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。

```
/*
```

```cpp
/*
#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;
}
```