加速你的程序，减小常数时间操作对渐近时间复杂度的影响；第三，有了位运算，我们能够方便地将状态表示为整数，从而降低了编程复杂度和时空复杂度。

当然，位运算也会有一些缺陷，比如在我们表示状态的时候虽然用整数可以解决很多事情，但是调试的时候却没有数组或者其它表示方式来得直观，对于一个整数，我们似乎还没有直接知道某几位的 01 情况；在移位操作时候还要格外注意不要移出界，正整数移成负数的可能会使得数组越界造成 201 错误；还有就是注意普通乘、除运算和位运算的优先级，在必要的地方加上括号。

六. 附录

1. SPFA 循环队列版主要程序段

```
fillchar(h,sizeof(h),false);
h[st]:=true;
filldword(dis,sizeof(dis) shr 2,maxlongint shr 1);
q[0]:=st; head:=0; tail:=0;
while head<=tail do
  begin
    p:=q[head and base];
    while p<>nil do
      begin
        if dis[q[head and base]]+p^.len<dis[p^.link]
          then begin
                 dis[p^.link]:=dis[q[head and base]]+p^.len;
                 if not(h[p^.link])
                   then begin
                          h[p^.link]:=true;
                          q[tail and base+1]:=p^.link;
                          tail:=tail+1;
                        end;
```

```
                end;
          p:=p^.next;
        end;
      h[q[head and base]]:=false; head:=head+1;
    end;
```

2. PIGS 代码

```
type point=^node;
     node=record
       c,f,loc:longint;
       next,op:point;
     end;
var a,first:array[0..1500] of point;
    key:array[1..100,1..32] of longint;
    level:array[0..1500] of longint;
    h:array[0..1500] of boolean;
    q:array[1..1000000] of longint;
    n,m,num,k,i,j,t,s,maxflow,st,ed:longint;
    p1,p2:point;

procedure addedge(x,y,z:longint);
  begin
    new(p1);
    p1^.loc:=y; p1^.c:=z; p1^.f:=0; p1^.next:=a[x]; a[x]:=p1;
    new(p2);
    p2^.loc:=x; p2^.c:=0; p2^.f:=0; p2^.next:=a[y]; a[y]:=p2;
    p1^.op:=p2; p2^.op:=p1;
  end;

procedure makelevel;
var i,head,tail:longint;
    p:point;
  begin
    fillchar(h,sizeof(h),false);
    for i:=st to ed do
      level[i]:=maxint;
    level[st]:=0; h[st]:=true;
    head:=1; tail:=1; q[1]:=st;
    while head<=tail do
      begin
        p:=a[q[head]];
        while p<>nil do
          begin
            if (p^.c>p^.f) and (level[q[head]]+1<level[p^.loc])
```

```
              then begin
                     level[p^.loc]:=level[q[head]]+1;
                     if h[p^.loc]=false
                       then begin
                              h[p^.loc]:=true;
                              tail:=tail+1;
                              q[tail]:=p^.loc;
                            end;
                   end;
            p:=p^.next;
          end;
        h[q[head]]:=false; head:=head+1;
      end;
  end;

function min(x,y:longint):longint;
  begin
    if x<y
      then min:=x
      else min:=y;
  end;

function dfs(k,flow:longint):longint;
var delta,r:longint;
    p:point;
  begin
    if k=ed
      then exit(flow)
      else begin
             p:=first[k];
             delta:=0;
             while p<>nil do
               begin
                 if (p^.c>p^.f) and (level[k]+1=level[p^.loc])
                   then begin
                          r:=dfs(p^.loc,min(p^.c-p^.f,flow-delta));
                          p^.f:=p^.f+r; p^.op^.f:=-p^.f;
                          delta:=delta+r;
                          if delta=flow
                            then break;
                        end;
                 p:=p^.next;
               end;
             first[k]:=p;
```

```
            dfs:=delta;
          end;
  end;

function match(x,y:longint):boolean;
var i:longint;
  begin
    for i:=1 to 32 do
      if key[x][i] and key[y][i]<>0
        then exit(true);
    match:=false;
  end;

begin
  assign(input,'pigs.in');
  assign(output,'pigs.out');
  reset(input);
  rewrite(output);
  readln(m,n);
  st:=0; ed:=n+m+1;
  for i:=1 to m do
    begin
      read(k);
      addedge(st,i,k);
    end;
  readln;
  fillchar(key,sizeof(key),0);
  for i:=1 to n do
    begin
      read(num);
      for j:=1 to num do
        begin
          read(k);
          t:=(k-1) shr 5+1;
          s:=k mod 32;
          key[i][t]:=key[i][t] or (1 shl s);
          addedge(k,i+m,maxlongint div 2);
        end;
      readln(k);
      addedge(i+m,ed,k);
    end;
  for i:=2 to n do
    for j:=1 to i-1 do
      begin
```

```
        if match(i,j)
          then addedge(m+j,m+i,maxlongint div 2);
      end;
  maxflow:=0;
  while true do
    begin
      makelevel;
      if level[ed]=maxint
        then break;
      first:=a;
      maxflow:=maxflow+dfs(st,maxlongint div 2);
    end;
  writeln(maxflow);
  close(input);
  close(output);
end.
```

3. 山贼集团代码

```
type point=^node;
     node=record
       loc:longint;
       next:point;
     end;
var a:array[1..100] of point;
    q:array[1..5000] of longint;
    h:array[0..1 shl 12] of boolean;
    visit:array[1..100] of boolean;
    profit,v:array[0..1 shl 12] of longint;
    cost:array[1..100,1..12] of longint;
    f:array[1..100,0..1 shl 12] of longint;
    s:array[1..1 shl 12,1..2] of longint;
    k,n,m,i,j,max,t,head,tail,state,tmp,x,y,c:longint;
    p:point;

procedure solve(k:longint);
var p:point;
    i,j,x:longint;
  begin
    p:=a[k];
    while p<>nil do
      begin
        if visit[p^.loc]=false
          then begin
                 visit[p^.loc]:=true;
```

```
                solve(p^.loc);
                for i:=1 shl m-1 downto 0 do
                  begin
                    if f[k,0]+f[p^.loc,i]>f[k,i]
                      then f[k,i]:=f[k,0]+f[p^.loc,i];
                    x:=i;
                    while x>0 do
                      begin
                        x:=(x-1) and i;
                        if f[k,i xor x]+f[p^.loc,x]>f[k,i]
                          then f[k,i]:=f[k,i xor x]+f[p^.loc,x];
                      end;
                  end;
              end;
        p:=p^.next;
      end;
    for i:=0 to 1 shl m-1 do
      f[k,i]:=f[k,i]+profit[i];
  end;

begin
  assign(input,'cateran.in');
  assign(output,'cateran.out');
  reset(input);
  rewrite(output);
  readln(n,m);
  for i:=1 to n do
    a[i]:=nil;
  for i:=1 to n-1 do
    begin
      readln(x,y);
      new(p);
      p^.loc:=y; p^.next:=a[x]; a[x]:=p;
      new(p);
      p^.loc:=x; p^.next:=a[y]; a[y]:=p;
    end;
  for i:=1 to n do
    begin
      for j:=1 to m do
        read(cost[i,j]);
      readln;
    end;
  readln(t);
  fillchar(v,sizeof(v),0);
```

```
for i:=1 to t do
  begin
    k:=0;
    read(x,c);
    for j:=1 to c do
      begin
        read(y);
        k:=k or (1 shl (y-1))
      end;
    v[k]:=v[k]+x;
    readln;
  end;
t:=0;
for i:=0 to (1 shl m-1) do
  if v[i]<>0
    then begin
           t:=t+1; s[t,1]:=i; s[t,2]:=v[i];
         end;
fillchar(profit,sizeof(profit),0);
for i:=0 to (1 shl m-1) do
  for j:=1 to t do
    if i or s[j,1]=i
      then profit[i]:=profit[i]+s[j,2];
fillchar(visit,sizeof(visit),false);
for k:=1 to n do
  begin
    f[k,0]:=0;
    fillchar(h,sizeof(h),false);
    h[0]:=true;
    head:=1; tail:=1; q[1]:=0;
    while head<=tail do
      begin
        state:=q[head];
        for i:=1 to m do
          if state and (1 shl (i-1))=0
            then begin
                   tmp:=state or (1 shl (i-1));
                   if h[tmp]=false
                     then begin
                            h[tmp]:=true; tail:=tail+1; q[tail]:=tmp;
                            f[k,tmp]:=f[k,state]-cost[k,i];
                          end;
                 end;
        head:=head+1;
```

```
        end;
    end;
  visit[1]:=true;
  solve(1);
  writeln(f[1,1 shl m-1]);
  close(input);
  close(output);
end.
```

4. bond 代码

```
var q:array[1..1500000] of longint;
    f:array[0..1500000] of double;
    a:array[1..20,1..20] of longint;
    h:array[0..1500000] of boolean;
    x,y,n,i,j,k,head,tail,t,step:longint;

begin
  assign(input,'bond.in');
  assign(output,'bond.out');
  reset(input);
  rewrite(output);
  readln(n);
  for i:=1 to n do
    begin
      for j:=1 to n do
        read(a[i,j]);
      readln;
    end;
  fillchar(f,sizeof(f),0);
  fillchar(h,sizeof(h),false);
  h[0]:=true; f[0]:=100; head:=1; tail:=1; t:=1; q[1]:=0; step:=0;
  while head<=tail do
    begin
      step:=step+1;
      for i:=head to tail do
        begin
          x:=q[i];
          for k:=1 to n do
            if (x and (1 shl (k-1))=0) and (f[x]*a[step,k]/100>f[x or (1 shl
(k-1))])
              then begin
                     y:=x or (1 shl (k-1));
                     f[y]:=f[x]*a[step,k]/100;
                     if h[y]=false
```

```
                    then begin
                           h[y]:=true;
                           t:=t+1; q[t]:=y;
                         end;
                end;
        end;
      head:=tail+1; tail:=t;
    end;
  writeln(f[(1 shl n)-1]:0:8);
  close(input);
  close(output);
end.
```

5. TV 代码

```
Program Tv;
  Const last=1 shl 2;
  Var q:array[1..2,1..1100000]of longint;
      h:array[0..1100000]of boolean;
      a:array[1..20]of longint;
      n,i,j,k,s,m,f,r:longint;
Begin
  assign(input,'tv.in');
  assign(output,'tv.out');
  reset(input);rewrite(output);
  readln(n);s:=0;
  for i:=0 to n-1 do begin
    read(j);
    if j=1 then s:=s or (1 shl i);
  end;
  for i:=1 to n do begin
    read(k);a[i]:=not 0;
    for j:=1 to k do begin
      read(m);dec(m);
      a[i]:=a[i] and not (1 shl m);
    end;
  end;
  fillchar(h,sizeof(h),true);
  h[s]:=false;q[1,1]:=s;
  q[2,1]:=0;f:=1;r:=1;
  repeat
    k:=q[1,r];
    for i:=0 to n-1 do if k and (1 shl i)=0
      then begin
        j:=k and a[i+1];
```

```
        j:=j or (1 shl i);
        if j=last then begin
          writeln(q[2,r]+1);
          close(input);
          close(output);
          halt
        end;
        if h[j] then begin
          h[j]:=false;inc(f);
          q[1,f]:=j;
          q[2,f]:=q[2,r]+1
        end;
      end;
    inc(r)
  until r>f;
End.
```

七. 参考资料

1. http://www.matrix67.com
2. http://hi.baidu.com/zfy0701/blog/item/6bd496894a77aeb60f244422.html#send
3. http://blog.csai.cn/user3/50125/archives/2009/35638.html
4. NOI 专刊《C++与位运算》、《树状数组简介》