多项式

生成函数

1. 序列 a 的**普通生成函数**: $F(x)=\sum a_n x^n$
2. 序列 a 的**指数生成函数**: $F(x)=\sum a_n \frac{x^n}{n!}$

泰勒展开式

1. $\frac{1}{1-x}=1+x+x^2+x^3+\cdots=\sum_{n=0}^{\infty} x^n$
2. $\frac{1}{1-x^2}=1+x^2+x^4+\cdots$
3. $\frac{1}{1-x^3}=1+x^3+x^6+\cdots$
4. $\frac{1}{(1-x)^2}=1+2x+3x^2+\cdots$
5. $e^x=1+\frac{x^1}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}+\cdots=\sum_{n=0}^{\infty} \frac{x^n}{n!}$
6. $e^{-x}=1-\frac{x^1}{1!}+\frac{x^2}{2!}-\frac{x^3}{3!}+\cdots$
7. $\frac{e^x+e^{-x}}{2}=1+\frac{x^2}{2!}+\frac{x^4}{4!}+\cdots$
8. $\frac{e^x-e^{-x}}{2}=x+\frac{x^3}{3!}+\frac{x^5}{5!}+\cdots$

有穷序列的生成函数

1. $1+x+x^2=\frac{1-x^3}{1-x}$
2. $1+x+x^2+x^3=\frac{1-x^4}{1-x}$

广义二项式定理

$$\frac{1}{(1-x)^n}=\sum_{i=0}^{\infty}\binom{n+i-1}{i}x^i$$

证明

1. 扩展域
   $(1+x)^n=\sum_{i=0}^{n}\binom{n}{i}x^i$，因 $i>n, \binom{n}{i}=0$。
2. 扩展指数为负数
   $\binom{-n}{i}=\frac{(-n)(-n-1)\cdots(-n-i+1)}{i!}=(-1)^i\times\frac{n(n+1)\cdots(n+i-1)}{i!}=(-1)^i\binom{n+i-1}{i}$
3. 括号内的加号变减号
   $(1-x)^{-n}=\sum_{i=0}^{\infty}(-1)^i\binom{n+i-1}{i}(-x)^i=\sum_{i=0}^{\infty}\binom{n+i-1}{i}x^i$

NTT快速数论变换 O(nlogn) 常数较大

```cpp
namespace ntt {

    const long long mod = 998244353;
    const long long G = 3;
    const long long Gi = 332748118; // G 在模 p 意义下的逆元

    class P {
    public:
        long long a{};

        P() = default;

        [[nodiscard]] long long get() const {
            return a;
        }

        explicit P(long long p) {
            a = p % mod;
            if (a < 0) {
                a += mod;
            }
        };

        P operator+(const P& rhs) const {
            return P{ a + rhs.a };
        }

        P operator-(const P& rhs) const {
            return P{ a - rhs.a };
        }

        P operator*(const P& rhs) const {
            return P{ a * rhs.a };
        }

        [[nodiscard]] P pow(long long n) const {
            P bas = P(a);
            P res = P(1);
            while (n > 0) {
                if (n % 2 == 1) {
```

```cpp
                res = res * bas;
            }
            bas = bas * bas;
            n /= 2;
        }
        return res;
    }

    [[nodiscard]] P inverse() const {
        return pow(mod - 2);
    }
};

class NTTMultiplier {
public:
    vector<int> input1;
    vector<int> input2;
    vector<int> bitInv;//位逆置换使用
    int size{};

    NTTMultiplier(const vector<int>& v1, const vector<int>& v2) {
        input1 = v1;
        input2 = v2;
        size = (int)(input1.size() + input2.size() - 1);
        int n = 1;
        while (n < size) {
            n <<= 1;
        }
        size = n;
        input1.resize(size);
        input2.resize(size);
        bitInv.resize(size);
        initBitInv();//位逆置换使用
    }

    void initBitInv() {
        bitInv[0] = 0;
        int log2n = (int)log2(size);
        for (int i = 1; i < size; i++) {
            int pre = (i & 1) << (log2n - 1);//第1位(奇数为1,偶数为0);
            int suf = bitInv[i >> 1] >> 1;   //第2到第n位(这是的递推公式);
            bitInv[i] = pre | suf;
        }
```

```
}

vector<int> multiply() {
    // 将输入转换为模数形式
    vector<P> nttInput1(input1.begin(), input1.end());
    vector<P> nttInput2(input2.begin(), input2.end());

    // 执行快速傅里叶变换
    fastNTT(nttInput1, false);
    fastNTT(nttInput2, false);
    // 对应位置相乘
    for (int i = 0; i < size; i++) {
        nttInput1[i] = nttInput1[i] * nttInput2[i];
    }
    // 执行反向快速傅里叶变换
    fastNTT(nttInput1, true);
    P invSize = P(size).inverse();
    for (int i = 0; i < size; i++) {
        nttInput1[i] = nttInput1[i] * invSize;
    }

    // 取实部并取整
    vector<int> result(size);
    for (int i = 0; i < size; i++) {
        result[i] = (int)nttInput1[i].get();
    }

    input1.clear();
    input2.clear();
    size = 0;

    return result;
}


void bitRev(vector<P>& arr) {
    for (int i = 1; i < arr.size(); i++) {
        if (i < bitInv[i]) {
            swap(arr[i], arr[bitInv[i]]);        //交换
        }
    }
}
```

```
void fastNTT(vector<P>& data, bool inverse) {
    int n = (int)data.size();
    bitRev(data);
    P bas = inverse ? P(Gi) : P(G);
    for (int len = 2; len <= n; len *= 2) {
        long long angle = (long long)(mod - 1) / len;
        P wn = bas.pow(angle);
        for (int i = 0; i < n; i += len) {
            P w(1);
            for (int j = i; j < i + len / 2; j++) {
                P evenVal = data[j];
                P oddVal = data[j + len / 2] * w;
                data[j] = evenVal + oddVal;
                data[j + len / 2] = evenVal - oddVal;
                w = w * wn;
            }
        }
    }
}

void ntt(vector<P>& data, bool inverse) { // NOLINT(misc-no-recursion)
    int n = (int)data.size();
    if (n == 1) {
        return;
    }
    vector<P> even(n / 2);
    vector<P> odd(n / 2);
    // 分离奇偶项
    for (int i = 0; i < n / 2; i++) {
        even[i] = data[2 * i];
        odd[i] = data[2 * i + 1];
    }
    // 递归进行快速傅里叶变换
    ntt(even, inverse);
    ntt(odd, inverse);

    P bas = inverse ? P(Gi) : P(G);
    long long angle = (long long)(mod - 1) / n; // NOLINT(cppcoreguidelines-narrowing-c
    P w(1);
    P wn = bas.pow(angle);
    // 合并结果
    for (int i = 0; i < n / 2; i++) {
```

```
                P evenVal = even[i];
                P oddVal = odd[i] * w;
                data[i] = evenVal + oddVal;
                data[i + n / 2] = evenVal - oddVal;
                w = w * wn;
            }
        }
    };
}
```