提高组5

万松园

题解

该题的核心是对询问离线处理。不难发现，能够保留的边数随着阈值的减小而增加，而且阈值较大能够保留的边在阈值较小时也能保留。如果将询问按照阈值的递减顺序排序，问题就变成了动态加边，询问图中某个点所在的连通块的大小。这可以使用并查集很方便地解决。使用一个记录集合大小的并查集维护即可。

标准代码

C++

```cpp
#include <algorithm>
#include <iostream>
#include <cstring>
#include <cstdio>
#include <string>
#include <vector>

#define forall(G,i) for (int i = 0, __for_siz__ = (int) (G).size(); i < __for_siz__; ++i)
#define DEBUG(x) std::cerr << #x << " = " << x << std::endl;
#define ALL(x) (x).begin(), (x).end()
#define MP std::make_pair
#define se second
#define fi first

using std::cin;
using std::cout;
using std::endl;

inline int read() {
    int s = 0, x = 1; char ch = getchar();
    while (!isdigit(ch)) { if (ch == '-') x = -x; ch = getchar(); }
    while (isdigit(ch)) { s = s * 10 + ch - '0'; ch = getchar(); }
    return s * x;
}

const int MAXN = 100000 + 10;
int n, q;

struct REdge {
    int u, v, w;

    REdge(int _u = 0, int _v = 0, int _w = 0) :
        u(_u), v(_v), w(_w) {}
} es[MAXN], qry[MAXN];

bool cmp1(REdge r1, REdge r2) {
    return r1.w > r2.w;
}
bool cmp2(REdge r1, REdge r2) {
    return r1.w < r2.w;
}

struct DSU {
```

```cpp
    int u[MAXN]; int siz[MAXN];

    void Init(int n) {
        for (int i = 1; i <= n; ++i) siz[i] = 1;
    }
    int Find(int x) { return !u[x] ? x : u[x] = Find(u[x]); }
    int Size(int x) { return siz[Find(x)];}
    bool Merge(int x, int y) {
        x = Find(x); y = Find(y);
        if (x == y) return false;
        u[x] = y; siz[y] += siz[x];
        siz[x] = 0; return true;
    }
} U;

int anss[MAXN];

int main() {
    n = read(); q = read();
    U.Init(n);
    for (int i = 1; i <= n - 1; ++i) {
        int u = read(); int v = read(); int w = read();
        es[i] = REdge(u, v, w);
    }
    for (int i = 1; i <= q; ++i) {
        int k = read(); int u = read();
        qry[i] = REdge(u, i, k);
    } std::sort(es + 1, es + 1 + n - 1, cmp1);
    std::sort(qry + 1, qry + 1 + q, cmp1);

    int ii = 1;

    for (int iq = 1; iq <= q; ++iq) {
        int k = qry[iq].w;
        for (; ii <= n - 1; ++ii) {
            if (es[ii].w >= k) {
                U.Merge(es[ii].u, es[ii].v);
            } else break;
        }
        anss[qry[iq].v] = U.Size(qry[iq].u);
    }
    for (int i = 1; i <= q; ++i) printf("%d\n", anss[i] - 1);
    return 0;
}
```

k进制

题解

普通的做法考虑模拟，对于每个 $2,3$ 询问进行暴力排序，每次查询答案时从后往前扫一遍统计答案即可，时间复杂度 $O(n^2\log n)$

对于随机数据在吸氧情况下是可以跑过去的，所以本题暴力最多可以拿到 30 分。

对于 $k=2$ 的情况，可以使用线段树进行区间推平并统计区间和，设区间 $[l,r]$ 的和为 sum ，先将 $[l,r]$ 全部赋值为 0 。升序时将 $[l,l+sum-1]$ 赋值为 0 ， $[l+sum,r]$ 赋值为 1 ，降序同理，可以在 $O(n\log n)$ 的时间复杂度内完成。

正解是 $k=2$ 情况的拓展，设 $t_{x,i}$ 表示在当前节点 x 统计的区间中， i 出现的次数，在升序的时候就分别修改 $[l,l+t_{x,0}-1]$ ， $[l+t_{x,0},l+t_{x,0}+t_{x,1}-1]......$ 以此类推，降序同理。设 s_x 表示节点 x 所统计这段区间转成十进制的结果。在从儿子合并时 $s_x=s_{ls(x)}\times k^{r-mid}+s_{rs(x)}$ ，在推平当前区间为 z 时 $s_x=(\sum_{i=0}^{r-l}k^i)\times z$ ，其中 $\sum_{i=0}^{r-l}k^i$ 和 k^{r-mid} 都可以事先预处理，所以最终单次操作的时间复杂度是 $O(k\log n)$ ，最终复杂度 $O(mk\log n)$

标准代码

C++

```
#include<cstdio>
#include<cstring>
#define N 50010
#define P 998244353
#define ll long long
#define ls(x) (x<<1)
#define rs(x) (x<<1|1)
using namespace std;
int n,k,m;
char s[N];
int t[N<<2][10],tag[N<<2],g[10];
ll ans;
ll sum[N<<2],p[N],a[N];
inline void read(int &g)
{
    g=0;register char c=getchar();
    while (c<'0' || c>'9') c=getchar();
    while (c>='0' && c<='9') g=(g<<1)+(g<<3)+(c&15),c=getchar();
}
inline void update(int x,int l,int r)
{
    for (int i=0;i<k;++i) t[x][i]=t[ls(x)][i]+t[rs(x)][i];
    int mid=(l+r)>>1;
    sum[x]=((sum[ls(x)]*a[r-mid]%P)+sum[rs(x)])%P;
}
inline void pushdown(int x,int l,int r)
{
    if (tag[x]==-1) return;
    int o=tag[x],mid=(l+r)>>1;
    for (int i=0;i<k;++i) t[ls(x)][i]=t[rs(x)][i]=0;
    t[ls(x)][o]=mid-l+1,t[rs(x)][o]=r-mid;
    tag[ls(x)]=tag[rs(x)]=o;
    sum[ls(x)]=p[mid-l]*(ll)o%P;
    sum[rs(x)]=p[r-mid-1]*(ll)o%P;
    tag[x]=-1;
}
void modify(int x,int l,int r,int X,int Y,int Z)
{
    if (X>Y) return;
    if (l>Y || r<X) return;
    if (X<=l && r<=Y)
    {
        sum[x]=p[r-l]*(ll)Z%P;
        for (int i=0;i<k;++i) t[x][i]=0;
```

```
        t[x][Z]=r-l+1;
        tag[x]=Z;
        return;
    }
    pushdown(x,l,r);
    int mid=(l+r)>>1;
    modify(ls(x),l,mid,X,Y,Z);
    modify(rs(x),mid+1,r,X,Y,Z);
    update(x,l,r);
}
void query(int x,int l,int r,int X,int Y,int wz)
{
    if (l>Y || r<X) return;
    if (X<=l && r<=Y){g[wz]+=t[x][wz];return;}
    pushdown(x,l,r);
    int mid=(l+r)>>1;
    query(ls(x),l,mid,X,Y,wz);
    query(rs(x),mid+1,r,X,Y,wz);
}
void getans(int x,int l,int r,int X,int Y)
{
    if (l>Y || r<X) return;
    if (X<=l && r<=Y){ans=(ans+(sum[x]*a[Y-r]%P))%P;return;}
    pushdown(x,l,r);
    int mid=(l+r)>>1;
    getans(ls(x),l,mid,X,Y);
    getans(rs(x),mid+1,r,X,Y);
}
int main()
{
    read(n),read(m),read(k);
    scanf("%s",s+1);
    memset(t,0,sizeof t);
    memset(tag,-1,sizeof tag);
    a[0]=p[0]=1;
    for (int i=1;i<=n;++i) a[i]=a[i-1]*k%P,p[i]=p[i-1]+a[i];
    for (int i=1;i<=n;++i) modify(1,1,n,i,i,(s[i]&15));
    while (m--)
    {
        register int op,x,y;
        read(op),read(x),read(y);
        if (op==1) modify(1,1,n,x,x,y);
        if (op==2)
        {
```

```
            memset(g,0,sizeof g);
            for (int i=0;i<k;++i) query(1,1,n,x,y,i);
            int L=x,R;
            for (int i=0;i<k;++i) R=L+g[i]-1,modify(1,1,n,L,R,i),L=R+1;
        }
        if (op==3)
        {
            memset(g,0,sizeof g);
            for (int i=0;i<k;++i) query(1,1,n,x,y,i);
            int L=x,R;
            for (int i=k-1;i>=0;--i) R=L+g[i]-1,modify(1,1,n,L,R,i),L=R+1;
        }
        if (op==4) ans=0,getans(1,1,n,x,y),printf("%lld\n",ans);
    }
    return 0;
}

```

喵喵与Rin

题解

我们把一个喵喵和他讨厌的喵喵连边，很显然这会构成一个由基环树组成的森林。

我们对这个基环树森林进行树形 dp ，对于每一棵基环树，任意相连的两个点不能同时取。

众所周知，要使一棵基环树转化为一棵树，必须要断掉其环上的一条边。

我们考虑环上的任意一条边，它的两个端点至少有一个不取。则分别以两个端点为根进行树形 dp ，分别存在 f 数组和 g 数组中， $f[i][0/1]$ 和 $g[i][0/1]$ 表达以点 i 为根的子树中，且当前节点 i 不取/取 的可爱值。

我们可以发现，对于每一条环上的边，它的贡献相同

令任意边两端点分别为 ui,vi

则对每个基环树（联通块）将答案累加上 $max(f[ui][0],g[vi][0])$ 即可。

注意要开 $longlong$ 。

标准代码

C++

```
#include <cstdio>
#include <cstdlib>
#include <algorithm>
#include <queue>
using namespace std;
struct knight{
    knight *p; int goin; long long f[2];
    inline void renew(const knight* x){
        f[0]+=max(x->f[0],x->f[1]);
        f[1]+=x->f[0];
        goin--;
    }
} no[1000001],*i,*j;
int N; long long f[2][2],tmp,Ans;
queue<knight*> q;
int main(){
    scanf("%u",&N);
    for(i=no;i<no+N;i++){
        static int P;
        scanf("%u%u",&i->f[1],&P);
        i->p=no+--P;
        no[P].goin++;
    }
    for(i=no;i<no+N;i++) if(!i->goin) q.push(i);
    while(!q.empty()){
        i=q.front();
        q.pop();
        i->p->renew(i);
        if(!i->p->goin) q.push(i->p);
    }
    for(i=no;i<no+N;i++) if(i->goin){
        i->goin--;
        f[0][0]=i->f[0];
        f[0][1]=f[1][0]=0;
        f[1][1]=i->f[1];
        for(j=i->p;j!=i;j=j->p){
            for(int b=0;b<2;b++){
                tmp=f[b][0];
                f[b][0]=max(tmp,f[b][1])+j->f[0];
                f[b][1]=tmp+j->f[1];
            }
            j->goin--;
        }
        Ans+=max(f[1][0],max(f[0][0],f[0][1]));
```

```
    }
    printf("%lld\n",Ans);
}


```

同态计数

题解

设 C 是可逆矩阵，映射 $f_C(X)=C^{-1}XC$ 显然是一个同态。我们断言，所有的同态要么是平凡同态 $\forall X, f(X)=0$ ，要么形如 f_C 。为了说明这一点，我们看：

如果你懂很多代数

注意到 $M_n(\mathbb{F}_p)$ 是单环，那么它上面的同态只能是平凡同态和同构，再注意到它是一个中心单代数，根据 Skolem-Noether 定理它的同构必然是内的。

如果你只懂微小的代数

我们当然只处理不平凡的同态，取 $\mathbb{Z}_p$ 上的 n 维向量空间 V ，取定一组基，把矩阵解释成线性算子。我们看 $P_i=f(E_{i,i})$ ，不难验证 $V=\bigoplus_i P_iV$ ，并且 P_i 是投影算子，于是 $\dim P_iV=1$ 。设 $P_1V=<A_1>$ ，$A_i=f(E_{i,1})A_1$ ，把 A_i 们在基下的坐标取成列向量，再把列向量排成行构成一个矩阵 A ， A^{-1} 即为所求矩阵。

接下来，我们只需要求不同的 f_C 的个数，注意到 $\forall X, C^{-1}XC=A^{-1}XA \Leftrightarrow \forall X, AC^{-1}X=XAC^{-1}$ ，也即 AC^{-1} 与所有矩阵乘积可交换，于是它是纯量阵，也即 $A=\lambda C$ 。故而 $f_C=f_A \Leftrightarrow A=\lambda C$ 。于是映射的个数就是全体 n 阶可逆矩阵 GL_n 的个数除以 $p-1$ 。为了计算 GL_n 的大小，注意到矩阵可逆等价于矩阵满秩，取 $\mathbb{Z}_p$ 上的 n 维坐标向量空间 V ，取定矩阵第一行 β_1 （把它看成一个向量），共有 p^n-1 种取法，而第二行可以取 $V\setminus<\beta_1>$ 里的所有向量，也即有 p^n-p 种取法，接下来也可以这样处理第三行。如此做下去，就可以算出来 GL_n 的大小为 $\prod_{i=0}^{n-1}(p^n-p^i)$ ，除以 $p-1$ 就是答案。

标准代码

C++

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

using namespace std;

typedef long long ll;

const int MAXN=1e5+5;
const ll MOD=1e9+7;

ll power[MAXN];
ll p,n;

ll Fast(ll a,ll b,ll mod)
{
    ll ans = 1;
    while(b){
        if(b&1){
            b--;
            ans=ans*a%mod;
        }
        else{
            a=a*a%mod;
            b/=2;
        }
    }
    return ans;
}

ll Rev(ll o)
{
    return Fast(o,MOD-2,MOD);
}

void First()
{
    power[0]=1;
    for(int i=1;i<=1e5;i++)
        power[i]=power[i-1]*p%MOD;
}

void Solve()
{
    ll ans=1;
    for(int i=0;i<n;i++)
```

```
            ans=(ans*((power[n]-power[i]+MOD)%MOD)%MOD);
        cout<<(ans*Rev(p-1)+1)%MOD<<endl;
    }

    void ReadData()
    {
        cin>>p>>n;
    }

    int main()
    {
        ReadData();
        First();
        Solve();
        return 0;
    }
```