目录

[有向图博弈 1](#_Toc24104628)

[Multi-SG 3](#_Toc24104629)

[Hackenbush博弈 4](#_Toc24104630)

[Anti-SG 5](#_Toc24104631)

[Every-SG 6](#_Toc24104632)

[斐波那契博弈 6](#_Toc24104633)

[Nim积 6](#_Toc24104634)

## 有向图博弈

**例1**

/\*

题意：

n点m边有向图，不能走者负。

A偏好:平局>赢>输

B偏好:赢>输>平局

分两行，分别输出n个位置A/B先走时,先手按上述偏好最终的结果

做法：

1、按此通用模版，初始全是平局，从deg为0的开始转移。按上述偏好顺序转移即可。

2、A能必平局一定平，B能必赢一定赢。A不能必定平局时，B一定阻止其平，因为是其最坏结果。

B不能必赢，A一定阻止其赢，亦然。

故A不能必平局，则必赢。B不能必赢，则必输。

拓扑排序标记点。

\*/

vector<int>edg[MAX];

const int lim=2e7;

int n,m,top,tail;

int dp[MAX][2],que[(int)2e7+5],deg[MAX],cnt[MAX][2][3];

inline int mk\_st(int lo,int id,int pre\_st,int now\_st){

return now\_st|(pre\_st<<2)|(id<<4)|(lo<<5);

}

inline void ad(int lo,int id,int st){

int tem=dp[lo][id];

if(tem!=st){

++top;

if(top>lim)top=0;

que[top]=mk\_st(lo,id,tem,st);

dp[lo][id]=st;

}

}

int main(){

freopen("game.in","r",stdin);

freopen("game.out","w",stdout);

read(n);read(m);

for(int i=1;i<=m;i++){

int u,v;read(u);read(v);

++deg[u];edg[v].pb(u);

}

for(int i=1;i<=n;i++){

dp[i][0]=dp[i][1]=1;

cnt[i][0][1]=cnt[i][1][1]=deg[i];

}

for(int i=1;i<=n;i++)

if(!deg[i])

ad(i,0,2),ad(i,1,0);

while(tail!=top){

++tail;

if(tail>lim)tail=0;

int tem=que[tail];

int lo=tem>>5,id=(tem>>4)&1,pre\_st=(tem>>2)&3,now\_st=tem&3;

int pid=id^1;///前一个人

for(int u:edg[lo]){

--cnt[u][pid][pre\_st];

++cnt[u][pid][now\_st];

int val=0;

if(pid==0){

if(cnt[u][pid][0])val=0;

else val=(cnt[u][pid][2]?2:0);

}

else{

if(cnt[u][pid][1])val=1;

else val=(cnt[u][pid][2]?2:0);

}

if(val!=dp[u][pid])ad(u,pid,val);

}

}

for(int i=1;i<=n;i++)

printf("%c",dp[i][1]==2?'W':(dp[i][1]==0?'L':'D'));

puts("");

for(int i=1;i<=n;i++)

printf("%c",dp[i][0]==0?'W':(dp[i][0]==2?'L':'D'));

return 0;

}

**例2**

/\*

题意：

两支队伍在一张n 个点m 条边的有向图上玩游戏，6 个人

轮流移动棋子行动一步，不能移动的人所属队伍输。

正常人希望赢，不能赢则希望平局；而演员希望输，不能输

则希望平局。

对于每个点计算棋子放在该点开始游戏时最后的游戏结果。

做法：

状态：fi;j 表示棋子位于i，目前轮到第j 个人行动时最终的

游戏结果。

定义-1 表示A 胜，0 表示平局，1 表示B 胜，则每个状

态的转移是min 或者max。

一开始假设所有状态的值都是0，并维护出cnt 数组表示每

个状态后继状态中有几个-1; 0; 1，用于O(1) 计算一个状态的值。

对于所有不能行动的状态，重新设定它的状态值，并加入队列。

用SPFA 的方式依次松弛每个值发生改变的状态的前驱状态，并更新cnt 数组。

每个状态的值只会在-1; 0; 1 之间切换常数次。

时间复杂度O(n + m)。

（实现中上述-1,0,1对应于 0,1,2）

\*/

const double eps=1e-8;

const int lim=(1<<23);

int t,n,m,top,pos;

int deg[MAX],who[MAX],que[(int)2e7+5],bel[MAX];

vector<int>edg[MAX];

char a[20];

int dp[MAX][8],cnt[MAX][8][4];

inline int mk\_st(int lo,int id,int tem,int st){

return st|(tem<<2)|(id<<4)|(lo<<7);

}

inline void ad(int lo,int id,int st){

int tem=dp[lo][id];///之前转移的状态

if(st!=tem){

++top;

if(top>lim)top=0;

que[top]=mk\_st(lo,id,tem,st);

dp[lo][id]=st;

}

}

int main(){

read(t);

while(t--){

read(n);read(m);

pos=top=0;

for(int u,v,i=1;i<=m;i++){

read(u);read(v);

++deg[u];

edg[v].pb(u);

}

scanf("%s",a);///sequence

for(int i=0;i<6;i++)

who[i]=(a[i]=='A'),bel[i]=who[i];

scanf("%s",a);///actor

for(int i=0;i<6;i++)

who[i]^=(a[i]=='1');

for(int i=1;i<=n;i++){

for(int j=0;j<6;j++){

dp[i][j]=1;

for(int k=0;k<3;k++)cnt[i][j][k]=0;

cnt[i][j][1]=deg[i];

}

}

for(int i=1;i<=n;i++)

if(!deg[i])

for(int j=0;j<6;j++)

ad(i,j,2\*bel[j]);

while(pos!=top){

++pos;if(pos>lim)pos=0;

int st=que[pos];

int lo=st>>7,id=(st>>4)&7,pre=(st>>2)&3,now=st&3;

int pid=id-1;///前一个人

if(pid<0)pid+=6;

for(int u:edg[lo]){

--cnt[u][pid][pre];

++cnt[u][pid][now];

int val=0;

if(who[pid]){///是A

if(cnt[u][pid][0])val=0;///能转移到0状态

else val=(cnt[u][pid][1]?1:2);

}

else{

if(cnt[u][pid][2])val=2;

else val=(cnt[u][pid][1]?1:0);

}

if(val!=dp[u][pid])

ad(u,pid,val);

}

}

for(int i=1;i<=n;i++){

char ans=(dp[i][0]==0?'A':(dp[i][0]==1?'D':'B'));

printf("%c",ans);

}

puts("");

for(int i=1;i<=n;i++){

deg[i]=0;

edg[i].clear();

}

}

return 0;

}

## Multi-SG

/\*题意

给出n堆石子, 每次可以选择将大于某个数f一堆平均分成多个堆, 最后不能操作的失败。

\*/

/\*做法

考虑每堆是单独的子游戏，互不影响，所以我只要对每堆做完，然后把每堆的SG给XOR起来就可以了。

而每堆的结果也只跟堆的石子数量有关，那么我就可以对于每种石子数量做好了。

而对于一种特定的石子数量的SG，我肯定是把他所有的后继状态的SG给XOR起来，那么枚举一下分成了多少堆，就可以直接算了。复杂度是O(石子数^2)。

进一步观察，发现若干个数的分出来的较小的那个块是相同的，而且这一段最多最多也只能贡献2种不同的SG值，所以就可以每块丢到一起做，算算2种不同的SG值，就可以了。

数论分块 O(n^3/2)

\*/

const int MAXN = 100011;

int n,F,SG[MAXN],ans,Tim,vis[MAXN];

inline int getint(){

int w=0,q=0; char c=getchar(); while((c<'0'||c>'9') && c!='-') c=getchar();

if(c=='-') q=1,c=getchar(); while (c>='0'&&c<='9') w=w\*10+c-'0',c=getchar(); return q?-w:w;

}

inline int calc(int x){

Tim++; int xiao,num,num2,now,nex;

for(int i=2;i<=x;i=nex+1) {//枚举分成的堆的数目

xiao=x/i;//数量较小的堆有多少个石子

num2=x%i;//注意计算方法

num=i-num2;

now=0;

//只与i奇偶性有关，那么只可能贡献两种SG值

if(num&1) now^=SG[xiao];

if(num2&1) now^=SG[xiao+1];

vis[now]=Tim;

nex=min(x/xiao,x);

if(i+1<=nex) {

now=0;

num2=x%(i+1);

num=(i+1)-num2;

if(num&1) now^=SG[xiao];

if(num2&1) now^=SG[xiao+1];

vis[now]=Tim;

}

}

int mex=0;

for(;vis[mex]==Tim;mex++) ;

return mex;

}

inline void init(){

for(int i=0;i<F;i++) SG[i]=0;

for(int i=F;i<=100000;i++)

SG[i]=calc(i);

}

inline void work(){

int T=getint(); F=getint(); int x;

init();

while(T--) {

ans=0; n=getint();

for(int i=1;i<=n;i++) x=getint(),ans^=SG[x];

if(T!=0) printf("%d ",ans==0?0:1);//不能直接输出SG值啊...

else printf("%d",ans==0?0:1);

}

}

int main()

{

work();

return 0;

}

## Hackenbush博弈

/\*

题意：

给定一棵 n 个节点的带权有根树，两人轮流操作，每次操作使某条边的边权减一，

若边权减至 0，则删去改边，并删去不含根的连通块，

判断先手是否必胜，并输出先手若要胜利，所有第一步可能的决策。

n,w <=1e6

做法:

考虑将题目转化为有根无向图删边游戏 (Hackenbush)，

每条树边的边权代表了无向图上两端点之间的边数量。

注意到对于边权大于1 时会形成环，按奇偶性缩环（无向图删边游戏的操作）即可。

在计算出 SG 值后，每一步操作会改动一条到根路径上的所有 SG 值，

自根向下确定每个子树如果修改那么需要修改得到的 SG 值是多少，即可检查某一种操作是否能使得先手必胜。

\*/

vector<int>edg[MAX];

int fa[MAX],w[MAX],sg[MAX];

int T,n,cnt;

int an[MAX];

void dfs(int now){///算原图的sg

sg[now]=0;

for(int v:edg[now]){

dfs(v);

if(w[v]==1)sg[now]^=sg[v]+1;

else{

sg[now]^=sg[v];

if(w[v]&1)sg[now]^=1;

}

}

}

void dfs(int now,int target){

if(target<0)return;

for(int v:edg[now]){

if(w[v]==1){///1->0

if((sg[now]^(sg[v]+1))==target)an[++cnt]=v;

}

else if(w[v]==2){///2->1

if((sg[now]^(sg[v])^(sg[v]+1)) ==target)an[++cnt]=v;

}

else{/// ? -> >1的状态

if((sg[now]^1)==target)an[++cnt]=v;

}

///去掉其余子树的影响更新target

if(w[v]==1)dfs(v,(target^(sg[now]^(sg[v]+1))) -1);

else dfs(v,target^(sg[now]^sg[v]));

}

}

int main(){

read(T);

while(T--){

read(n);

for(int i=1;i<=n;i++)edg[i].clear();

for(int u,i=2;i<=n;i++){

read(u);read(w[i]);

edg[u].pb(i);

}

dfs(1);

if(!sg[1])puts("0\n");

else{

cnt=0;

dfs(1,0);

sort(an+1,an+1+cnt);

printf("%d\n",cnt);

for(int i=1;i<=cnt;i++)printf("%d%c",an[i]," \n"[i==cnt]);

}

}

return 0;

}

## Anti-SG

antiSG的定义是这样的：

1、决策集合为空的游戏者赢

2、其余规则与SG游戏相同

SJ定理（亦即csy模版里的Anti-SG Game）

建立在：规定当局面中所有的单一游戏的SG值为0时，游戏结束。

（也可以替换成：当局面中所有的单一游戏的SG值为0时，存在一个单一游戏它的SG函数能通过一次操作变为1）

“因为笔者发现这样可以出题，我们可以将题目模型设成这样：游戏中存在一个按钮，游戏双方都可以触动按钮，当其中一个人触动按钮时，触动按钮的人每次必须移动对方上次移动的棋子。如果触动按钮的人能保证他能够使得对方无路可走，那么他同样获胜！”

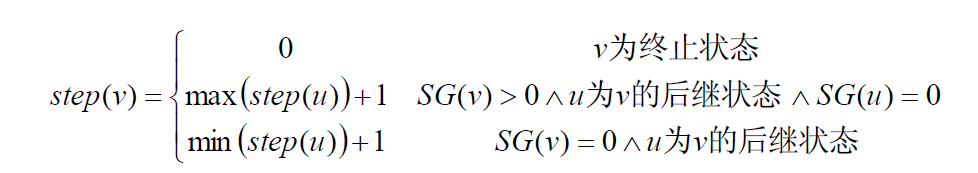
## Every-SG

**定义：**

Every-SG 游戏规定，对于还没有结束的单一游戏，游戏者必须对该游戏进行一步决策；

Every-SG 游戏的其他规则与普通SG 游戏相同

在通过拓扑关系计算某一个状态点的SG 函数时（事实上，我们只需要计算该状态点的SG 值是否为0），对于SG 值为0 的点，我们需要知道最快几步能将游戏带入终止状态，对于SG 值不为0 的点，我们需要知道最慢几步游戏会被带入终止状态，我们用函数来表示这个值。



**[定理]** 对于Every-SG 游戏先手必胜当且仅当单一游戏中最大的为奇数。

## 斐波那契博弈

有一堆石子，两个顶尖聪明的人玩游戏，先取者可以取走任意多个，但不能全取完，以后每人取的石子数不能超过上个人的两倍

结论：先手必败，当且仅当石子数为斐波那契数

## Nim积

我们对于一些二维 Nim游戏（好像更高维也行），可以拆分成两维单独的 Nim 然后求 Nim 积。

定义为

x⊗y=mex{(a⊗b)⊕(a⊗y)⊕(x⊗b),0≤a<x,0≤b<y}

其中 ⊗ 定义为 NimNim 积，⊕ 定义为异或。

**运算性质：**

x⊗0=0⊗x=0

x⊗1=1⊗x=x

x⊗y=y⊗x

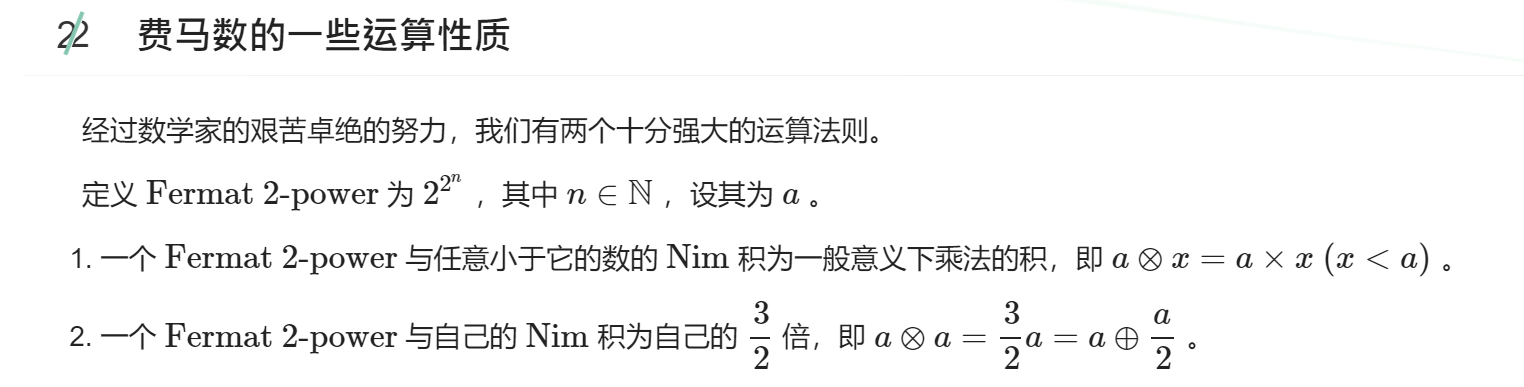
即 0 与所有数做 Nim 积仍然为 0 ， 1仍然是单位元，并且满足交换律。

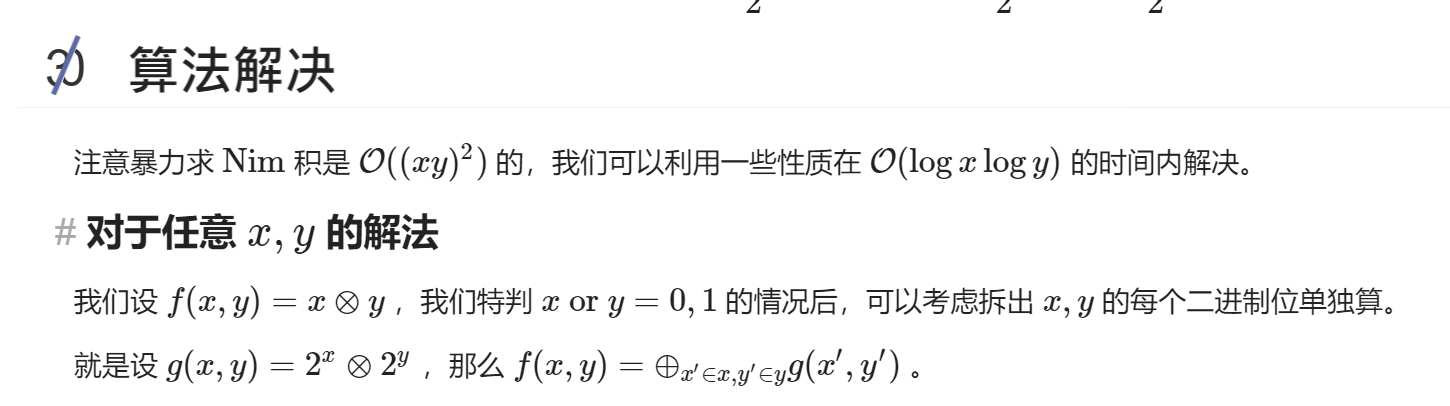
不会证明的两个结论：

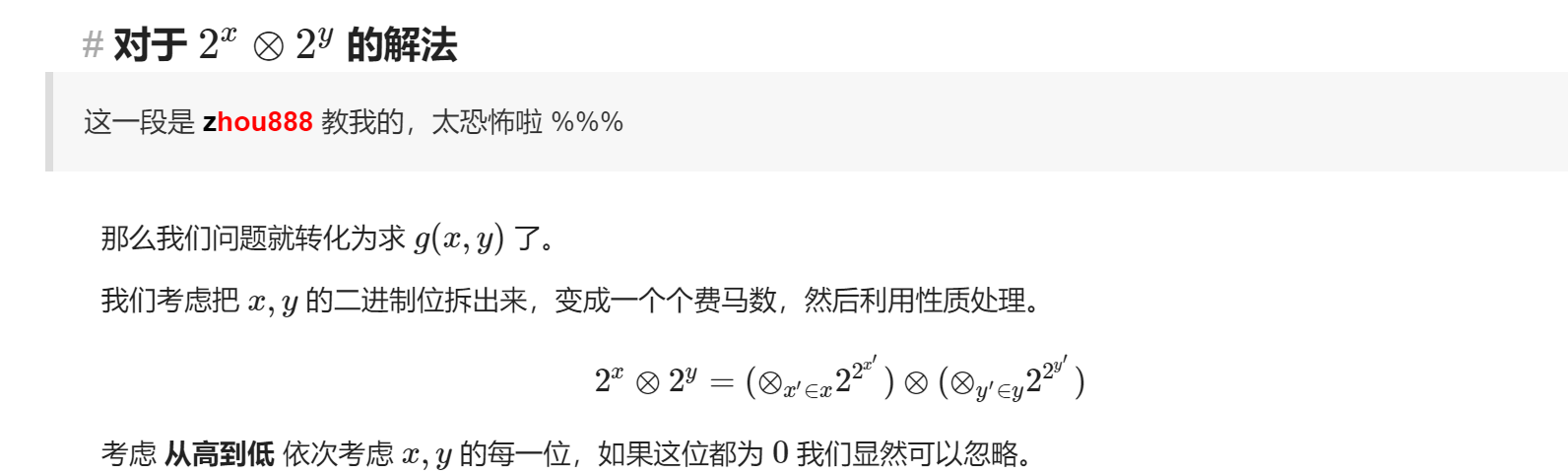
x⊗(y⊗z) =(x⊗y)⊗z

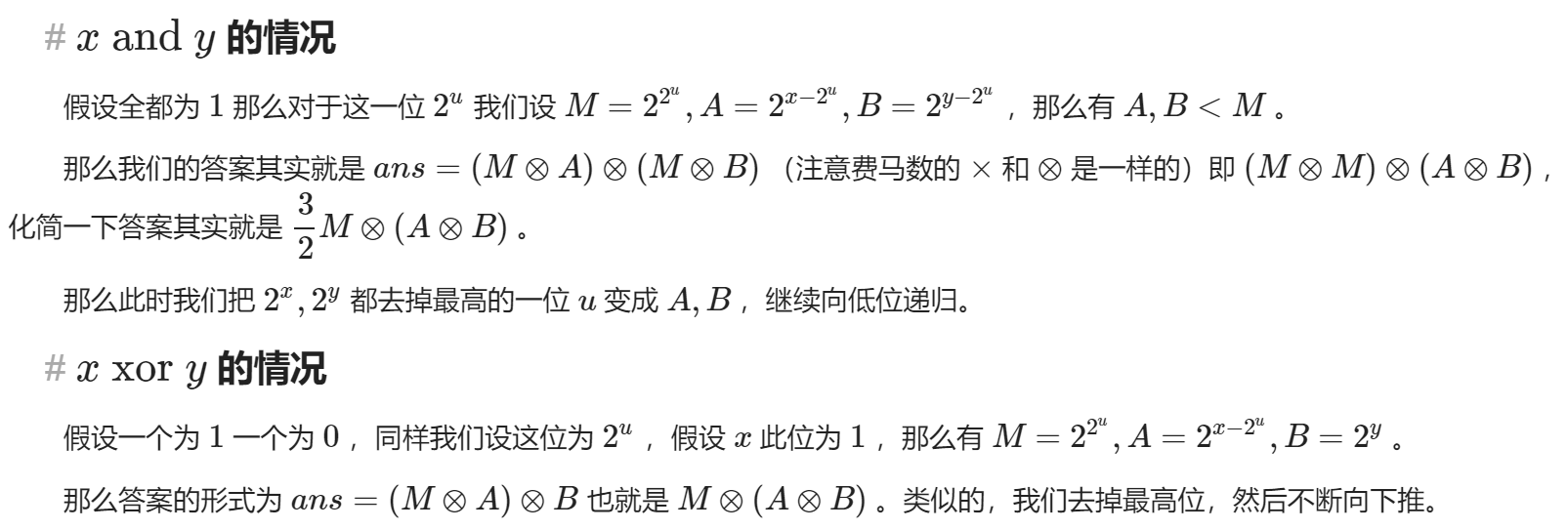
x⊗(y⊕z) =(x⊗y)⊕(x⊗z)

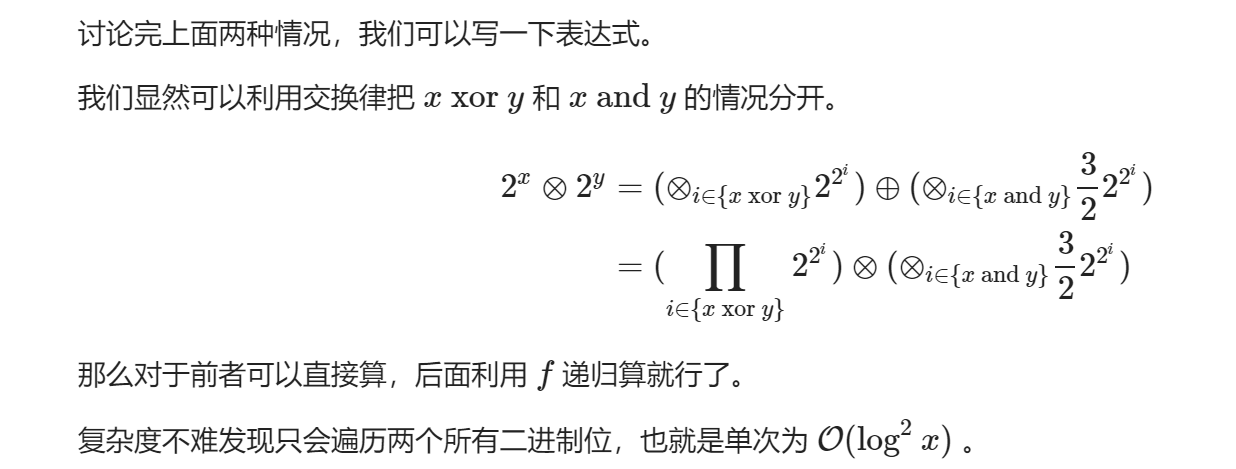
就是说满足乘法交换律，和乘法分配率（把 ⊗看作 × 以及 ⊕ 看做 + ）。











/\*

在一个二维平面中，有 n 个灯亮着并告诉你坐标，每回合需要找到一个矩形，

这个矩形 (x,y) 坐标最大的那个角落的点必须是亮着的灯，然后我们把四个角落的灯状态反转，不能操作为败。

Turning Corners 是裸的二维 Nim 问题，直接上模板就好了。

复杂度是 O(Tnlogxlogy) 的。

\*/

#define Resolve(i, x) for (int u = (x), i = 0; (1ll << i) <= u; ++ i) if (u >> i & 1)

ll f(ll x, ll y);

ll g(int x, int y) {

if (!x || !y) return 1ll << (x | y);

if (~ tab[x][y]) return tab[x][y];

ll res = 1;

Resolve(i, x ^ y) res <<= (1 << i);

Resolve(i, x & y) res = f(res, 3ll << ((1 << i) - 1));

return tab[x][y] = res;

}

ll f(ll x, ll y) {

if (!x || !y) return x | y;

if (x == 1 || y == 1) return max(x, y);

ll res = 0;

Resolve(i, x) Resolve(j, y) res ^= g(i, j);

return res;

}