多项式

zhx

复数

- $i = \sqrt{-1}$
- $z = a + bi$
- 可以对应复平面上一个点(a, b)
- 所以长度角度也可以表示一个点
- $(r\cos\theta, r\sin\theta)$

复数运算

- $z_1 = a + bi, z_2 = c + di$
- $z_1 z_2 = (ac - bd) + (ad + bc)i$
- 加法：向量加法
- $z_1 = (r_1 \cos\theta_1, r_1 \sin\theta), z_2 = (r_2 \cos\theta_2, r_2 \sin\theta_2)$
- $z_1 z_2 = r_1 r_2(\cos\theta_1 \cos\theta_2 - \sin\theta_1 \sin\theta_2) + r_1 r_2(\cos\theta_1 \sin\theta_2 + \sin\theta_1 \cos\theta_2)i = r_1 r_2 \cos(\theta_1 + \theta_2) + r_1 r_2 \sin(\theta_1 + \theta_2)\, i$
- $(r_1 r_2, \theta_1 + \theta_2)$
- 乘法：向量旋转

复数单位根

- $x^n = 1$
- $\omega_n = \cos\frac{2\pi}{n} + \sin\frac{2\pi}{n} \cdot i$
- $\omega_n^k = \cos\frac{2\pi k}{n} + \sin\frac{2\pi k}{n} \cdot i$

单位根性质

- $\omega_n^k = \omega_{2n}^{2k}$
- $\omega_n^{k+\frac{n}{2}} = -\omega_n^k$
- $\omega_n^0 = \omega_n^n = 1$

点值表示法（DFT）

- 原始表示法：$f(x)=\sum_{i=0}^{n} a_i x^i$
- 点值表示法：$\{(x_1,f(x_1)),(x_2,f(x_2)),\cdots,(x_n,f(x_n))\}$
- 使用点值表示法进行多项式乘法：
- $\{(x_1,g(x_1)f(x_1)),(x_2,g(x_2)f(x_2)),\cdots,(x_n,g(x_n)f(x_n))\}$

快速傅里叶变换（FFT）

- 首先假设$n = 2^k$
- 多项式$A(x) = \sum_{i=0}^{N-1} a_i x^i$
- 点值表示法：使用$\omega_n^0, \omega_n^1, \cdots, \omega_n^{n-1}$为根带进去

快速傅里叶变换（FFT）

- $A(x) = \left(a_0 + a_2x^2 + \cdots\right) + \left(a_1x + a_3x^3 + \cdots\right)$
- 令
- $A_1(x) = a_0 + a_2x + a_4x^2 + \cdots + a_{n-2}x^{\frac{n}{2}-1}$
- $A_2(x) = a_1 + a_3x + a_5x^2 + \cdots + a_{n-1}x^{\frac{n}{2}-1}$

- $A(x) = A_1\left(x^2\right) + xA_2\left(x^2\right)$

快速傅里叶变换（FFT）

- $A(\omega_n^k) = A_1(\omega_n^{2k}) + \omega_n^k A_2(\omega_n^{2k})$
- $= A_1\left(\omega_{\frac{n}{2}}^k\right) + \omega_n^k A_2\left(\omega_{\frac{n}{2}}^k\right)$
- 同理可得

- $A\left(\omega_n^{k+\frac{n}{2}}\right) = A_1\left(\omega_{\frac{n}{2}}^k\right) - \omega_n^k A_2\left(\omega_{\frac{n}{2}}^k\right)$
- 于是便可递归求解，总共$\log n$层

快速傅里叶逆变换（IDFT）

- 目标是把点值表示法换回多项式表示法
- 令$y = DFN(f), c_k = \sum_{i=0}^{n-1} y_i \left(\omega_n^{-k}\right)^i$
- 则
- $c_k = \sum_{i=0}^{n-1} y_i \left(\omega_n^{-k}\right)^i = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_j \left(\omega_n^i\right)^j \left(\omega_n^{-k}\right)^i$
- $= \sum_{j=0}^{n-1} a_j \sum_{i=0}^{n-1} \left(\omega_n^{j-k}\right)^i$

快速傅里叶逆变换（IDFT）

- 令$S(x)=\sum_{i=0}^{n-1}x^i$
- $k\neq 0$时
- $S(\omega_n^k)=1+(\omega_n^k)+(\omega_n^k)^2+\cdots$
- $\omega_n^k S(\omega_n^k)=\omega_n^k+(\omega_n^k)^2+(\omega_n^k)^3+\cdots$
- $S(\omega_n^k)=\frac{(\omega_n^k)^n-1}{\omega_n^k-1}=0$
- $k=0$时
- $S(\omega_n^k)=n$

快速傅里叶逆变换（IDFT）

- $c_k = \sum_{j=0}^{n-1} a_j \sum_{i=0}^{n-1} \left(\omega_n^{j-k}\right)^i$
- $c_k = a_k n$
- $a_k = \frac{c_k}{n}$

- 因此再反着做一遍FFT即可
- https://noip.ac/show_problem/3206

快速傅里叶变换（FFT）

```
void fft(complex_ *z,int n,int inv)
{
    if (n==1) return;
    int h=n>>1;
    static complex_ y[maxn];
    for (int a=0;a<h;a++)
    {
        y[a]=z[a<<1];
        y[a+h]=z[a<<1|1];
    }
    for (int a=0;a<n;a++)
        z[a]=y[a];

    fft(z,h,inv);
    fft(z+h,h,inv);
    for (int k=0;k<h;k++)
    {
        complex_ x=complex_(cos(2*k*pi/n),sin(2*k*pi/n));
        if (inv) x.b=-x.b;
        y[k] = z[k] + z[k+h] * x;
        y[k+h] = z[k] - z[k+h] * x;
    }
    for (int a=0;a<n;a++)
        z[a]=y[a];
}
```

非递归版优化

- 实验一下可以发现最后一层的位置是二进制翻转之后的结果
- 可以提前预处理
- $i \to j$
- 考虑$i \gg 1$，类递归处理

非递归版优化

```
void fft(complex_ *z,int n,int inv)
{
    int bit=0;
    while ((1<<bit)<n) bit++;
    for (int a=0;a<n;a++)
    {
        rev[a] = (rev[a>>1]>>1) | ((a&1)<<(bit-1));
        if (a<rev[a]) swap(z[a],z[rev[a]]);
    }

    for (int l=1;l<n;l<<=1)
    {
        complex_ x = complex_(cos(pi/l),sin(pi/l));
        if (inv) x.b=-x.b;
        for (int a=0;a<n;a+=(l<<1))
        {
            complex_ v = complex_(1,0);
            for (int b=0;b<l;b++,v=v*x)
            {
                complex_ v1=z[a+b],v2=z[a+b+l]*v;
                z[a+b] = v1+v2;
                z[a+b+l] = v1-v2;
            }
        }
    }
}
```

数论变换（NTT）

- 在模意义下的多项式乘法
- 需要通过类似的操作是复杂度降到$O(n \log n)$

单位根

- 在FFT中用到了的性质:
- $\omega_n^n = 1$
- $\omega_n^1, \omega_n^2, \cdots, \omega_n^n$互不相等
- $\omega_n^k = \omega_{2n}^{2k}, \omega_n^{k+\frac{n}{2}} = -\omega_n^k$
- $\sum_{i=0}^{n-1} \left(\omega_n^k\right)^i = \begin{cases} 0 & k \neq 0 \\ n & k = 0 \end{cases}$

原根

- 设g为模质数p的原根
- $p-1=q\times 2^r=q\times n$
- 设$\omega_n=g^q$
- 则$\omega_n,\omega_n^2,\cdots,\omega_n^n$两两不同

- 且$\omega_n^n=g^{nq}=g^{p-1}=1$
- 性质一二满足

性质三

- $p = q \times n + 1 = \frac{q}{2} \times 2n + 1, \omega_n = g^q$
- $\omega_{2n} = g^{\frac{q}{2}}$
- $\omega_{2n}^{2k} = g^{kq} = \omega_n^k$

- $\left(\omega_n^{\frac{n}{2}}\right)^2 = 1 \Rightarrow \omega_n^{\frac{n}{2}} = -1 \Rightarrow \omega_n^{k+\frac{n}{2}} = -\omega_n^k$

- 性质三满足

性质四

- 若$k \neq 0$
- $\sum_{i=0}^{n-1}\left(\omega_n^k\right)^i = \frac{\left(\omega_n^k\right)^n-1}{\omega_n^k-1} = 0$

- 若$k = 0$
- $\sum_{i=0}^{n-1}\left(\omega_n^k\right)^i = n$

- 性质四满足

数论变换（NTT）

- 实际使用中
- 使用模数$998244353 = 119 * 2^{23} + 1$
- 可以处理所有序列长度$\leq 2^{23}$的情况

数论变换（NTT）

```
void ntt(int *z,int n,int inv)
{
    int bit=0;
    while ((1<<bit)<n) bit++;
    for (int a=0;a<n;a++)
    {
        rev[a] = (rev[a>>1]>>1) | ((a&1)<<(bit-1));
        if (a<rev[a]) swap(z[a],z[rev[a]]);
    }

    for (int l=1;l<n;l<<=1)
    {
        int x = mul(g,(mo-1)/(l<<1),mo);
        if (inv) x=mul(x,mo-2,mo);
        for (int a=0;a<n;a+=(l<<1))
        {
            int v=1;
            for (int b=0;b<l;b++,v=1ll*v*x%mo)
            {
                int v1=z[a+b],v2=1ll*z[a+b+l]*v%mo;
                z[a+b] = v1+v2; if (z[a+b]>=mo) z[a+b]-=mo;
                z[a+b+l] = v1-v2; if (z[a+b+l]<0) z[a+b+l]+=mo;
            }
        }
    }
}
```

快速沃尔什变换（FWT）

- 原来的多项式乘法：$C_{i+j} = \sum_i \sum_j A_i \times B_j$

- 新的多项式乘法：$C_{i \oplus j} = \sum_i \sum_j A_i \times B_j$
- $\oplus$为某种位运算

快速沃尔什变换（FWT）

- 目标
- 若$C = A \cdot B$
- 直接计算是$O\left(n^2\right)$
- 希望通过FWT变换为
- $FWT(C) = FWT(A) \cdot FWT(B)$
- 再转回去就可以得到C

快速沃尔什变换（FWT）

- 考虑$\oplus$为or运算
- $FWT(A) = \begin{cases} \big(FWT(A_0), FWT(A_0) + FWT(A_1)\big), n > 1 \\ A,\ n = 1 \end{cases}$
- 其中A_0为A前半段，A_1为A后半段
- 其中有另外一个重要的性质是$i|k = k, j|k = k \Rightarrow (i|j)|k = k$
- 可以发现$FWT(A)_i = \sum_{j\ or\ i=i} A_j$

快速沃尔什变换（FWT）

- 我们需要证明$FWT(A|B) = FWT(A) \times FWT(B)$
- 数学归纳法
- $FWT(A|B) = FWT\big((A|B)_0, (A|B)_1\big)$
- $= FWT(A_0|B_0, A_0|B_1 + A_1|B_0 + A_1|B_1)$
- $= \big(FWT(A_0|B_0), FWT(A_0|B_0 + A_0|B_1 + A_1|B_0 + A_1|B_1)\big)$
- $= \big(FWT(A_0) \times FWT(B_0), FWT(A_0) \times FWT(B_0) + FWT(A_0) \times FWT(B_1) + FWT(A_1) \times FWT(B_0) + FWT(A_1) \times FWT(B_1)\big)$

快速沃尔什变换（FWT）

- $FWT(A|B) = \Big(FWT(A_0) \times FWT(B_0), \big(FWT(A_0) + FWT(A_1)\big) \times \big(FWT(B_0) + FWT(B_1)\big)\Big)$
- $= \big(FWT(A_0) \times FWT(B_0), FWT(A_0 + A_1) \times FWT(B_0 + B_1)\big)$
- $= \big(FWT(A_0), FWT(A_0 + A_1)\big) \times \big(FWT(B_0), FWT(B_0 + B_1)\big)$
- $= FWT(A) \times FWT(B)$

快速沃尔什逆变换（IFWT）

- $FWT(A) = \left(FWT(A_0), FWT(A_0) + FWT(A_1)\right)$
- $\Rightarrow$
- $IFWT(A) = \left(IFWT(A_0), IFWT(A_1) - IFWT(A_0)\right)$

快速沃尔什变换（FWT）

- 考虑$\oplus$为and运算
- 类似的
- $FWT(A)=\begin{cases}\big(FWT(A_0)+FWT(A_1),FWT(A_1)\big),n>1\\ A,\ n=1\end{cases}$
- 可以发现$FWT(A)_i=\sum_{j\ and\ i=i}A_j$

快速沃尔什变换（FWT）

- 异或怎么弄？
- 我们希望
- $FWT(A \otimes B) = FWT(A) \times FWT(B)$

- $A \otimes B = (A_0 \otimes B_0 + A_1 \otimes B_1, A_0 \otimes B_1 + A_1 \otimes B_0)$

- 但是如果$i \otimes k = k, j \otimes k = k$是没办法使用结合律的
- 所以要换个东西

异或

- 我们的目标:
- $FWT(A \otimes B) = FWT(A) \times FWT(B)$
- $FWT(A \otimes B) = FWT(A_0 \otimes B_0 + A_1 \otimes B_1, A_0 \otimes B_1 + A_1 \otimes B_0)$
- 设$FWT(A) = \left(aFWT(A_0) + bFWT(A_1), cFWT(A_0) + dFWT(A_1)\right)$
- 接下来省略FWT
- $FWT(A) \times FWT(B) = (aA_0 + bA_1, cA_0 + dA_1) \times (aB_0 + bB_1, cB_0 + dB_1) = ((aA_0 + bA_1) \times (aB_0 + bB_1), (cA_0 + dA_1) \times (cB_0 + dB_1))$

异或

- $FWT(A_0 \otimes B_0 + A_1 \otimes B_1, A_0 \otimes B_1 + A_1 \otimes B_0)$
- $= \big(aFWT(A_0 \otimes B_0 + A_1 \otimes B_1) + bFWT(A_0 \otimes B_1 + A_1 \otimes B_0), cFWT(A_0 \otimes B_0 + A_1 \otimes B_1) + dFWT(A_0 \otimes B_1 + A_1 \otimes B_0)\big)$
- $= (aA_0B_0 + aA_1B_1 + bA_0B_1 + bA_1B_0, cA_0B_0 + cA_1B_1 + dA_0B_1 + dA_1B_0)$
- $= ((aA_0 + bA_1) \times (aB_0 + bB_1), (cA_0 + dA_1) \times (cB_0 + dB_1))$

异或

- $$\begin{cases} a^2 = a \\ b^2 = a \\ ab = b \\ c^2 = c \\ cd = c \\ d^2 = c \end{cases}$$
- 怎么解? ? ?

异或

- 首先$a, b, c, d \neq 0$，因为前后两部分都和A_0, A_1有关系
- 则$a = 1, b = \pm 1$
- $c = 1, d = \pm 1$
- 但是若$b = d$则前后完全一致
- 所以$a = b = c = 1, d = -1$或者$a = d = c = 1, b = -1$
- 所以
- $FWT(A) = (A_0 + A_1, A_0 - A_1)$
- $IFWT(A) = \left(\frac{A_0+A_1}{2}, \frac{A_0-A_1}{2}\right)$

快速沃尔什变换（FWT）

```
void fwt(int *z,int n,int inv,int type)
{
    for (int l=1;l<n;l<<=1)
        for (int a=0;a<n;a+=(l<<1))
            for (int b=0;b<l;b++)
                if (!inv)
                {
                    z[a+b+l] += z[a+b];
                    if (z[a+b+l]>=mo) z[a+b+l]-=mo;
                }
                else
                {
                    z[a+b+l] -= z[a+b];
                    if (z[a+b+l]<0) z[a+b+l]+=mo;
                }
}
```

多项式求逆

- $A(x)B(x) \equiv 1(mod\ x^n)$
- 则$B(x)$叫做$A(x)$的逆

- $n = 1$直接是逆元的情况
- 对于$n > 1$划归为一个更小的子问题

多项式求逆

- 设$A(x)C(x) \equiv 1\left(mod\ x^{\left\lceil\frac{n}{2}\right\rceil}\right)$
- 则仍然有$A(x)B(x) \equiv 1\left(mod\ x^{\left\lceil\frac{n}{2}\right\rceil}\right)$
- 所以$B(x) - C(x) \equiv 0\left(mod\ x^{\left\lceil\frac{n}{2}\right\rceil}\right)$
- $B(x)^2 - 2B(x)C(x) + C(x)^2 \equiv 0(mod\ x^n)$
- $A(x)B(x)^2 - 2A(x)B(x)C(x) + A(x)C(x)^2 \equiv 0(mod\ x^n)$
- $B(x) - 2C(x) + A(x)C(x)^2 \equiv 0(mod\ x^n)$
- $B(x) \equiv 2C(x) - A(x)C(x)^2(mod\ x^n)$
- 多项式乘法用NTT加速即可
- https://noip.ac/show_problem/3207

多项式开根

- $B(x)^2 \equiv A(x)(mod\ x^n)$
- 则称$B(x)$为$A(x)$的根

多项式开根

- 类似的方法
- $B^2(x) \equiv A(x)\left(mod\ x^{2n}\right)$
- $C^2(x) \equiv A(x)(mod\ x^n)$
- $C^4(x) - 2C^2(x)A(x) + A^2(x) \equiv 0\left(mod\ x^{2n}\right)$
- $C^4(x) + 2C^2(x)A(x) + A^2(x) \equiv 4C^2(x)A(x)\left(mod\ x^{2n}\right)$
- $\left(\frac{C^2(x)+A(x)}{2C(x)}\right)^2 \equiv A(x)\left(mod\ x^{2n}\right)$
- 多项式求逆+多项式乘法
- https://noip.ac/show_problem/3208

多项式除法

- 给定$A(x), B(x)$
- 求$C(x), D(x)$
- 使得
- $A(x) = C(x)B(x) + D(x)$

多项式除法

- 我们希望直接用多项式乘法和求逆来解决
- 所以要想办法把余数给搞掉
- 设$A(x)$最高项为x^n而$B(x)$为x^m
- 令$inv\big(f(x)\big) = x^n f\left(\frac{1}{x}\right)$
- 则$A(x) = B(x)C(x) + D(x)$
- $\Rightarrow x^n A\left(\frac{1}{x}\right) = x^m B\left(\frac{1}{x}\right) x^{n-m} C\left(\frac{1}{x}\right) + x^{n-m+1} \cdot x^{m-1} D\left(\frac{1}{x}\right)$

多项式除法

- $x^n A\left(\frac{1}{x}\right) = x^m B\left(\frac{1}{x}\right) x^{n-m} C\left(\frac{1}{x}\right) + x^{n-m+1} \cdot x^{m-1} D\left(\frac{1}{x}\right)$
- $\Rightarrow rev\big(A(x)\big) \equiv rev\big(B(x)\big) rev\big(C(x)\big) \big(mod\ x^{n-m+1}\big)$
- 所以只用求出$rev\big(B(x)\big)$的多项式逆
- 就可以用多项式乘法求出$C(x), D(x)$
- https://noip.ac/show_problem/3209

多项式取对数

- 求$g(x) \equiv \ln f(x) \ (mod\ x^n)$

多项式取对数

- 这个式子在一般意义下无法定义
- 所以我们要把取对数消掉

- $g'(x) = \frac{f'(x)}{f(x)}$
- 多项式求导、除法、积分之后即可
- https://noip.ac/show_problem/3210

多项式exp

- 求$g(x) \equiv e^{f(x)} (mod\ x^n)$

多项式exp

- 首先两边取对数
- $\ln g(x) = f(x) \Rightarrow \ln g(x) - f(x) = 0$

- 设$h\big(g(x)\big) = \ln g(x) - f(x)$，两边对$g$求导可得
- $h'\big(g(x)\big) = \frac{1}{g(x)}$
- 我们求$g(x)$可以转换为求$h\big(g(x)\big) = 0$ 的解

泰勒展开

- 函数$f(x)$在x_0处的展开式为
- $f(x) = \frac{f(x_0)}{0!} + \frac{f^{(1)}(x_0)}{1!} \cdot (x - x_0) + \frac{f^{(2)}(x_0)}{2!} \cdot (x - x_0)^2 + \cdots$

多项式exp

- 结合泰勒展开、倍增
- 假设已经求出来了$g\left(f(x)\right) \equiv 0(mod\ x^n)$的解$f_0(x)$
- 现在要解$g\left(f(x)\right) \equiv 0\left(mod\ x^{2n}\right)$
- 将$g\left(f(x)\right)$在$f_0(x)$处泰勒展开有
- $g\left(f(x)\right) = g\left(f_0(x)\right) + g'\left(f_0(x)\right)\left(f(x) - f_0(x)\right) + \frac{g''\left(f_0(x)\right)\left(f(x)-f_0(x)\right)^2}{2} + \cdots$

多项式exp

- $g\big(f(x)\big) = g\big(f_0(x)\big) + g'\big(f_0(x)\big)\big(f(x) - f_0(x)\big) + \frac{g''\big(f_0(x)\big)\big(f(x) - f_0(x)\big)^2}{2} + \cdots$
- 由于$f(x)$和$f_0(x)$最低的n项相同，所以两边对x^{2n}取模之后有
- $0 \equiv g\big(f(x)\big) \equiv g\big(f_0(x)\big) + g'\big(f_0(x)\big)\big(f(x) - f_0(x)\big)\big(mod\ x^{2n}\big)$

- 所以有
- $f(x) \equiv f_0(x) - \frac{g\big(f_0(x)\big)}{g'\big(f_0(x)\big)}\big(mod\ x^{2n}\big)$

多项式exp

- $g(x) \equiv g_0(x) - \frac{h\left(g_0(x)\right)}{h'\left(g_0(x)\right)} \left(mod\ x^{2n}\right)$

- $h\left(g(x)\right) = \ln g(x) - f(x)$
- $h'\left(g(x)\right) = \frac{1}{g(x)}$
- 带入可得
- $g(x) \equiv g_0(x) - \frac{\ln g_0(x) - f(x)}{\frac{1}{g_0(x)}} \equiv g_0(x)\left(1 - \ln g_0(x) + f(x)\right)\left(mod\ x^{2n}\right)$
- 多项式乘法、多项式对数、倍增即可
- https://noip.ac/show_problem/3211

多项式快速幂

- 求$f(x)^k \ mod \ x^n$

多项式快速幂

- $f(x)^k \ mod \ x^n = e^{k \ln f(x)} \ mod \ x^n$
- 多项式取对数、多项式exp

- 或者直接快速幂+多项式乘法即可
- https://noip.ac/show_problem/3212

Problem 1

- 给定小写字母串S
- 给定小写字母串T
- T中可能有通配符的存在
- 问S和T有多少个位置能够匹配

- $|S|, |T| \leq 10^5$

Problem 1

- 转化式子
- $\sum(S[i]-T[i])^2\times T[i]=0$
- 展开之后分别求卷积
- NTT

- https://noip.ac/show_problem/3213

Problem 2

- 给定n根木棍
- 长度为a_i
- 问任选三根能够组成三角形的概率

- $n, a_i \leq 10^5$

Problem 2

- 用一个桶记录每种长度的数量
- 卷积
- 枚举最长边长度
- 减去不合法的情况：另两条边都更长、有一条边更长、有一条边相等

- https://noip.ac/show_problem/3214

Problem 3

- $N \times M$的矩形
- 每个点有一个权值$p(i, j)$
- 每个点可以对周围距离严格小于r的格点贡献权值除以欧几里得距离+1
- 问最后所有点中价值最大的是多少
- $n, m \le 500, r \le 300$

Problem 3

- 将原网格上下左右扩充

- 令$A[i \times M + j] = p[i][j]$
- $B[(x + R) \times M + y + R] = \frac{1}{\sqrt{x^2+y^2+1}}$
- 则我们考虑$(i + x, j + y)$对(i, j)的贡献为
- $A[(i + x) \times M + j + y] \times B[(-x + R) \times M - y + R]$
- $C[(i + R) \times M + j + R]$
- https://noip.ac/show_problem/3215