GYM 链接：Matrix Exponentiation

F - Min Path

给一个无向图，求包含 k 条边的最小路径。

考虑 dp定义 $f[i][j][k]$ 为从第 i 个点出发经过 k 条边到第 j 个点的最短路。则会有如下的转移方程

$$f[i][j][k] = min_u\{f[i][u][k-1] + g[u][j]\}$$

但 k 值很大，所以不能直接dp因为取最小值的操作和乘法都满足结合律交换律，且上述形式和矩阵乘法类似，所以可以把矩阵乘法改写成上面的转移方程，再用矩阵快速幂加速。

```
#include<bits/stdc++.h>
#define ALL(x) (x).begin(),(x).end()
#define ll long long
#define db double
#define ull unsigned long long
#define pii_ pair<int,int>
#define mp_ make_pair
#define pb push_back
#define fi first
#define se second
#define rep(i,a,b) for(int i=(a);i<=(b);i++)
#define per(i,a,b) for(int i=(a);i>=(b);i--)
#define show1(a) cout<<#a<<" = "<<a<<endl
#define show2(a,b) cout<<#a<<" = "<<a<<"; "<<#b<<" = "<<b<<endl
using namespace std;
const ll INF = 1LL<<60;
const int inf = 1<<30;
const int maxn = 2e5+5;

inline void fastio() {ios::sync_with_stdio(false);cin.tie(0);cout.tie(0);}

int n = 100;

struct Matrix
{
    ll mat[100][100];
    //Matrix() {memset(mat,0,sizeof(mat));}
    /*Matrix operator * (Matrix b)
    {
        Matrix res;
        rep(i,0,n-1) rep(j,0,n-1) rep(k,0,n-1) res.mat[i][j] =
(res.mat[i][j] + mat[i][k]*b.mat[k][j]%M)%M;
        return res;
    }*/
    Matrix() {rep(i,0,n-1) rep(j,0,n-1) mat[i][j] = INF;}
    Matrix operator * (Matrix b)
```

```
    {
        Matrix res;
        rep(k,0,n-1){
            rep(i,0,n-1) rep(j,0,n-1) {
                res.mat[i][j] = min(res.mat[i][j],mat[i][k]+b.mat[k][j]);
            }
        }
        return res;
    }
    Matrix operator ^ (ll b)
    {
        Matrix res,A=*this;
        rep(i,0,n-1) rep(j,0,n-1) res.mat[i][j] = (i!=j)*INF;
        while(b){
            if(b&1) res = res*A;
            A = A*A;
            b>>=1;
        }return res;
    }
};
int main()
{
    fastio();
    int m,k; cin>>n>>m>>k;
    Matrix A;
    while(m--){ int u,v,w;
        cin>>u>>v>>w; u--,v--;
        A.mat[u][v] = w;
    }
    A = A^k;
    ll ans = INF;
    rep(i,0,n-1) rep(j,0,n-1) ans = min(ans,A.mat[i][j]);
    if(ans>1e18) cout<<"IMPOSSIBLE"<<endl;else cout<<ans<<endl;
    return 0;
}
```

G - Recurrence With Square

求序列

$$a_i = c_1\cdot a_{i-1} + c_2\cdot a_{i-2} + \cdots + c_n\cdot a_{i-n} + p + i\cdot q + i^2 \cdot r$$

的第 k 项，也就是归纳一般的线性递推式的矩阵构造方法。

```
/*
```

```
#pragma GCC optimize(2)
#pragma GCC optimize(3,"Ofast","inline")
*/
#include<bits/stdc++.h>
#define ALL(x) (x).begin(),(x).end()
#define ll long long
#define db double
#define ull unsigned long long
#define pii_ pair<int,int>
#define mp_ make_pair
#define pb push_back
#define fi first
#define se second
#define rep(i,a,b) for(int i=(a);i<=(b);i++)
#define per(i,a,b) for(int i=(a);i>=(b);i--)
#define show1(a) cout<<#a<<" = "<<a<<endl
#define show2(a,b) cout<<#a<<" = "<<a<<"; "<<#b<<" = "<<b<<endl
using namespace std;
const ll INF = 1LL<<60;
const int inf = 1<<30;
const int maxn = 2e5+5;
const int M = 1e9+7;
inline void fastio() {ios::sync_with_stdio(false);cin.tie(0);cout.tie(0);}
int n; ll k;
struct Matrix
{
    ll mat[15][15];
    Matrix() {memset(mat,0,sizeof(mat));}
    Matrix operator * (const Matrix other) const
    {
        Matrix product;
        rep(i,1,n+3) rep(j,1,n+3) rep(k,1,n+3) product.mat[i][j] =
(product.mat[i][j] + mat[i][k]*other.mat[k][j])%M;
        return product;
    }
    Matrix operator ^ (ll b) const
    {
        Matrix res,A=*this;
        rep(i,1,n+3) res.mat[i][i] = 1;
        while(b){
            if(b&1) res = res*A;
            A = A*A;
            b>>=1;
        }return res;
    }
};
ll a[15];
int main()
{
    fastio();
    cin>>n>>k; k -= n-1;
```

```
    rep(i,1,n) cin>>a[n-i+1]; a[n+1] = 1,a[n+2] = n,a[n+3] = n*n;
    Matrix single;
    rep(i,1,n+3) cin>>single.mat[1][i];
    rep(i,2,n) single.mat[i][i-1] = 1;
    single.mat[n+1][n+1] = 1;
    single.mat[n+2][n+1] = single.mat[n+2][n+2] = 1;
    single.mat[n+3][n+1] = 1,single.mat[n+3][n+2] = 2,single.mat[n+3][n+3]
= 1;
    Matrix total = single^k;
    ll ans = 0;
    rep(i,1,n+3) ans = (ans + total.mat[1][i]*a[i])%M;
    cout<<ans<<endl;
    return 0;
}
```

H - String Mood Updates

给一个只包含 26 个大写字母或者 $?$ 的字符串。一开始 $Limak$ 的心情是好的，接着从左往右遍历，如果遇到 $AEIOU$ 中的字母则心情翻转，如果遇到 H 则心情变好，如果遇到 S 和 D 则心情变差。字符 $?$ 可以是任何一种字母。

问遍历了字符串后$Limak$ 的心情仍然是好的的情况有多少种。并且会给出 q 次修改，每次修改某个位置的字符，并询问最后的结果。

如果没有修改则可以线性的 dp 求解。在有修改的情况下，考虑 dp 的过程是一个矩阵的乘法，即每个字母对应了一种矩阵，那修改就可以用线段树来维护了。

```
/*
#pragma GCC optimize(2)
#pragma GCC optimize(3,"Ofast","inline")
*/
#include<bits/stdc++.h>
#define ALL(x) (x).begin(),(x).end()
#define ll long long
#define db double
#define ull unsigned long long
#define pii_ pair<int,int>
#define mp_ make_pair
#define pb push_back
#define fi first
#define se second
#define rep(i,a,b) for(int i=(a);i<=(b);i++)
#define per(i,a,b) for(int i=(a);i>=(b);i--)
#define show1(a) cout<<#a<<" = "<<a<<endl
#define show2(a,b) cout<<#a<<" = "<<a<<"; "<<#b<<" = "<<b<<endl
using namespace std;
```

```
const ll INF = 1LL<<60;
const int inf = 1<<30;
const int maxn = 2e5+5;
const int M = 1e9+7;
inline void fastio() {ios::sync_with_stdio(false);cin.tie(0);cout.tie(0);}
char s[maxn];
int n,q;
struct Matrix
{
    ll mat[2][2];
    Matrix() {memset(mat,0,sizeof(mat));}
    Matrix operator * (const Matrix other) const
    {
        Matrix product;
        rep(i,0,1)rep(j,0,1)rep(k,0,1) product.mat[i][j] =
(product.mat[i][j] + mat[i][k] * other.mat[k][j])%M;
        return product;
    }
}tr[maxn<<2];
void push_up(int id)
{
    tr[id] = tr[id<<1] * tr[id<<1|1];
}
void build(int id,int l,int r)
{
    if(l==r){
        if(s[n-l+1]=='A' || s[n-l+1]=='E' || s[n-l+1]=='I' || s[n-l+1]=='O'
|| s[n-l+1]=='U'){
            tr[id].mat[0][1] = tr[id].mat[1][0] = 1;
        }else if(s[n-l+1]=='H'){
            tr[id].mat[0][0] = tr[id].mat[0][1] = 1;
        }else if(s[n-l+1]=='S' || s[n-l+1]=='D'){
            tr[id].mat[1][0] = tr[id].mat[1][1] = 1;
        }else if(s[n-l+1]=='?'){
            tr[id].mat[0][0] = 19,tr[id].mat[0][1] = 6;
            tr[id].mat[1][0] = 7,tr[id].mat[1][1] = 20;
        }else {
            tr[id].mat[0][0] = tr[id].mat[1][1] = 1;
        }
        return ;
    }
    int mid = (l+r)>>1;
    build(id<<1,l,mid);build(id<<1|1,mid+1,r);
    push_up(id);
}
void update(int id,int stl,int str,int pos,char o)
{
    if(stl==str){
        memset(tr[id].mat,0,sizeof(tr[id].mat));
        if(o=='A' || o=='E' || o=='I' || o=='O' || o=='U'){
            tr[id].mat[0][1] = tr[id].mat[1][0] = 1;
```

```
        }else if(o=='H'){
            tr[id].mat[0][0] = tr[id].mat[0][1] = 1;
        }else if(o=='S' || o=='D'){
            tr[id].mat[1][0] = tr[id].mat[1][1] = 1;
        }else if(o=='?'){
            tr[id].mat[0][0] = 19,tr[id].mat[0][1] = 6;
            tr[id].mat[1][0] = 7,tr[id].mat[1][1] = 20;
        }else {
            tr[id].mat[0][0] = tr[id].mat[1][1] = 1;
        }
        return ;
    }
    int mid = (stl+str)>>1;
    if(pos<=mid) update(id<<1,stl,mid,pos,o);
    else update(id<<1|1,mid+1,str,pos,o);
    push_up(id);
}
int main()
{
    fastio();
    cin>>n>>q>>s+1;
    build(1,1,n);
    cout<<tr[1].mat[0][0]<<endl;
    while(q--){ int pos;char o;
        cin>>pos>>o;
        update(1,1,n,n-pos+1,o);
        cout<<tr[1].mat[0][0]<<endl;
    }
    return 0;
}
```

I - Count Paths Queries

给一个有向图和 q 次询问，每次询问是求 s 到 t 并经过 k 条边的路径数。

因为 $n,q \le 200$所以每次都用快速幂求一次是不行的。因为每次询问只要求特定两点之间的路径，所以很多乘法是没有意义的，考虑如何只维护矩阵的第 s 行向量：

可以先倍增求出邻接矩阵的 2 的幂次的幂次。然后每次询问时只需要维护第 s 行的向量即可。

```
/*
#pragma GCC optimize(2)
#pragma GCC optimize(3,"Ofast","inline")
*/
#include<bits/stdc++.h>
#define ALL(x) (x).begin(),(x).end()
```

```cpp
#define ll long long
#define db double
#define ull unsigned long long
#define pii_ pair<int,int>
#define mp_ make_pair
#define pb push_back
#define fi first
#define se second
#define rep(i,a,b) for(int i=(a);i<=(b);i++)
#define per(i,a,b) for(int i=(a);i>=(b);i--)
#define show1(a) cout<<#a<<" = "<<a<<endl
#define show2(a,b) cout<<#a<<" = "<<a<<"; "<<#b<<" = "<<b<<endl
using namespace std;
const ll INF = 1LL<<60;
const int inf = 1<<30;
const int maxn = 2e5+5;
const int M = 1e9+7;
inline void fastio() {ios::sync_with_stdio(false);cin.tie(0);cout.tie(0);}
int n,m,q;
struct Matrix
{
    ll mat[201][201];
    Matrix() {memset(mat,0,sizeof(mat));}
    Matrix operator * (const Matrix other) const
    {
        Matrix product;
        rep(i,1,n) rep(j,1,n) rep(k,1,n) product.mat[i][j] =
(product.mat[i][j] + mat[i][k]*other.mat[k][j])%M;
        return product;
    }
}A[35];
int ans[205],tmp[205];
int main()
{
    fastio();
    cin>>n>>m>>q;
    while(m--){
        int u,v; cin>>u>>v;
        A[0].mat[u][v] = 1;
    }
    rep(i,1,30) A[i] = A[i-1]*A[i-1];
    while(q--){ int s,t,k;
        cin>>s>>t>>k;
        rep(i,1,n) ans[i] = (i==s);
        rep(b,0,30){
            if((k>>b)&1){
                rep(i,1,n) tmp[i] = 0;
                rep(i,1,n) rep(j,1,n) tmp[i] = (tmp[i] + ans[j] *
A[b].mat[j][i])%M;
                rep(i,1,n) ans[i] = tmp[i];
            }
```

```
        }
        cout<<ans[t]<<endl;
    }
    return 0;
}
```