

多项式 4

循环卷积

定义

$$c_k = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} [i+j \bmod p=k] a_i b_j$$

性质

对序列 A, B 做长度为 n 的 FFT 等价于求序列 A, B 的循环卷积。

考虑单位根反演证明

$$\begin{aligned} c_k &= \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} [i+j \bmod p=k] a_i b_j \\ &= \frac{1}{n} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{d=0}^{n-1} w_n^{d(i+j-k)} a_i b_j \\ &= \frac{1}{n} \sum_{d=0}^{n-1} w_n^{-dk} \sum_{i=0}^{n-1} w_n^{di} a_i \sum_{j=0}^{n-1} w_n^{dj} b_j \\ &= \frac{1}{n} \sum_{d=0}^{n-1} w_n^{-dk} \left(\sum_{i=0}^{n-1} w_n^{di} a_i \right) \left(\sum_{j=0}^{n-1} w_n^{dj} b_j \right) \end{aligned}$$

不难发现

$\left(\sum_{i=0}^{n-1} w_n^{di} a_i \right) \left(\sum_{j=0}^{n-1} w_n^{dj} b_j \right)$ 即为原来的 DFT 过程 $\frac{1}{n} \sum_{d=0}^{n-1} w_n^{-dk}$ 即为原来的 IDFT 证毕。

事实上普通的卷积计算相当于长度为 2^n 的循环卷积计算，只是循环卷积长度大于 C 序列的长度，所以循环卷积结果即为普通卷积结果。

Bluestein's Algorithm

考虑 DFT 过程，有

$$\begin{aligned} y_k &= \sum_{i=0}^{n-1} a_i w_n^{-ki} \\ &= \sum_{i=0}^{n-1} a_i w_n^{\binom{k+i}{2} - \binom{k}{2} - \binom{i}{2}} \\ &= w_n^{-\binom{k}{2}} \sum_{i=0}^{n-1} \left(a_i w_n^{\binom{i}{2}} \right) w_n^{\binom{k+i}{2}} \end{aligned}$$

其中 $\binom{n}{2}$ 表示组合数，易知上式可以通过普通卷积求解。

考虑 IDFT 过程，同样有

$$\begin{aligned} a_k &= \frac{1}{n} \sum_{i=0}^{n-1} y_i w_n^{ki} \\ &= \frac{1}{n} \sum_{i=0}^{n-1} y_i w_n^{-\binom{k+i}{2} + \binom{k}{2} + \binom{i}{2}} \\ &= \frac{1}{n} w_n^{\binom{k}{2}} \sum_{i=0}^{n-1} \left(y_i w_n^{\binom{i}{2}} \right) w_n^{-\binom{k+i}{2}} \end{aligned}$$

值得一提的是，循环卷积的求逆、快速幂等操作直接对 DFT 的点值进行相应运算即可。

算法例题

[洛谷p5293](#)

题意

给定一个二维 $[L+1] \times n$ 的空间，其中 $(u_1, v_1) \to (u_2, v_2)$ 有 w_{v_1, v_2} 条重边。

假设起点为 $(0, x)$ 终点为 (L, y) (L, y 为任意值)，路径长度 m 定义为路径的边数。

对每个 $0 \leq t \leq L$ 求满足所有 $m \equiv t \pmod k$ 且横坐标单增的路径数模 p 意义下的值。

数据保证 p 为素数 $10^8 \leq p \leq 2^{30}, k \mid p-1, 1 \leq k \leq 65536, 1 \leq n \leq 3, L \leq 10^9$

题解

假设 $f_{a,b}$ 表示 $m=a$ 且 $y=b$ 的路径数 $g_{a,b}$ 表示将空间的 X 维消去后 $m=a$ 且 $y=b$ 的路径数。

于是有状态转移方程

$$g_{a,b} = \sum_{i=1}^n g_{a-1,i} w_{i,b} \quad f_{a,b} = \binom{L}{a} g_{a,b}$$

设矩阵

$$W = \begin{bmatrix} w_{1,1} & \dots & w_{1,n} \\ \vdots & \ddots & \vdots \\ w_{n,1} & \dots & w_{n,n} \end{bmatrix}$$

设 $G_i = (g_{i,1} \dots g_{i,n})$ 于是有 $G_i = G_0 W^i$

考虑单位根反演，设 $w_k \equiv g \pmod p, g$ 为 p 的原根，有

$$\begin{aligned} \text{ans}_t &= \sum_{i=0}^L f_{i,y} [i \bmod k = t] = \frac{1}{k} \sum_{i=0}^L f_{i,y} \sum_{j=0}^{k-1} w_k^{j(i-t)} = \frac{1}{k} \sum_{j=0}^{k-1} w_k^{-jt} \sum_{i=0}^L f_{i,y} w_k^{ij} \\ \sum_{i=0}^L w_k^{ij} (f_{i,1} \dots f_{i,n}) &= \sum_{i=0}^L w_k^{ij} \binom{L}{i} (g_{i,1} \dots g_{i,n}) = G_0 \sum_{i=0}^L \binom{L}{i} (w_k^j W)^i = G_0 (w_k^j W + I)^L \end{aligned}$$

于是根据矩阵快速幂可以 $O(kn^3 \log L)$ 计算出所有 $h_j = \sum_{i=0}^L f_{i,y} w_k^{ij} (0 \leq j \leq k)$

于是有

$$\text{ans}_t = \frac{1}{k} \sum_{i=0}^{k-1} w_k^{-ti} h_i$$

发现上式就是 $\text{Bluestein's Algorithm}$ 的 IDFT 过程，直接求解时间复杂度 $O(k \log k)$

总时间复杂度 $O(kn^3 \log L)$ 主要用于计算矩阵快速幂。

```
const int MAXN=1<<16|5;
const long double pi=acos(-1.0);
int p,sz,gw[MAXN];
int quick_pow(int a,int b){
    int ans=1;
    while(b){
        if(b&1)ans=1LL*ans*a%p;
    }
}
```

```

        a=1LL*a*a%p;
        b>>=1;
    }
    return ans;
}
vector<int> pdiv;
bool check(int x){
    _for(i,0,pdiv.size()){
        if(quick_pow(x,(p-1)/pdiv[i])==1)
            return false;
    }
    return true;
}
void get_G(int k){
    int temp=p-1,g;
    for(int i=2;i*i<=temp;i++){
        if(temp%i==0){
            pdiv.push_back(i);
            while(temp%i==0)temp/=i;
        }
    }
    if(temp!=1)pdiv.push_back(temp);
    _for(i,2,p){
        if(check(i)){
            g=i;
            break;
        }
    }
    int w=quick_pow(g,(p-1)/k);
    gw[0]=1;
    _rep(i,1,k)gw[i]=1LL*gw[i-1]*w%p;
}
struct Matrix{
    int ele[3][3];
    Matrix(int x=0){
        mem(ele,0);
        _for(i,0,sz)ele[i][i]=x;
    }
    Matrix operator + (const Matrix b){
        Matrix c;
        _for(i,0,sz)
            _for(j,0,sz)
                c.ele[i][j]=(ele[i][j]+b.ele[i][j])%p;
        return c;
    }
    Matrix operator * (const int b){
        Matrix c;
        _for(i,0,sz)
            _for(j,0,sz)
                c.ele[i][j]=1LL*ele[i][j]*b%p;
        return c;
    }
};

```

```
    }
    void operator *= (const Matrix b){
        Matrix c=*this;
        _for(i,0,sz)
        _for(j,0,sz){
            ele[i][j]=0;
            _for(k,0,sz)
                ele[i][j]=(ele[i][j]+1LL*c.ele[i][k]*b.ele[k][j])%p;
        }
    }
};

Matrix quick_pow(const Matrix &a,int k){
    Matrix Ans(1),Base;
    Base=a;
    while(k){
        if(k&1)Ans*=Base;
        k>>=1;
        Base*=Base;
    }
    return Ans;
}

struct complex{
    long double x,y;
    complex(long double x=0.0,long double y=0.0):x(x),y(y){}
    complex operator + (const complex &b){
        return complex(x+b.x,y+b.y);
    }
    complex operator - (const complex &b){
        return complex(x-b.x,y-b.y);
    }
    complex operator * (const complex &b){
        return complex(x*b.x-y*b.y,x*b.y+y*b.x);
    }
};

int rev[MAXN<<2];
int build(int k){
    int n,pos=0;
    while((1<<pos)<=k)pos++;
    n=1<<pos;
    _for(i,0,n)rev[i]=(rev[i>>1]>>1)|((i&1)<<(pos-1));
    return n;
}

void FFT(complex *f,int n,int type){
    _for(i,0,n)if(i<rev[i])
        swap(f[i],f[rev[i]]);
    complex t1,t2;
    for(int i=1;i<n;i<=<1){
        complex w(cos(pi/i),type*sin(pi/i));
        for(int j=0;j<n;j+=(i<<1)){
            complex cur(1.0,0.0);
```

```

        _for(k,j,j+i){
            t1=f[k],t2=cur*f[k+i];
            f[k]=t1+t2,f[k+i]=t1-t2;
            cur=cur*w;
        }
    }
}
if(type==-1)_for(i,0,n)
f[i].x/=n,f[i].y/=n;
}
void FFT2(complex *f1,complex *f2,int n){
    FFT(f1,n,1);
    f2[0].x=f1[0].x,f2[0].y=-f1[0].y;
    _for(i,1,n)
    f2[i].x=f1[n-i].x,f2[i].y=-f1[n-i].y;
    complex t1,t2;
    _for(i,0,n){
        t1=f1[i],t2=f2[i];
        f1[i]=complex((t1.x+t2.x)*0.5,(t1.y+t2.y)*0.5);
        f2[i]=complex((t1.y-t2.y)*0.5,(t2.x-t1.x)*0.5);
    }
}
void MTT(int *f,int n1,int *g,int n2,int *ans,int mod){
    static complex
    f1[MAXN<<2],f2[MAXN<<2],g1[MAXN<<2],g2[MAXN<<2],temp[2][MAXN<<2];
    int n=build(n1+n2),m=sqrt(mod)+1;
    _rep(i,0,n1)f1[i].x=f[i]/m,f1[i].y=f[i]%m;
    _rep(i,0,n2)g1[i].x=g[i]/m,g1[i].y=g[i]%m;
    FFT2(f1,f2,n);FFT2(g1,g2,n);
    complex I(0.0,1.0);
    _for(i,0,n){
        temp[0][i]=f1[i]*g1[i]+I*f2[i]*g1[i];
        temp[1][i]=f1[i]*g2[i]+I*f2[i]*g2[i];
    }
    FFT(temp[0],n,-1);FFT(temp[1],n,-1);
    LL a,b,c;
    _rep(i,0,n1+n2){
a=temp[0][i].x+0.5,b=temp[0][i].y+temp[1][i].x+0.5,c=temp[1][i].y+0.5;
        ans[i]=((a%mod*m%mod*m%mod+b%mod*m%mod+c%mod)%mod+mod)%mod;
    }
}
}
Matrix W;
int a[MAXN<<2],b[MAXN<<2],c[MAXN<<2];
int main()
{
    int k,L,x,y;
    sz=read_int(),k=read_int(),L=read_int(),x=read_int()-1,y=read_int()-1,p=rea
d_int();
    get_G(k);
    _for(i,0,sz)
    _for(j,0,sz)

```

```
W.ele[i][j]=read_int();
_for(i,0,k){
    Matrix temp=quick_pow(W*gw[i]+Matrix(1),L);
    a[i]=1LL*temp.ele[x][y]*gw[1LL*i*(i-1)/2%k]%p;
}
_for(i,0,2*k-1)b[i]=gw[k-1LL*i*(i-1)/2%k]%p;
reverse(a,a+k);
MTT(a,k-1,b,2*k-2,c,p);
int div=quick_pow(k,p-2);
_for(i,0,k)
enter(1LL*div*gw[1LL*i*(i-1)/2%k]%p*c[k-1+i]%p);
return 0;
}
```

线性递推

From: <https://wiki.cvbbacm.com/> - CVBB ACM Team

Permanent link: https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:jxm2001:%E5%A4%9A%E9%A1%B9%E5%BC%8F_4&rev=1598199229

Last update: 2020/08/24 00:13