设斐波那契数列第 k 项为 F_k，则递推公式:

$$F_k = F_{k-1} + F_{k-2}$$

则:

$$\begin{bmatrix} F_k \\ F_{k-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F_{k-1} \\ F_{k-2} \end{bmatrix}$$

则:

$$\begin{bmatrix} F_k \\ F_{k-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{k-1} \begin{bmatrix} F_1 \\ F_0 \end{bmatrix}$$

那么:

$$F_k = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{k-1}_{1,1}$$

问题转化为求一个矩阵的 n 次幂。

由于矩阵运算满足结合律，我们可以采用矩阵快速幂。

```
Matrix operator ^ (Matrix A,int k){
    Matrix base=A;
    while (k){
        if (k&1) base=base*A;
        A=A*A;
        k>>=1;
    }
    return base;
}
```

编写代码如下

```
#include <fraction.h>
#define Num frac
#include <matrix.h>
using namespace std;
int main(){
    Matrix A(2,2);
    A[1][1]=1,A[1][2]=1,A[2][1]=1;
    for (int i=1;i<=10;++i){
        cout<<"Fib"<<i<<"="<<(A^(i-1))[1][1]<<endl;
    }
}
```

如果要求解析解，我们可以采用相似对角化，求出 P，使得 $P^{-1}AP = \Lambda$，则:

$$A^n = (P\Lambda P^{-1})^n = P\Lambda^n P^{-1}$$

首先，计算特征值:

$$|A - \lambda E| = 0$$

$$\begin{vmatrix} 1-\lambda & 1 \\ 1 & -\lambda \end{vmatrix} = -\lambda+\lambda^2-1$$

有两个解：

$$\lambda_1=\frac{1}{2}+\frac{1}{2}\sqrt{5},\lambda_2=\frac{1}{2}-\frac{1}{2}\sqrt{5}$$

对应 C++ 代码：

```
Matrix A(2,2);
A.Message="A";
A[1][1]=1,A[1][2]=1,A[2][1]=1;
out<<begin_latex()<<endl<<endl;
out<<to_latex(A)<<endl<<endl;
Matrix B=A;
for (int i=1;i<=A.row;++i){
    B[i][i]=B[i][i]-Num("l");
}
cout<<"Eigen Poly"<<endl;
_poly x=Determinant(B).x;//计算行列式
cout<<x<<endl;
out<<to_latex(B,"vmatrix")<<"="<<to_latex(x)<<endl<<endl;
upoly _x;
_x.init_from_poly(x);
cpoly v=Factorization(_x);//因式分解为一次和二次因式
v.sort();
cout<<"Factorization"<<endl<<v<<endl;
out<<"Factorize it then we have "<<to_latex(v)<<endl<<endl;
```

对于每一个特征值，用基础解系求得一个特征向量。

```
for (int i=0;i<v.v.size();++i){
    if (v.v[i].first.deg()==1){
        Num lambda=poly(poly_ele((frac)(0)-v.v[i].first[0]));
        out<<"For eigen value $\\lambda="<<to_latex(lambda,0)<<"$ ,we have
vectors,"<<endl<<endl;
        cout<<"lambda="<<lambda<<endl;
        cout<<"n="<<v.v[i].second<<endl;//代数重数
        Matrix B=A-lambda*Matrix(A.row,A.col,1);
        vector<Matrix>baseS=baseSolution(B);
        // baseS=Schmidt(baseS); 如果需要正交矩阵的话需要施密特正交化
        cout<<"m="<<baseS.size()<<endl;//几何重数，几何重数不超过代数重数
        cout<<B<<endl;
        for (int i=0;i<baseS.size();++i){
            cout<<baseS[i]<<endl;
            s.push_back(baseS[i]);
            out<<to_latex(baseS[i])<<endl<<endl;
        }
    }
    else if (v.v[i].first.deg()==2){
        Num lambda_1,lambda_2;
        frac a=v.v[i].first[2],b=v.v[i].first[1],c=v.v[i].first[1];
        frac delta=b*b-4*a*c;
        lambda_1=poly_ele(sqrtNum(-b/(2*a),1/(2*a),delta));
        lambda_2=poly_ele(sqrtNum(-b/(2*a),-1/(2*a),delta));
```

```
        out<<"For eigen value $\\lambda="<<to_latex(lambda_1,0)<<"$ ,we have vectors,"<<endl<<endl;
        cout<<"lambda="<<lambda_1<<endl;
        cout<<"n="<<v.v[i].second<<endl;//代数重数
        Matrix B=A-lambda_1*Matrix(A.row,A.col,1);
        vector<Matrix>baseS=baseSolution(B);
        cout<<"m="<<baseS.size()<<endl;//几何重数，几何重数不超过代数重数
        cout<<B<<endl;
        for (int i=0;i<baseS.size();++i){
            cout<<baseS[i]<<endl;
            s.push_back(baseS[i]);
            out<<to_latex(baseS[i])<<endl<<endl;
        }
        out<<"For eigen value $\\lambda="<<to_latex(lambda_2,0)<<"$ ,we have vectors,"<<endl<<endl;
        cout<<"lambda="<<lambda_2<<endl;
        cout<<"n="<<v.v[i].second<<endl;//代数重数
        B=A-lambda_2*Matrix(A.row,A.col,1);
        baseS=baseSolution(B);
        cout<<"m="<<baseS.size()<<endl;//几何重数，几何重数不超过代数重数
        cout<<B<<endl;
        for (int i=0;i<baseS.size();++i){
            cout<<baseS[i]<<endl;
            s.push_back(baseS[i]);
            out<<to_latex(baseS[i])<<endl<<endl;
        }
    }
}
```

得到：

$$\eta_1=\begin{bmatrix}\frac{1}{2}+\frac{1}{2}\sqrt{5}\\1\end{bmatrix},\eta_2=\begin{bmatrix}\frac{1}{2}-\frac{1}{2}\sqrt{5}\\1\end{bmatrix}$$

组合形成

$$P=\begin{bmatrix}\frac{1}{2}+\frac{1}{2}\sqrt{5}&\frac{1}{2}-\frac{1}{2}\sqrt{5}\\1&1\end{bmatrix}$$

且：

$$\Lambda=PAP^{-1}=\begin{bmatrix}\frac{1}{2}+\frac{1}{2}\sqrt{5}&0\\0&\frac{1}{2}-\frac{1}{2}\sqrt{5}\end{bmatrix}$$

解得：

$$F_n=(\frac{1}{2}+\frac{1}{10}\sqrt{5})(\frac{1}{2}+\frac{1}{2}\sqrt{5})^{n-1}+(\frac{1}{2}-\frac{1}{10}\sqrt{5})(\frac{1}{2}-\frac{1}{2}\sqrt{5})^{n-1}$$

对应代码：

```
Matrix P=s[0];
out<<"Combine the vectors together then we get, "<<endl<<endl;
for (int i=1;i<s.size();++i){
    P=addH(P,s[i]);
}
cout<<P.message("P")<<endl;
```

```
cout<<(P^(-1)).message("P^{-1}")<<endl;
cout<<((P^-1)*A*P).message("P^{-1}AP")<<endl;
Matrix Lambda=(P^-1)*A*P;
out<<to_latex(P.message("P"))<<endl<<endl;
out<<to_latex(((P^-1)*A*P).message("\\Lambda=PAP^{-1}"))<<endl<<endl;
out<<"So $A^n=P^{-1}\\Lambda^n P$"<<endl<<endl;
out<<"For calculating $A^n$, assume $\\Lambda^n$=[a,0;0,b], then we have:"
<<endl<<endl;
Matrix _A(2,2);
_A[1][1]=poly("a"),_A[2][2]=poly("b");
cout<<P*_A*(P^-1)<<endl;
out<<to_latex((P*_A*(P^-1))[1][1])<<endl<<endl;
out<<"Then, substitute $a=("<<to_latex(Lambda[1][1],0)<<")^{n-1}$ and $b=("
<<to_latex(Lambda[2][2],0)<<")^{n-1}$, we have the final answer"<<endl<<endl;
```

一些小小的推广

求 $F_k=F_{k-1}+2F_{k-2}$ 的通项。

则:

$$A=\begin{bmatrix}1 & 2\\1 & 0\end{bmatrix}$$

利用程序自动生成结果:

$A = \begin{bmatrix} 1 & 2 \\ 1 & 0 \end{bmatrix}$

$A = \begin{vmatrix} 1-l & 2 \\ 1 & -l \end{vmatrix} = -l + l^2 - 2$

Factorize it then we have $(l-2)(l+1)$

For eigen value $\lambda = 2$,we have vectors,

$\eta_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$

For eigen value $\lambda = -1$,we have vectors,

$\eta_1 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$

Combine the vectors together then we get,

$P = \begin{bmatrix} 2 & -1 \\ 1 & 1 \end{bmatrix}$

$\Lambda = PAP^{-1} = \begin{bmatrix} 2 & 0 \\ 0 & -1 \end{bmatrix}$

So $A^n = P^{-1}\Lambda^n P$

For calculating A^n, assume Λ^n=[a,0;0,b], then we have:

$\frac{2}{3}a + \frac{1}{3}b$

Then, substitute $a = (2)^{n-1}$ and $b = (-1)^{n-1}$, we have the final answer

求 $F_k = -2F_{k-1} + F_{k-2} + 2F_{k-3}$ 的通项。

则:

$$A = \begin{bmatrix} -2 & 1 & 2 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$$A=\begin{bmatrix}-2&1&2\\1&0&0\\0&1&0\end{bmatrix}$$

$$A=\begin{vmatrix}-2-l&1&2\\1&-l&0\\0&1&-l\end{vmatrix}=-2l^2-l^3+l+2$$

Factorize it then we have $(l-1)(l+1)(l+2)$

For eigen value $\lambda=1$,we have vectors,

$$\eta_1=\begin{bmatrix}1\\1\\1\end{bmatrix}$$

For eigen value $\lambda=-1$,we have vectors,

$$\eta_1=\begin{bmatrix}1\\-1\\1\end{bmatrix}$$

For eigen value $\lambda=-2$,we have vectors,

$$\eta_1=\begin{bmatrix}4\\-2\\1\end{bmatrix}$$

Combine the vectors together then we get,

$$P=\begin{bmatrix}1&1&4\\1&-1&-2\\1&1&1\end{bmatrix}$$

$$\Lambda=PAP^{-1}=\begin{bmatrix}1&0&0\\0&-1&0\\0&0&-2\end{bmatrix}$$

So $A^n=P^{-1}\Lambda^nP$

For calculating A^n, assume $\Lambda^n=[a,0,0;0,b,0;0,0,c]$, then we have:

$\frac{1}{6}a-\frac{1}{2}b+\frac{4}{3}c$

Then, substitute $a=(1)^{n-1}$ and $b=(-1)^{n-1}$ and $c=(-2)^{n-1}$, we have the final answer

注意到，程序算出的 A 的特征多项式就是我们常说的数列的特征多项式，在后面严格证明。

更进一步推广：常系数齐次线性递推

一个 k 阶常系数齐次递推数列满足：

$$f_n=\sum_{i=1}^{k}a_if_{n-i}$$

对于斐波那契数列：

$$a_1 = a_2 = 1$$

转移矩阵：

$$A = \begin{pmatrix} a_1 & a_2 & a_3 & \cdots & a_{k-2} & a_{k-1} & a_k \\ 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 & 0 \end{pmatrix}$$

初值：

$$F = \begin{pmatrix} f_{k-1} \\ f_{k-2} \\ \vdots \\ f_0 \end{pmatrix}$$

则答案变成 $(A^n F)_{k,1}$。还是转化为计算 A^n。

不妨设多项式

$$g(\lambda) = |\lambda E - A|$$

在第一行第一列展开，我们得到 λ^k，在第一行第二列展开，我们得到 $-\lambda^{k-1}a_1$，在第一行第三列展开，我们得到 $-\lambda^{k-2}a_2$，之后一直到 $-\lambda^0 a_k$。

于是

$$g(\lambda) = \lambda^k - a_1\lambda^{k-1} - \cdots - a_k$$

由 Cayley-Hamilton 定理，我们也知道 $g(A) = 0$，那么我们希望凑出 $g(A)$ 的形式，就可以进行抵消。

不妨设 $A^n = f(A)g(A) + r(A)$，其中 f, r 都是关于 A 的多项式。那么就可以转化为多项式除法和取余的问题。答案是 $r(A)$。

```
#include <poly.h>
#define Num frac
#include <matrix.h>
#define MAXN 1005
using namespace std;
frac a[MAXN];
int main(){
    int n,k;
    cin>>n>>k;
    upoly g;
    g.v.resize(k+1);
    g[k]=frac(1);
    for (int i=1;i<=k;++i){
        cin>>a[i];
        g[k-i]=-a[i];
    }
    Matrix A(k,k);
    for (int i=1;i<=k;++i) A[1][i]=a[i];
    for (int i=1;i<=k-1;++i) A[i+1][i]=1;
    upoly M;
```

```
    M.v.resize(n+1);
    M[n]=frac(1);
    upoly r=M%g;
    Matrix An=Matrix(k,k);
    for (int i=r.v.size()-1;i>=0;--i) An=An*A+r.v[i]*Matrix(k,k,1);
    Matrix F(k,1);
    for (int i=1;i<=k;++i) cin>>F[k-i+1][1];
    cout<<(An*F)[k][1]<<endl;
}
```

例如，计算 fib_8，则输入

```
8 2
1 1
0 1
```

不用线性代数的另解：生成函数

构造$f(x)=\sum_{j=1} fib_j x^j$

所以$f(x)\times x=\sum_{j=2} fib_{j-1} x^j$

$f(x)-f(x)\times x=fib_1 x+\sum_{j=3} fib_{j-2} x^j=x+x^2\times f(x)$

所以$f(x)=\frac{x}{1-x-x^2}$

令$\phi=\frac{1+\sqrt{5}}{2},\phi'=\frac{1-\sqrt{5}}{2}$。

分解：$\frac{x}{1-x-x^2}=\frac{x}{(1-\phi' x)(1-\phi x)}=\frac{1}{\sqrt{5}}(\frac{1}{1-\phi x}-\frac{1}{1-\phi' x})$

根据公式$\frac{1}{1-x}=\sum_{j=0} x^j$，前面一项化为$(\phi)^n$后面一项化成$(\phi')^n$。

于是我们得到$fib_n=\frac{1}{\sqrt{5}}(\phi^n-\phi'^n)$。

即$fib_n=-\frac{1}{\sqrt{5}}(\frac{1-\sqrt{5}}{2})^n+\frac{1}{\sqrt{5}}(\frac{1+\sqrt{5}}{2})^n$。