博弈论

博弈图

定理1：没有后继状态的状态是必败状态

定理2：一个状态是必胜状态当且仅当存在至少一个必败状态为它的后继状态

定理3：一个状态是必败状态当且仅当它的所有后继状态均为必胜状态

如果博弈图是一个有向无环图，则我们在绘制出博弈图的情况下用 $O(N+M)$ 的时间得出每个状态是必胜状态还是必败状态。其中 N 为状态种数 M 为边数。

对于 nim 游戏，当异或和为 0 时，状态为必败状态，否则为必胜状态。显然对于异或和为 0 的局面，一定不存在某一种移动让异或和仍然为 0 。而对于异或和不为 0 的局面，我们总有某种移动让异或和变为 0 。如果能证明这个，我们就能证明最终结论。

比如现在所有石子异或和为 t 我们显然要对 t 做二进制分解，取出最高位，所以原来的石子中，一定有奇数堆石子二进制这一位为 1 ，我们任选一堆，取这一堆，让最高位为 0 ，之后对于原来的 t 哪一位为 1 ，就让取的这一堆剩余的二进制和原来的二进制相反就可以了，可以知道这个数一定存在，且一定比原来少（因为刚才的最高位被取没了，且更高的位没动，所以一定变少）。所以小定理得证，大定理也就得证。其实设取的这一堆原来是 a_{i} 则 $a_{i}'=a_{i}$ ^ t

有向图游戏与 SG 函数

在一个有向无环图中，只有一个起点，上面有一个棋子，两个玩家可以沿着有向边，轮流移动棋子，不能移动的玩家判负。

定义 mex 函数的值为不属于集合 S 中的最小非负整数。

对于状态 x 和它的所有 k 个后继状态 $y_{1},y_{2},...,y_{k}$ 定义 SG 函数：

$SG(x)=mex\{SG(y_{1}),SG(y_{2}),...,SG(y_{k})\}$

SG 值为 0 是必败态。

对于由 n 个有向图游戏组成的游戏，设起点分别为 $s_{1},s_{2},...,s_{n}$ 则有定理，当且仅当 $SG(s_{1})$ ^ $SG(s_{2})$ ^ $...$ ^ $SG(s_{n})≠0$ 时，先手必胜，同时这是组合游戏状态 x 的 SG 值。

此证明省去。

对于 nim 游戏，一个堆取走任意不为 0 的数量，都会变成后继状态，所以 $SG(x)=x$ 即可满足题意，也就验证了上面的结论。

$Anti-SG$ 游戏

$Anti-SG$ 游戏规定，决策集合为空的游戏者赢。

$Anti-SG$ 游戏其他规则与 SG 游戏相同。

$Anti-Nim$ 游戏

现在我们把取完石子定为失败状态。

显然当每堆只有一个石子时，石子为偶数堆也就是异或和为 0 时，先手必胜，奇数堆时先手必败。

当有一堆石子不为 1 时，显然先手可以控制将这堆石子取剩下 1 或者 0 ，来控制 1 的堆数，根据上面的推论，先手必胜。

当至少有两堆石子数大于 1 时，还是考虑异或和，举了两个特例，提出异或和为 0 时，先手必败，不为 0 时，先手必胜的猜想。不为 0 的情况先手必胜和为 0 先手必败可以看成等价，所以只证明后者即可。

注意到当剩余大于 1 的石子堆数为 1 时，异或和不可能为 0 。当后手面对异或和为 0 的情况，只要他操作之后大于 1 的石子堆数大于 1 ，只需要将异或和还原为 0 即可。步数有限，后手早晚要面对到大于 1 的石子堆数等于 2 ，且不得不将其中一堆石子变为 1 的情况，这时根据情况 2 ，先手必胜，证毕。

SJ 定理

对于任意一个 $Anti-SG$ 游戏，如果我们规定当局面中所有的单一游戏的 SG 值为 0 时，游戏结束，则先手必胜当且仅当：

1.游戏的 SG 函数不为 0 且游戏中某个单一游戏的 SG 函数大于 1 （对应 $Anti-Nim$ 游戏中的石子数大于 1 的堆数只有一堆的情况，和有至少两堆石子数大于 1 ，且石子异或和不为 0 ）

2.游戏的 SG 函数为 0 且游戏中没有单一游戏的 SG 函数大于 1 （所有石子堆均为 1 个石子，这时候偶数堆必胜）

$Every-SG$ 游戏

$Every-SG$ 游戏规定，每轮操作要对所有单一游戏进行操作，其他规则和普通 SG 游戏相同。

显然有一个贪心，对于先手而言，先手希望自己必胜的小游戏进行的轮数尽量多，必败的小游戏尽量短。也就是说对于 SG 值为 0 的点，我们需要知道最快几步能让游戏终止，对于 SG 不为 0 的点，我们需要知道最慢几步游戏终止，用 $step$ 函数来表示这个值。

$step(u)=0$ 如果 u 是终止状态。

$step(u)=max\{step(v)\}+1$ $sg(u)≠0 ∧ v$ 为 u 的后继 $ ∧ sg(v)=0$

$step(u)=min\{step(v)\}+1$ $sg(u)=0 ∧ v$ 为 u 的后继

我们有定理：对于 $Every-SG$ 游戏先手必胜当且仅当单一游戏中最大的 $step$ 为奇数。证明显然。

$Multi-SG$ 游戏

$Multi-SG$ 游戏指在符合拓扑原则的前提下，一个单一游戏的后继可以为多个单一游戏，其他规则和 SG 游戏相同。

比如 $Multi-Nim$ 游戏，定义为：有 n 堆石子，两个人可以从任意一堆石子中多拿任意多个石子（不能不拿）或把一堆数量不少于 2 石子分为两堆不为空的石子，没发拿的人失败。

操作一和普通的 Nim 游戏等价。操作二实际上是将一个游戏分为两个游戏，于是相比于最开始的 Nim 游戏，我们增加了若干个操作二带来的后继，可以将这两个游戏的 sg 值异或起来作为一个新的游戏后继，再对所有后继求一个 mex 就知道现在这个点的 sg 值了，这个可以打表解决。

```
int sg[101],vis[305];
int main() {
    sg[0]=0,sg[1]=1;
    for(int i=2;i<=100;i++) {
        memset(vis,0,sizeof(vis));
        for(int j=0;j<i;j++) vis[sg[j]]=1;
        for(int j=1;j<=i/2;j++) vis[sg[j]^sg[i-j]]=1;
        for(int j=0;j<305;j++) {
            if(!vis[j]) {
                sg[i]=j;
                printf("%d %d\n",i,j);
                break;
            }
        }
    }
    return 0;
}
```

对于这道题而言，我们可以找到规律：

$SG(x)=x-1(x\ mod\ 4=0)$

$SG(x)=x(x\ mod\ 4=1||x\ mod\ 4=2)$

$SG(x)=x+1(x\ mod\ 4=3)$

可以背下来，但是没必要 $\times$

翻硬币游戏

一般的游戏规则是这样的 N 枚硬币排成一排，有的正面向上，有的反面向上，从左到右按照 1 到 N 编号。

游戏者按照某些约束翻硬币，但是他翻动的硬币中，最右边的必须是从正面到反面，谁不能翻谁输。

有一个结论，不管限制是怎么样的，局面的 SG 值为局面中每个正面朝上的棋子单一存在时的 SG 值

的异或和。比如现在有 $DUUDDU$ ，U 代表证明朝上，D 代表反面朝上。这等价于 $DUDDDD+DDUDDD+DDDDDU$ 三个游戏的和，同时 sg 值有 $sg[DUUDDU]=sg[DUDDDD]^sg[DDUDDD]^sg[DDDDDU]$ ，所以重点是如何求单个硬币朝上的 sg 值。

条件一：每次只能翻一个。

显然这种情况下，每个的异或值等同于朝上硬币数的奇偶。

条件二：每次能翻转一个或两个硬币（不用连续）。

每个硬币的 sg 值为它的编号，当前局面的 sg 值是所有正面向上的硬币编号异或和。

类似于 Nim 游戏，我们总能让一个异或和不为零的局面异或和变为零，最终局面的异或值为 0 。

对应于游戏中，比如当前异或和为 $x≠0$ ，我们取 x 二进制为 1 的最高位，再在所有 sg 值这一位为 1 的数中任选一个，将其变为 t^x ，

如果 t^x 在 sg 值中没出现，则相当于是翻一上和一下，且上的编号一定大于下，所以满足题意。

如果出现了，就相当于将两个向上的硬币翻到下面。

如果 $t^x=0$ 则只翻这一个硬币，其余不动。

条件三：每次必须连续翻转 k 个硬币

比如 $k=3$ ，

由上，只考虑单枚硬币向上的情况。显然 $sg[1]=sg[2]=0,sg[3]=mex(sg[2]^sg[1])=1$

$N=3t$ 时，先手必赢，$sg[3t]=1$ ，其余必输，$sg[3t+1]=sg[3t+2]=0$ ，

因为 $3t$ 翻完之后，会变成 $3t-1$ 和 $3t-2$ 这两种情况的异或和，也就是 $sg[3t]=mex(sg[3t-1]^sg[3t-2])=1$ ，对于 $3t+1$ ，$sg[3t+1]=mex(sg[3t]^sg[3t-1])=mex(1)=0$ ，对于 $3t+2$ ，同理。

推广到别的值也成立。

条件四：每次翻动一个硬币后，必须翻动其左侧最近三个硬币中的一个，除非这个位置编号小于等于 3 （编号小的话可以翻左边任意一个不一定是三个中的一个，也可以不翻）

$N=1$ 时，先手必胜，$sg[1]=1$

$N=2$ 时，先手必胜，$sg[2]=mex(sg[1],sg[0])=2$

$N=3$ 时，先手必胜，$sg[3]=mex(sg[0],sg[1],sg[2])=3$

$N=4$ 时，先手必败，$sg[4]=mex(sg[3],sg[2],sg[1])=0$

继续往下推可以发现是 $1\ 2\ 3\ 0$ 循环的，所以就推出来了。

条件五：每次可以翻动一个、两个或三个硬币

初始编号为 0

显然对于一个硬币的情况先手必胜

$N=1$ 时， $sg[0]=mex(0)=1$

$N=2$ 时， $sg[1]=mex(0,1)=2$

$N=3$ 时， $sg[2]=mex(0,1,2,1^2=3)=4$

$N=4$ 时， $sg[3]=mex(0,1,2,4,1^2=3,1^4=5,2^4=6)=7$

剩下太麻烦了，打个表

```
int sg[101],vis[1005];
int main() {
    sg[0]=1,sg[1]=2;
    for(int i=2; i<=200; i++) {
        memset(vis,0,sizeof(vis));
        vis[0]=1;
        for(int j=0; j<i; j++) {
            vis[sg[j]]=1;
            for(int k=j; k<i; k++) {
                vis[sg[j]^sg[k]]=1;
            }
        }
        for(int j=0; j<1005; j++) {
            if(!vis[j]) {
                sg[i]=j;
                printf("%d %d\n",i,j);
                break;
            }
        }
    }
    for(int i=0; i<=200; i++) {
        if(sg[i]==i*2) {
            printf("%d ",i);
            int j=i,cnt=0;
            while(j) {
                if(j&1) {
                    cnt++;
                }
                j>>=1;
            }
            if(cnt&1) printf("%d",);
            puts("");
```

```
        }

    }
    return 0;

}
```

太屑了，打完上面的表都没找到规律…其实注意到 2 的幂次 sg 值都是 原数的二倍就应该知道和二进制的关系了，硬生生没看出来…规律是二进制为 1 的位如果是奇数个就是 $sg[x]=2x$ 不然是 $sg[x]=2x+1$

然后我们考虑 $sg[x_{1}]$ ^ $sg[x_{2}]$ ^ $...$ ^ $sg[x_{n}]=0$ 有一个很显然的事，我们先规定二进制为 1 的位数是奇数的数是 j 数，为偶数的数为 o 数。显然 j ^ $j=o$ ^ $o=o,j$ ^ $o=o$ ^ $j=j$

注意到， 0 是一个 o 数，假设 sg 中 j 的个数为奇数，那么异或起来总共 1 的位数必为奇数，矛盾！所以 j 的个数为偶数。又注意到，不管原来的 x 为什么数 $sg[x]$ 都一定是 j 数，所以我们有结论，这样的 n 一定是奇数。

第二个结论，因为最后一位为 1 的位一定是偶数个，所以可以不看，可以认为所有的 $sg[x]=2x$ 于是都相当于把二进制后面加了个 0，所以其实是 x_{1} ^ x_{2} ^ $\ldots$ ^ $x_{n}=0$

反之，如果我们有 x_{1} ^ x_{2} ^ $x_{n}=0$ 且 n 为偶数。则我们应该有二进制位为 1 的位数为奇数的数为偶数，偶数减偶数为偶数，所以偶数的也是偶数。所以这些变成 $2x+1$ 末尾异或起来也是 0，所以也有 $sg[x_{1}]$ ^ $sg[x_{2}]$ ^ $...$ ^ $sg[x_{n}]=0$ 然后剩下的问题就变成了 Nim 游戏的那一套，只不过必败的条件多了一个硬币向上的数为偶数个。

条件六：每次可以连续翻动任意个硬币，至少翻一个

初始编号从 1 开始。

$sg[x]=mex(0,sg[x-1],sg[x-1]$ ^ $sg[x-2],...,sg[x-1]$ ^ $...$ ^ $sg[1])$

直接打表

```
int sg[101],vis[1005];
int main() {
    sg[1]=1;
    for(int i=2; i<=200; i++) {
        memset(vis,0,sizeof(vis));
        int tmp=sg[i-1];
        vis[0]=1,vis[tmp]=1;
        for(int j=i-2;j>=1;j--) {
            tmp^=sg[j];
            vis[tmp]=1;
        }
        for(int j=0;j<1005;j++) {
            if(!vis[j]) {
                printf("%d %d\n",i,j);
```

```
                sg[i]=j;
                break;
            }
        }
    }
    return 0;
}
```

这回找到规律了！就是这个数二进制最后一个为 1 的位代表的值。

阶梯 Nim 游戏

问题：有 n 个位置 $1,2,...,n$ ，i 位置上有 a_{i} 个石子，两个人轮流操作，每次可以挑选任意一个存在石子的位置 i ，将至少一个石子移动到 $i-1$ 的位置（所以终局是所有石子都移动到了 0 这个位置），谁不能移动就输。

显然这个是 Nim 游戏的改版，我们将奇数位置挪动到偶数位置看成减少石子，偶数位置移动到奇数位置看成增加石子，也就是说我们只关心奇数位置上的石子。然后就看成简单的 Nim 游戏就可以了。

比如最开始奇数位置异或起来不为零。由 Nim 游戏，先手可以移动某一奇数位置的石子，让这个异或和为零，然后后手无论怎么操作，先手都让异或和维持为零，然后又因为所有石子距离零这个位置的总距离总在不断减少，所以游戏早晚会结束，所以结论得证。

树上删边游戏

给出一个 N 个点的有根树，轮流删边，并且将不与根节点相连的部分移走，无边可删者输。

显然叶子节点的 sg 值为 0 。开始构造其他简单情况。

注意到一个结点连接奇数个叶子节点的 sg 值不应为 0 ，连接偶数个叶子结点的 sg 值为 0 。所以貌似父亲的 sg 值不是所有儿子的 sg 值的简单异或，但好像又有联系。

比如一个结点连接两个儿子，两个儿子的子树都是一条链，左边长度为 $lenl$ ，右边长度为 $lenr$ ，当 $lenl==lenr$ 时先手必败，好像是异或为 0 的情况，反之必胜，好像是异或值不为 0 的情况。

注意到一条链长为 L 的话， sg 值为 L 。

又比如一个根节点，有三个儿子， $len1=0,len2=1,len3=2$ （指链长），这个异或和肯定不为 0 ，但是先手必败，注意到都加一的异或和好像是 0 ，然后这么猜想一下 $\times$ ，就能猜到结论了：父亲的 sg 值是所有儿子 $sg+1$ 异或起来。然后树形 dp 一下就好了。复杂度是 $O(N)$ 的。

证明如下：

注意到 sg 值一样的游戏是可以相互转化的，如果我们求出来儿子的 sg 值（类似于归纳法），我们就将其等价于一个长度为此 sg 值的链，然后注意到父亲到儿子还有一条边，这边的 sg 值就是儿子 sg 值加一，然后所有的游戏互相独立，所以就是异或起来就可以了。

图删边游戏

刚才的树变成了图，其余规则不变。

相比于树，图多了一些环，有些是奇环，有些是偶环。

对于偶环，删掉一个边之后，剩下的两个链长度不同奇偶，所以异或之后的值不可能为 0 ，所以 sg 值必为 0 ，所以等效于一个点的情况 sg 值也为 0 ）

对于奇环，删掉一个边之后，剩下的两个链长度同奇偶，所以异或的 sg 值不可能为奇数，又因为可以拆成两个长度相同的链，所以 mex 值必为 1 ，所以相当于一个长度为 1 的链。剩下就变成上一个问题了。

威佐夫博弈

问题：两堆糖果，分别有 n,m 个，两个人轮流，可以从某一堆取走至少一个，也可以从两堆中取走同样多的物品，都是至少取一个，最后取光者获胜。

手动玩一下发现 $(1,1)$ 是必胜的， $(1,2)$ 是必败的，剩余的 $(1,m)$ 都是必胜的，然后除了 $(2,1)$ 都是必胜的，因为都可以变为 $(2,1)$ 的局面。其实到这里应该发现，我们如果找到必败态，所有能够到达必败态的状态都是必胜态，然后如果到达一个状态，如果这个状态没有打上必胜标记，那他就是必败的，然后他所到达的一切状态都是必胜的。

这好像似曾相识？没错就是埃氏筛，没打上必胜标记就对应质数。然后获得新质数之后就把所有的能到达的都变成必胜态（合数）。于是可以在 $O(n^{2})$ 的时间内筛出来所有的必胜必败态。但是威佐夫博弈的板子上的数据范围巨大，巨大到 $O(n)$ 都做不了，这需要一些结论。

这里的结论是 $(a_{i},b_{i})=(a_{i},a_{i}+i),i≥1,a_{1}=1$ 因为差恒定的都会被提前扫掉，所以每个解的差值都不一样。并且一定是当前没出现的最小的数和这个数加上这个差值。

这里有一个很漂亮的式子，是 $a_{i}=(int)(i/fi),b_{i}=(int)(i/fi/fi)$ 其中 fi 是 $(sqrt(5)-1)/2$ 也就是黄金分割率。于是验证威佐夫博弈是否必胜，只需要先看两堆的差值，然后除以 fi 看是不是 a 就可以了。代码中 0 代表必败， 1 代表必胜。

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const double fi=(sqrt(5)-1)/2;
ll a,b;
int main() {
    scanf("%lld %lld",&a,&b);
    if(a>b) swap(a,b);
    if((int)((b-a)*1.0/fi)!=a) puts("1");
    else puts("0");
    return 0;
}
```

这里引申出一个贝蒂定理：如果 a,b 是正无理数，且 ${\frac 1 a}+{\frac 1 b}=1$ 则 $(int)[na]$ 和

$(int)[nb]$ 可以取遍所有的正整数。这个是高中数竞的内容这里不证明了。在这里 $a={\frac 1 {fi}},b={\frac 1 {fi^{2}}}$ 所以包含了所有的必败解。

对称性构造

现在有 n 个硬币围成一圈，每个硬币放在一个凹槽里，操作是每次可以选 1 到 k 个连续硬币取走。取完的赢，问先后手谁赢。

分类讨论：

1. $k≥n$ 先手必胜，直接全部取走就可以了。

2. $k==1$ 根据 n 的奇偶性 n 奇先手赢，偶后手赢

3. 其余的 k 都是后手赢。

关于第三种情况，为什么？

先手显然不会一下子取走一大半，这时后手可以把剩下的都取走。

那么不妨设先手取走一小部分，然后后手只需要在剩下圆弧的中间取走一个或两个（根据剩余数量的奇偶性而定），然后剩余两个部分是完全相同的就可以了，这时两边是对称的，先手怎么做，后手在另一堆怎么做就可以了，于是后手必胜，证毕。

二分图博弈

两个人在二分图上玩游戏，每次一个人沿边移动棋子，棋子待过的点不能再走，不能走的输，问谁赢。

结论：考虑二分图最大匹配，如果最大匹配一定包含起点，先手胜，此外后手胜利。然后只需要算一下包含和不包含这个点的最大匹配相不相等即可。

然后如果最大匹配一定包含这个点，对于先手，沿着最大匹配走，如果后手能走到另一个点结束，先手走不动，说明那个点完全可以代替这个点，则这个点并不是最大匹配所必须包含的，然后后手走完先手必然还能走，如果这个点没有匹配的，也说明起点不是最大匹配所必须包含的，也矛盾，所以走到的这个点必然可以进行匹配，则先手可以继续走，周而复始，后手早晚会走不动，所以先手必胜，证毕。

反之，如果可以不包含这个点，这个点走出去的点，一定在最大匹配中，不然加上这两个点最大匹配更大了，矛盾，则由上，后手必胜，所以先手必败，证毕。

斐波那契博弈

一堆石子有 n 个，两人轮流取，先取者第一次可以取任意多个，但是不能取完，以后每次取的石子数不能超过上次取石子数的二倍，取完者赢。

结论：先手胜当且仅当 n 不是斐波那契数。

因为 $i=2$ 时，先手只能取 1 颗，显然必败，结论成立。

假设当取到斐波那契的第 k 项都已经成立：

当为 $k+1$ 项时，可以拆分为 $f_{k}+f_{k-1}$

如果拿走的石子数大于等于 f_{k-1} 则可以直接拿走剩下的，因为 $f_{k}≤2f_{k-1}$

不然，我就先玩完 f_{k-1} 的游戏，由归纳法知，一定是我结束了 f_{k-1} 的游戏，所以接下来剩 f_{k} 又是你先玩，并且，我这一下最多拿走 ${\frac 2 3}f_{k-1}$ 所以你最多拿 ${\frac 4 3}f_{k-1}f_{k}$ 这个可以由归纳法证明，所以你一次拿不完，由归纳法知你输了，所以 $k+1$ 的情况也递归出来了，所以得证。

不是斐波那契数列的情况由某定理可知，我可以将其分为若干个斐波那契项之和，每次取尽量大的。因为每次都取尽可能大的，所以这些斐波那契项必不相邻，所以相邻的倍数差两倍以上。然后先手取完这个最小的斐波那契数那么多。后手显然只能在倒数第二堆里取，且取不完，然后根据上面的结论，我一定可以取走这一堆中的最后一个石子，然后你又要面对下一堆…，这样每一堆的最后一个石子都是我取走的，所以最后也是我取走的，所以先手必胜，得证。

代码练习

1.2020ACM-ICPC澳门区域赛G

题目大意：给定一个长度为 $n,1≤n≤200000$ 的数组和 $m,1≤m≤200000$ 个操作，每个数的范围为 $[0,255]$ 。

操作 1 是在数组末尾添加一个数，操作 2 是再给一个位置，问从这个位置按照规则两人博弈谁能获得胜利。

博弈规则是，每次只能选下标比当前数大的数字并且这个数字的二进制要和原来数的二进制不同的位数不能超过 1 ，跳到那个位置，谁不能跳谁输。

注意到数字较小，我们可以对值进行操作。又发现对于出现了很多次的同一个数字，如果我们当前所在位置不是这个数最后一次出现的位置，则这个位置是必胜态，因为如果这个数出现的最后一个位置是必败态，则我们可以跳到这个位置，于是现在这个位置必胜。如果最后那个位置是必胜态，那说明后面必存在必败态，我们只需要到达那个必败态就可以了。

综上，如果不是这个值最后一次出现的位置，都是必胜态。

然后对于这个值最后一次出现的位置，我们看看他能到哪些值，这些值都是可以预处理的，然后因为上面的结论，我们不可能选择去能到达值的非最终位置的，因为相对于我来说是必败的，所以我们也是去这些值的最后位置，所以我们只需要记录每个值最后一次出现的位置就可以了。

然后对于查询的位置，先看是不是最后一个位置，不是的话直接输出先手必胜。是最后一个位置就 dfs 一下能到达哪些位置，然后记忆化搜索一下，看看有没有必败态就可以了。

```
const int maxn=400100,maxm=260;
vector<int> nxt[260];
int n,m;
int las[maxm],a[maxn],vis[maxm];
bool ok[maxm];

bool dfs(int pos) {
    if(vis[a[pos]]) return ok[a[pos]];
    vis[a[pos]]=true;
    bool flag=false;
    for(int i=0;i<nxt[a[pos]].size();i++) {
        if(las[nxt[a[pos]][i]]<las[a[pos]]) continue;
```

```
        flag|=(!dfs(las[nxt[a[pos]][i]]));
    }
    return ok[a[pos]]=flag;
}

int main() {
    scanf("%d %d",&n,&m);
    for(int i=0;i<=255;i++) {
        for(int j=0;j<8;j++) {
            nxt[i].push_back(i^(1<<j));
        }
    }
    for(int i=1;i<=n;i++) {
        scanf("%d",&a[i]);
        las[a[i]]=i;
    }
    for(int i=1,op,key;i<=m;i++) {
        scanf("%d %d",&op,&key);
        if(op==1) {
            a[++n]=key;
            las[a[n]]=n;
        }else {
            if(las[a[key]]!=key) {
                puts("Grammy");
                continue;
            }
            memset(vis,0,sizeof(vis));
            memset(ok,0,sizeof(ok));
            bool flag=dfs(key);
            printf("%s\n",flag?"Grammy":"Alice");
        }
    }
    return 0;
}
```

1.2021牛客1A

题目大意：给定两堆石子，有两个人，每次可以在两堆中分别取 k 和 $s\times k$ 个石子，其中 $k\square 0,s\geq 0$ $\square$ 谁先取完谁赢，问先后手谁必胜。石子数量都不超过 5000 。

这个数据范围大概率是筛出来的，仿照前面的威佐夫博弈和线性筛思想打个表就好了，对于必败态，能到达必败态的都是必胜态。

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

const int maxn=5000;
bool dp[5010][5010];
```

```
void get_ab() {
    for(int i=0;i<=maxn;i++) {
        for(int j=0;j<=maxn;j++) {
            if(!dp[i][j]) {
                for(int k=1;k<=maxn;k++) {
                    if(i+k>maxn&&j+k>maxn) break;
                    for(int s=0;s<=maxn;s++) {
                        if(i+k*s>maxn&&j+k*s>maxn) break;
                        if(i+k*s<=maxn&&j+k<=maxn) dp[i+k*s][j+k]=true;
                        if(i+k<=maxn&&j+k*s<=maxn) dp[i+k][j+k*s]=true;
                    }
                }
            }
        }
    }
}

int main() {
    get_ab();
    int t,a,b;
    scanf("%d",&t);
    while(t--) {
        scanf("%d %d",&a,&b);
        if(dp[a][b]) puts("Alice");
        else puts("Bob");
    }
    return 0;
}
```

ps:这玩意本地跑得太慢了，还是牛客的机器好用 $\times$

From:
https://wiki.cvbbacm.com/ - **CVBB ACM Team**

Permanent link:
https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:%E7%8E%8B%E6%99%BA%E5%BD%AA:%E5%8D%9A%E5%BC%88%E8%AE%BA&rev=1629015025

Last update: **2021/08/15 16:10**