A. MaPa游戏

关键点(i,x)：做完第i次操作后当前值为x。只需要考虑满足$a_i+x<0$的关键点，维护倍增。单组数据时间复杂度：$\mathcal{O}((n+\sum|a_i|)logn)$。

参考代码：

```
#include "bits/stdc++.h"
#define rep(i, a, n) for (auto i = a; i <= (n); ++i)

#define all(a) a.begin(), a.end()
#define sz(a) (int)(a).size()
template<class T> inline bool chmax(T &a, T b) { if (a < b) { a = b; return 1; }
return 0; }
template<class T> inline bool chmin(T &a, T b) { if (b < a) { a = b; return 1; }
return 0; }
using namespace std;

using ll = long long;
using pii = pair<int, int>;
using vi = vector<int>;
using vvi = vector<vi>;


int main() {
    ios::sync_with_stdio(0); cin.tie(0);

    int cas; cin >> cas; while (cas--) {
        int n, q; cin >> n >> q;
        vi as(n);
        for (auto &x: as) cin >> x;
        int bsz = sqrt(n) + 0.5, btot = n / bsz + 1;
        vi ls(btot, n), rs(btot, 0);
        rep(i, 0, n - 1) {
            int bid = i / bsz;
            chmin(ls[bid], i);
            chmax(rs[bid], i);
        }
        vvi pres(btot);
        rep(bid, 0, btot - 1) {
            int S = 0;
            rep(i, ls[bid], rs[bid]) if (as[i] < 0) S -= as[i];
            vi &pre = pres[bid];
            pre.resize(S + 1);
            rep(x, 0, S) {
                int now = x;
                rep(i, ls[bid], rs[bid]) {
                    int a = as[i];
                    if (now + a >= 0) now += a;
                }
                pre[x] = now;
            }
        }
```

```
        while (q--) {
            int ql, qr, x; cin >> ql >> qr >> x;
            qr--;
            int L = ql / bsz, R = qr / bsz;
            rep(bid, L, R) {
                if (ql <= ls[bid] && rs[bid] <= qr) {
                    auto &pre = pres[bid];
                    int S = sz(pre) - 1;
                    if (x <= S) x = pre[x];
                    else x = pre[S] + x - S;
                } else {
                    rep(i, max(ql, ls[bid]), min(qr, rs[bid])) {
                        if (x + as[i] >= 0) x += as[i];
                    }
                }
            }
            printf("%d\n", x);
        }
    }
    return 0;
}
```

B. 快速冒泡排序

假设$P = n_1\lambda_1 n_2\lambda_2 \cdots n_k\lambda_1 k$，则$B(P) = \lambda_1 n_1\lambda_2 n_2 \cdots \lambda_k n_k$，其中$n_1, \cdots, n_k$为从左到右的局部最大值且有$n_1 < n_2 < \cdots < n_k$，则不难证明答案为非空$\lambda_i$的个数。

将询问离线，每次从n到1倒序扫描左端点l并回答所有左端点为l的询问。对于每个固定的左端点l，$[l, n]$中从左到右的局部最大值可以通过单调栈维护，局部最大值插入/删除对于答案的影响可以用树状数组/线段树快速更新/求解。单组数据时间复杂度为$\mathcal{O}((n+q)logn)$。

参考代码：

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

#define int long long
#define rep(i,n) for (int i=0;i<(int)(n);++i)
#define endl '\n'

#define pb push_back
#define F first
#define S second

typedef pair<int, int> pii;

const int N=100017;

int n,q;
int p[N];
int ans[N],C[N],now[N];
vector<pii> info[N];
int st[N];

int lowbit(int u){return u&(-u);}
```

```
void update(int u,int w){for (int i=u;i<=n+7;i+=lowbit(i)) C[i]+=w;}
int query(int u){int res=0; for (int i=u;i;i-=lowbit(i)) res+=C[i]; return res;}
signed main(){
  ios::sync_with_stdio(false), cin.tie(0), cout.tie(0);
  int _;
  cin>>_;
  while (_--){
    cin>>n>>q;
    rep(i,n) cin>>p[i];
    for (int i=1;i<=n+5;++i) C[i]=0,now[i]=0;
    rep(i,N) info[i].clear();
    p[n]=n+1;
    rep(i,q){
      int u,v;
      cin>>u>>v;
      u--, v--;
      info[u].pb({i,v});
    }
    int cnt=0;
    st[cnt++]=n;
    auto upd=[&](int u){
      update(u+1,(now[u+1]?-1:1)), now[u+1]^=1;
      update(u+2,(now[u+2]?-1:1)), now[u+2]^=1;
    };
    for (int i=n-1;i>-1;--i){
      while (cnt&&p[i]>p[st[cnt-1]]) {
        cnt--; upd(st[cnt]);
      }
      st[cnt++]=i, upd(i);
      for (auto c:info[i]){
        int id=c.F, r=c.S;
        if (i==r) ans[id]=0;
        else ans[id]=(query(r+1)-query(i))/2;
      }
    }
    rep(i,q) cout<<ans[i]<<"\n";
  }
  return 0;
}
```

C. 和加积

可以观察到$\prod_{i=1}^{n}(a_i+1)$是一个不变量，因此答案是$\prod_{i=1}^{n}(a_i+1)-1$。时间复杂度为$\mathcal{O}(n)$。

参考代码：

```
#include<bits/stdc++.h>

#define MAXN 100005
#define MOD 998244353

using namespace std;

int T,n,a[MAXN];
```

```
void dec(int &a,int b) {a-=b; if(a<0) a+=MOD;}
int main()
{
    scanf("%d",&T);
    while(T--)
    {
        scanf("%d",&n);
        int ans=1;
        for(int i=1;i<=n;i++)
        {
            scanf("%d",&a[i]); ans=1LL*ans*(a[i]+1)%MOD;
        }
        dec(ans,1);
        printf("%d\n",ans);
    }
    return 0;
}
```

D. 网络画家

求期望的正方形个数等价于求所有方案下的正方形个数之和。可以先枚举形成的正方形边长，有$\sqrt{nm}$种，之后再枚举四个角的小正方形种类，有$4^4=256$种。之后可以将剩余的小正方形分为三类：能拼竖边的，能拼横边的，以及都能拼的（即十字）。可以考虑枚举大正方形的竖边中有几个是十字小方块，然后用剩余的十字小方块和只能拼横边的小方块去拼大正方形的横边。用组合数计算答案即可这个可能的十字小方块个数也是$\sqrt{nm}$种。

时间复杂度：$\mathcal{O}(256nm)$，空间复杂度：$\mathcal{O}(nm\sqrt{nm})$或$\mathcal{O}(nm)$。

参考代码：

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

typedef long long LL;
typedef pair<int, int> PII;
#define PB push_back
#define fi first
#define se second
#define MP make_pair
const int oo = 1000000000;
const int P = 1000000007;
const int N = 320;
const int V = N + 10;

int Pow(int x, int y) {
    int res = 1;
    while (y) {
        if (y & 1) res = (LL) res * x % P;
        x = (LL) x * x % P; y /= 2;
    }
    return res;
}

const int maskU = 8;
const int maskR = 4;
```

```
const int maskD = 2;
const int maskL = 1;
vector<int> LU, RU, RD, LD, VEC, HOR;
int C[V * V][V * 2];
int fac[V * V], pt[V * V];

int _, n, m, a[30], b[30], c[30];
int main() {
    scanf("%d", &_);
    for (int i = 0; i < (1 << 4); ++i) {
        bool hasU = (i & maskU) != 0;
        bool hasR = (i & maskR) != 0;
        bool hasD = (i & maskD) != 0;
        bool hasL = (i & maskL) != 0;
        if (hasL && hasU) LU.PB(i);
        if (hasR && hasU) RU.PB(i);
        if (hasR && hasD) RD.PB(i);
        if (hasL && hasD) LD.PB(i);
        if (hasU && hasD) VEC.PB(i);
        if (hasL && hasR) HOR.PB(i);
    }
    for (int i = 0; i <= N * N; ++i) {
        C[i][0] = 1;
        if (i <= 2 * N) C[i][i] = 1;
        for (int j = 1; j < min(2 * N + 1, i); ++j) C[i][j] = (C[i - 1][j - 1] +
C[i - 1][j]) % P;
    }
    fac[0] = 1;
    for (int i = 1; i <= N * N; ++i) fac[i] = (LL) fac[i - 1] * i % P;
    pt[0] = 1;
    for (int i = 1; i <= N * N; ++i) pt[i] = pt[i - 1] * 2 % P;
    for (int ca = 1; ca <= _; ++ca) {
        scanf("%d%d", &n, &m);
        for (int i = 0; i < (1 << 4); ++i) scanf("%d", &a[i]);
        int ans = 0;
        for (int lu = 0; lu < LU.size(); ++lu) {
            for (int ru = 0; ru < RU.size(); ++ru) {
                for (int rd = 0; rd < RD.size(); ++rd) {
                    for (int ld = 0; ld < LD.size(); ++ld) {
                        int xs = 1;
                        for (int i = 0; i < 16; ++i) c[i] = 0;
                        ++c[LU[lu]];
                        ++c[RU[ru]];
                        ++c[RD[rd]];
                        ++c[LD[ld]];
                        for (int i = 0; i < 16; ++i) if (a[i] < c[i]) xs = 0;
                        if (xs == 0) continue;
                        for (int i = 0; i < 16; ++i) {
                            xs = (LL) xs * C[a[i]][c[i]] % P;
                            xs = (LL) xs * fac[c[i]] % P;
                            b[i] = a[i] - c[i];
                        }
                        int hor = 0, vec = 0, cross = b[15];
                        //printf("Cal %d %d %d %d\n", xs, hor, vec, cross);
```

```
                        for (int i = 0; i < HOR.size(); ++i) if (HOR[i] != 15)
hor += b[HOR[i]];
                        for (int i = 0; i < VEC.size(); ++i) if (VEC[i] != 15)
vec += b[VEC[i]];
                        for (int L = 0; L <= min(n, m) - 2; ++L) {
                            int tmp = 0;
                            for (int cro = 0; cro <= 2 * L; ++cro) {
                                int v_cnt = 2 * L - cro;
                                int h_remain = cross - cro + hor;
                                if (v_cnt > vec || h_remain < 2 * L) continue;
                                tmp += (LL) C[cross][cro] * C[vec][v_cnt] % P *
C[h_remain][2 * L] % P;
                                tmp %= P;
                            }
                            tmp = (LL) tmp * fac[2 * L] % P * fac[2 * L] % P;
                            ans += (LL) tmp * fac[n * m - 4 * (L + 1)] % P * (n
- L - 1) % P * (m - L - 1) % P * xs % P;
                            ans %= P;
                        }
                    }
                }
            }
        }
        ans = (LL) ans * Pow(fac[n * m], P - 2) % P;
        printf("%d\n", ans);
    }
    return 0;
}
```

E. 算术序列

首先如果某个数出现次数大于等于3则不存在解。然后如果所有数字均为偶数，我们可将所有数除以二；如果所有数字均为奇数，我们可将所有数减去一；否则，我们将所有奇数放在左边，所有偶数放在右边，对奇数/偶数分治解决。单组数据时间复杂度$\mathcal{O}(nlogmaxA_i)$。

参考代码：

```
#include<bits/stdc++.h>

#define int long long
#define double long double

using namespace std;

#define rep(i,n) for (int i=0;i<(int)(n);++i)
#define rep1(i,n) for (int i=1;i<=(int)(n);++i)
#define range(x) begin(x), end(x)
#define sz(x) (int)(x).size()
#define pb push_back
#define F first
#define S second

typedef long long ll;
typedef unsigned long long ull;
```

```cpp
typedef pair<int, int> pii;
typedef vector<int> vi;

int dx[]={1,-1,0,0};
int dy[]={0,0,1,-1};
const int mod=998244353;
const double EPS=1e-9;
const double pi=acos(-1);
const int INF=1e18;
const int N=5007;

int n;
vi ans;
void solve(vi &x){
    vector<pair<int,int>> y;
    int n=sz(x);
    for (int i=0;i<n;++i){
        y.push_back({0,x[i]});
        for (int j=0;j<30;++j){
            if (x[i]>>j&1) y[i].first|=(1ll<<(29-j));
        }
    }
    sort(range(y));
    for (int i=0;i<n;++i) ans.push_back(y[i].second);
}
signed main(){
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    cout.precision(15);
    int _;
    cin>>_;
    while (_--){
        cin>>n;
        vi a(n,0);
        rep(i,n) cin>>a[i];
        map<int,int> cnt;
        ans.clear();
        rep(i,n){
            cnt[a[i]]++;
            if (cnt[a[i]]>2){cout<<"NO\n"; goto cont;}
        }
        cout<<"YES\n";
        solve(a);
        for (auto c:ans) cout<<c<<" ";
        cout<<"\n";
        cont:;
    }
    return 0;
}
```

F. 函数和

数位DP，用$dp[i][j][0/1][0/1]$表示已经统计了前i位，其中有j个位上是d，是否取数字上界，以及是否有前导零即可。用记忆化搜索维护转移。转移时需分四种情况考虑：

1. 当前位卡住上界，且之前也卡住数字上界。
2. 当前位为0，且目前还在维护前导零。
3. 当前位为d。
4. 以上都不满足。

要注意上述几种情况可能也会合并成一种。用这种讨论的形式能够优化转移的复杂度。最终复杂度为$\mathcal{O}(log_B^2 r)$。

参考代码：

```
#include<bits/stdc++.h>
using namespace std;
#define fi first
#define se second
#define SZ(x) ((int)x.size())
#define lowbit(x) x&-x
#define pb push_back
#define ALL(x) (x).begin(),(x).end()
#define UNI(x) sort(ALL(x)),x.resize(unique(ALL(x))-x.begin())
#define GETPOS(c,x) (lower_bound(ALL(c),x)-c.begin())
#define LEN(x) strlen(x)
#define MSO(x) memset((x),0,sizeof((x)))
#define Rint register int
#define ls (u<<1)
#define rs (u<<1|1)
typedef unsigned int unit;
typedef long long ll;
typedef unsigned long long ull;
typedef double db;
typedef pair<int,int> pii;
typedef pair<ll,ll> pll;
typedef vector<int> Vi;
typedef vector<ll> Vll;
typedef vector<pii> Vpii;
template<class T> void _R(T &x) { cin >> x; }
void _R(int &x) { scanf("%d", &x); }
void _R(ll &x) { scanf("%lld", &x); }
void _R(ull &x) { scanf("%llu", &x); }
void _R(double &x) { scanf("%lf", &x); }
void _R(char &x) { scanf(" %c", &x); }
void _R(char *x) { scanf("%s", x); }
void R() {}
template<class T, class... U> void R(T &head, U &... tail) { _R(head);
R(tail...); }
template<class T> void _W(const T &x) { cout << x; }
void _W(const int &x) { printf("%d", x); }
void _W(const ll &x) { printf("%lld", x); }
void _W(const double &x) { printf("%.16f", x); }
void _W(const char &x) { putchar(x); }
void _W(const char *x) { printf("%s", x); }
template<class T,class U> void _W(const pair<T,U> &x) {_W(x.fi);putchar(' ');
_W(x.se);}
template<class T> void _W(const vector<T> &x) { for (auto i = x.begin(); i !=
x.end(); _W(*i++)) if (i != x.cbegin()) putchar(' '); }
void W() {}
```

```cpp
template<class T, class... U> void W(const T &head, const U &... tail) {
_W(head); putchar(sizeof...(tail) ? ' ' : '\n'); W(tail...); }
const int MOD=1e9+7,mod=998244353;
ll qpow(ll a,ll b) {ll res=1;a%=MOD; assert(b>=0); for(;b;b>>=1)
{if(b&1)res=res*a%MOD;a=a*a%MOD;}return res;}
const int MAXN=5e5+10,MAXM=1e7+10;
const int INF=INT_MAX,SINF=0x3f3f3f3f;
const ll llINF=LLONG_MAX;
const int inv2=(MOD+1)/2;
const int Lim=1<<20;

template <int _P>
struct Modint
{
    static constexpr int P=_P;
private :
    int v;
public :
    Modint() : v(0){}
    Modint(ll _v){v=_v%P;if(v<0)v+=P;}
    explicit operator int() const {return v;}
    explicit operator long long() const {return v;}
    explicit operator bool() const {return v>0;}
    bool operator == (const Modint &o) const {return v==o.v;}
    bool operator != (const Modint &o) const {return v!=o.v;}
    Modint operator - () const {return Modint(v?P-v:0);}
    Modint operator + () const {return *this;}
    Modint & operator ++ (){v++;if(v==P)v=0;return *this;}
    Modint & operator -- (){if(v==0)v=P;v--;return *this;}
    Modint operator ++ (int){Modint r=*this;++*this;return r;}
    Modint operator -- (int){Modint r=*this;--*this;return r;}
    Modint & operator += (const Modint &o){v+=o.v;if(v>=P)v-=P;return *this;}
    Modint operator + (const Modint & o)const{return Modint(*this)+=o;}
    Modint & operator -= (const Modint & o){v-=o.v;if(v<0)v+=P;return *this;}
    Modint operator - (const Modint &o)const {return Modint(*this)-=o;}
    Modint & operator *=(const Modint & o){v=(int)(((ll)v)*o.v%P);return *this;}
    Modint operator * (const Modint & o)const {return Modint(*this)*=o;}
    Modint & operator /= (const Modint & o){return (*this)*=o.Inv();}
    Modint operator / (const Modint & o)const{return Modint(*this)/=o;}
    friend Modint operator + (const Modint &x,const ll &o) {return x+(Modint)o;}
    friend Modint operator + (const ll &o,const Modint &x) {return x+(Modint)o;}
    friend Modint operator - (const Modint &x,const ll &o) {return x-(Modint)o;}
    friend Modint operator - (const ll &o,const Modint &x) {return (Modint)o-x;}
    friend Modint operator * (const Modint &x,const ll &o) {return x*(Modint)o;}
    friend Modint operator * (const ll &o,const Modint &x) {return x*(Modint)o;}
    friend Modint operator / (const Modint &x,const ll &o) {Modint c=o;return
x*c.Inv();}
    friend Modint operator / (const ll &o,const Modint &x) {Modint c=o;return
c*x.Inv();}
    Modint operator ^ (ll o)const{Modint r=1,t=v;while(o)
{if(o&1)r*=t;t*=t;o>>=1;}return r;}
    Modint operator ~ (){return (*this)^(P-2);}
    Modint Inv() const{return (*this)^(P-2);}
};
```

```
using mi=Modint<MOD>;

template<int P>
void _W(Modint<P> x){printf("%d",(int)x);}

template<int P>
void _R(Modint<P> &x){ll t;scanf("%lld",&t);x=t;}

mi dp[75][75][2][2],vis[75][75][2][2];;
ll t;
int s[75],k,b,d,n,m,c;

mi dfs(int dep,int tot,int lim,bool zero)
{
    if(dep==m+1&&tot==0)return 1;
    if(dep==m+1)return 0;
    if(tot<0)return 0;
    if(vis[dep][tot][lim][zero])return dp[dep][tot][lim][zero];
    vis[dep][tot][lim][zero]=1;
    int up=lim?s[dep]:b-1;
    int ct=0,i=0;
    int c=(i==d);
    if(zero&&(d==0))c=0;
    dp[dep][tot][lim][zero]+=dfs(dep+1,tot-c,lim&&(s[dep]==i),zero);
    ct++;
    if(i!=d&&d<=up)
    {
        ct++;
        i=d;
        int c=(i==d);
        if(zero&&(d==0))c=0;
        dp[dep][tot][lim][zero]+=dfs(dep+1,tot-c,lim&&(s[dep]==i),0);
    }
    if(i!=up)
    {
        ct++;
        i=up;
        dp[dep][tot][lim][zero]+=dfs(dep+1,tot,lim&&(s[dep]==i),zero&&(i==0));
    }
    dp[dep][tot][lim][zero]+=dfs(dep+1,tot,0,0)*max(0,up-ct+1);
    return dp[dep][tot][lim][zero];
}


mi calc(bool f)
{
    MSO(dp);MSO(vis);
    R(t);
    m=0;
    t-=f;
    while(t)
    {
        s[++m]=t%b;
        t/=b;
    }
```

```
    reverse(s+1,s+m+1);
    mi ans=0;
    for(int i=1;i<=m;i++)
    {
        mi t=i;
        ans+=dfs(1,i,1,1)*(t^k);
    }
    return ans;
}

void solve()
{
    R(k,b,d);
    mi ans=calc(1);
    ans=calc(0)-ans;
    W(ans);
}

int main()
{
    srand(time(0));
    int T=1;
    scanf("%d",&T);
    for(int kase=1;kase<=T;kase++)
        solve();
    return 0;
}
```

G. 黑魔法

分类讨论，以下的L, R, E, B含义和题目中一样。对于连通块最多的情况，要让能够相连的方块对尽量少。可以发现，所有L可以全放在最左边，所有R可以全放在最右边，这样他们就不会和中间的方块相连，之后尽可能用E把B隔开即可。对于连通块最少的情况，要让能够相连的方块对尽可能多。可以先把所有B拼在一起，之后可以把一个L和一个R分为一组拼起来。注意如果有至少一组LR并且也有至少一块B的话，可以把一组LR拆开拼在B的两边，这样可以再减少一个连通块。

同时存在一种线性DP的解法，在此不多赘述。

时空复杂度均为$\mathcal{O}(1)$。

参考代码：

```
#include <bits/stdc++.h>

using namespace std;

int _, ca, E, L, R, B;
int main() {
    scanf("%d", &_);
    for (int ca = 1; ca <= _; ++ca) {
        scanf("%d%d%d%d", &E, &L, &R, &B);
        int n = E + L + R + B;
        int mi = min(L, R);
        if (max(L, R) > 0) mi += B;
        else mi += max(0, B - 1);
```

```
        mi = n - mi;
        int mx = max(B - E - 1, 0);
        mx = n - mx;
        printf("%d %d\n", mi, mx);
    }
    return 0;
}
```

H. GCD图中的最短路径

首先最短路长度不超过2，因为对任意(x,y)，沿路径$x \to 1 \to y$即可得到一条长度为2的路径。最短路长度为1当且仅当$gcd(x,y)=1$，否则等价于询问$[1,n]$中满足$gcd(x,z)=1$且$gcd(y,z)=1$的z的数量，分解x,y后用容斥可以解决。注意$gcd(x,y)=2$时，直接$x \to y$也是一条长度为2的路径。单组数据时间复杂度: $\mathcal{O}(n+Q\cdot 2^p)$，其中$p \le 12$为$x,y$的不同质因子个数。

参考代码：

```
#include<bits/stdc++.h>
#define pb push_back
using namespace std;
typedef long long ll;
const int maxn=1e7+5;
typedef vector<int> vi;
int prime[maxn/10],f[maxn],n,Q;
bool p[maxn];
void sieve(int n){
    int cnt=0;
    for (int i=2;i<=n;i++){
        if (!p[i]) prime[++cnt]=i,f[i]=i;
        for (int j=1;j<=cnt&&prime[j]*i<=n;j++){
            p[prime[j]*i]=1;
            f[prime[j]*i]=prime[j];
            if (i%prime[j]==0) break;
        }
    }
}
int m,ret;
set<int> S;
vi a;
void pv(int x){
    while (x>1){
        int y=f[x];
        if (S.find(y)==S.end()) S.insert(y);
        while (x%y==0) x/=y;
    }
}
void dfs(int x,int s,int o){
    if (x==m){
        ret+=o*(n/s);
        return;
    }
    dfs(x+1,s,o);
    if (s<=n/a[x]) dfs(x+1,s*a[x],-o);
}
int calc(int x,int y){
```

```
    S.clear(); pv(x); pv(y);
    a.clear(); for (auto x:S) a.pb(x);
    m=a.size(); ret=0;
    dfs(0,1,1);
    return ret;
}
int main(){
    ios::sync_with_stdio(false);
    cin >> n >> Q;
    sieve(n);
    while (Q--){
        int x,y; cin >> x >> y;
        assert(x!=y);
        if (__gcd(x,y)==1){
            cout << 1 << ' ' << 1 << endl;
            continue;
        }
        int ans=calc(x,y);
        if (__gcd(x,y)==2) ans++;
        cout << 2 << ' ' << ans << endl;
    }
    return 0;
}
```

I. 石头树

一个显然的想法是先点分治，求经过分治中心的联通块，只需要对于每个子树求出深度不超过 $i=0,1,\ldots,k-1$ 的最大的包含根的联通块，最后在分治中心合并各子树并计算答案即可。

本题用长链剖分同样可以通过。

参考代码：

```
#include<bits/stdc++.h>
using namespace std;
#define cs const
#define pb push_back
#define ll long long
#define gc getchar
inline int read(){
    char ch=gc();
    int res=0;bool f=1;
    while(!isdigit(ch))f^=ch=='-',ch=gc();
    while(isdigit(ch))res=(res*10)+(ch^48),ch=gc();
    return f?res:-res;
}
template<typename tp>inline bool chemx(tp &a,tp b){return (a<b)?(a=b,1):0;}
cs int N=100005;
int len,a[N];
vector<int> e[N];
struct seg{
    #define lc (u<<1)
    #define rc ((u<<1)|1)
    #define mid ((l+r)>>1)
    int *vl,*tg,*cov,n;
```

```
void init(int _n){
    n=_n;
    vl=new int[(n+1)*4+1];
    tg=new int[(n+1)*4+1];
    cov=new int[(n+1)*4+1];
    for(int i=0;i<=n*4;i++)vl[i]=tg[i]=cov[i]=0;
}
void covernow(int u){
    cov[u]=1,vl[u]=0,tg[u]=0;
}
void pushnow(int u,int k){
    tg[u]+=k,vl[u]+=k;
}
void pushdown(int u){
    if(cov[u]){
        covernow(lc);
        covernow(rc);
        cov[u]=0;
    }
    if(tg[u]){
        pushnow(lc,tg[u]);
        pushnow(rc,tg[u]);
        tg[u]=0;
    }
}
void update(int u,int l,int r,int st,int des,int k){
    if(st<=l&&r<=des)return pushnow(u,k);
    pushdown(u);
    if(st<=mid)update(lc,l,mid,st,des,k);
    if(mid<des)update(rc,mid+1,r,st,des,k);
    vl[u]=vl[rc];
}
void update(int l,int r,int k){
    assert(l>=0);
    assert(l<=r);
    if(l>n)return;
    update(1,0,n,l,min(r,n),k);
}
void cover(int u,int l,int r,int st,int des){
    if(st<=l&&r<=des)return covernow(u);
    pushdown(u);
    if(st<=mid)cover(lc,l,mid,st,des);
    if(mid<des)cover(rc,mid+1,r,st,des);
    vl[u]=vl[rc];
}
int query(int u,int l,int r,int p){
    if(l==r)return vl[u];
    pushdown(u);
    if(p<=mid)return query(lc,l,mid,p);
    else return query(rc,mid+1,r,p);
}
int query(int p){
    assert(p>=0);
    return query(1,0,n,min(p,n));
}
```

```
int find(int u,int l,int r,int st,int k){
    if(l==r){
        if(vl[u]<k)return l+1;
        return l;
    }
    pushdown(u);
    if(mid<st)return find(rc,mid+1,r,st,k);

    if(vl[lc]>=k)return find(lc,l,mid,st,k);
    return find(rc,mid+1,r,st,k);
}
void Chemx(int u,int l,int r,int p,int &k){
    if(l==r){
        if(vl[u]>k)k=vl[u];
        else vl[u]=k;
        return;
    }
    pushdown(u);
    if(p<=mid)Chemx(lc,l,mid,p,k);
    else Chemx(rc,mid+1,r,p,k);
    vl[u]=vl[rc];
}
void Chemx(int p,int &k){
    assert(p>=0);
    if(p>n)return;
    Chemx(1,0,n,p,k);
}
void Setmx(int l,int k){
    if(l>n)return;
    assert(l<=n);
    int ps=find(1,0,n,l,k);
    assert(ps<=n+1);
    if(ps>l)cover(1,0,n,l,ps-1),update(1,0,n,l,ps-1,k);
}
int find2(int u,int l,int r,int st,int des,int k){
    if(l==r){
        if(vl[u]<k)return l+1;
        return l;
    }
    pushdown(u);
    if(mid<st)return find2(rc,mid+1,r,st,des,k);
    if(des<=mid)return find2(lc,l,mid,st,des,k);
    if(vl[lc]>=k)return find2(lc,l,mid,st,des,k);
    return find2(rc,mid+1,r,st,des,k);
}
void Setmx2(int l,int r,int k){
    if(l>n)return;
    if(r>n)r=n;
    if(l>r)return;
    int ps=find2(1,0,n,l,r,k);
    assert(ps<=n+1);
    if(ps>l)cover(1,0,n,l,ps-1),update(1,0,n,l,ps-1,k);
}
void Set(int p,int k){
    assert(p<=n);
```

```
    //  cout<<p<<'\n';
        if(k<0){
            update(1,0,n,0,n,k);
            Setmx(0,0);
        }
        else{
            update(1,0,n,p,n,k);
        }
    }
    void clear(){
        delete[] vl;
        vl=NULL;
        delete[] tg;
        tg=NULL;
        delete[] cov;
        cov=NULL;
    }
    #undef lc
    #undef rc
    #undef mid
}tr[N];


vector<int> f[N];
int maxdep;
int ans;
int siz[N],dep[N],mxdep[N];
int son[N],top[N];

int F(int u,int i){
    if(i<0)return 0;
    return tr[top[u]].query(dep[u]-dep[top[u]]+i);
}

void dfs1(int u,int fa){
    for(int v:e[u])if(v!=fa){
        dep[v]=dep[u]+1,dfs1(v,u);
        if(mxdep[v]>mxdep[son[u]])son[u]=v;
    }
    mxdep[u]=mxdep[son[u]]+1;
}
void dfs2(int u,int fa,int tp){
    top[u]=tp;if(u==tp)tr[u].init(mxdep[u]);
    if(son[u])dfs2(son[u],u,tp);
    for(int v:e[u])if(v!=son[u]&&v!=fa)dfs2(v,u,v);
}
void merge(int u,int v){
    int ps2=max(len-mxdep[v],mxdep[v]+2);
    vector<int> now(mxdep[v]+2);
    for(int i=2;i<=mxdep[v]+1;i++){
        chemx(now[i],F(u,i)+F(v,min(i-1,len-i)));
        chemx(now[i],F(v,i-1)+F(u,min(i,len-i+1)));
    }
    for(int i=2;i<=mxdep[v]+1;i++){
        tr[top[u]].Chemx(dep[u]-dep[top[u]]+i,now[i]);
```

```
    }

    if(mxdep[v]+1+1<ps2){
        tr[top[u]].update(dep[u]-dep[top[u]]+(mxdep[v]+2),dep[u]-dep[top[u]]+
(ps2-1),F(v,mxdep[v]));
    }

    for(int i=ps2;i<=len&&i<=mxdep[u];i++){
        tr[top[u]].update(dep[u]-dep[top[u]]+i,dep[u]-dep[top[u]]+i,F(v,len-i));
    }

    int res=0;
    for(int i=2;i<=mxdep[v]+1;i++)
    if(min(i-1,len-i)>=0){
        tr[top[u]].Chemx(dep[u]-dep[top[u]]+i,res);
    }
    if(mxdep[v]+1+1<ps2){
        tr[top[u]].Setmx2(dep[u]-dep[top[u]]+(mxdep[v]+2),ps2-1,res);
    }
    if(ps2<=mxdep[u]&&ps2<=len){
        if(ps2)res=max(res,F(u,ps2-1));
        for(int i=ps2;i<=len&&i<=mxdep[u];i++)tr[top[u]].Chemx(dep[u]-
dep[top[u]]+i,res);
    }
    tr[top[u]].Setmx(dep[u]-dep[top[u]]+min(len,mxdep[u])+1,res);
    tr[v].clear();

}
void dfs3(int u,int fa){
    if(son[u])dfs3(son[u],u);
    for(int v:e[u])if(v!=fa){
        if(v==son[u])continue;
        dfs3(v,u),merge(u,v);
    }
    tr[top[u]].Set(dep[u]-dep[top[u]]+1,a[u]);
    chemx(ans,F(u,min(len,mxdep[u])));
    chemx(ans,a[u]);
}
void dfs4(int u,int fa){
    siz[u]=dep[u]=mxdep[u]=son[u]=top[u]=0;
    for(int v:e[u])if(v!=fa){
        dfs4(v,u);
    }
}
void dp(int u,int fa){
    dep[u]=1;
    dfs1(u,fa);
    chemx(maxdep,mxdep[u]);
    dfs2(u,fa,u);
    dfs3(u,fa);
    f[u].resize(mxdep[u]+1);
    tr[u].clear();
    dfs4(u,fa);
}
```

```
int n;
int main(){
    int T=read();int tt=clock(),sn=0;;
    while(T--){
        ans=-1e9;
        n=read(),len=read();
        sn+=n;int ok=1;
        for(int i=1;i<=n;i++)a[i]=read(),ok&=(a[i]<0);
        for(int i=1;i<n;i++){
            int u=read(),v=read();
            e[u].pb(v),e[v].pb(u);
        }
        if(ok==1){
            for(int i=1;i<=n;i++)chemx(ans,a[i]);
            cout<<ans<<'\n';
        }
        else{
            dp(1,0);
            cout<<ans<<'\n';
        }
        for(int i=1;i<=n;i++)e[i].clear();
    }
}
```

J. 保龄球

题目可以视作凸多边形不动，点沿着反方向平移。单独研究一个点，可以发现如果要碰到凸多边形，其运动方向应当夹在点与凸多边形的两条切线之间。将两条切线之间的方向范围视作一段区间，那么题目可以转换成给出多个区间，询问一个方向向量被多少区间覆盖。一种处理方式是将所有切线与询问的方向向量离线存储并离散化，然后使用差分处理区间覆盖问题。本题的关键在于求出点到凸多边形的切线，使用二分的方式可以在$\mathcal{O}(logn)$的复杂度求出一个点的切线，总复杂度为$\mathcal{O}(mlogn+q)$。

参考代码：

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

using _T=long long;

constexpr _T eps=1e-8;
constexpr long double PI=3.14159265358979323841;

// 点与向量
template<typename T> struct point
{
    T x,y;

    bool operator==(const point &a) const {return (abs(x-a.x)<=eps && abs(y-a.y)
<=eps);}
    bool operator<(const point &a) const {if (abs(x-a.x)<=eps) return y<a.y-eps;
return x<a.x-eps;}
    bool operator>(const point &a) const {return !(*this<a || *this==a);}
    point operator+(const point &a) const {return {x+a.x,y+a.y};}
    point operator-(const point &a) const {return {x-a.x,y-a.y};}
    point operator-() const {return {-x,-y};}
```

```
    point operator*(const T k) const {return {k*x,k*y};}
    point operator/(const T k) const {return {x/k,y/k};}
    T operator*(const point &a) const {return x*a.x+y*a.y;}  // 点积
    T operator^(const point &a) const {return x*a.y-y*a.x;}  // 叉积，注意优先级
    int toleft(const point &a) const {const auto t=(*this)^a; return (t>eps)-
(t<-eps);}  // to-left 测试
    T len2() const {return (*this)*(*this);}  // 向量长度的平方
    T dis2(const point &a) const {return (a-(*this)).len2();}  // 两点距离的平方
};

using Point=point<_T>;

// 极角排序
struct argcmp
{
    bool operator()(const Point &a,const Point &b) const
    {
        const auto quad=[](const Point &a)
        {
            if (a.y<-eps) return 1;
            if (a.y>eps) return 4;
            if (a.x<-eps) return 5;
            if (a.x>eps) return 3;
            return 2;
        };
        const int qa=quad(a),qb=quad(b);
        if (qa!=qb) return qa<qb;
        const auto t=a^b;
        // if (abs(t)<=eps) return a*a<b*b-eps;  // 不同长度的向量需要分开
        return t>eps;
    }
};

// 直线
template<typename T> struct line
{
    point<T> p,v;  // p 为直线上一点，v 为方向向量

    bool operator==(const line &a) const {return v.toleft(a.v)==0 && v.toleft(p-
a.p)==0;}
    int toleft(const point<T> &a) const {return v.toleft(a-p);}  // to-left 测试
    bool operator<(const line &a) const  // 半平面交算法定义的排序
    {
        if (abs(v^a.v)<=eps && v*a.v>=-eps) return toleft(a.p)==-1;
        return argcmp()(v,a.v);
    }
};

using Line=line<_T>;

//线段
template<typename T> struct segment
{
    point<T> a,b;
```

```
    // 判定性函数建议在整数域使用

    // 判断点是否在线段上
    // -1 点在线段端点 | 0 点不在线段上 | 1 点严格在线段上
    int is_on(const point<T> &p) const
    {
        if (p==a || p==b) return -1;
        return (p-a).toleft(p-b)==0 && (p-a)*(p-b)<-eps;
    }

    // 判断线段直线是否相交
    // -1 直线经过线段端点 | 0 线段和直线不相交 | 1 线段和直线严格相交
    int is_inter(const line<T> &l) const
    {
        if (l.toleft(a)==0 || l.toleft(b)==0) return -1;
        return l.toleft(a)!=l.toleft(b);
    }

    // 判断两线段是否相交
    // -1 在某一线段端点处相交 | 0 两线段不相交 | 1 两线段严格相交
    int is_inter(const segment<T> &s) const
    {
        if (is_on(s.a) || is_on(s.b) || s.is_on(a) || s.is_on(b)) return -1;
        const line<T> l{a,b-a},ls{s.a,s.b-s.a};
        return l.toleft(s.a)*l.toleft(s.b)==-1 && ls.toleft(a)*ls.toleft(b)==-1;
    }
};

using Segment=segment<_T>;

// 多边形
template<typename T> struct polygon
{
    vector<point<T>> p;  // 以逆时针顺序存储

    size_t nxt(const size_t i) const {return i==p.size()-1?0:i+1;}
    size_t pre(const size_t i) const {return i==0?p.size()-1:i-1;}

    // 回转数
    // 返回值第一项表示点是否在多边形边上
    // 对于狭义多边形，回转数为 0 表示点在多边形外，否则点在多边形内
    pair<bool,int> winding(const point<T> &a) const
    {
        int cnt=0;
        for (size_t i=0;i<p.size();i++)
        {
            const point<T> u=p[i],v=p[nxt(i)];
            if (abs((a-u)^(a-v))<=eps && (a-u)*(a-v)<=eps) return {true,0};
            if (abs(u.y-v.y)<=eps) continue;
            const Line uv={u,v-u};
            if (u.y<v.y-eps && uv.toleft(a)<=0) continue;
            if (u.y>v.y+eps && uv.toleft(a)>=0) continue;
            if (u.y<a.y-eps && v.y>=a.y-eps) cnt++;
            if (u.y>=a.y-eps && v.y<a.y-eps) cnt--;
        }
```

```
        return {false,cnt};
    }

    // 多边形面积的两倍
    // 可用于判断点的存储顺序是顺时针或逆时针
    T area() const
    {
        T sum=0;
        for (size_t i=0;i<p.size();i++) sum+=p[i]^p[nxt(i)];
        return sum;
    }

    // 多边形的周长
    long double circ() const
    {
        long double sum=0;
        for (size_t i=0;i<p.size();i++) sum+=p[i].dis(p[nxt(i)]);
        return sum;
    }
};

using Polygon=polygon<_T>;

//凸多边形
template<typename T> struct convex: polygon<T>
{
    // 闵可夫斯基和
    convex operator+(const convex &c) const
    {
        const auto &p=this->p;
        vector<Segment> e1(p.size()),e2(c.p.size()),edge(p.size()+c.p.size());
        vector<point<T>> res; res.reserve(p.size()+c.p.size());
        const auto cmp=[](const Segment &u,const Segment &v) {return argcmp()
(u.b-u.a,v.b-v.a);};
        for (size_t i=0;i<p.size();i++) e1[i]={p[i],p[this->nxt(i)]};
        for (size_t i=0;i<c.p.size();i++) e2[i]={c.p[i],c.p[c.nxt(i)]};
        rotate(e1.begin(),min_element(e1.begin(),e1.end(),cmp),e1.end());
        rotate(e2.begin(),min_element(e2.begin(),e2.end(),cmp),e2.end());
        merge(e1.begin(),e1.end(),e2.begin(),e2.end(),edge.begin(),cmp);
        const auto check=[](const vector<point<T>> &res,const point<T> &u)
        {
            const auto back1=res.back(),back2=*prev(res.end(),2);
            return (back1-back2).toleft(u-back1)==0 && (back1-back2)*(u-
back1)>=-eps;
        };
        auto u=e1[0].a+e2[0].a;
        for (const auto &v:edge)
        {
            while (res.size()>1 && check(res,u)) res.pop_back();
            res.push_back(u);
            u=u+v.b-v.a;
        }
        if (res.size()>1 && check(res,res[0])) res.pop_back();
        return {res};
    }
```

```
    // 旋转卡壳
    // func 为更新答案的函数，可以根据题目调整位置
    template<typename F> void rotcaliper(const F &func) const
    {
        const auto &p=this->p;
        const auto area=[](const point<T> &u,const point<T> &v,const point<T>
&w){return (w-u)^(w-v);};
        for (size_t i=0,j=1;i<p.size();i++)
        {
            const auto nxti=this->nxt(i);
            func(p[i],p[nxti],p[j]);
            while (area(p[this->nxt(j)],p[i],p[nxti])>=area(p[j],p[i],p[nxti]))
            {
                j=this->nxt(j);
                func(p[i],p[nxti],p[j]);
            }
        }
    }

    // 凸多边形的直径的平方
    T diameter2() const
    {
        const auto &p=this->p;
        if (p.size()==1) return 0;
        if (p.size()==2) return p[0].dis2(p[1]);
        T ans=0;
        auto func=[&](const point<T> &u,const point<T> &v,const point<T> &w)
{ans=max({ans,w.dis2(u),w.dis2(v)});};
        rotcaliper(func);
        return ans;
    }

    // 判断点是否在凸多边形内
    // 复杂度 O(logn)
    // -1 点在多边形边上 | 0 点在多边形外 | 1 点在多边形内
    int is_in(const point<T> &a) const
    {
        const auto &p=this->p;
        if (p.size()==1) return a==p[0]?-1:0;
        if (p.size()==2) return segment<T>{p[0],p[1]}.is_on(a)?-1:0;
        if (a==p[0]) return -1;
        if ((p[1]-p[0]).toleft(a-p[0])==-1 || (p.back()-p[0]).toleft(a-p[0])==1)
return 0;
        const auto cmp=[&](const Point &u,const Point &v){return (u-
p[0]).toleft(v-p[0])==1;};
        const size_t i=lower_bound(p.begin()+1,p.end(),a,cmp)-p.begin();
        if (i==1) return segment<T>{p[0],p[i]}.is_on(a)?-1:0;
        if (i==p.size()-1 && segment<T>{p[0],p[i]}.is_on(a)) return -1;
        if (segment<T>{p[i-1],p[i]}.is_on(a)) return -1;
        return (p[i]-p[i-1]).toleft(a-p[i-1])>0;
    }

    // 凸多边形关于某一方向的极点
    // 复杂度 O(logn)
```

```
    // 参考资料：https://codeforces.com/blog/entry/48868
    template<typename F> size_t extreme(const F &dir) const
    {
        const auto &p=this->p;
        const auto check=[&](const size_t i){return dir(p[i]).toleft(p[this-
>nxt(i)]-p[i])>=0;};
        const auto dir0=dir(p[0]); const auto check0=check(0);
        if (!check0 && check(p.size()-1)) return 0;
        const auto cmp=[&](const Point &v)
        {
            const size_t vi=&v-p.data();
            if (vi==0) return 1;
            const auto checkv=check(vi);
            const auto t=dir0.toleft(v-p[0]);
            if (vi==1 && checkv==check0 && dir0.toleft(v-p[0])==0) return 1;
            return checkv^(checkv==check0 && t<=0);
        };
        return partition_point(p.begin(),p.end(),cmp)-p.begin();
    }

    // 过凸多边形外一点求凸多边形的切线，返回切点下标
    // 复杂度 O(logn)
    // 必须保证点在多边形外
    pair<size_t,size_t> tangent(const point<T> &a) const
    {
        const size_t i=extreme([&](const point<T> &u){return u-a;});
        const size_t j=extreme([&](const point<T> &u){return a-u;});
        return {i,j};
    }

    // 求平行于给定直线的凸多边形的切线，返回切点下标
    // 复杂度 O(logn)
    pair<size_t,size_t> tangent(const line<T> &a) const
    {
        const size_t i=extreme([&](...){return a.v;});
        const size_t j=extreme([&](...){return -a.v;});
        return {i,j};
    }
};

using Convex=convex<_T>;

vector<int> solve(const Convex &ball, const vector<Point> &pins, const
vector<Point> &ques)
{
    auto dirs=ques; dirs.reserve(ques.size()+pins.size()+pins.size());
    vector<pair<size_t,size_t>> tans; tans.reserve(pins.size());
    for (const auto &pin:pins)
    {
        const auto t=ball.tangent(pin);
        const auto i=t.first,j=t.second;
        tans.push_back(t);
        dirs.push_back(pin-ball.p[j]);
        dirs.push_back(pin-ball.p[i]);
    }
```

```
    sort(dirs.begin(),dirs.end(),argcmp());
    const auto eq=[&](const Point &u,const Point &v){return u*v>eps && abs(u^v)
<=eps;};
    dirs.erase(unique(dirs.begin(),dirs.end(),eq),dirs.end());
    const int siz=dirs.size();
    vector<int> diff(siz),sum(siz);
    for (size_t i=0;i<pins.size();i++)
    {
        const auto pin=pins[i];
        const auto t=tans[i];
        const Point u=pin-ball.p[t.second],v=pin-ball.p[t.first];
        const int l=lower_bound(dirs.begin(),dirs.end(),u,argcmp())-
dirs.begin();
        const int r=lower_bound(dirs.begin(),dirs.end(),v,argcmp())-
dirs.begin();
        if (l<=r)
        {
            diff[l]++;
            if (r+1<siz) diff[r+1]--;
        }
        else
        {
            diff[l]++; diff[0]++;
            if (r+1<siz) diff[r+1]--;
        }
    }
    partial_sum(diff.begin(),diff.end(),sum.begin());
    vector<int> ans; ans.reserve(ques.size());
    for (const auto &que:ques)
    {
        const int i=lower_bound(dirs.begin(),dirs.end(),que,argcmp())-
dirs.begin();
        ans.push_back(sum[i]);
    }
    return ans;
}

int main()
{
    int T;
    scanf("%d",&T);
    while (T--)
    {
        int n;
        scanf("%d",&n);
        Convex ball; ball.p.resize(n);
        for (int i=0;i<n;i++)
        {
            int x,y;
            scanf("%d%d",&x,&y);
            ball.p[i]={x,y};
        }
        int m;
        scanf("%d",&m);
        vector<Point> pins(m);
```

```
        for (int i=0;i<m;i++)
        {
            int x,y;
            scanf("%d%d",&x,&y);
            pins[i]={x,y};
        }
        int q;
        scanf("%d",&q);
        vector<Point> ques(q);
        for (int i=0;i<q;i++)
        {
            int x,y;
            scanf("%d%d",&x,&y);
            ques[i]={x,y};
        }
        auto ans=solve(ball,pins,ques);
        for (int i:ans) printf("%d\n",i);
    }
    return 0;
}
```

K. 三角形游戏

结论是：Alice获胜当且仅当$(a-1)\oplus(b-1)\oplus(c-1)\neq 0$。其中$\oplus$为异或运算。

不妨令x,y,z分别为三角形最短，次短和最长的边长。由于$x+y>z$且x,y,z都是正整数，则有$(x-1)+(y-1)\geq(z-1)$。

不妨定义如下状态：

- (L态) $(x-1)\oplus(y-1)\oplus(z-1)=0$
- (W态) $(x-1)\oplus(y-1)\oplus(z-1)\neq 0$

当玩家目前处于L态时，由于$(x-1)\oplus(y-1)=(z-1)$，则有$(x-1)+(y-1)\geq(x-1)\oplus(y-1)=(z-1)$，此时一定为一个非退化三角形。

在L态时，转移有如下情况：

- $x=1$，则$y=z$。这种情况任何玩家移动都会判负，因此为必败态。
- $x>1$，则$1<x<y<z$，此时一定存在一种减少边长的方案，使得减少边长后三角形不退化。不妨考虑x减少为x'，则由于减去的是正整数，会有$(y-1)\oplus(z-1)=(x-1)\neq(x'-1)$。则$(x'-1)\oplus(y-1)\oplus(z-1)\neq 0$。如果改变的是其他边，也可以类似地利用此式。

综上，L态一定转移到某必败态或W态。

当玩家处于W态时，有$(x-1)\oplus(y-1)\neq(z-1)$。令$r=(x-1)\oplus(y-1)\oplus(z-1)$，则$((x-1)\oplus r)+1<x$或$((y-1)\oplus r)+1<y$或$((z-1)\oplus r)+1<z$，三式中必有一式成立。考虑$r$的二进制中每一位1都一定出现了奇数次，对于最高位的1，将其异或r后一定会变小。因此可以将成立的不等式作为这一步操作进行代换，即可转为L态。

由此，每次移动三角形某边长均会减小，W态会转化为L态，而L态均转化为某些必败态和W态。故所有W态均为必胜态，L态均为必败态。结论证毕。

时空复杂度均为$\mathcal{O}(1)$。

参考代码：

```cpp
#include <cstdio>

void Solve()
{
    int a, b, c;
    scanf("%d %d %d", &a, &b, &c);
    puts(((a - 1) ^ (b - 1) ^ (c - 1)) ? "Win" : "Lose");
    return ;
}

int main()
{
    int T; scanf("%d", &T);
    while (T--) Solve();
    return 0;
}
```

L. 优雅的数组

考虑固定n和m，此时数组中都是m的因子，且最后一个位置为m。将每个a_i除以a_{i-1}之后对每个位置上的数做质因子分解，可以观察到答案对于每个质因子是互相独立的，即固定n时答案是一个关于m的积性函数。固定m时，设m中质因子p的幂为e，则方案数等于将e个p分给$n-1$个位置的方案数。显然这是一个组合数，且可以视作关于n的一个e次多项式。因此任选$e+1$个n，对每个n用min25算一次即可将答案插值出来。单次询问复杂度为$\mathcal{O}(m^{3/4})$。

参考代码：

```cpp
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;

const int N = 500001;
const int P = 1'000'000'007;

typedef vector<int> vi;

inline int add(int a, int b) { int r = a + b; return r < P ? r : r - P; }
inline int sub(int a, int b) { int r = a - b; return r < 0 ? r + P : r; }
inline int mul(int a, int b) { return 1ll * a * b % P; }
inline int qpm(int a, int b) {
    int r = 1;
    do if (b & 1) r = mul(r, a);
    while (a = mul(a, a), b >>= 1);
    return r;
}
inline int inv(int x) { return qpm(x, P - 2); }

int invs[N], fac[N], ifac[N];
int binom(int n, int k) { return mul(fac[n], mul(ifac[n - k], ifac[k])); }
void ginv() {
    invs[1] = 1; fac[0] = ifac[0] = 1;
    for (int i = 2; i != N; ++i) invs[i] = mul(invs[P % i], (P - P / i));
    for (int i = 1; i != N; ++i) fac[i] = mul(fac[i - 1], i);
    for (int i = 1; i != N; ++i) ifac[i] = mul(ifac[i - 1], invs[i]);
```

```cpp
}

int lintp(const vector<int>& x, const vector<int>& y, int k) {
    int ans = 0;
    for (int i = 0; i != x.size(); ++i) {
        if (k == x[i])
            return y[i];
        int u = 1, v = 1;
        for (int j = 0; j != x.size(); ++j) {
            if (i == j) continue;
            u = mul(u, sub(k, x[j]));
            v = mul(v, sub(x[i], x[j]));
        }
        ans = add(ans, mul(y[i], mul(u, inv(v))));
    }
    return ans;
}

namespace min25 {

const int N = 1000005;
bool ip[N]; vector<int> ps;
void sieve() {
    fill_n(ip, N, 1); ip[1] = 0;
    for (int i = 2; i < N; ++i) {
        if (ip[i]) ps.push_back(i);
        for (int j : ps) {
            if (i * j >= N) break;
            ip[i * j] = 0;
            if (i % j == 0) break;
        }
    }
}

ll n, sq, w[N]; int c;
ll g[N];

inline int id(ll x) { return x ? x <= sq ? c - x + 1 : n / x : 0; }

void cal_g(ll n_) {
    n = n_; sq = sqrt(n_); c = 0;
    for (ll l = 1, r; l <= n; l = r + 1) {
        ll v = w[++c] = n / l; r = n / v;
        g[c] = (v - 1) % P;
    }
    for (int p : ps) {
        if (1ll * p * p > n) break;
        for (int j = 1; 1ll * p * p <= w[j]; ++j)
            g[j] -= g[id(w[j] / p)] - g[id(p - 1)];
    }
}

int cal_s(int n, int i, ll x) {
    int p = ps[i];
    if (x < p) return 0;
```

```
    int sum = 0;
    if (1ll * p * p > x)
        sum = mul(n, sub(g[id(x)], i == 0 ? 0 : g[id(ps[i - 1])]));
    else {
        sum = cal_s(n, i + 1, x);
        ll q = p;
        for (int e = 1; q <= x; e++, q *= p)
            sum = add(sum, mul(binom(n + e - 1, n - 1), add(cal_s(n, i + 1, x /
q), 1)));
    }
    return sum;
}

int cal_sf(ll n, ll m) {
    cal_g(m);
    vector<int> x, y;
    for (int i = 1; i <= 32; ++i) {
        x.push_back(i);
        y.push_back(add(y.empty() ? 0 : y.back(), add(cal_s(i, 0, m), 1)));
    }
    return lintp(x, y, n);
}

}


int main(void) {
    ginv();
    min25::sieve();
    int T;
    scanf("%d", &T);
    while (T--) {
        int n, m;
        scanf("%d %d", &n, &m);
        printf("%d\n", min25::cal_sf(n, m));
    }

    return 0;
}
```

M. 布娃娃

$dp(x,y)$表示将前x个套娃分成y组的方案数，转移为

$$dp(x,y)=dp(x-1,y-1)+dp(x-1,y)\cdot max0,y-f(x)$$

其中$f(x)$表示满足$1\le z<x$且$a_x-r<a_z\le a_x$的z的个数。单组数据时间复杂度：$\mathcal{O}(n^2)$。

参考代码：

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int M=998244353;
int n,k,r,a[5005],dp[5005][5005];
```

```
void solve(){
    memset(dp,0,sizeof(dp));
    cin >> n >> k >> r;
    for (int i=1;i<=n;i++) cin >> a[i];
    int p=0; dp[0][0]=1;
    for (int i=1;i<=n;i++){
        while (p+1<i&&a[p+1]+r<=a[i]) ++p;
        int s=i-p-1;
        for (int j=s+1;j<=i;j++)
            dp[i][j]=(1ll*(j-s)*dp[i-1][j]+dp[i-1][j-1])%M;
    }
    cout << dp[n][k] << endl;
}
int main(){
    int T; cin >> T;
    while (T--) solve();
}
```