组合游戏的简单分析

—四川省绵阳南山中学 王晓珂

【关键字】

组合游戏 游戏的和 nim 和 Sprague-Grundy 函数

【摘要】

本文的主要内容是分析组合游戏的方法。
在介绍这些方法之前，会先介绍相关的概念
对于每一种方法,本文附有相应的例题帮助理解。

【组合游戏与相关的概念】

1．组合游戏

游戏是一个常见的概念，通常它是指娱乐性质的游戏。不过它也可以有一些其它的内涵比如竞争、寻找最优，像商业竞争、外交谈判就是这样的意义下的游戏，尽管不那么有趣、不过战争也可以算作是游戏。

在 OI 中有许多题目从游戏中提出问题。Alice & Bob 就因此出了名。在这些游戏中组合游戏占了很大一部分，组合游戏其实对我们来说并不陌生，它是是满足这样一些条件的游戏:

1） 游戏有 2 名参与者。
2） 游戏过程中任意时刻有确定的状态。
3） 参与者操作时可以的操作时将游戏从当前状态改变为另一状态，规则规定了在任意一状态时，可以到达的状态集合。
4） 参与者轮流进行操作。
5） 在游戏出于某状态，当前参与者不能进行操作时，游戏结束。此时参照规则决定胜负。
6） 无论参与者做出怎样的操作游戏在有限部数之内结束(没有平局)。
7） 参与者拥有游戏本身，和游戏过程的所有信息，比如规则、以前自己和对手的操作。

本文讨论的组合游戏还有一个公平性的限制、即 2 名参与者的操作规则
是相同的。

组合游戏也可以用一个有向图来表示 G=(X，F)X 为游戏的状态集合 F(X)为 X 可以到达的状态集合，而结束状态均宣布当前参与者失败。
如果原游戏的结束状态为胜利，那么有必要将图改一下适应题目规则。

组合游戏必然是一个无环图(无平局)。

2．必胜、必败状态

在组合游戏中必胜状态定义为当前玩家有策略能使无论对手做什么操作也可以保证自己胜利的状态。相反，如果之前操作的玩家能够保证胜利、则它是必败的状态

在组合游戏中一个状态如果不是必败状态就是必胜状态，这是显而易见的。通常我们分析做和游戏的目的就是判断一个状态为必胜还是必败，如果必胜，则找出必胜策略。完成这一点仅凭以上定义是不够的，而以下的性质(可以用来判定一组划分是否为必败状态与必胜状态的划分)则明确的指出了一种递推的方法。

1） 结束状态的性质由规则决定。

2） 一个非结束状态，如果它能到达任何必败状态，那么它是必胜状态，否则它就是必败状态。

尽管这对每个组合游戏都适用，但它可不是万能的，有时一个游戏的状态非常多，计算机无法处理，这时你就得借用别的方法了。

3．**Sprague-Grundy** 函数

它是定义在组合游戏状态上的函数，用 g (X)表示 X 状态的函数值。它的定如下：

g (X)= min｛ n| n∈ N ,n≠ for y∈ F(x) ｝

形象的说就是 X 的函数值为与 X 所能到达的任意点函数值不同的最小自然数。

我想细心的读者已经发现 g (X)= 0 当且仅当 X 为必败状态，这用其定义很容易验证：

1） 当 X 为结束状态时，由于 g (X)= 0 它是必败状态。

2） 当 X 不是结束状态时，如果它能到达必败状态，那么 g (X)＞ 0 ， X 是必胜状态。

3） 如果 X 不是结束状态且它到达的所有状态未必败状态，那么 g (X)= 0 它是必败状态。

这个冗长的定义，看上去毫无意义，因为直接判断一个状态的性质比使用这个函数要方便多了。但实际上与感觉的相反，这确是许多方法的基石，他的一些性质，使我们分析组合游戏的重要工具。

【概述】

组合游戏是一直是一个有趣的话题，有了电脑，我们可以更轻松的分析这些游戏。这些游戏渐渐进入了信息学竞赛，掌握合理分析组合游戏的方法的必要性也与日俱增。

我们总是有兴趣了解一个组合游戏是否有必胜的策略，如果有那应该怎么操作才能保证胜利？这是我们分析组合游戏的主要目的。

组合游戏有着许多普遍的规律，这是我们分析组合游戏的关键。怎么样利用这些规律，对于特殊的游戏，又怎么利用它的特殊性给我们的分析带来便利？

笔者将对许多组合游戏的分析过程加以总结，得出一套有效方法。不过任何方法都有其局限性，希望读者在掌握运用这些方法的同时，不要让思维局限于其中，从中得到启发的

同时能锻炼思维，使眼光敏锐，拓宽思维。

本文介绍的三种基本手段分别是分解游戏，归纳，转化游戏。

【分解游戏】

1、 分解游戏与游戏的和

游戏的和是指这样的游戏：两名参与者轮流操作若干子游戏，每次操作可以选择任意一个子游戏进行操作，最后后操作者胜利。

nim 游戏就是这样的典型。

nim 游戏为两人轮流从若干堆石子中取走石子，每次可以取走任意一堆中任意数目的棋子，但必须取走至少一枚，取走最后一枚石子的人胜利。它是由若干个一堆石子的特殊 nim 组成的，每次操作都只能按照子游戏的规则操作一个子游戏。

我们用 G=(X，F)表示 $G_1+ G_2+ \dots \dots G_n$．其中 $G_i=(X_i，F_i)$是一个简单的组合游戏。则

$X=X_1 * X_2 * X_3 *\dots\dots X_n$

对于 $X= (X_1，X_2，X_3\dots\dots X_n)$

$F(X)= F(X_1，X_2，X_3\dots\dots X_n)=$

$F_1(X_1) * \{ X_2 \} * \{ X3 \} * \dots \dots \{ X_n \} \cup$

$\{ X_1 \} * F_2(X_2) * \{ X_3 \} * \dots \dots \{ X_n \} \cup$

$\dots \dots$

$\{ X_1 \} * \{ X_2 \} * \{ X_3 \} * \dots \dots F_3(X_n)$

2、 Sprague-Grundy 定理

游戏的和往往比它的子游戏复杂许多。比如 nim，它的子游戏相当简单：除了没有石子的情况为必败，其他的都是必胜的状态。令 X 表示有 X 个石子的状态。但是多堆的情况不仅状态数陡增，而且关系更为复杂使它很难分析。

我们知道一个状态位必败状态当且仅当它的 SG 函数值为 0，所有状态的 SG 函数值可以完成必胜、必败状态的划分。Sprague-Grundy 定理就是用来寻找这样游戏的 SG 值的。

作为经典游戏 nim 的规律其实非常简单。对于每一堆石子 n 堆石子，第 i (1 <= i <= n)堆石子的个数是 X i，该状态为必败状态当且仅当 X1 **XOR** X2 **XOR** ……X n = 0。这些状态因为是必败状态，所以 SG 值也为 0

看到这里其实定理的内容很容易猜到了。

Sprague-Grundy Theorem:

对于 $G=G_1+G_2+ \dots \dots G_n$

$G(X_1，X_2，\dots \dots X_n) =G_1(X_1)$ **XOR** $G_2(X_2)$ $\dots \dots G_n(X_n)$

证明：

XOR 运算有一些重要的运算性质与加法相似，这里做一下必要的介绍(用符号 xor 表示)：

a xor b=c，c 的 2 二进制形式的第 i 位是 a，b 二进制形式该位置上的值队 2 取余的结果。 xor 的运算率有结合率，交换率。除此以外，若 a xor b=c 则 a xor d=c 当且仅当 b=c；a xor a = 0；a xor 0 = a；

令 $b=G_1(X_1)$ xor … … $G_n(X_n)$，则只需证明:

1） 于任意 **a**∈ N 且 **a < b ,** 一定存在 **X'**∈ **F(X)**使 **G(X') = a** 。

2） 对于任意 **X'**∈ **F(X)** ，那么有 **G(X') ≠ b。**

证明 1):

令 d=a xor b，令 d 的 2 二进制位数为 k 。必然存在 i 使 $G_i(X_i)$的二进制第 k 位为 1 。显然 $G_i(X_i)$ xor d < $G_i(X_i)$,因此存在 X_i'∈ $F_i(X_i)$ 使 $G_i(X_i')$ =$G_i(X_i)$ xor d 。

b*d = a => a = $G_1(X_1)$ xor … …$G_i(X_i)$ xor d … …$G_n(X_n)$= $G(X_1$… … X_i'，… …，$X_n)$

因为 (X_1… …X_i' … …X_n)∈F(X) 。所以 1) 得证。

证明 2):

假设 2)不成立，即对于一个任意的 X 存在 X′ ∈F(X)使 G(X ')= b 。那么令 X'=(X_1，… …，X_i',… …，X_n)，G(X')=$G_1(X_1)$ xor … …$G_i(X_i')$… …$G_n(X_n)$=b。因此 $G_i(X_i')$=$G_i(X_i)$与 SG 函数的定义矛盾， 2)得证。

由于在证明中用到了自身，因此需要用数学归纳法。

看，并不是所有的时候 SG 函数都比必败状态难求吧。将一个游戏分解，就是将它看作数个游戏的和，利用对每一个单独游戏分析的结果来研究整个游戏 。几乎所有的子游戏都比游戏的和要简单，因此在面对大多数复杂的游戏之和时，用分解的方法，所有问题都能迎刃而解。

用分解的方法来分析组合游戏

例题 1 :

ACM ICPC 2006 Asia Regional Contest,Beijing

A Funny Stone Game

David 玩一个石子游戏 。游戏中，有 n 堆石子,被编号为 0..n-1 。两名玩家轮流取石子 。每一轮游戏，每名玩家选取 3 堆石子 i,j,k(i<j,j<=k,且至少有一枚石子在第 i 堆石子中)，从 i 中取出一枚石子，并向 j，k 中各放入一枚石子(如果 j=k 则向 k 中放入 2 颗石子) 。最先不能取石子的人输。

请编程帮助 David。

石子堆的个数不会超过 23，每一堆石子不超过 1000 个。

输入:

输入包含多组数据，每组数据由 2 行构成，第一行为一个整数 n，第二行包含 n 个整数 S0..Sn-1 表示每一堆石子的个数。输入以包含一个 0 的一行结束。

输出:

每组数据的输出依次各占一行。输出格式为“Game t: i j k”,t 为数据的序号，i,j,k 表示一个保证 David 能获胜的开局。如果有多种方案，输出字典顺序最小的一组，如果没有这样的方案，i、j、k、均为-1。

分析:

这个游戏用另一种角度来看，可以将每一颗石子看作是一堆石子,如果它是第 p 堆中的石子，把么它所代表的这堆石子的个数为 n-1-p 。从而，操作变为拿走一个非 0 的石堆，并

放入 2 个规模小于他的石堆(可以为 0)。这便成了另一个游戏。之所以这么做是因为，转化后的游戏与经典的 take&break 游戏很相似。

因为石子堆是互不干扰的,因此这个游戏可以看作由若干个只有一堆石子的游戏组成。

先分析子游戏。求子游戏某状态 x 的 SG 函数值，我们需要它后继状态的 SG 函数值，子游戏的后继状态大多数为含有 2 堆石子的状态，不过 2 堆均小于 x 石子数。在了解石子数小于 x 的状态函数值的条件下，用 SG 定理可以求得任意后继状态的函数值。

用(p)表示只剩一堆规模为 p 的石子的状态，(p,q)表示剩下 2 堆石子，规模分别为 p,q 的状态。g (i，j) = g (i) xor g (j) ; g(i) = min{n∈ N | n ≠ g (p,q) for n ≥ p≥ 0 且 n ≥ q ≥ 0}。

应用以上结论，我们可以递推求得子游戏任意状态的 SG 函数值。用 SG 定理可以求得和游戏的任意状态的 SG 函数值。SG=0,David 可以保证他的胜利，否则就不行。至于策略，只要操作之后留下的状态 SG 值为 0 就行了。

时间复杂度分析:

以上算法包含 O (n)个 SG 值的计算，计算每一个的时间最多为 $O(n^2)$，判断必胜状态需要 $O(\sum S_i)$，寻找最优策略需要 $O(n^3)$的时间，综上,该算法的时间复杂度为 $O(n^3+\sum Si)$。

【归纳】

归纳也是分析组合游戏的一种常见方法。通过归纳，我们可以在很短的时间内完成对许多状态的分析。例如对于 nim 游戏的 1 堆情形，我们在递推的简单基础上，我们可以归纳得出规模为 x 的状态，其 SG 函数值为 x，x 必败当且仅当 x=0。由于归纳结果往往简单得出人意料，所以我想读者们在自己分析组合游戏的时候一定会自然而然地用上这种方法。

归纳的对象除了我们常见的必败状态还可以是刚刚介绍的 SG 函数。

通常我们做出归纳的基础都是游戏中的简单情形的特点。不过，游戏中简单的情形给予我们的启示是有限的。有时，我们需要在经验而不是事实的基础上做出猜想。许多游戏与经典游戏有着种种相似而又不完全相同，找出其模型上的相似之处，进行合理的猜想有时比从原始数据上进行归纳更直接有效。

例题 2:

staircase nim 经典组合游戏

游戏开始时有许多硬币任意分布在楼梯上，共 n 阶楼梯从地面由下向上编号为 0 到 n。游戏者在每次操作时可以将楼梯 j(1<=j<=n)上的任意多但至少一个硬币移动到楼梯 j-1 上。游戏者轮流操作，将最后一枚硬币移至地上的人获胜。

分析:

这个问题与 nim 游戏的区别在于移走的硬币不是被扔掉而是被放进了另一堆硬币之中，考虑能否将这一部分楼梯排除考虑范围。奇数号楼梯只能将硬币扔到偶数号楼梯之中，同样偶数号楼梯上的硬币也只会被扔上奇数号楼梯。只考虑奇数号楼梯 nim，若偶数楼梯只作容器，那么游戏变为 nim。当偶数号楼梯上的硬币可以将硬币移出时，我们是不是仍然可以用 nim 的方法判断必败状态?

将奇数台阶的硬币数 nim 和为 0 称作条件 A,结束状态满足条件 A；任何满足条件 A 的状态都到达满足条件 A 的状态；任何不满足条件 A 的状态都可以到达满足条件 A 的状态(nim)。因此一个状态为必败状态当且仅当它满足条件 A。

例题 3

翻转硬币(经典组合游戏)

在一条直线上排列着一行硬币，有的正面朝上、有的背面朝上。2 名游戏者轮流对硬币进行翻转。翻转时，先选一枚正面朝上的硬币翻转，然后，如果愿意，可以从这枚硬币的左边选取一枚硬币翻转。最后翻转使所有硬币反面朝上的玩家胜利。

T	H	T	T	H	T	T	T	H	H	T	H	T
1	2	3	4	5	6	7	8	9	10	11	12	13

图 1

例如图 1 所示的状态，将 2 和 12 同时翻转就构成一个合法的操作。

分析：

基于转化的思想，如果将位置 i 上的 H 看作一堆规模为 i 的石子，我们就能看到游戏与 nim 的相似点。不过有许多 nim 中的合法操作在这个游戏中是无法实现的，比如图一状态中，我们就无法从一堆规模为 10 的石子中取出其中的 5 个。不过我们将 10 与 5 同时翻转，所得到的状态。对应的 nim 状态 SG 值与从 10 中取 5 是相同的。因此我们猜想这个游戏的状态 SG 函数值与对应的 nim 相同。由于事实上虽然这个游戏无法实现 nim 的所有操作，但它的状态所能到达的状态 SG 函数值集合与对应的 nim 游戏状态相同。因此，它的状态 SG 函数值与对应 nim 相同。

【转化游戏】

现实中有许多游戏在一定的形式的掩饰下，隐藏了许多我们分析所必要的特点，有的游戏状态本身包含多种元素和关系异常复杂。面对这些令我们无法下手的情况时，我们应该想到对游戏的转化。

本文介绍 2 种方法，一是转化游戏的形式(等价转化)，而是对状态进行一定条件下的不等价转化。

一、等价转化

同一个组合游戏往往有不同表现形式，比如例题一中所提到的 take &break 游戏就可以转换成形式截然不同的石子游戏(只要将例题一稍加改变就能与 take &break 游戏等价)。

同一个游戏，从不同的角度，我们能够观察到的性质也不同。比如例 1 在原型下很难体现其游戏的和的本质，经过巧妙的转化后这一性质就浮出水面了。

例题 4 ：

POI2003/2004 stage I

Game

该游戏在 m*1 的平板上进行。平板的一些单位格上放有一枚硬币。两名游戏者轮流移动硬币。移动时，选取一枚硬币，将它放在右边最近的空格里，如果不存在，则将它扔掉，扔掉最后一枚硬币的人获胜

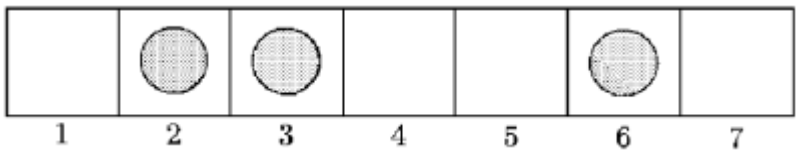

图 2

在图一表示的状态下，当千玩家可以将 2 的硬币移动到 4，3 到 4，6 到 7。

对于给出的状态输出先行者能保证胜利的第一步策略数。

输入

第一行输入 m,n 。m 如题是般的大小，n 为硬币的数量。第二行 n 个数分别表示每一枚硬币

所在的位置。
输出
进一行，保证先行者胜利的策略数。
分析：
首先每一枚硬币都相互影响，因此很难将其看作任何子游戏的和。状态繁多、关系复杂很难直接找到他的必败状态。考虑转换该游戏，经过多次尝试后可以这样转化一下，将任意空格与其左边最近的空格之间的空间看作一阶楼梯，最后一个空格的右边看作一个地面每段空间中的硬币数为楼梯数那么这个游戏就与 staircase nim 游戏完全等价了。
小结：
如果，我们不能将例二成功转化，那么对它的分析过程可以让人非常沮丧。面对组合游戏，从不同的角度来观察(转化)它应该作为我们的基本手段之一。我们最熟悉取石子/硬币这类游戏，它可以作为我们的首选。无论怎样转化，我们都要实现对游戏本质和特点的挖掘。

二、不等价的转化

我们分析组合游戏，只需要 SG 函数值就可以完成，将游戏原汁原味地转化，保持游戏本质不变是不是有些浪费呢?
由于，只需要 SG 函数值，我们可以期望在保证状态 SG 函数值不变时，将复杂状态化进行不等价的转化，化简到最朴素的状态，最后轻松解决。
例题 5

IPSC 2003 Got Root?

Alice & Bob 有一天发现了一株奇怪的灌木、他的枝分叉之后又可以汇合、更奇特的是来自不同的不仅是同一点的分叉可以汇合、即使分叉点相差很远、两根枝条仍然可以生长到一起(组织连在一块儿了)。在我们看来他可以抽象为一个无向图。其中有唯一结点是根。
他们认为这是用来玩游戏的绝佳材料。于是他们打算在这棵树上玩伐木游戏，规则如下：
1.两人轮流从灌木上截下枝条(图上的边)，每截下一根枝条就将与根不再相连的部分去掉。
2.截去最后一根枝条的人获胜
Alice 先手。
对于给出的无向图输出输出胜利的人的名字。
分析：
既然要转化、那么我们就要有一个目标，一个更简单的状态。我们先考虑树的情形是否会更简单。经过初步分析以后树形的状态也难以得到直接的结论、它还需要进一步的简化。那么再来考虑图构成一条长链的情况。这时，终于有了令人满意的结果，问题变为了典型的单堆 nim，可以直接求得 SG 函数值。
那么怎样将图转化成链呢? 我们分两步进行，先把图转化成树，在将树转化成链，整个过程保持状态的 SG 函数值不变。
先做第二步，在只有一个分叉点时，这是一个简单的 nim，可以转化为一个长为每根单链异或运算结果的单链。在更复杂一些的树上，猜想从末端开始进行这样的操作，将分叉的地方合并成一条链，长度为每条链的异或运算结果，不会改变整个状态的 SG 值。
这种合并方式可以推广为更一般的结论：对于任意 2 棵树 H1,H2 如果他们玩伐木游戏的 SG 值相同，则将他们接到两个相同图的对应点后，产生的新图 SG 值是一样的。
它的证明对于证明 2 个状态的 SG 函数值相通这样的命题具有一般性，令产生的 2 个图

为 G1,和 G2。先证明 G1+G2 后行者有必胜策略：player1 截下的是不是 H1 或 H2 的边，player2 截下相同的边；如果 player1 截下的是 H1 或 H2 的边，则 player2 截下 H1、H2 中 SG 值较高一个的某一边让 SG 相同。因此 g(G1+G2)=0 即 g(G1)=g(G2)。

接下来要将图转化成树。

图与树的区别在于图有环而树没有，把图转化成树最朴素的思想就是将图中的环一次缩成单个的点，那么相关的边怎么处理？图中的边是游戏操作的对象，我们最好将它们全都留下来，至于端点，如果已经被合并，那么边就变成一个到自己的环。

我们希望这样的转化，可以保持状态的 SG 函数值。这个结论证明比较长，这里就不再证明，只附在附录中。

不过，这样操作最终会留下大量的 self-loop，不难发现，一个 self-loop，和一条长为 1 的单链是等价的，这样，图到树的转化，我们就有了理论依据。

算法：

有了以上的结论，我们可以设计算法将图在 SG 值不变的条件下转化成一条单链。我们知道，在无向图中，有这样一个划分，每一个集合内删除任意一条边都能保持连通，集合之间则存在桥。每一个集合，可以通过缩环的操作收缩成一个点与成堆的自己到自己的边。我们的算法可以将图用 O(e)的时间对图进行这样的收缩，再用 O(n)的时间完成到树和链的转化，整个过程保持状态 SG 函数值不变。完成转化后，问题就很容易解决了。

该算法时间复杂度为 O(e)。

【小结】

三种方法一一讲述之后，笔者希望读者对组合游戏和它的的分析过程有了比较全面的认识，在竞赛中能灵活地运用这些方法。

组合游戏包含的内容很多，这些方法有许多问题是无法解决的，感兴趣的读者不要拘泥于本文的范围，可以从各种相关的书籍中更多地了解组合游戏。

解决组合游戏并不困难，重要的是拓宽知识，多做总结，灵活运用、扩展已知方法。

【感谢】

感谢叶诗富老师，刘汝佳老师对我的帮助和指导。
感谢何森与古楠同学给我提出的宝贵建议

【参考资料】

《Game Theory》 ferguson

【附录】

1、 融合原则的证明(摘自 ISPC 官方网站,其中所提到的 colon principle 是指把树转化为链所用到的结论)

Proof of the Fusion Principle. Suppose the contrary. Take the counterexample with the least number of edges, if there are more such counterexamples, choose the one with the least number of vertices. Call this graph **G**. We can deduce the following facts about **G**:

- For arbitrary vertices **x**, **y** on some cycle in **G** if we fuse **x** and **y**, SG value of **G** changes (otherwise we would get a smaller counterexample).
- When we remove the vertex 1, **G** stays connected, otherwise one of the components and the vertex 1 would form a smaller counterexample.
- No pair of nodes in **G** is connected by three edge-disjoint paths. Suppose the contrary, let the vertices **x**, **y** in **G** be connected by three edge-disjoint paths. Fuse **x** and **y**. The resulting graph **H** has to have a different SG value. The SG values of **G** and **H** differ, therefore **G+H** is a winning position for the player to move. We play a winning move. We then play the next move by removing the corresponding edge in the other component. (E.g. if the winning move cut an edge in **G**, cut the "same" edge in **H**.) We obtain a position **G'+H'** which has to be winning again. Note that **x** and **y** are still connected by at least two edge-disjoint paths in **G'**. However, **G'** is smaller than **G** and therefore the Fusion Principle holds for **G'**. We may fuse **x** and **y** without changing the SG value of **G'**. But by fusing **x** and **y** in **G'** we get **H'**. It follows that **G'** and **H'** have the same SG value. In other words, **G'+H'** is losing, which is a contradiction.
- Every cycle in **G** contains the vertex 1. Again, suppose the contrary. **G** is connected, therefore at least one of the vertices on the cycle has to be connected to the vertex 1 (without using edges of the cycle).

  If there is exactly one such point, it is an articulation. Take the "component" of **G** containing our cycle. It is smaller than **G**, therefore Fusion Principle holds and we may fuse the vertices on our cycle without changing the SG value of the "component". However from the Colon Principle we may replace the original "component" with the fused one without changing the SG value of **G**, which is a contradiction.

  If there are more such points, take any two of them. These points are connected by at least three edge-disjoint paths. (Twice by the edges of the cycle, at least once through the rest of the graph.) But **G** contains no such pair of vertices.

- **G** contains only one cycle. If there were two cycles with a common vertex **x** other than the vertex 1, then 1 and **x** would be connected by three edge-disjoint paths. Now suppose there are two cycles with only the vertex 1 in common. When we remove the vertex 1, **G** has to stay connected. Therefore there is a path leading through the rest of the graph that connects some vertex **x** on the first cycle to some vertex **y** on the second cycle. But then these two vertices are again connected by three edge-disjoint paths.

We now know that **G** contains only one cycle passing through the vertex 1. There may be additional vertices and edges in **G**, they form trees rooted in vertices of the cycle. To finish our proof we will show that the Fusion Principle in fact holds for **G**.

We now have to show that **G** has the same SG value as the graph **G'** obtained from **G** by fusing the vertices of the only cycle in **G**. Take the sum of these two games, we have to show that the position is losing for the first player. If he cuts an edge in one of the trees in **G** or **G'**, the second player responds by cutting the corresponding edge in the other graph. Both graphs are now smaller than **G**, moreover we still get one of them by fusing the vertices of the only cycle in the other one. But for graphs smaller than **G** the Fusion Principle applies. Therefore their SG value is the same and thus the whole position is losing.

So the first player has two options left: to cut an edge of the cycle in **G** or to cut one of the corresponding loops in **G'**. Suppose he cut an edge of the cycle in **G**. Thereby **G** is split into two trees. We have to show that this position is winning, i.e. that its SG value is non-zero. We will show that the SG value of this position is odd. Recall that when we computed the SG value of a tree, at each vertex we xor-ed some values. Now if we replace all the xors by addition, the last binary digit of the result will be the same. Knowing this, we can easily prove by induction that the parity of the SG value of a tree is the same as the parity of its number of edges. Our position consists of three trees and together they have an odd number of edges, therefore its SG value is odd.

The last option for the first player is to cut one of the loops at the vertex 1 in **G'**. We omit this part of the (already long) proof and leave it as an exercise to the reader.

2、 ISPC 2003 Got Root?

Internet Problem Solving Contest

IPSC 2003

Problem G - Got Root?

- Easy input data set - G1
- Difficult input data set - G2
- Solution

Famous biologist Herbert Greenstuff recently discovered a new kind of a plant, the *graphius vulgaris*, which is apparently endemic to the Pezinska

Baba National Wilderness in western Slovakia. This plant is quite peculiar. On the first sight it looks just like a normal bush. However, on the second sight you may notice that its structure is far more complicated. When two branches of the bush touch each other they are able to grow together. In fact, they often do. The result is a bush that is incredibly twisted and dense. (If Herbert were not a biologist but a computer scientist instead, he would notice that when you consider the plant as a graph, it is no longer a tree but a general undirected connected graph.)

A few days later the bushes were discovered for the second time by two computer science students, Alice and Bob. You probably know them, they are famous for playing games (and executing cryptographic protocols) day and night. They were in the middle of a simple game of NIM when Alice noticed the bushes. Immediately the simple NIM was forgotten. These bushes were an ideal opportunity to play a far more interesting game.

Given is a connected graph (the bush) on **N** vertices. The vertices are numbered from **1** to **N**. Edges represent branches of the bush, vertex **1** represents the ground. Players alternate in making moves, Alice moves first. A move consists of two steps. In the first step the player selects an edge and removes it from the graph. In the second step he/she removes all the edges that are no longer connected to the ground. (The player cuts a branch of the bush and removes the part of the bush he cut away.) The first player who

has no legal move (nothing to cut) loses. You may assume that both Alice and Bob play optimally.

Your task is so simple you probably expect it already. You have to save the bushes before Alice and Bob destroy them. For each of the bushes you have to find out who will win if they play their game on it. After you tell them the results you will spoil them all the fun and they'll leave the bushes alone. Please hurry, the bushes are really rare!

Input file specification

On the first line of the input file is the number **B** of bushes. **B** blocks of data follow, each of them describes one bush.

Each block begins with two numbers **N**, **M** separated by whitespaces, where **N** is the number of vertices and **M** the number of edges of the respective bush. Descriptions of **M** edges follow. For each edge the input file contains the numbers of the vertices it connects. All these numbers are separated by whitespaces.

Output file specification

For each block in the input file, output a single line containing the name of the player who will win the game on the given bush (e.g. either *Alice* or *Bob*).

Example

Input file

```
2

8 7

1 2
1 3
3 4
1 5
5 6
6 7
7 8

5 6

1 2
1 3
3 2
1 4
5 1
4 5
```

Output file:

```
Alice
```

Bob

3、 ACM ICPC 2006 Asia Regional Contest,Beijing

Problem A
Robot
Input File: robot.in

There is a robot in each grid of an $N \times M$ Grid. These robots can execute commands. Commands include:

NORTH All robots move one grid north.

SOUTH All robots move one grid south.

WEST All robots move one grid west.

EAST All robots move one grid east.

If a robot stays outside of the Grid after executing the command, it will be destroyed immediately.

Given the total number of each type of command, you task is to arrange an order of these commands, so that the maximized number of commands (i.e. the maximal sum of the number of commands executed by each robot) are executed by these $N \times M$ robots. (Note: a destroyed robot can not execute commands anymore.)

Input

Input contains several cases.

The first line of each case contains two positive integers N and M indicating the number of rows and columns in the grid ($1 \leq N, M \leq 10^6$). The second line contains four integers indicating the number of each type of command : NORTH, SOUTH, WEST and EAST respectively. Each of the four numbers will not exceed 10^5.

The last case is followed by a line containing two zeros.

Output

For each case, output an integer indicating the maximized number of commands that these robots can execute. The answer will be fit in a 64-bit signed integer.

Sample Input	Output for the Sample Input
2 2 1 0 0 0 2 2 1 1 1 1 0 0	Case 1: 4 Case 2: 9

四、**POI 2003/2004 GAME**

Task: gra

Game

I stage competition

Source file: gra.*xxx* (*xxx*=pas,c,cpp)

Memory limit: 32 MB

Let us consider a game on a rectangular board m x 1 consisting of m elementary squares numbered successively from 1 to m. There are n pawns on the board, each on a distinct square. None of them occupies the square with number m. Each single move in the is the following action: the moving player picks a pawn from any occupied square chosen at will and places it on the first unoccupied square with a larger number. The two players make moves in turn. The one who puts a pawn on the last square, i.e. the square with a number m, wins.

In the case presented in the figure (m = 7), a player is allowed to move a pawn from square no. 2 to 4, from square no. 3 to 4 or from square no. 6 to 7. The latter ends the game.

We say a player's move is winning if after making it he can win the game, no matter what moves his opponent makes.

Task

Write a programme that:

- reads the size of a board and the initial setup of pawns from the standard input,
- determines the number of distinct winning moves the starting player may choose in the given initial situation,
- writes the result to the standard output.

Input

The first line of the input contains two integers m and n ($2 <= m <= 10^9$, $1 <= n <= 10^6$, $n < m$) separated by a single space. The second line contains n increasing numbers - these are the numbers

of squares the pawns are set on. Numbers in the line are separated by single spaces.

Output

The first and only output line should contain the number of distinct winning moves possible for the starting player in the given initial situation.

Example

For the following input data:

```
5 2
1 3
```

the correct answer is:

```
1
```

For the following input data:

```
5 2
2 3
```

the correct answer is:

```
0
```