

[比赛链接](#)

补题情况

题目	蒋贤蒙	王赵安	王智彪
A	0	1	2
B	0	0	0
C	1	0	2
D	2	1	0
E	2	0	0
G	0	1	2
J	2	0	1
K	2	0	0

题解

A. Contracting Convex Hull

题意

给了 $n(n \leq 1000)$ 个点的凸包，这个凸包每条边以 1 的速度向内收缩。给 q 组询问，每组询问一个时间，求这个时间凸包的面积 $\square$ $(q \leq 100000)$ $\square$

题解

这绝对是最容易理解的解法，没有之一！（因为看了别人的神仙代码看不懂，于是自己亲自动手了...）

由角分线定理知，每个点的移动轨迹其实是这个点在凸包内的夹角的角平分线。

于是我们可以求每个角，也就知道每个点的轨迹，知道轨迹就可以知道这个边什么时候收缩到一个点。

烦人的是因为有的边收缩成点，会改变凸包形状，于是相邻的角会变，然后相邻的边的收缩时间会变，所以这是个动态的过程，这也是我考场上不知道怎么解决的地方，对不起我的队友...

于是我这个暴力狂 $\times$ ，自然每次都暴力求一下所有点的收缩时间，选出最短的时间，同时有一点很显然，就是将询问时间排序，模拟凸包收缩的过程。当当前询问的时间在现在时间+最近删除时间之后时，说明我们要删除一个点，于是我们模拟删除的过程，并更新凸包周长面积和周长缩小速度（暴力更新即可，因为每次删除也是 $O(n)$ 的，除非你闲出屁...）。

然后面积就是删除后的当前时间距离询问时间这个时间内缩小之后的周长（是时间的一次函数，可以直接 $O(1)$ 得到）+之前周长（也就是梯形的上底加下底）乘时间除以 2 即可，也就是我们把两个凸包之间的区域看成 n 个梯形，搞一下就出来了~。

别忘了，当删剩下的点不足 2 的时候，直接输出 0 ，因为已经没有面积，不然样例都过不去 $\times$ 。

```
//这绝对是最粗暴并且最易懂的做法!!!
#include<bits/stdc++.h>
```

```
using namespace std;
const int M=100100;
const long double eps=1e-12;
int n,q,erase;
long double ans[M],ti,tot,V,Len,dans;
int sgn(long double a) {
    if(fabs(a)<eps)return 0;
    if(a>0)return 1;
    else return -1;
}
struct QQ {
    int id;
    long double t;
} Q[M];
int cmp(QQ a,QQ b) {
    return a.t<b.t;
}

int nxt(int i,int n) {
    i++;
    if(i==n+1) return 1;
    return i;
}

struct node {
    long double x,y,dx,dy;
    node operator +(const node&tmp)const {
        return (node) {x+tmp.x,y+tmp.y,dx+tmp.dx,dy+tmp.dy};
    }
    node operator -(const node&tmp)const {
        return (node) {x-tmp.x,y-tmp.y,dx-tmp.dx,dy-tmp.dy};
    }
    node operator /(const long double &a)const {
        return (node) {x/a,y/a,dx/a,dy/a};
    }
    long double length() {
        return sqrt(x*x+y*y);
    }
    long double v() {
        return sqrt(dx*dx+dy*dy);
    }
    long double dis(node nd) {
        return sqrt((x-nd.x)*(x-nd.x)+(y-nd.y)*(y-nd.y));
    }
} A[M],now;
long double Angle(node a,node b) {
    return acos((a.x*b.x+a.y*b.y)/a.length()/b.length());
}
node rotate(node a,long double t) {
    return (node) {a.x*cos(t)-a.y*sin(t),a.x*sin(t)+a.y*cos(t),0,0};
}
```

```

}
void init(long double &a,long double &b,long double &c) {
    if(n<3)return;
    a=0,b=0,c=0;
    int m=0;
    while(erase&&n>=3) {
        for(int i=erase; i<n; i++) A[i]=A[i+1]; //把删除的点挤掉
        n--; //少了一个点
        erase=0; //把消除标记打掉
        A[0]=A[n],A[n+1]=A[1];
        for(int i=1; i<=n; i++) {
            long double t;
            now=A[i+1]-A[i];
            t=now.length()/now.v();
            if(sgn(t-ti)==0) erase=i; //这个点也应该这个点删掉 那就继续删
        }
    }
    if(n<3)return;
    for(int i=1; i<=n; i++) {
        A[i].x=A[i].x+A[i].dx*ti; //移动速度*时间
        A[i].y=A[i].y+A[i].dy*ti;
    }
    A[0]=A[n],A[n+1]=A[1];
    for(int i=1; i<=n; i++) {
        long double th=Angle(A[i-1]-A[i],A[i+1]-A[i])/2; //移动按照角分线移动
        now=rotate(A[i+1]-A[i],th); //旋转到角分线位置
        now=now/now.length()/sin(th); //如果凸包缩小1 这个点沿角分线移动now的距离
        A[i].dx=now.x; //dx赋值
        A[i].dy=now.y;
    }
    A[0]=A[n],A[n+1]=A[1];
    tot+=ti;
    ti=1e9;
    for(int i=1; i<=n; i++) {
        now=A[i+1]-A[i];
        long double t=now.length()/now.v(); //发现速度垂直于线的分量被两边消掉了 只剩下沿线方向的 于是就直接除就是这个边消失的时间
        if(sgn(t-ti)<0) { //看看接下来应该删什么
            ti=t;
            erase=i;
        }
    }
}
V=0;
Len=0;
dans=0;
for(int i=1;i<=n;i++) {
    V+=(A[i+1]-A[i]).v(); //暴力计算周长缩小的速度
    Len+=(A[i+1]-A[i]).length(); //暴力计算当前周长
    dans+=(A[i].x*A[nxt(i,n)].y-A[i].y*A[nxt(i,n)].x)/2; //暴力计算当前面积
}
dans=fabs(dans); //三角剖分求面积需要绝对值

```

```

}

void solve() {
    long double a=0,b=0,c=0;
    init(a,b,c);
    for(int i=1; i<=q; i++) {
        while(n>=3&&sgn(tot+ti-Q[i].t)<=0) init(a,b,c); //当 当前时间+最快要删除
        的点的的时间和小于等于这个时间说明要删这个点
        if(n<3) ans[Q[i].id]=0; //不足三个点 图形都围不成直接0
        else ans[Q[i].id]=dans-(Len+Len-(Q[i].t-tot)*V)*(Q[i].t-tot)/2; //不然
        就是当前面积-所有的梯形面积就是 (原周长+现在周长)*时间/2
    }
}

int main() {
    scanf("%d",&n);
    for(int i=1; i<=n; i++) {
        A[i].dx=A[i].dy=0;
        scanf("%LF%LF",&A[i].x,&A[i].y);
    }
    scanf("%d",&q);
    for(int i=1; i<=q; i++) {
        Q[i].id=i; //标记是哪个问题
        scanf("%LF",&Q[i].t);
    }
    sort(Q+1,Q+1+q,cmp); //将点的移动按照时间排序
    solve();
    for(int i=1; i<=q; i++) printf("%.10LF\n",ans[i]);
    return 0;
}

```

D. Gambling Monster

题意

给定一个数 x 初始值为 0 。每次操作随机获得一个数 $y \in [0 \sim n-1]$ 其中 $y=i$ 的概率为 p_i

每次操作 $x \leftarrow \max(x, x \oplus y)$ 求 x 变成 $n-1$ 的期望操作次数。保证 n 为 2 的幂次。

题解

设 $dp(i)$ 表示从 i 到 $n-1$ 的期望操作次数，则 $dp(n-1)=0$ 题目答案为 $dp(0)$ 不难得到以下状态转移

$$dp(i) = 1 + \sum_{j \oplus i > i} p_j dp(j \oplus i) + \sum_{j \oplus i \leq i} p_j dp(i)$$

移项，得

$$dp(i) = \frac{1 + \sum_{j \oplus i > i} p_j}{\sum_{j \oplus i > i} p_j}$$

考虑满足 $j \oplus i \geq i$ 的 j 的范围。设 $i = 010100$ 从高位到低位考虑合法的 j

不难发现 $j \in [100000 \sim 111111] \cup [001000 \sim 001111] \cup [000010 \sim 000011] \cup [000001 \sim 000001]$

预处理 p_j 的前缀和，不难计算出 $\sum_{j \oplus i > i} p_j$ 接下来考虑怎么计算 $\sum_{j \oplus i > i} p_j dp(j \oplus i)$

实际上，可以从后往前 dp 刷表法更新 $\sum_{j \oplus i > i} p_j dp(j \oplus i)$

当计算完 $i = 010100$ 时 $dp(010100 \sim 010111)$ 正好全部算出，取 $j = [000100 \sim 000111]$

利用 FWT 计算 $dp(010100 \sim 010111)$ 对 $dp(010000 \sim 010011)$ 的贡献。

贡献计算的时间复杂度 $O(\sum_{i=1}^n \text{lowbit}(i) \log \text{lowbit}(i) \log n) \sim O(n \log^2 n)$

ps. 标程是 CDQ 分治套 FWT 突然感觉简单了不少。

```
const int MAXN=1<<17,mod=1e9+7;
int quick_pow(int n,int k){
    int ans=1;
    while(k){
        if(k&1)ans=1LL*ans*n%mod;
        n=1LL*n*n%mod;
        k>>=1;
    }
    return ans;
}
void XOR(int *f,int n,int type){
    int t1,t2,t3=type==1?1:quick_pow(2,mod-2);
    for(int i=1;i<n;i<=1)
        for(int j=0;j<n;j+=(i<<1))
            _for(k,j,j+i){
                t1=f[k],t2=f[k+i];
                f[k]=1LL*(t1+t2)*t3%mod;
                f[k+i]=1LL*(t1-t2)*t3%mod;
            }
    }
#define lowbit(x) ((x)&(-x))
int a[MAXN],pre[MAXN],dp[MAXN],temp1[MAXN],temp2[MAXN];
void solve(){
    int n=read_int(),s=0,bt=0;
    while(n!=(1<<bt))bt++;
    bt--;
    _for(i,0,n){
        a[i]=read_int();
        s+=a[i];
    }
```

```

        pre[i]=s;
    }
    s=quick_pow(s,mod-2);
    _for(i,0,n){
        a[i]=1LL*a[i]*s%mod;
        pre[i]=1LL*pre[i]*s%mod;
    }
    mem(dp,0);
    for(int i=n-2;i>=0;i--){
        dp[i]++;
        int d=0;
        for(int j=bt;j>=0;j--){
            if((i&(1<<j))==0)
                d=(d+pre[(1<<(j+1))-1]-pre[(1<<j)-1])%mod;
        }
        dp[i]=1LL*dp[i]*quick_pow(d,mod-2)%mod;
        int len=lowbit(i);
        _for(j,0,len){
            temp1[j]=dp[i+j];
            temp2[j]=a[j^len];
        }
        XOR(temp1,len,1);
        XOR(temp2,len,1);
        _for(j,0,len)
            temp1[j]=1LL*temp1[j]*temp2[j]%mod;
        XOR(temp1,len,0);
        _for(j,0,len)
            dp[(i^len)+j]=(dp[(i^len)+j]+temp1[j])%mod;
    }
    enter((dp[0]+mod)%mod);
}
int main(){
    int T=read_int();
    while(T--){
        solve();
    }
    return 0;
}

```

E. Growing Tree

题意

给定一棵树，初始时只有一个颜色为 c 的 1 号结点，接下来 m 个操作：

1. 对结点 u 添加一个颜色为 c 的叶子结点 $n+1$ (设当前共有 n 个结点)
2. 查询结点 u 子树颜色为 c 的结点数量

题解

维护每个结点 u 的最新儿子 $\text{hson}(u)$ 以及每个结点的子树在单括号序列的终止结点 $\text{dfr}(u)$ (子树不包含该点)。

于是新插入叶子结点 $n+1$ 时, 如果 u 没有叶子结点, 则有 $\text{dfr}(n+1) \leftarrow \text{dfr}(u)$ 否则有 $\text{dfr}(n+1) \leftarrow \text{hson}(u)$

然后将 $\text{hson}(u)$ 更新为 $n+1$ 即可实现序列的维护。 u 的子树查询等价于查询序列 $[\text{pos}(u), \text{pos}(\text{dfr}(u))]$ 的信息。

接下来为每种颜色开一个平衡树, 每棵平衡树维护该颜色结点的所有位置。

则操作 2 等价于查询颜色为 c 的平衡树中位置位于 $[\text{pos}(u), \text{pos}(\text{dfr}(u))]$ 的结点个数。

发现 $\text{pos}(u)$ 是动态更新的, 但是 $\text{pos}(u), \text{pos}(v)$ 的相对大小是不会改变的, 因此考虑建立一棵平衡树维护所有 $\text{pos}(u)$

如果用 splay 维护 $\text{pos}(u)$ 则每次查询 $\text{pos}(u)$ 时间复杂度为 $O(\log n)$ 在颜色为 c 的平衡树查询的结点数的时间复杂度为 $O(\log^2 n)$

考虑 $O(1)$ 查询 $\text{pos}(u)$ 一种想法是将 $\text{pos}(u)$ 映射到实数空间, 每次插入叶子结点则将 $\text{pos}(n+1)$ 赋值为 $\frac{\text{pos}(u) + \text{pos}(v)}{2}$

其中 u, v 表示 $n+1$ 在单括号序列的左右相邻结点, 但这样会导致精度误差。

考虑替罪羊树维护 long long 空间, 替罪羊树定期重构重新赋值部分 $\text{pos}(u)$ 保证了树的深度, 不会导致精度误差。

算法总时间复杂度变为 $O(n \log n)$

```
const int MAXN=5e5+5;
const LL MAXV=1LL<<62;
LL val[MAXN];
namespace Tree1{
    #define lch(k) node[node[k].lch]
    #define rch(k) node[node[k].rch]
    const double alpha=0.70;
    struct Node{
        int lch,rch,sz;
    }node[MAXN];
    int root,a[MAXN],n;
    bool isbad(int k){
        return alpha*node[k].sz<max(lch(k).sz,rch(k).sz);
    }
    void build(int &k,int lef,int rig,LL vl,LL vr){
        if(lef>rig) return k=0,void();
        int mid=lef+rig>>1;
        k=a[mid];
        val[k]=vl+vr>>1;
        if(lef==rig){
```

```
        node[k].lch=node[k].rch=0;
        node[k].sz=1;
        return;
    }
    build(node[k].lch,lef,mid-1,vl,val[k]-1);
    build(node[k].rch,mid+1,rig,val[k]+1,vr);
    node[k].sz=lch(k).sz+rch(k).sz+1;
}
void dfs(int k){
    if(!k)return;
    dfs(node[k].lch);
    a[++n]=k;
    dfs(node[k].rch);
}
void rebuild(int &k,LL vl,LL vr){
    n=0;
    dfs(k);
    build(k,1,n,vl,vr);
}
void check(int &k,int id,LL vl,LL vr){
    if(k){
        if(isbad(k))
            rebuild(k,vl,vr);
        else if(val[id]<val[k])
            check(node[k].lch,id,vl,val[k]-1);
        else
            check(node[k].rch,id,val[k]+1,vr);
    }
}
void Insert(int &k,int id){
    if(!k){
        k=id;
        node[k].lch=node[k].rch=0;
        node[k].sz=1;
        return;
    }
    node[k].sz++;
    if(val[id]<val[k])
        Insert(node[k].lch,id);
    else
        Insert(node[k].rch,id);
}
void Insert(int id){
    Insert(root,id);
    check(root,id,0,MAXV);
}
#undef lch
#undef rch
}
namespace Tree2{
```

```
struct Node{
    int r,sz,ch[2];
}node[MAXN];
#define lch(k) node[node[k].ch[0]]
#define rch(k) node[node[k].ch[1]]
void push_up(int k){
    node[k].sz=lch(k).sz+rch(k).sz+1;
}
void split(int k,int &k1,int &k2,LL v){
    if(!k) return k1=k2=0,void();
    if(val[k]<=v){
        k1=k;
        split(node[k].ch[1],node[k1].ch[1],k2,v);
        push_up(k1);
    }else{
        k2=k;
        split(node[k].ch[0],k1,node[k2].ch[0],v);
        push_up(k2);
    }
}
void merge(int &k,int k1,int k2){
    if(!k1||!k2)return k=k1|k2,void();
    if(node[k1].r>node[k2].r){
        k=k1;
        merge(node[k].ch[1],node[k1].ch[1],k2);
        push_up(k);
    }else{
        k=k2;
        merge(node[k].ch[0],k1,node[k2].ch[0]);
        push_up(k);
    }
}
void Insert(int &root,int id){
    node[id].r=rand();
    node[id].sz=1;
    node[id].ch[0]=node[id].ch[1]=0;
    int lef,rig;
    split(root,lef,rig,val[id]);
    merge(lef,lef,id);
    merge(root,lef,rig);
}
int rank(int root,LL v){
    int lef,rig,ans;
    split(root,lef,rig,v-1);
    ans=node[lef].sz+1;
    merge(root,lef,rig);
    return ans;
}
int query(int root,LL vl,LL vr){
    return rank(root,vr)-rank(root,vl);
}
```

```
#undef lch
#undef rch
};
int root[MAXN],col[MAXN],hson[MAXN],dfr[MAXN];
void solve(){
    int n=1;
    col[1]=read_int();
    val[1]=0;
    val[0]=MAXV;
    Tree1::Insert(1);
    Tree2::Insert(root[col[1]],1);
    int m=read_int(),lastans=0;
    while(m--){
        int t=read_int()^lastans,u=read_int()^lastans,c=read_int()^lastans;
        if(t==1){
            int v=hson[u]?hson[u]:dfr[u];
            hson[u]=++n;
            col[n]=c;
            dfr[n]=v;
            val[n]=val[u]+val[v]>>1;
            Tree1::Insert(n);
            Tree2::Insert(root[c],n);
        }
        else{
            lastans=Tree2::query(root[c],val[u],val[dfr[u]]);
            enter(lastans);
        }
    }
    Tree1::root=0;
    _rep(i,1,n){
        root[col[i]]=0;
        hson[i]=dfr[i]=0;
    }
}
int main()
{
    int T=read_int();
    while(T--){
        solve();
        return 0;
    }
}
```

G. Hasse Diagram

题意

给一个根节点的值，然后它延伸出去的点的值都是这个值恰好除以它的一个素因子得到的值，我们把一个点的 $f(x)$ 函数值记做这个图的边数，求 $f(x)$ 的前缀和。

题解

题干给了 12^2 的图，可以观察到，如果只是一个 3^2 的图，也就是 3^2 和 1^2 两个点的图。

12^2 的图在这个基础上，可以分为 12^2 和 4^2 ， 6^2 和 2^2 这两组点，每组都和 3^2 的图一样，并且这些图之间有边相连，比如 12^2 连 6^2 ， 6^2 连 3^2 ， 4^2 连 2^2 ， 2^2 连 1^2 ，这些边的个数是 3^2 的因子数乘 12^2 除以 3^2 是 12^2 除以 6^2 的多少次幂。

如果看到这里，可以推出递推式 $f(n)=(e+1)f(\frac{n}{p^e})+e(\frac{n}{p^e})$

经过这道题才知道从 $f(x)$ 推出 $f(\frac{n}{p^e})$ 的结构也可以用 Min-25 筛来做，原来一直以为必须是积性函数的...

于是我们对板子魔改一下，变成同时求 f 和 d 的。 d 显然是积性函数，并且 $d(p)$ 和 $d(p^t)$ 都是很好算的，于是在筛中一起解决 f 用到 d 的结果，一并算出就结束了~

```
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<cmath>
#define ll long long
using namespace std;
const ll MOD=1145140019,inv2=(MOD+1)>>1,N=3e6,maxn=5e6;
ll prime[maxn],num,sp0[maxn];
ll n,Sqr,tot,g0[maxn],w[maxn],ind1[maxn],ind2[maxn];
bool flag[maxn];
void pre(int n) { //预处理，线性筛
    flag[1]=1;
    for(int i=1; i<=n; i++) {
        if(!flag[i]) {
            prime[++num]=i;
            sp0[num]=(sp0[num-1]+1)%MOD;
        }
        for(int j=1; j<=num&&prime[j]*i<=n; j++) {
            flag[i*prime[j]]=1;
            if(i%prime[j]==0)break;
        }
    }
}
pair<ll,ll> S(ll x,int y) { //第二部分
    if(prime[y]>=x)return make_pair(0,0);
    ll k=x<=Sqr?ind1[x]:ind2[n/x];
    ll ansf=(g0[k]-sp0[y]+MOD)%MOD,ansd=ansf*2%MOD;
    for(int i=y+1; i<=num&&prime[i]*prime[i]<=x; i++) {
        ll pe=prime[i];
        for(int e=1; pe<=x; e++,pe=pe*prime[i]) {
            pair<ll,ll> pp=S(x/pe,i);
            ansf=(ansf+1ll*(e+1)*pp.first%MOD+1ll*e*(pp.second+(e!=1))%MOD)%MOD;
            ansd=(ansd+(e+1)*(pp.second+(e!=1))%MOD);
        }
    }
}
```

```
ansf%=MOD;
ansd%=MOD;
return make_pair(ansf,ansd);
}
int main() {
    int t;
    scanf("%d",&t);
    pre(N);
    while(t--) {
        scanf("%lld",&n);
        Sqr=sqrt(n);
        tot=0;
        for(ll i=1; i<=n; i++) {
            ll j=n/(n/i);
            w[++tot]=n/i;
            g0[tot]=(w[tot]%MOD-1+MOD)%MOD;
            if(n/i<=Sqr)ind1[n/i]=tot;
            else ind2[n/(n/i)]=tot;
            i=j+1;
        }
        for(int i=1; i<=num; i++) {
            for(int j=1; j<=tot&&prime[i]*prime[i]<=w[j]; j++) {
                ll
k=w[j]/prime[i]<=Sqr?ind1[w[j]/prime[i]]:ind2[n/(w[j]/prime[i])];
                g0[j]=(g0[j]-(g0[k]-sp0[i-1])%MOD+MOD)%MOD;
            }
        }
        printf("%lld\n",S(n,0).first);
    }
    return 0;
}
```

J. Defend Your Country

题意

给定 n 个点 m 条边的连通图，要求删去任意条边，最大化图的点权和。

其中第 i 个点的点权的绝对值为 a_i 。如果该点所在连通块大小为偶数则权值为正，否则为负。

题解

不难发现如果 n 为偶数则不需要任何删边。

如果 n 为奇数，如果最优解中存在一个大小不为 1 的奇连通块，任取该连通块的一个点删去所有相关连边一定不会使得答案变劣。

因此不妨考虑强制删一个点，如果这个点不是割点，显然删除该点后不需要额外删点，统计此时的答案。

如果这个点是割点，则需要考虑删去这个点得到的剩余连通分量，如果每个连通分量大小都是偶数，显然也不需要额外删点。

否则，删去该点后得到至少两个奇连通分量，还要继续在奇连通分量中删点，事实上，这种策略一定不是最优的，下面给出证明：

假设这是最优策略，则所有删去的点一定是原图中的割点，否则如果存在非割点一开始删去该点才是最优。

另外，每次删去割点一定得到的都是奇连通块，否则考虑 (偶连通块-第二个割点-偶连通块)-第一个割点-奇连通块的情况。

其中原策略是删除第一个割点后得到奇连通块 (偶连通块-第二个割点-偶连通块)，再删除第二个割点。这样一定不如直接删去第二个割点。

于是由于每个割点删完后一定存在与他相邻的奇连通块，然后又要在奇数连通块中删割点，于是每个割点一定有两个相邻割点。

考虑在原图建立点双树，易知点双树的叶子割点一定没有两个相邻割点，矛盾。因此假设不成立。

至于维护删除割点后的其他连通分量奇偶性可以在跑 dfs 树时顺便维护子树大小，根据子树奇偶性判断。

特别注意即使 u 是割点，删除 u 也不能保证 u 的每个子树都构成独立连通分量。

事实上如果 $\text{low}[v] < \text{dfn}[u]$ 则说明 v 和 u 的祖先结点属于同一个点双连通分量，不要重复判定。比赛的时候就这里假了，长了个教训

时间复杂度 $O(n+m)$

```
const int MAXN=1e6+5,MAXM=2e6+5,Inf=1e9;
struct Edge{
    int to,next;
}edge[MAXM<<1];
int head[MAXN],edge_cnt;
void Insert(int u,int v){
    edge[++edge_cnt]=Edge{v,head[u]};
    head[u]=edge_cnt;
}
int low[MAXN],dfn[MAXN],sz[MAXN],dfs_t;
bool iscut[MAXN],fib[MAXN];
void dfs(int u,int fa){
    low[u]=dfn[u]=++dfs_t;
    int child=0;
    sz[u]=1;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==fa)continue;
        if(!dfn[v]){
            dfs(v,u);
            sz[u]+=sz[v];
            if(sz[v]%2&&low[v]>=dfn[u])
                fib[u]=true;
        }
    }
}
```

```
        low[u]=min(low[u],low[v]);
        if(low[v]>=dfn[u]&&u!=fa)
            iscut[u]=true;
        if(u==fa)child++;
    }
    low[u]=min(low[u],dfn[v]);
}
if(u==fa&&child>=2)
    iscut[u]=true;
}
int a[MAXN];
void solve(){
    int n=read_int(),m=read_int();
    LL ans=0;
    edge_cnt=0,dfs_t=0;
    _rep(i,1,n){
        a[i]=read_int();
        head[i]=0;
        dfn[i]=0;
        iscut[i]=0;
        fib[i]=0;
        ans+=a[i];
    }
    while(m--){
        int u=read_int(),v=read_int();
        Insert(u,v);
        Insert(v,u);
    }
    if(n%2==0){
        enter(ans);
        return;
    }
    dfs(1,1);
    int det=Inf;
    _rep(i,1,n){
        if(!iscut[i])
            det=min(det,a[i]);
        else if(!fib[i])
            det=min(det,a[i]);
    }
    enter(ans-det*2);
}
int main(){
    int T=read_int();
    while(T--){
        solve();
    }
    return 0;
}
```

K. Starch Cat

题意

给定一棵点权树，每次询问一条路径的最大权值，强制在线。其中，路径的权值定义为该路径上的最大权点集，满足点集中所有点都不相邻。

题解

考虑建立点分树。对固定的根节点 rt 设 $\text{dp}(u, 0/1, 0/1)$ 表示从根节点出发到 u 的路径，选/不选根节点以及选/不选 u 结点得到的最大权值。

不难得到状态转移方程，进一步，设 $f(d, u, 0/1)$ 表示 u 在点分树上根节点深度为 d 的结点到 u 的路径，选/不选根节点的最大权值。

最后对于询问操作，可以找到 u, v 在点分树上的 LCA 设 LCA 在点分树的深度为 d 则答案为

$$\max(f(d, u, 0) + f(d, v, 0), f(d, u, 1) + f(d, v, 1) - a_{\text{LCA}})$$

时间复杂度 $O((n+m)\log n)$ 尽管 m 很大，但常数很小而且时限比较宽松，所以能过。

```
struct Rand{
    unsigned int n, seed;
    Rand(unsigned int n, unsigned int seed)
    : n(n), seed(seed){}
    int get(long long lastans){
        seed ^= seed << 13;
        seed ^= seed >> 17;
        seed ^= seed << 5;
        return (seed ^ lastans) % n + 1;
    }
};

const int MAXN = 5e5 + 5, MAXD = 20;
const LL inf = 1e15;
struct Edge{
    int to, next;
} edge[MAXN << 1];
int head[MAXN], edge_cnt;
void Insert(int u, int v){
    edge[++edge_cnt] = Edge{v, head[u]};
    head[u] = edge_cnt;
}

int sz[MAXN], mson[MAXN], tot_sz, root, root_sz;
bool vis[MAXN];
int a[MAXN], rt[MAXD][MAXN];
LL dp[MAXN][2][2], dp2[MAXD][MAXN][2];
void dfs(int u, int fa, int dep){
```

```
rt[dep][u]=root;
if(fa==0){
    dp[u][0][0]=0;
    dp[u][1][1]=a[u];
    dp[u][1][0]=dp[u][0][1]=-inf;
}
else{
    dp[u][0][0]=max(dp[fa][0][0],dp[fa][0][1]);
    dp[u][0][1]=dp[fa][0][0]+a[u];
    dp[u][1][0]=max(dp[fa][1][0],dp[fa][1][1]);
    dp[u][1][1]=dp[fa][1][0]+a[u];
}
dp2[dep][u][0]=max(dp[u][0][0],dp[u][0][1]);
dp2[dep][u][1]=max(dp[u][1][0],dp[u][1][1]);
for(int i=head[u];i;i=edge[i].next){
    int v=edge[i].to;
    if(vis[v]||v==fa)
        continue;
    dfs(v,u,dep);
}
}
void solve2(int u,int dep){
    dfs(u,0,dep);
}
void find_root(int u,int fa){
    sz[u]=1;mson[u]=0;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(vis[v]||v==fa)
            continue;
        find_root(v,u);
        sz[u]+=sz[v];
        mson[u]=max(mson[u],sz[v]);
    }
    mson[u]=max(mson[u],tot_sz-sz[u]);
    if(mson[u]<root_sz){
        root=u;
        root_sz=mson[u];
    }
}
void solve(int u,int dep){
    int cur_sz=tot_sz;
    vis[u]=true;
    solve2(u,dep);
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(vis[v])
            continue;
        tot_sz=sz[v]>sz[u]?cur_sz-sz[u]:sz[v];
        root_sz=MAXN;
    }
}
```

```

        find_root(v,u);
        solve(root,dep+1);
    }
}
LL query(int u,int v){
    int d=0;
    while(rt[d][u]&&rt[d][u]==rt[d][v])d++;
    d--;
    return max(dp2[d][u][0]+dp2[d][v][0],dp2[d][u][1]+dp2[d][v][1] -
a[rt[d][u]]);
}
int main(){
    int n=read_int(),m=read_int(),seed=read_int();
    _rep(i,1,n)a[i]=read_int();
    _rep(i,2,n){
        int f=read_int();
        Insert(f,i);
        Insert(i,f);
    }
    root_sz=MAXN;
    tot_sz=n;
    find_root(1,0);
    solve(root,0);
    long long lastans=0,ans=0;
    constexpr int P=998244353;
    Rand rand(n,seed);
    for(int i=0;i<m;i++){
        int u=rand.get(lastans);
        int v=rand.get(lastans);
        int x=rand.get(lastans);
        lastans=query(u,v);//calculate the answer
        ans=(ans+lastans%P*x)%P;
    }
    enter(ans);
    return 0;
}

```

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

Permanent link:
https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:%E7%BB%84%E9%98%9F%E8%AE%AD%E7%BB%83%E6%AF%94%E8%B5%9B%E8%AE%B0%E5%BD%95:contest10

Last update: 2021/08/11 16:00

