题解

food

不难发现题目所求即为$ans_k=\prod_{i=1}^{n}C_k^{a_i}$

【30%】

直接循环暴力即可

【100%】

考虑到$\sum a_i<10^5$，因此可知不同种类的a_i最多也只会有$\sqrt{10^5}$种，因此将相同的a_i统计次数，利用快速幂一起计算，直观复杂度为$O(n\sqrt{n}logn)$，实际上可以证明时间复杂度为$O(n\sqrt{n})$

```
#include<bits/stdc++.h>
#define N 110000
#define db double
#define ll long long
#define ldb long double
#define ull unsigned long long
using namespace std;
const int h=3,ki=149,mo=998244353;
int mod(int x){return (x%mo+mo)%mo;}
int inc(int x,int k){x+=k;return x<mo?x:x-mo;}
int dec(int x,int k){x-=k;return x>=0?x:x+mo;}
int ksm(int x,int k)
{
  int ans=1;
  while(k){if(k&1)ans=1ll*ans*x%mo;k>>=1;x=1ll*x*x%mo;}
  return mod(ans);
}
int inv(int x){return ksm(x,mo-2);}
int read()
{
  char ch=0;int x=0,flag=1;
  while(!isdigit(ch)){ch=getchar();if(ch=='-')flag=-1;}
  while(isdigit(ch)){x=(x<<3)+(x<<1)+ch-'0',ch=getchar();}
  return x*flag;
}
void write(int x)
{
  if(!x)return (void)putchar(48);
  if(x<0)putchar(45),x=-x;
  int len=0,p[20];
  while(x)p[++len]=x%10,x/=10;
  for(int i=len;i>=1;i--)putchar(p[i]+48);
}
```

```
void writeln(int x){write(x),putchar('\n');}
int a[N],cnt[N],fac[N],vac[N];
int C(int n,int m)
{
  if(n<0||n<m||n<m)return 0;
  return 1ll*fac[n]*vac[m]%mo*vac[n-m]%mo;
}
int main()
{
  int n=read(),m=read();
  for(int i=1;i<=n;i++)a[i]=read();
  sort(a+1,a+n+1);

  int tot=0;
  a[0]=-1;
  for(int i=1;i<=n;i++)
  if(a[i]==a[i-1])cnt[tot]++;
  else a[++tot]=a[i],cnt[tot]=1;

  fac[0]=1;
  for(int i=1;i<=m;i++)fac[i]=1ll*fac[i-1]*i%mo;
  vac[m]=inv(fac[m]);
  for(int i=m;i>=1;i--)vac[i-1]=1ll*vac[i]*i%mo;

  for(int i=1;i<=m;i++)
  {
    int x=1;
    for(int j=1;j<=tot;j++)x=1ll*x*ksm(C(i,a[j]),cnt[j])%mo;
    writeln(x);
  }
  return 0;
}
```

history

【100%】

尝试迭代函数$f(x)$，可以发现$f(x)$迭代四次后变为x本身。因此求解最短路时只需要考虑额外记录当前步数$mod\ 4$的结果即可。分层图最短路即可解决。

```
#include<bits/stdc++.h>
#define T 4
#define N 220000
#define M 440000
#define db double
#define ll long long
#define ldb long double
#define ull unsigned long long
using namespace std;
```

```
ll mo;
const double pi=acos(-1),eps=1e-7,inf=1e18+7;
ll mod(ll x){return (x%mo+mo)%mo;}
ll inc(ll x,ll k){x+=k;return x<mo?x:x-mo;}
ll dec(ll x,ll k){x-=k;return x>=0?x:x+mo;}
ll ksm(ll x,ll k)
{
  ll ans=1;
  while(k){if(k&1)ans=1ll*ans*x%mo;k>>=1;x=1ll*x*x%mo;}
  return mod(ans);
}
ll inv(ll x){return ksm(mod(x),mo-2);}
ll read()
{
  char ch=0;ll x=0,flag=1;
  while(!isdigit(ch)){ch=getchar();if(ch=='-')flag=-1;}
  while(isdigit(ch)){x=(x<<3)+(x<<1)+ch-'0',ch=getchar();}
  return x*flag;
}
void write(ll x)
{
  if(!x)return (void)putchar(48);
  if(x<0)putchar(45),x=-x;
  ll len=0,p[20];
  while(x)p[++len]=x%10,x/=10;
  for(ll i=len;i>=1;i--)putchar(p[i]+48);
}
void writeln(ll x){write(x),putchar('\n');}
struct edge{ll to,nxt,w;}e[M];
ll num,head[N];
inline void add(ll x,ll y,ll z){e[++num]={y,head[x],z};head[x]=num;}
struct node{ll x,t,d;};
bool operator<(node a,node b){return a.d>b.d;}
priority_queue<node>q;
ll dis[N][T];
ll f(ll x,ll t)
{
  for(ll i=1;i<=t;i++)x=1ll*(1+x)*inv(1-x)%mo;
  return x;
}
void dijkstra(ll n,ll m,ll rt)
{
  for(ll i=0;i<=n;i++)for(ll t=0;t<4;t++)dis[i][t]=inf;
  dis[rt][0]=0;
  q.push((node){rt,0,dis[rt][0]});
  while(!q.empty())
  {
    node k=q.top();
    ll x=k.x,t=k.t;q.pop();
```

```
    if(k.d!=dis[x][t])continue;
    for(ll i=head[x];i!=-1;i=e[i].nxt)
    {
      ll to=e[i].to,tt=(t+1)%4;
      if(dis[to][tt]>dis[x][t]+f(e[i].w,t))
      dis[to][tt]=dis[x][t]+f(e[i].w,t),q.push((node){to,tt,dis[to][tt]});
    }
  }
}
int main()
{
  ll n=read(),m=read();mo=read();
  memset(head,-1,sizeof(head));
  for(ll i=1;i<=m;i++){ll x=read(),y=read(),z=read();add(x,y,z);}
  dijkstra(n,m,1);
  ll ans=inf;
  for(ll i=0;i<4;i++)ans=min(ans,dis[n][i]);
  if(ans!=inf)write(ans);else write(-1);
  return 0;
}
```

tree

【40%】

由于$k \leq 30$，所以我们可以通过预处理每一层子树的大小，每次从根开始往下跳，二分找到往哪个叶子上跳，来实现一个点标号$O(k \log n)$的定位。

而如果已知两个点在每一层子树所对应的叶子节点，则可以用类似于倍增求lca的方法，每次先把所在层数较深的点往上跳，跳到同一颗子树时再用传统方法求$lca/distance$，复杂度$O(k+\log n)$。

【100%】

由于点的标号小于等于10^9，而如果T_0有至少两个叶子的话，30天过后整颗树的大小就超过10^9了，所以k很大时是没用的，只有最靠左下的30层子树和它往上到根的一条链是有用的。

记录一下链长度，套用【40%】做法即可。

```
#include<iostream>
#include<cstdio>

using namespace std;

const int maxn = 1e5+5;
vector<int> G[maxn];
int h[maxn], F[maxn][25], leaf_x[40], leaf_y[40], w[maxn], fa[maxn];
long long g[40], link_len;
int k, q, n, leaf_cnt, max_h, ans, cnt;
vector<int> leaf;
```

```
void dfs(int x, int fa) {
    w[x] = cnt++;
    for (int i = 0; i < G[x].size(); ++i) {
        int y = G[x][i];
        h[cnt] = h[w[x]] + 1;
        dfs(y, x);
    }
    if (G[x].size() == 0) leaf.push_back(w[x]);
}

void calc(int x, int &L_x, int &v_x, int leaf_x[]) {
    if (x <= link_len) {
        ans += link_len - x + 1;
        x = 1;
    } else {
        x -= link_len;
    }
    while (L_x <= max_h) {
        int ll = 0, rr = leaf_cnt - 1;
        while (ll <= rr) {
            int mid = (rr + ll) / 2;
            if (x <= leaf[mid] + 1 + (mid + 1) * (g[max_h - L_x] - 1)) rr = mid - 1;
            else ll = mid + 1;
        }
        if (x <= leaf[ll] + 1 + ll * (g[max_h - L_x] - 1)) {
            v_x = x - ll * (g[max_h - L_x] - 1) - 1;
            break;
        }
        x -= leaf[ll] + ll * (g[max_h - L_x] - 1);
        leaf_x[L_x++] = leaf[ll];
    }
}

int dis(int x, int y) {
    long long S = h[x] + h[y];
    if (h[x] < h[y]) swap(x, y);
    for (int i = 19; i >= 0; --i) {
        if (h[F[x][i]] >= h[y]) x = F[x][i];
    }
    if (x == y) return S - 2 * h[x];
    for (int i = 19; i >= 0; --i) {
        if (F[x][i] != F[y][i]) {
            x = F[x][i];
            y = F[y][i];
        }
    }
    return S - 2 * (h[x] - 1);
}
```

```
int main() {
    cin >> n;
    for (int i = 1; i < n; ++i) {
        cin >> fa[i];
        G[fa[i]].push_back(i);
    }
    cin >> k >> q;
    h[0] = 0;
    int cnt = 0;
    dfs(0, 0);

    F[0][0] = 0;
    for (int i = 1; i < n; ++i) F[w[i]][0] = w[fa[i]];
    for (int j = 1; j < 20; ++j) {
        for (int i = 1; i <= n; ++i) {
            F[i][j] = F[F[i][j-1]][j-1];
        }
    }

    leaf_cnt = leaf.size();
    if (leaf_cnt != 1) {
        g[0] = 1;
        max_h = 1;
        for (; max_h <= k && g[max_h - 1] <= 1e9; ++max_h) {
            g[max_h] = n + leaf_cnt * (g[max_h - 1] - 1);
        }
        --max_h;
        link_len = 1LL * (k - max_h) * leaf[0];
    }

    while (q--) {
        int x, y;
        cin >> x >> y;
        ans = 0;
        if (leaf_cnt == 1 || x <= link_len && y <= link_len) {
            cout << abs(x-y) << endl;
            continue;
        }

        int Lx = 0, v_x;
        calc(x, Lx, v_x, leaf_x);
        int Ly = 0, v_y;
        calc(y, Ly, v_y, leaf_y);

        bool flag = false;
        for (int i = 0; i <= max(Lx, Ly); ++i) {
            if (flag) {
                if (i < Lx) ans += h[leaf_x[i]];
                if (i == Lx) ans += h[v_x];
```

```
                if (i < Ly) ans += h[leaf_y[i]];
                if (i == Ly) ans += h[v_y];
            } else {
                flag = true;
                if (i >= Lx && i >= Ly) {
                    ans += dis(v_x, v_y);
                } else if (i >= Lx) {
                    ans += dis(v_x, leaf_y[i]);
                } else if (i >= Ly) {
                    ans += dis(leaf_x[i], v_y);
                } else if (leaf_x[i] != leaf_y[i]) {
                    ans += dis(leaf_x[i], leaf_y[i]);
                } else {
                    flag = false;
                }
            }
        }
        cout << ans << endl;
    }
    return 0;
}
```

stone

【100%】

$dp(l,r)$表示$[a_l,a_{l+1}...a_r]$的胜负情况

$dp(l,r)=1$时，我们定义f函数和g函数。

$f(l,r)=x$表示唯一x使得$[x,a_l,a_{l+1}...a_r]$为必败态

$g(l,r)=y$表示唯一y使得$[a_l,a_{l+1}...a_r,y]$为必败态

我们考虑基于

$dp(l+1,r-1)$，$dp(l+1,r)$，$dp(l,r-1)$

$f(l+1,r-1)$，$f(l+1,r)$，$f(l,r-1)$

$g(l+1,r-1)$，$g(l+1,r)$，$g(l,r-1)$

推导$dp(l,r),f(l,r),g(l,r)$

我们直接考虑最为困难的情况$dp(l+1,r-1)=dp(l+1,r)=dp(l,r-1)=1$

在进行讨论之前，我们先引入一个问题。

有两堆石子，数量分别为$x,y>0$，给定两个参数$p,q>0$

双方轮流操作，每次可以从恰好一堆中取出任意数量的石子。

当其中任意一堆为空时游戏结束。

胜负判定条件为：

当且仅当$x=p,y=0$或者$x=0,y=q$，最后操作的人获胜。

这个问题通过画出二维的一个矩阵图稍加讨论便可以解决。

不妨设$p\leq q$，答案为$ans(x,y)$

首先给出一个结论，对于任意x，存在唯一y使得$ans(x,y)=0$

同理，对于任意y存在唯一x使得$ans(x,y)=0$

对于这个问题，我们可以$O(1)$求解。

此外，我们还可以$O(1)$解决，给定x，求y使得$ans(x,y)=0$这个问题。

然后我们回到原来的问题。

我们考虑先计算$dp(l,r)$

这个很容易，由于我们已经已知了$f(l+1,r-1)$和$g(l+r,r-1)$

因此$[f(l+1,r-1),a_{l+1}\dots a_{r-1}]$和$[a_{l+1}\dots a_{r-1},g(l+r,r-1)]$为必败态。

而对于其余任意$i,j>0$，$[i,a_{l+1}\dots a_{r-1}]$，$[a_{l+1}\dots a_{r-1},j]$均为必胜态。

因此这个问题便转化为了上面提到的那个问题。

我们令$x=a[l],y=a[r],p=f(l+1,r-1),q=g(l+1,r-1)$

即可计算出$dp(l,r)$

考虑如何计算$f(l,r)$和$g(l,r)$

我们不妨以$f(l,r)$为例。

此时我们可知$[f(l,r-1),a_l\dots a_{r-1}]$和$[a_l\dots a_{r-1},g(l,r-1)]$为必败态。

而对于其余任意$i,j>0$，$[i,a_l\dots a_{r-1}]$，$[a_l\dots a_{r-1},j]$均为必胜态。

于是问题又转化成了一个与上面提到的问题类似的问题。

我们令$y=a[r],p=f(l,r-1),q=g(l,r-1)$

求解x满足$ans(x,y)=0$，这个x便是$f(l,r)$

同理我们可以计算出$g(l,r)$

最后$dp(l,r)=0$的情况，由于没有$f(l,r)$和$g(l,r)$的定义，比较暴力的想法是进行类似的分类讨论。

但实际上我们可以直接令$f(l,r)=g(l,r)=+\infty$，然后递推规则不变即可。

```
#include<bits/stdc++.h>
#define N 1100
#define db double
#define ll long long
#define ldb long double
using namespace std;
int read()
{
  char ch=0;
```

```
  int x=0,flag=1;
  while(!isdigit(ch)){ch=getchar();if(ch=='-')flag=-1;}
  while(isdigit(ch)){x=(x<<3)+(x<<1)+ch-'0';ch=getchar();}
  return x*flag;
}
bool dp[N][N];
int a[N],f[N][N],g[N][N];
bool cal(int x,int y,int p,int q)
{
  if(p>q)swap(x,y),swap(p,q);
  if(x==p)return true;
  if(y==q)return true;
  if(x<p||y>q)return x^y;
  return x^(y+1);
}
int get(int k,int p,int q,bool fg)
{
  if(p>q)fg^=1,swap(p,q);
  if(!fg)
  {
    if(k==p)return 0;
    if(k<p||k>q)return k;
    return k-1;
  }
  else
  {
    if(k==q)return 0;
    if(k<p||k>q)return k;
    return k+1;
  }
}
const int inf=1e9+7;
void work()
{
  int n=read();
  for(int i=1;i<=n;i++)a[i]=read();
  for(int i=1;i<=n;i++)
  {
    dp[i][i]=true;
    f[i][i]=g[i][i]=a[i];
  }
  for(int l=n;l>=1;l--)
  for(int r=l+1;r<=n;r++)
  {
    if(r-l==1)dp[l][r]=a[l]^a[r];
    else dp[l][r]=cal(a[l],a[r],f[l+1][r-1],g[l+1][r-1]);
    if(dp[l][r])
    {
      f[l][r]=get(a[r],f[l][r-1],g[l][r-1],1);
```

```
      g[l][r]=get(a[l],f[l+1][r],g[l+1][r],0);
    }
    else f[l][r]=g[l][r]=inf;
  }
  if(dp[1][n])printf("YES\n");
  else printf("NO\n");
}
int main()
{
  int t=read();
  for(int i=1;i<=t;i++)work();
  return 0;
}
```