线性递推数列算法研究

引言

常系数递推关系是组合数学中的一个研究方向，线性递推关系是最常见的递推关系，本文将介绍递推数列的相关理论与算法

1. 线性递推数列

1.1 线性递推数列与递推方程

定义1 如果数列数列 $\{F_n\}$ 满足**递推方程**

$$F_n - b_1F_{n-1} - b_2F_{n-2} - \cdots - b_kF_{n-k} = 0(0 < k < n)$$

其中 $b_1, b_2, \ldots, b_k$ 是常数，则称数列 $\{F_n\}$ 是**线性递推数列**

注意，$\{F_n\}$ 满足任何一个线性递推多项式即可，例如 $F_n - F_{n-1} = 1$ 不是线性递推方程，但我们不难得到 $F_n - 2F_{n-1} + F_{n-2} = 0$，因此 $\{F_n\}$ 是线性递推数列

1.2 线性递推数列的通项公式

线性递推数列的通项公式可以通过如下方式求解：

先求递推公式对应的特征方程

$$q^k - b_1q^{k-1} - b_2q^{k-2} - \cdots - b_k = 0$$

的解，根据代数基本定理，k 次多项式有 k 个根（重根重复计算）

如果特征方程有 k 个单根 $p_1, p_2, \ldots, p_k$，则递推方程的解为

$$F_n = c_1q_1^n + c_2q_2^n + \cdots + c_kq_k^n$$

如果有 m 重根 p_i，以 $c_1p_i^n + c_1np_i^n + \cdots + c_mn^{m-1}p_i^n$ 替代 $c_ip_i^n$ 即可

将解代入递推方程即可验证其正确性（证明略），而 $c_1, c_2, \ldots, c_k$ 是由数列的初值条件 $F(0), F(1), \ldots, F(k-1)$ 确定的

这个求解过程与常微分方程的通解的求解过程几乎异曲同工，读者可自行研究其中的联系，本文不作讨论

例1 求斐波那契数列 $F_n = F_{n-1} + F_{n-2}, F_0 = F_1 = 1$ 的通项公式

解 线性递推方程 $F_n = F_{n-1} + F_{n-2}$ 的特征方程为

$$q^2 - q - 1 = 0$$

特征方程有两个实根

$$q_1 = \frac{1+\sqrt{5}}{2}, q_2 = \frac{1-\sqrt{5}}{2}$$

因此线性递推方程的通解为

$$F_n = c_1\left(\frac{1+\sqrt{5}}{2}\right)^n + c_2\left(\frac{1-\sqrt{5}}{2}\right)^n$$

将 $F_0 = F_1 = 1$ 代入上式求得

$$c_1 = \frac{\sqrt{5}}{5}, c_2 = -\frac{\sqrt{5}}{5}$$

因此斐波那契数列的通项公式为

$$F_n = \frac{\sqrt{5}}{5}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right]$$

2. 线性递推公式求解算法

根据递推方程求通项公式的过程中需要求高次多项式的根，目前还没有有效的求解高次多项式的根的精确解的方法，而即使求出了所有的根，根也有可能分布在复数域上，通项公式的形式也就很复杂，要求 $\{F_n\}$ 的第 n 项，递推公式往往更可行，根据递推公式构造矩阵，利用快速幂等分治乘法即可以 $O(k^3 \log n)$ 的复杂度求出第 n 项，本节介绍如何根据线性递推数列中的足够多项来求其递推公式

2.1 线性方程组解法

根据 $\{F_n\}$ 的前 $2k$ 项，可得线性方程组

$$\begin{cases} b_k F_0 + b_{k-1} F_1 + \cdots + b_1 F_{k-1} = F_k \\ b_k F_1 + b_{k-1} F_2 + \cdots + b_1 F_k = F_{k+1} \\ \cdots \\ b_k F_{k-1} + b_{k-2} F_1 + \cdots + b_1 F_{2k-2} = F_{2k-1} \end{cases}$$

再利用高斯消元或其他线性方程组求解算法即可求得 $b_1, b_2, \ldots, b_k$ ，如果用高斯消元法，则算法的复杂度为 $O(k^3)$

2.2 BM算法

Berlekamp–Massey算法是一个鲜为人知的算法，笔者在大多数组合数学的书籍中，并未找到相关的资料，相关的文献也是很少，因此只能根据维基百科提供的描述撰写本节，若有不严谨之处，还请读者包涵

算法伪代码：

```
数列{F_n}的前N项 F[0],F[1],...,F[N-1]
多项式 C(x)=1,B(x)=1
L=0,m=1,b=1
for n from 0 to N-1 do
    d=0;
    for i from 0 to L do
        d=d+C[i]*F[n-i];
    if d==0
        m=m+1;
    else if 2*L<=n
        temp=C(x);
        C(x)=C(x)-d/b*x^m*B(x);
        L=n+1-L;
        B(x)=temp;
        b=d;
        m=1;
    else
```

```
            C(x)=C(x)-d/b*x^m*B(x);
            m=m+1;
    return C(x);
```

程序返回的多项式 $C(x)$ 的系数即为递推方程的系数，算法的复杂度为 $O(k^2)$

3. 特殊的线性递推数列与优化技巧

对于某些线性递推数列，可以用更低的复杂度来求解第 n 项

3.1 周期性数列

定义2 如果存在 $T \in \mathbb{N}$，使得 $F_n = F_{n+T}$ 对任意 $n \in \mathbb{N}$ 成立，则称 $\{F_n\}$ 是**周期性数列**

不难发现，$\{F_n\}$ 是周期性数列当且仅当满足递推方程

$$F_n - F_{n-T} = 0$$

那么

$$F_n = F_{(n \mod T)}$$

如果 T 不是很大，预处理前 T 项，即可以 $O(1)$ 的复杂度求解 $\{F_n\}$ 的第 n 项

3.2 多项式数列

定义3 如果 $\{F_n\}$ 的通项公式为 n 的多项式，则称 $\{F_n\}$ 是**多项式数列**

$\{F_n\}$ 是多项式数列当且仅当其递推方程系数是正负交错的杨辉三角的某一行

例2 $F(n) = n^3$ 的递推公式为

$$F_n - 4F_{n-1} + 6F_{n-2} - 4F_{n-3} + F_{n-4} = 0$$

多项式数列可用多项式插值算法（Lagrange插值、Newton插值等）计算，笔者的另一篇文章中有详细的介绍

https://blog.csdn.net/qq_39515621/article/details/83188074

3.3 复合数列

定义4 设 $\{F_n\}$ 与 $\{G_n\}$ 是两个线性递推数列，称数列

$$H_n = \begin{cases} F_{n/2}, n\text{是偶数} \\ G_{(n-1)/2}, n\text{是奇数} \end{cases}$$

是 $\{F_n\}$ 与 $\{G_n\}$ 的复合数列

线性递推数列的复合数列也是线性递推数列，如果递推方程中下标为奇数的系数都为 0，那么 $\{F_n\}$ 是复合数列

对于复合数列，可以将其拆开分类求解

4. 附录

线性递推数列模板，可供ACM竞赛使用

```
#include <bits/stdc++.h>
#define rep(i,a,b) for(ll i=a;i<=b;i++)
```

```cpp
using namespace std;
typedef long long ll;
const ll N=25;
const ll mo=1e9+7;
//快速幂
ll fpow(ll a,ll b){
    ll ans=1;
    while(b>0){if(b&1)ans=ans*a%mo;b>>=1;a=a*a%mo;}
    return ans;
}
//BM算法 求线性递推数列的递推公式
vector<ll> BM(const vector<ll> &s){
    vector<ll> C={1},B={1},T;
    ll L=0,m=1,b=1;
    rep(n,0,s.size()-1){
        ll d=0;
        rep(i,0,L)d=(d+s[n-i]%mo*C[i])%mo;
        if(d==0)m++;
        else{
            T=C;
            ll t=mo-fpow(b,mo-2)*d%mo;
            while(C.size()<B.size()+m)C.push_back(0);
            rep(i,0,B.size()-1)C[i+m]=(C[i+m]+t*B[i])%mo;
            if(2*L>n)m++;
            else L=n+1-L,B=T,b=d,m=1;
        }
    }
    return C;
}
//矩阵快速幂求解数列单项值,s为初值,C为递推公式
ll MatrixSolve(const vector<ll> &s,const vector<ll> &C,ll n){
    if(n<s.size())return s[n];
    ll k=C.size()-1,b=n-k+1;
    static ll B[N][N],t[N],a[N],c[N][N];
    rep(i,0,k-1)a[i]=s[i];
    rep(i,0,k-1)rep(j,0,k-1)B[i][j]=j==0?((C[i+1]>0)*mo-C[i+1]):(i==j-1);
    while(b){
        if(b&1){
            rep(i,0,k-1)t[i]=0;
            rep(i,0,k-1)rep(j,0,k-1)t[i]=(t[i]+a[k-j-1]*B[j][k-i-1])%mo;
            rep(i,0,k-1)a[i]=t[i];
        }
        b>>=1;
        rep(i,0,k-1)rep(j,0,k-1)c[i][j]=0;
        rep(r,0,k-1)rep(j,0,k-1)if(B[r][j])rep(i,0,k-1)c[i][j]=(c[i][j]+B[i]
[r]*B[r][j])%mo;
        rep(i,0,k-1)rep(j,0,k-1)B[i][j]=c[i][j];
    }
    return a[k-1];
}
//更快的矩阵快速幂求解数列单项值,s为初值,C为递推公式
ll fastMatrixSolve(const vector<ll> &s,const vector<ll> &C,ll n){
    if(n<s.size())return s[n];
    ll k=C.size()-1,b=n-k+1;
    static ll B[N][N][64],t[N],a[N],c[N][N],init_flag=1;
    if(init_flag){
        init_flag=0;
        rep(i,0,k-1)rep(j,0,k-1)B[i][j][0]=j==0?((C[i+1]>0)*mo-C[i+1]):(i==j-1);
```

```
        for(ll it=1;it<64;it++){
            rep(i,0,k-1)rep(j,0,k-1)B[i][j][it]=0;
            rep(r,0,k-1)rep(j,0,k-1)if(B[r][j][it-1])rep(i,0,k-1)B[i][j][it]=
(B[i][j][it]+B[i][r][it-1]*B[r][j][it-1])%mo;
        }
    }
    rep(i,0,k-1)a[i]=s[i];
    ll it=0;
    while(b){
        if(b&1){
            rep(i,0,k-1)t[i]=0;
            rep(i,0,k-1)rep(j,0,k-1)t[i]=(t[i]+a[k-j-1]*B[j][k-i-1][it])%mo;
            rep(i,0,k-1)a[i]=t[i];
        }
        b>>=1;
        it++;
    }
    return a[k-1];
}
//牛顿多项式插值
class NewtonPoly{
public:
    ll f[N],d[N],x[N],n=0;
    void add(ll X,ll Y){
        x[n]=X,f[n]=Y%mo;
        rep(i,1,n)f[n-i]=(f[n-i+1]-f[n-i])%mo*fpow((x[n]-x[n-i])%mo,mo-2)%mo;
        d[n++]=f[0];
    }
    ll operator () (ll X)const{
        ll ans=0,t=1;
        rep(i,0,n-1)ans=(ans+d[i]*t)%mo,t=(X-x[i])%mo*t%mo;
        return ans+mo*(ans<0);
    }
};
//检验线性递推数列的通项公式是否为多项式,C为递推公式
bool polyCheck(const vector<ll> &C){
    ll m=mo-C[1],last=1;
    rep(i,1,C.size()-1){
        last=last*(m-i+1)%mo*fpow(i,mo-2)%mo;
        if(last!=(i&1?(mo-C[i]):C[i]))return false;
    }
    return true;
}
//矩阵
class intMatrix{
public:
    ll a[N][N],n,m;
    bool operator < (const intMatrix &b)const{
        rep(i,0,n-1)rep(j,0,m-1)if(a[i][j]!=b.a[i][j])return a[i][j]<b.a[i][j];
        return false;
    }
    bool operator == (const intMatrix &b)const{
        return !(*this<b) && !(b<*this);
    }
    static intMatrix Eye(ll n){
        intMatrix c;
        c.n=c.m=n;
        rep(i,0,n-1)rep(j,0,n-1)c.a[i][j]=i==j;
```

```
        return c;
    }
    intMatrix operator * (const intMatrix &b)const{
        assert(m==b.n);
        intMatrix c;
        c.n=n,c.m=b.m;
        rep(i,0,c.n-1)rep(j,0,c.m-1)c.a[i][j]=0;
        rep(i,0,c.n-1)rep(j,0,c.m-1)rep(k,0,m-1)c.a[i][j]=(c.a[i][j]+a[i]
[k]*b.a[k][j])%mo;
        return c;
    }
    intMatrix operator ^ (int t)const{
        assert(n==m);
        intMatrix c=Eye(n),b=*this;
        while(t){if(t&1)c=c*b;b=b*b;t>>=1;}
        return c;
    }
    void print()const{
        rep(i,0,n-1)rep(j,0,m-1)cout<<a[i][j]<<" \n"[j==m-1];
        cout<<endl;
    }
};
//求同余意义下不依赖初值的周期,C为递推公式,d为步长
ll periodSolve(const vector<ll> &C,ll d){
    static map<intMatrix,ll> tab;
    tab.clear();
    ll k=C.size()-1;
    assert(C[k]!=0);
    intMatrix A,B,T=intMatrix::Eye(k),D=T;
    A.n=A.m=B.n=B.m=k;
    rep(i,0,k-1)rep(j,0,k-1)A.a[i][j]=j==0?((C[i+1]>0)*mo-C[i+1]):(i==j-1);
    rep(i,0,k-1)rep(j,0,k-1)B.a[i][j]=j==k-1?((mo-C[i])*fpow(C[k],mo-2)%mo):
(i==j+1);
    rep(i,1,d){
        T=T*B;
        if(T==D)return i;
        tab[T]=i;
    }
    intMatrix Ad=A^d;
    ll x=0;
    while(1){
        if(tab.find(D)!=tab.end())return x*d+tab[D];
        D=D*Ad;
        x++;
    }
    return -1;
}
ll polySolve(const vector<ll> &s,const vector<ll> &C,ll n){
    if(n<s.size())return s[n];
    static ll g[N],f[N],d[N];
    ll k=(ll)C.size()-1,w=1;
    rep(i,0,k)f[i]=i==1,d[i]=i==k?1:C[k-i];
    while((w<<1)<=n)w<<=1;
    while(w>>=1){
        rep(i,0,k+k-2)g[i]=0;
        rep(i,0,k-1)if(f[i])rep(j,0,k-1)(g[i+j]+=f[i]*f[j])%=mo;
        for(ll i=k+k-2;i>=k;i--)if(g[i])rep(j,1,k)(g[i-j]-=g[i]*d[k-j])%=mo;
        rep(i,0,k-1)f[i]=g[i];
```

```
        if(w&n)for(ll i=k;i>=0;i--)f[i]=i==k?f[i-1]:(i==0?-f[k]*d[i]:(f[i-1]-
f[k]*d[i]))%mo;
    }
    ll ans=0;
    rep(i,0,k-1)(ans+=f[i]*s[i])%=mo;
    return ans+(ans<0)*mo;
}
class Sequence{
public:
    ll poly;
    vector<ll> A,C;
    NewtonPoly P;
    void build(const vector<ll> &s){
        A=s;
        C=BM(A);
        poly=polyCheck(C);
        if(poly)rep(i,0,s.size()-1)P.add(i,s[i]);
    }
    ll operator () (ll n)const{
        return poly?P(n):polySolve(A,C,n);
    }
    friend ostream &operator << (ostream &o,const Sequence &b){
        o<<"f(n)";
        rep(i,1,b.C.size()-1)o<<"+("<<b.C[i]<<")*f(n-"<<i<<")";
        o<<"=0 (mod "<<mo<<")";
        return o;
    }
    ll period()const{
        ll M=periodSolve(C,(ll)sqrt(mo)),ans=M;
        return M;
        for(ll i=1;i*i<=M;i++)if(M%i==0){
            ll d=i,flag=1;
            rep(i,0,(ll)C.size()-2)if((*this)(d+i)!=A[i])flag=0;
            if(flag)ans=min(ans,d);
            d=M/i,flag=1;
            rep(i,0,(ll)C.size()-2)if((*this)(d+i)!=A[i])flag=0;
            if(flag)ans=min(ans,d);
        }
        return ans;
    }
}F;

int main(){
    ios::sync_with_stdio(false);
    F.build({1,1,2,3,5,8,13,21});
    cout<<F<<endl;
    F.build({0,1,4,9,16,25,36,49});
    cout<<F<<endl;
    F.build({0,1,5,15,35,70,126,210,330,495,715});
    cout<<F<<endl;
    F.build({0,1,2,0,1,2,0,1,2});
    cout<<F<<endl;
    return 0;
}
```