USTC 2020 年秋季学期集训2题解

A(CF1333B).

对于$\forall i, 1 \leq i \leq n$，当$a_i > b_i$时，必须$\exists j < i$且$a_j = -1$；当$a_i < b_i$时，必须$\exists j < i$且$a_j = 1$。

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
const int maxn=1e5+10;
int t;
int n,a[maxn],b[maxn];

int main(){
    scanf("%d",&t);
    while(t--){
        scanf("%d",&n);
        for(int i=1;i<=n;i++)scanf("%d",&a[i]);
        for(int i=1;i<=n;i++)scanf("%d",&b[i]);
        bool f1=false,f2=false,f=true;
        for(int i=1;i<=n;i++){
            if(a[i]>b[i]){
                if(!f2){
                    f=false;
                    break;
                }
            }
            else if(a[i]<b[i]){
                if(!f1){
                    f=false;
                    break;
                }
            }
            if(a[i]==1)f1=true;
            else if(a[i]==-1)f2=true;
            if(f1&&f2)break;
        }
        if(f)cout<<"YES"<<endl;
        else cout<<"NO"<<endl;
    }
    return 0;
}
```

B(CF1295D).

设$gcd(a, m) = gcd(a + x, m) = d$，令$a = a'd, m = m'd, x = x'd$,则有$gcd(a', m') = gcd(a' + x', m') = 1, 0 \leq x' < m'$，有$a' \leq a' + x' < m' + a'$，令$i = a' + x'$，枚举$i$。

$ans = \sum_{i=a'}^{m'-1+a'} [gcd(i, m') = 1]$(若$gcd(i, m') = 1$成立，则$[gcd(i, m') = 1]$的值为$1$，反之)

$$= \sum_{i=a'}^{m'-1}[gcd(i,m')=1] + \sum_{i=m'}^{m'-1+a'}[gcd(i,m')=1]$$

$$= \sum_{i=a'}^{m'-1}[gcd(i,m')=1] + \sum_{i=0}^{a'-1}[gcd(i,m')=1], 因为(gcd(i,m')=gcd(i+m',m'))$$

$$= \sum_{i=0}^{m'-1}[gcd(i,m')=1]$$

$= \phi(m')$，上一个式子表示对0-m'-1内与m'互素的个数计数，即为欧拉函数

对于本题，求欧拉函数可以直接$O(\sqrt{m'})$的暴力方法来求。

```cpp
#include<bits/stdc++.h>
using namespace std;
#define ll long long
ll a,n;
int t;

inline ll phi(ll x){
    ll s=x,y=x;
    for(ll i=2;i*i<=x;i++)if(y%i==0){
        while(y%i==0)y/=i;
        s=s/i*(i-1);
    }
    if(y!=1)s=s/y*(y-1);
    return s;
}

int main(){
    cin>>t;
    while(t--){
        cin>>a>>n;
        n/=__gcd(n,a);
        cout<<phi(n)<<endl;
    }
    return 0;
}
```

C(CF1300E).

显然最后的答案a_i单调不降，所以构建单调栈，栈中元素为区间信息{l,r,v}，l为区间左端点，r为区间右端点，v为区间元素和，每次输入新的a_i时，向栈中加入$\{i,i,a_i\}$，每次加入后，都不断取栈顶的2个元素，若栈顶的元素区间的平均值小于等于栈顶下面一个的元素区间的平均值，则讲两个元素弹出并合并放入栈顶，否则不做操作。

```cpp
#include<bits/stdc++.h>
using namespace std;
#define ll long long
const int maxn=1e6+10;
int n,cnt=0;
struct point{
    int l,r;
    ll v;
    point(int l=0,int r=0,ll v=0):l(l),r(r),v(v){}
```

```
}a[maxn];

int main(){
    cin>>n;
    ll x;
    for(int i=1;i<=n;i++){
        scanf("%I64d",&x);
        a[++cnt]=point(i,i,x);
        while(cnt>1){
            if(a[cnt-1].v*(a[cnt].r-a[cnt].l+1)>=a[cnt].v*(a[cnt-1].r-a[cnt-1].l+1)){
                a[cnt-1].v+=a[cnt].v;
                a[cnt-1].r=a[cnt].r;
                cnt--;
            }
            else break;
        }
    }
    for(int i=1;i<=cnt;i++)
        for(int j=a[i].l;j<=a[i].r;j++)printf("%.9lf\n",a[i].v/(a[i].r-a[i].l+1.0));
    return 0;
}
```

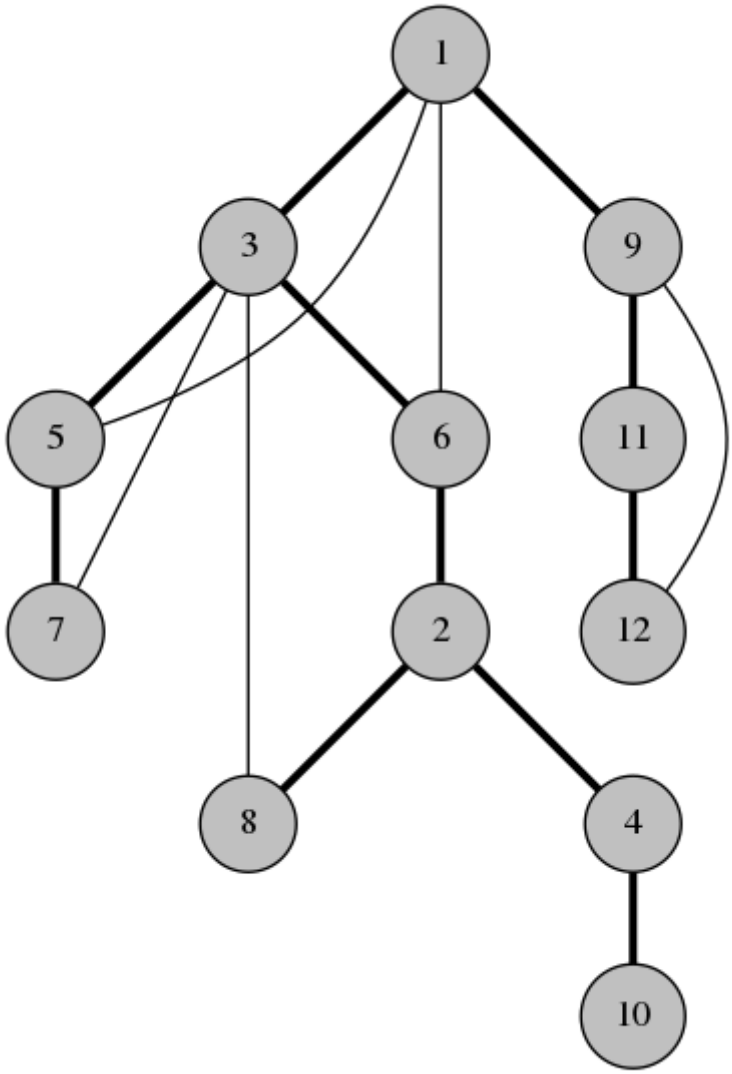
D(CF1342E).

显然当$n \leq k$，答案为0（因为就算所有棋子放一行，也只有n-1对棋子可以互相攻击）。当$k=0$时，特判答案为$n!$。又因为每个格子都要被攻击到，所以必须保证每一行或每一列都有一个棋子，所以不妨考虑每一行都有一个棋子的方案。当每一行都有一个棋子时，若有某一列有a_i个棋子，则有a_i-1对棋子互相攻击，设有m列有棋子，则$\sum\limits_{i,\text{第}i\text{列有棋子}}(a_i-1)=k$，即为$n-m=k$，所以m已知，从n列中选择m列有$\binom{n}{m}$种，剩下需要讲每一行填一个棋子使得m列中的每一列都有棋子，如果m个列中有i个列是空的，则n行的棋子只有m-i种列可选，所以有$\binom{m}{i}(m-i)^n$种情况，于是由容斥原理可得每一行填一个棋子使得m列中的每一列都有棋子的方案数有$\sum\limits_{i=0}^{m}(-1)^i\binom{m}{i}(m-i)^n$。最终$0<k<n$的答案为$2\binom{n}{m}\sum\limits_{i=0}^{m}(-1)^i\binom{m}{i}(m-i)^n$，直接枚举i计算即可。

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
const int maxn=2e5+10;
const ll mod=998244353;
ll f[maxn],n,k;

inline void prepare(){
    f[0]=1;
    for(ll i=1;i<maxn;i++)f[i]=f[i-1]*i%mod;
}

inline ll qpow(ll x,ll y){
    ll ans=1;
```

```
    while(y){
        if(y&1)ans=ans*x%mod;
        y>>=1;
        x=x*x%mod;
    }
    return ans;
}

int main(){
    prepare();
    cin>>n>>k;
    if(k==0)printf("%I64d\n",f[n]);
    else if(k>=n)printf("0\n");
    else{
        ll ans=0,m=n-k;
        for(ll i=0;i<=m;i++){
            if(i%2==0)ans=(ans+f[m]*qpow(f[i],mod-2)%mod*qpow(f[m-i],mod-2)%mod*qpow(m-i,n)%mod)%mod;
            else ans=(ans-f[m]*qpow(f[i],mod-2)%mod*qpow(f[m-i],mod-2)%mod*qpow(m-i,n)%mod+mod)%mod;
        }
        ans=ans*f[n]%mod*qpow(f[m],mod-2)%mod*qpow(f[k],mod-2)%mod*2%mod;
        printf("%I64d\n",ans);
    }
    return 0;
}
```

E(CF1325F).



该图为dfs树。

https://codeforces.com/blog/entry/68138 该链接介绍了dfs树。

令$sq=\lceil\sqrt{n}\rceil$先求出该无向图的dfs树，标记每个点的深度，因为dfs树的性质：**一条非树边连接了一个点和它在生成树中的一个后代**，所以如果存在一条非树边使得两个端点的深度差$\geq sq-1$，就能形成满足条件的环。若找不出环则每个点最多有$sq-2$个非树边，就一定存在$\geq sq$个点的最大独立集，寻找独立集染色即可。

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
const int maxn=4e5+10;
int cnt=0,head[maxn];
int n,m;
int sq;
struct edge{
    int v,nxt;
    edge(int v=0,int nxt=0):v(v),nxt(nxt){}
}e[maxn<<1];
int d[maxn];
int vis[maxn];
vector<int>p1;
vector<int>p;
```

```cpp

inline void addedge(int u,int v){
    e[++cnt]=edge(v,head[u]);head[u]=cnt;
    e[++cnt]=edge(u,head[v]);head[v]=cnt;
}

inline bool dfs1(int u){
    p1.push_back(u);
    d[u]=p1.size();
    for(int i=head[u];i;i=e[i].nxt){
        int v=e[i].v;
        if(!d[v]&&dfs1(v))return true;
        else if(d[v]+sq-1<=d[u]){
                printf("2\n");
                printf("%d\n",d[u]-d[v]+1);
                for(int i=d[v]-1;i<d[u];i++)printf("%d ",p1[i]);
                printf("\n");
                return true;
            }
    }
    if(!vis[u]){
        p.push_back(u);
        for(int i=head[u];i;i=e[i].nxt){
            int v=e[i].v;
            vis[v]=1;
        }
    }
    p1.pop_back();
    return false;
}

int main(){
    cin>>n>>m;
    sq=sqrt(n-1)+1;
    for(int i=1;i<=m;i++){
        int u,v;
        scanf("%d%d",&u,&v);
        addedge(u,v);
    }
    if(!dfs1(1)){
        printf("1\n");
        for(int j=0;j<sq;j++)printf("%d ",p[j]);
        printf("\n");
    }
    return 0;
}
```

F(CF1336D).

$a_i \to a_i+1$时,$\Delta triplet = \frac{a_i(a_i-1)}{2}$，$\Delta straight = a_{i-2}a_{i-1}+a_{i-1}a_{i+1}+a_{i+1}a_{i+2}$，所以至多放两次牌就知道$a_i$,知道$a_{i-2},a_{i-1},a_{i+1}$就可以推出$a_{i+2}$。 但是当$a_{i+1}=0(i\geq 2)$时，无法求出$a_{i+2}$。

所以先往$a_{n-1}\to a_3$都投一张牌，此时$a_i(i\in[3,n-1])$的$\Delta straight=a_{i-2}a_{i-1}+a_{i-1}(a_{i+1}+1)+(a_{i+1}+1)(a_{i+2}+1)$。此时可以用三次牌确定出$a_1,a_2,a_3$了：往$a_1$里投一张牌得到$a_2(a_3+1)$，往$a_2$里投一张牌得到$(a_3+1)(a_1+a_4+2)$，再往$a_1$里投一张牌得到$(a_2+1)(a_3+1)$和$a_1$。顺推就可以解出$a_i$,刚好n次。

需要特判$n=4$的情况，因为$n=4$时，a_2的$\Delta straight=(a_3+1)(a_1+a_4+1)$。

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
const int maxn=100+10;
int ans1,ans2,n;
int s[maxn*maxn];
struct point{
    int x,y;
    point(int x=0,int y=0):x(x),y(y){}
}a[maxn];
int c[maxn];

inline point query(int x){
    printf("+ %d\n",x);
    fflush(stdout);
    int ans3,ans4;
    scanf("%d%d",&ans3,&ans4);
    a[x]=point(ans3-ans1,ans4-ans2);
    ans1=ans3;
    ans2=ans4;
    return a[x];
}

int main(){
    for(int i=1;i<maxn;i++)s[i*(i-1)/2]=i;
    scanf("%d%d%d",&n,&ans1,&ans2);
    for(int i=n-1;i>=3;i--)query(i);
    point x1,x2,x3;
    x1=query(1);
    x2=query(2);
    x3=query(1);
    c[3]=x3.y-x1.y-1;
    if(x1.y==0)c[2]=0;
    else c[2]=x1.y/(c[3]+1);
    if(x1.x==0){
        if(x3.x==0)c[1]=0;
        else c[1]=1;
    }
    else c[1]=s[x1.x];
    if(n==4)c[4]=x2.y/(c[3]+1)-c[1]-1;
    else c[4]=x2.y/(c[3]+1)-c[1]-2;
    for(int i=3;i<=n-3;i++)c[i+2]=(a[i].y-c[i-2]*c[i-1]-c[i-1]*
(c[i+1]+1))/(c[i+1]+1)-1;
    if(n>4)c[n]=(a[n-2].y-c[n-4]*c[n-3]-c[n-3]*(c[n-1]+1))/(c[n-1]+1);
    printf("! ");
    for(int i=1;i<=n;i++)printf("%d ",c[i]);
    printf("\n");
    return 0;
}
```

Edited by jyz