Fast Fourier Transform and Polynomial Operations

wcysai

Nanjing University

wcysai@foxmail.com

2018 年 4 月 22 日

Overview

1. FFT
   - Review on FFT
   - FFT with divide and conquer
   - FFT with CDQ divide and conquer
2. Polynomial Operations
   - Polynomial Inverse
   - Newton's Method
   - Polynomial Square Root
   - Polynomial Logarithm and Exponentiation

Just the basic FFT......

Just the basic FFT......

- 可以在$O(n \log n)$时间内快速求出两个多项式的卷积

Just the basic FFT......

- 可以在$O(n\log n)$时间内快速求出两个多项式的卷积

$$C_i = \sum_{k=0}^{i} A_k \cdot B_{i-k}$$

Just the basic FFT......

- 可以在$O(n \log n)$时间内快速求出两个多项式的卷积

$$C_i = \sum_{k=0}^{i} A_k \cdot B_{i-k}$$

- 用Complex类实现，可能会存在精度误差

Fast Number-Theoretic Transform

Fast Number-Theoretic Transform

- 在模数为$P = Q \cdot 2^K + 1$的情况下,可以运用快速数论变换(NTT)

Fast Number-Theoretic Transform

- 在模数为$P = Q \cdot 2^K + 1$的情况下,可以运用快速数论变换(NTT)
- 对于P的原根g,$\{1, g^Q, g^{2Q}, ...\}$构成2^K阶循环群，此时可以用g替代ω

Fast Number-Theoretic Transform

- 在模数为$P = Q \cdot 2^K + 1$的情况下,可以运用快速数论变换(NTT)
- 对于P的原根g,$\{1, g^Q, g^{2Q}, ...\}$构成2^K阶循环群，此时可以用g替代ω
- 没有精度误差

Some simple problems

Some simple problems

- 如果模数为非NTT质数(比如$1e9 + 7$),而复数类FFT精度又不够怎么办?

Some simple problems

- 如果模数为非NTT质数(比如$1e9+7$),而复数类FFT精度又不够怎么办?
- 一般来说有两种处理方法

Some simple problems

- 如果模数为非NTT质数(比如$1e9+7$),而复数类FFT精度又不够怎么办?
- 一般来说有两种处理方法
  1. 减少复数类FFT的精度误差
  2. 使得NTT能够处理任意模数的情况

Reduce precision errors

Reduce precision errors

- 改变计算ω_n的方法

Reduce precision errors

- 改变计算ω_n的方法
  - $\omega_n = \omega_{n-1} \cdot \omega$ add up rounding errors n times

Reduce precision errors

- 改变计算ω_n的方法
  - $\omega_n = \omega_{n-1} \cdot \omega$ add up rounding errors n times
  - $\omega_n = \omega_{n/2} \cdot \omega_{(n+1)/2}$ add up rounding errors $\log n$ times

Reduce precision errors

- 改变计算ω_n的方法
  - $\omega_n = \omega_{n-1} \cdot \omega$ add up rounding errors n times
  - $\omega_n = \omega_{n/2} \cdot \omega_{(n+1)/2}$ add up rounding errors $\log n$ times
- 将卷积数组分成$A[i] = a_0[i]m + a_1[i]$的形式,其中$m = O(\sqrt{MOD})$

Reduce precision errors

- 改变计算ω_n的方法
  - $\omega_n = \omega_{n-1} \cdot \omega$ add up rounding errors n times
  - $\omega_n = \omega_{n/2} \cdot \omega_{(n+1)/2}$ add up rounding errors $\log n$ times
- 将卷积数组分成$A[i] = a_0[i]m + a_1[i]$的形式,其中$m = O(\sqrt{MOD})$
  - $(a_0 m + a_1)(b_0 m + b_1) = a_0 b_0 m^2 + (a_1 b_0 + a_0 b_1)m + a_1 b_1$

Reduce precision errors

- 改变计算ω_n的方法
  - $\omega_n = \omega_{n-1} \cdot \omega$ add up rounding errors n times
  - $\omega_n = \omega_{n/2} \cdot \omega_{(n+1)/2}$ add up rounding errors $\log n$ times
- 将卷积数组分成$A[i] = a_0[i]m + a_1[i]$的形式,其中$m = O(\sqrt{MOD})$
  - $(a_0 m + a_1)(b_0 m + b_1) = a_0 b_0 m^2 + (a_1 b_0 + a_0 b_1)m + a_1 b_1$
  - 需要进行几次FFT?

Reduce precision errors

- 改变计算ω_n的方法
  - $\omega_n = \omega_{n-1} \cdot \omega$ add up rounding errors n times
  - $\omega_n = \omega_{n/2} \cdot \omega_{(n+1)/2}$ add up rounding errors $\log n$ times
- 将卷积数组分成$A[i] = a_0[i]m + a_1[i]$的形式,其中$m = O(\sqrt{MOD})$
  - $(a_0m + a_1)(b_0m + b_1) = a_0b_0m^2 + (a_1b_0 + a_0b_1)m + a_1b_1$
  - 需要进行几次FFT?
  - 4次卷积=12次FFT?

Reduce precision errors

- 改变计算ω_n的方法
  - $\omega_n = \omega_{n-1} \cdot \omega$ add up rounding errors n times
  - $\omega_n = \omega_{n/2} \cdot \omega_{(n+1)/2}$ add up rounding errors $\log n$ times
- 将卷积数组分成$A[i] = a_0[i]m + a_1[i]$的形式,其中$m = O(\sqrt{MOD})$
  - $(a_0m + a_1)(b_0m + b_1) = a_0b_0m^2 + (a_1b_0 + a_0b_1)m + a_1b_1$
  - 需要进行几次FFT?
  - 4次卷积=12次FFT? 其实只要4次就可以了

Reduce precision errors

- 改变计算ω_n的方法
  - $\omega_n = \omega_{n-1} \cdot \omega$ add up rounding errors n times
  - $\omega_n = \omega_{n/2} \cdot \omega_{(n+1)/2}$ add up rounding errors $\log n$ times
- 将卷积数组分成$A[i] = a_0[i]m + a_1[i]$的形式,其中$m = O(\sqrt{MOD})$
  - $(a_0m + a_1)(b_0m + b_1) = a_0b_0m^2 + (a_1b_0 + a_0b_1)m + a_1b_1$
  - 需要进行几次FFT?
  - 4次卷积=12次FFT? 其实只要4次就可以了
    $FFT(a_0, a_1)$ $FFT(b_0, b_1)$ $FFT^{-1}(a_0b_0, a_1b_1)$ $FFT^{-1}(a_0b_1 + a_1b_0)$

Reduce precision errors

- 改变计算ω_n的方法
  - $\omega_n = \omega_{n-1} \cdot \omega$ add up rounding errors n times
  - $\omega_n = \omega_{n/2} \cdot \omega_{(n+1)/2}$ add up rounding errors $\log n$ times
- 将卷积数组分成$A[i] = a_0[i]m + a_1[i]$的形式,其中$m = O(\sqrt{MOD})$
  - $(a_0 m + a_1)(b_0 m + b_1) = a_0 b_0 m^2 + (a_1 b_0 + a_0 b_1)m + a_1 b_1$
  - 需要进行几次FFT?
  - 4次卷积=12次FFT? 其实只要4次就可以了
    $FFT(a_0, a_1)$ $FFT(b_0, b_1)$ $FFT^{-1}(a_0 b_0, a_1 b_1)$ $FFT^{-1}(a_0 b_1 + a_1 b_0)$
- 进行上述两项处理后的FFT在long long范围内进行整多项式卷积都不会出现浮点误差

NTT with arbitrary modulos

NTT with arbitrary modulos

- 假设模数为m,NTT变换的长度为n,那么每个数的大小不会超过$n(m-1)^2$

NTT with arbitrary modulos

- 假设模数为m,NTT变换的长度为n,那么每个数的大小不会超过$n(m-1)^2$
- 这样我们可以选取k个NTT模数$p_1, p_2, ..., p_k$,要求满足

$$\prod_{i=1}^{k} p_k > n(m-1)^2$$

NTT with arbitrary modulos

- 假设模数为m,NTT变换的长度为n,那么每个数的大小不会超过$n(m-1)^2$
- 这样我们可以选取k个NTT模数$p_1, p_2, ..., p_k$,要求满足

$$\prod_{i=1}^{k} p_k > n(m-1)^2$$

- 如此我们便可以分别在 $\bmod p_k$的剩余系下做变换,最后使用中国剩余定理合并

NTT with arbitrary modulos

- 假设模数为m,NTT变换的长度为n,那么每个数的大小不会超过$n(m-1)^2$
- 这样我们可以选取k个NTT模数$p_1, p_2, ..., p_k$,要求满足

$$\prod_{i=1}^{k} p_k > n(m-1)^2$$

- 如此我们便可以分别在 $\mod p_k$的剩余系下做变换,最后使用中国剩余定理合并
- 可能需要高精度或者$__int128$

Application: Stirling number of the second kind

Application: Stirling number of the second kind

- $S(n,k)$ 为第二类斯特林数,表示将n个数划分为k个非空集合的方案数

Application: Stirling number of the second kind

- $S(n,k)$ 为第二类斯特林数,表示将n个数划分为k个非空集合的方案数
- $S(n,k) = S(n-1,k-1) + S(n-1,k) \times k$

Application: Stirling number of the second kind

- $S(n, k)$ 为第二类斯特林数,表示将n个数划分为k个非空集合的方案数
- $S(n, k) = S(n-1, k-1) + S(n-1, k) \times k$
- $S(n, k) = \sum_{i=1}^{n} S(n-i, k-1) \times \binom{n-1}{i-1}$

Application: Stirling number of the second kind

- $S(n,k)$ 为第二类斯特林数,表示将n个数划分为k个非空集合的方案数
- $S(n,k) = S(n-1,k-1) + S(n-1,k) \times k$
- $S(n,k) = \sum_{i=1}^{n} S(n-i,k-1) \times \binom{n-1}{i-1}$
- 运用容斥原理得到$S(n,k) = \frac{1}{k!}\sum_{i=0}^{k}(-1)^k\binom{k}{i}(k-i)^n = \sum_{i=0}^{k}\frac{(-1)^i}{i!}\frac{(k-i)^n}{(k-i)!}$

Application: Stirling number of the second kind

- $S(n,k)$ 为第二类斯特林数,表示将n个数划分为k个非空集合的方案数
- $S(n,k) = S(n-1,k-1) + S(n-1,k) \times k$
- $S(n,k) = \sum\limits_{i=1}^{n} S(n-i,k-1) \times \binom{n-1}{i-1}$
- 运用容斥原理得到$S(n,k) = \frac{1}{k!}\sum\limits_{i=0}^{k}(-1)^k\binom{k}{i}(k-i)^n = \sum\limits_{i=0}^{k}\frac{(-1)^i}{i!}\frac{(k-i)^n}{(k-i)!}$
- 可利用FFT快速计算一行的第二类斯特林数,复杂度$O(n\log n)$

Helvetic Coding Contest 2018 F3 Lightsabers(hard)

Helvetic Coding Contest 2018 F3 Lightsabers(hard)

- 有n件物品,每种物品的颜色为$1-m$间的一个整数,现从这n件物品中取出k件,求取出的物品的不同组合的个数,答案对1009取模

Helvetic Coding Contest 2018 F3 Lightsabers(hard)

- 有n件物品,每种物品的颜色为$1-m$间的一个整数,现从这n件物品中取出k件,求取出的物品的不同组合的个数,答案对1009取模
- $1 \leq n \leq 2 \cdot 10^5, 1 \leq m \leq n, 1 \leq k \leq n$

Helvetic Coding Contest 2018 F3 Lightsabers(hard)

- 有n件物品,每种物品的颜色为$1-m$间的一个整数,现从这n件物品中取出k件,求取出的物品的不同组合的个数,答案对1009取模
- $1 \le n \le 2 \cdot 10^5, 1 \le m \le n, 1 \le k \le n$

  Solution

Helvetic Coding Contest 2018 F3 Lightsabers(hard)

- 有n件物品,每种物品的颜色为$1-m$间的一个整数,现从这n件物品中取出k件,求取出的物品的不同组合的个数,答案对1009取模
- $1 \le n \le 2 \cdot 10^5, 1 \le m \le n, 1 \le k \le n$

Solution

- 本质是多重集组合数

Helvetic Coding Contest 2018 F3 Lightsabers(hard)

- 有n件物品,每种物品的颜色为$1-m$间的一个整数,现从这n件物品中取出k件,求取出的物品的不同组合的个数,答案对1009取模
- $1 \leq n \leq 2 \cdot 10^5, 1 \leq m \leq n, 1 \leq k \leq n$

Solution

- 本质是多重集组合数
- 对于每一种颜色$l \in \{1, 2, ..., m\}$,定义多项式$p_l(x) = \sum_{j=0}^{c_m} x^j$

Helvetic Coding Contest 2018 F3 Lightsabers(hard)

- 有n件物品,每种物品的颜色为$1-m$间的一个整数,现从这n件物品中取出k件,求取出的物品的不同组合的个数,答案对1009取模
- $1 \leq n \leq 2 \cdot 10^5, 1 \leq m \leq n, 1 \leq k \leq n$

Solution

- 本质是多重集组合数
- 对于每一种颜色$l \in \{1, 2, ..., m\}$,定义多项式$p_l(x) = \sum_{j=0}^{c_m} x^j$
- 答案是多项式$p(x) = \prod_{l=1}^{m} p_l(x)$中$x^k$的系数

FFT with divide and conquer

FFT with divide and conquer

- 如何快速计算$p(x) = \prod_{l=1}^{m} p_l(x)$?

FFT with divide and conquer

- 如何快速计算$p(x) = \prod_{l=1}^{m} p_l(x)$?
- 运用分治的思想，递归地处理前一半多项式乘积和后一半多项式乘积，利用FFT进行合并

FFT with divide and conquer

- 如何快速计算$p(x) = \prod_{l=1}^{m} p_l(x)$?
- 运用分治的思想，递归地处理前一半多项式乘积和后一半多项式乘积，利用FFT进行合并
- 时间复杂度$O(n \log^2 n)$

FFT with divide and conquer

- 如何快速计算$p(x) = \prod_{l=1}^{m} p_l(x)$?
- 运用分治的思想，递归地处理前一半多项式乘积和后一半多项式乘积，利用FFT进行合并
- 时间复杂度$O(n \log^2 n)$
- 空间复杂度为$O(n \log n)/O(n)$

FFT with divide and conquer

- 如何快速计算$p(x) = \prod_{l=1}^{m} p_l(x)$?
- 运用分治的思想，递归地处理前一半多项式乘积和后一半多项式乘积，利用FFT进行合并
- 时间复杂度$O(n \log^2 n)$
- 空间复杂度为$O(n \log n)/O(n)$
- 其实也可以每次贪心地选取系数最小的两个多项式进行合并,时间复杂度也为$O(n \log^2 n)$

Application: Stirling number of the first kind

Application: Stirling number of the first kind

- $s(n,k)$ 为第一类斯特林数,表示将n个数划分为k个圆排列的方案数

Application: Stirling number of the first kind

- $s(n,k)$ 为第一类斯特林数,表示将n个数划分为k个圆排列的方案数
- $s(n,k) = s(n-1,k-1) + s(n-1,k) \times (n-1)$

Application: Stirling number of the first kind

- $s(n,k)$ 为第一类斯特林数,表示将n个数划分为k个圆排列的方案数
- $s(n,k) = s(n-1,k-1) + s(n-1,k) \times (n-1)$
- $s(n,k) = \sum_{i=1}^{n} s(n-i,k-1) \times (i-1)! \times \binom{n-1}{i-1}$

Application: Stirling number of the first kind

- $s(n,k)$ 为第一类斯特林数,表示将n个数划分为k个圆排列的方案数
- $s(n,k) = s(n-1,k-1) + s(n-1,k) \times (n-1)$
- $s(n,k) = \sum_{i=1}^{n} s(n-i,k-1) \times (i-1)! \times \binom{n-1}{i-1}$
- $x^{\bar{n}} = \prod_{i=1}^{n} (x+i-1) = \sum_{i=0}^{n} s(n,i) \times x^i$

Application: Stirling number of the first kind

- $s(n,k)$ 为第一类斯特林数,表示将n个数划分为k个圆排列的方案数
- $s(n,k) = s(n-1,k-1) + s(n-1,k) \times (n-1)$
- $s(n,k) = \sum_{i=1}^{n} s(n-i,k-1) \times (i-1)! \times \binom{n-1}{i-1}$
- $x^{\bar{n}} = \prod_{i=1}^{n} (x+i-1) = \sum_{i=0}^{n} s(n,i) \times x^i$
- 可利用分治FFT快速计算一行的第一类斯特林数,复杂度$O(n\log^2 n)$

2017 CCPC Hangzhou Onsite G Marriage

2017 CCPC Hangzhou Onsite G Marriage

- S市里有n个家庭,其中第i个家庭中有a_i个男孩和b_i个女孩(a_i和b_i可能为0),现在S市的政府决定包办婚姻,指定男孩与女孩的配对方式,问避免近亲结婚的配对方式的种数，答案对998244353取模

2017 CCPC Hangzhou Onsite G Marriage

- S市里有n个家庭,其中第i个家庭中有a_i个男孩和b_i个女孩(a_i和b_i可能为0),现在S市的政府决定包办婚姻,指定男孩与女孩的配对方式,问避免近亲结婚的配对方式的种数，答案对998244353取模
- $1 \leq n \leq 10^5, a_i, b_i \leq 10^5, 0 < \sum a_i = \sum b_i \leq 10^5$

2017 CCPC Hangzhou Onsite G Marriage

- S市里有n个家庭,其中第i个家庭中有a_i个男孩和b_i个女孩(a_i和b_i可能为0),现在S市的政府决定包办婚姻,指定男孩与女孩的配对方式,问避免近亲结婚的配对方式的种数，答案对998244353取模
- $1 \leq n \leq 10^5, a_i, b_i \leq 10^5, 0 < \sum a_i = \sum b_i \leq 10^5$

Solution

2017 CCPC Hangzhou Onsite G Marriage

- S市里有n个家庭,其中第i个家庭中有a_i个男孩和b_i个女孩(a_i和b_i可能为0),现在S市的政府决定包办婚姻,指定男孩与女孩的配对方式,问避免近亲结婚的配对方式的种数，答案对998244353取模
- $1 \le n \le 10^5, a_i, b_i \le 10^5, 0 < \sum a_i = \sum b_i \le 10^5$

Solution

- 注意到$a_i = b_i = 1$的时候是经典的错排问题,所以可以很自然地想到用容斥来做

2017 CCPC Hangzhou Onsite G Marriage

- S市里有n个家庭,其中第i个家庭中有a_i个男孩和b_i个女孩(a_i和b_i可能为0),现在S市的政府决定包办婚姻,指定男孩与女孩的配对方式,问避免近亲结婚的配对方式的种数，答案对998244353取模
- $1 \le n \le 10^5, a_i, b_i \le 10^5, 0 < \sum a_i = \sum b_i \le 10^5$

Solution

- 注意到$a_i = b_i = 1$的时候是经典的错排问题,所以可以很自然地想到用容斥来做
- 对于每一个k,要求出至少有k对近亲的方案数

2017 CCPC Hangzhou Onsite G Marriage

- S市里有n个家庭,其中第i个家庭中有a_i个男孩和b_i个女孩(a_i和b_i可能为0),现在S市的政府决定包办婚姻,指定男孩与女孩的配对方式,问避免近亲结婚的配对方式的种数，答案对998244353取模
- $1 \le n \le 10^5, a_i, b_i \le 10^5, 0 < \sum a_i = \sum b_i \le 10^5$

Solution

- 注意到$a_i = b_i = 1$的时候是经典的错排问题,所以可以很自然地想到用容斥来做
- 对于每一个k,要求出至少有k对近亲的方案数
- 对每个家庭定义多项式$f_i(x) = \sum_j c_j x^j$,其中c_j表示这个家庭中出现j对近亲的方案数,之后分治FFT即可

FFT with CDQ divide and conquer

FFT with CDQ divide and conquer

- 如何快速计算形如$f_n = \sum_{i=1}^{n-1} f_i \times A_{n-i}$的递归式?

FFT with CDQ divide and conquer

- 如何快速计算形如$f_n = \sum_{i=1}^{n-1} f_i \times A_{n-i}$的递归式?
- 直接递推时间复杂度为$O(n^2)$

FFT with CDQ divide and conquer

- 如何快速计算形如$f_n = \sum_{i=1}^{n-1} f_i \times A_{n-i}$的递归式?
- 直接递推时间复杂度为$O(n^2)$
- 利用CDQ分治的思想，对于求解一个区间$[left, right]$的函数值，我们先递归地求解左半部分$[left, mid]$的值，再利用FFT计算左半部分的函数值对于右半部分函数值的贡献，接着递归求解右半部分$[mid+1, right]$

FFT with CDQ divide and conquer

- 如何快速计算形如$f_n = \sum_{i=1}^{n-1} f_i \times A_{n-i}$的递归式?
- 直接递推时间复杂度为$O(n^2)$
- 利用CDQ分治的思想，对于求解一个区间$[left, right]$的函数值，我们先递归地求解左半部分$[left, mid]$的值，再利用FFT计算左半部分的函数值对于右半部分函数值的贡献，接着递归求解右半部分$[mid+1, right]$
- 时间复杂度为$O(n\log^2 n)$

CS Academy Round 9(Beta) G Jetpack

CS Academy Round 9(Beta) G Jetpack

- Raccoon在二维坐标平面上从$(0,0)$走到$(N,0)$,他随身带着一个jetpack。假设他某一刻在(x,y)，每一步他可以走到：

CS Academy Round 9(Beta) G Jetpack

- Raccoon在二维坐标平面上从$(0,0)$走到$(N,0)$,他随身带着一个jetpack。假设他某一刻在(x,y)，每一步他可以走到：
  - $(x+1,y+1)$如果他使用jetpack
  - $(x+1,max(0,y-1))$如果他不使用jetpack

CS Academy Round 9(Beta) G Jetpack

- Raccoon在二维坐标平面上从$(0,0)$走到$(N,0)$,他随身带着一个jetpack。假设他某一刻在(x,y)，每一步他可以走到：
  - $(x+1,y+1)$如果他使用jetpack
  - $(x+1,max(0,y-1))$如果他不使用jetpack
- jetpack有K点充能点数，所以Raccoon不能在x轴上方停留超过$2\times K$步。与此同时，当Raccoon每次接触地面时，jetpack的能量点数又会重新充满，问从$(0,0)$走到$(N,0)$的不同的方案数，答案对10^9+7取模

CS Academy Round 9(Beta) G Jetpack

- Raccoon在二维坐标平面上从$(0,0)$走到$(N,0)$,他随身带着一个jetpack。假设他某一刻在(x,y)，每一步他可以走到：
  - $(x+1,y+1)$如果他使用jetpack
  - $(x+1,max(0,y-1))$如果他不使用jetpack
- jetpack有K点充能点数，所以Raccoon不能在x轴上方停留超过$2\times K$步。与此同时，当Raccoon每次接触地面时，jetpack的能量点数又会重新充满，问从$(0,0)$走到$(N,0)$的不同的方案数，答案对10^9+7取模
- $1\le N,K\le 10^5$

CS Academy Round 9(Beta) G Jetpack

Solution

CS Academy Round 9(Beta) G Jetpack

Solution

- 考虑一段长度为k的飞行，显然k是偶数，这段飞行的方案数为$C_{k/2}$(C_n为卡塔兰数)

Solution

- 考虑一段长度为k的飞行，显然k是偶数，这段飞行的方案数为$C_{k/2}$(C_n为卡塔兰数)
- 令dp_i表示从$(0,0)$走到$(i,0)$的方案数,同时

$$令 coef_i = \begin{cases} 0 & \text{i为奇数} \\ C_{i/2} & \text{i为偶数} \end{cases}$$

CS Academy Round 9(Beta) G Jetpack

Solution

- 考虑一段长度为k的飞行，显然k是偶数，这段飞行的方案数为$C_{k/2}$(C_n为卡塔兰数)
- 令dp_i表示从$(0,0)$走到$(i,0)$的方案数,同时

$$令coef_i = \begin{cases} 0 & \text{i为奇数} \\ C_{i/2} & \text{i为偶数} \end{cases}$$

- 这样可以得到递归式$dp_i = \sum_{j=0}^{i-1} dp_{i-j} \times coef_j$

CS Academy Round 9(Beta) G Jetpack

Solution

- 考虑一段长度为k的飞行，显然k是偶数，这段飞行的方案数为$C_{k/2}$(C_n为卡塔兰数)
- 令dp_i表示从$(0,0)$走到$(i,0)$的方案数,同时

  令$coef_i = \begin{cases} 0 & \text{i为奇数} \\ C_{i/2} & \text{i为偶数} \end{cases}$
- 这样可以得到递归式$dp_i = \sum_{j=0}^{i-1} dp_{i-j} \times coef_j$
- 利用CDQ分治+FFT,时间复杂度为$O(n\log^2 n)$

CS Academy Round 9(Beta) G Jetpack

Solution

- 考虑一段长度为k的飞行，显然k是偶数，这段飞行的方案数为$C_{k/2}$(C_n为卡塔兰数)
- 令dp_i表示从$(0,0)$走到$(i,0)$的方案数,同时

$$令coef_i = \begin{cases} 0 & \text{i为奇数} \\ C_{i/2} & \text{i为偶数} \end{cases}$$

- 这样可以得到递归式$dp_i = \sum_{j=0}^{i-1} dp_{i-j} \times coef_j$
- 利用CDQ分治+FFT,时间复杂度为$O(n\log^2 n)$
- 需要利用之前提到的技巧处理精度/模数

CS Academy Round 24 G Colored Forests

CS Academy Round 24 G Colored Forests

- 对于一个带标号的树，我们将它的每一个节点都染上M种不同颜色中的一种，如果一棵树上M种颜色都至少出现了一次，我们就称这个树是colorful的。一个森林是colorful的当且仅当这个森林中的所有树都是colorful的。

CS Academy Round 24 G Colored Forests

- 对于一个带标号的树，我们将它的每一个节点都染上M种不同颜色中的一种，如果一棵树上M种颜色都至少出现了一次，我们就称这个树是colorful的。一个森林是colorful的当且仅当这个森林中的所有树都是colorful的。
- 给定N和M,对于所有$1 \leq i \leq N$,求出i个点的colorful的森林的种数，结果对924844033取模。

CS Academy Round 24 G Colored Forests

- 对于一个带标号的树，我们将它的每一个节点都染上M种不同颜色中的一种，如果一棵树上M种颜色都至少出现了一次，我们就称这个树是colorful的。一个森林是colorful的当且仅当这个森林中的所有树都是colorful的。
- 给定N和M,对于所有$1 \le i \le N$,求出i个点的colorful的森林的种数，结果对924844033取模。
- $1 \le N \le 10^5, 1 \le M \le 50$

CS Academy Round 24 G Colored Forests

Solution

CS Academy Round 24 G Colored Forests

Solution

- 令T_n表示n个节点的colorful树的数量，根据Cayley's formula,n个节点的带标号树的数量是n^{n-2},因此$T_n = n^{n-2} \times S(n, m)$

Solution

- 令T_n表示n个节点的colorful树的数量，根据Cayley's formula,n个节点的带标号树的数量是n^{n-2},因此$T_n = n^{n-2} \times S(n,m)$
- 令F_n表示n个节点的colorful森林的数量,容易看出$F_n = \sum_{k=1}^{n} \binom{n-1}{k-1} \times T_k \times F_{n-k}$

CS Academy Round 24 G Colored Forests

Solution

- 令T_n表示n个节点的colorful树的数量，根据Cayley's formula,n个节点的带标号树的数量是n^{n-2},因此$T_n = n^{n-2} \times S(n, m)$
- 令F_n表示n个节点的colorful森林的数量,容易看出$F_n = \sum\limits_{k=1}^{n} \binom{n-1}{k-1} \times T_k \times F_{n-k}$
- $F_n = \sum\limits_{k=1}^{n} (n-1)! \times \frac{T_k}{(k-1)!} \times \frac{F_{n-k}}{(n-k)!}$

Solution

- 令T_n表示n个节点的colorful树的数量，根据Cayley's formula,n个节点的带标号树的数量是n^{n-2},因此$T_n = n^{n-2} \times S(n,m)$
- 令F_n表示n个节点的colorful森林的数量,容易看出$F_n = \sum\limits_{k=1}^{n} \binom{n-1}{k-1} \times T_k \times F_{n-k}$
- $F_n = \sum\limits_{k=1}^{n} (n-1)! \times \frac{T_k}{(k-1)!} \times \frac{F_{n-k}}{(n-k)!}$
- $\frac{F_n}{n!} \times n = \sum\limits_{k=1}^{n} (n-1)! \times \frac{T_k}{(k-1)!} \times \frac{F_{n-k}}{(n-k)!}$

CS Academy Round 24 G Colored Forests

Solution

- 令T_n表示n个节点的colorful树的数量，根据Cayley's formula,n个节点的带标号树的数量是n^{n-2},因此$T_n = n^{n-2} \times S(n,m)$
- 令F_n表示n个节点的colorful森林的数量,容易看出$F_n = \sum\limits_{k=1}^{n} \binom{n-1}{k-1} \times T_k \times F_{n-k}$
- $F_n = \sum\limits_{k=1}^{n} (n-1)! \times \frac{T_k}{(k-1)!} \times \frac{F_{n-k}}{(n-k)!}$
- $\frac{F_n}{n!} \times n = \sum\limits_{k=1}^{n} (n-1)! \times \frac{T_k}{(k-1)!} \times \frac{F_{n-k}}{(n-k)!}$
- 利用CDQ分治+FFT,时间复杂度为$O(n\log^2 n + nm)$

Polynomial Operations

Polynomial Operations

- 利用FFT解决问题的思路在于将问题转化为多项式(生成函数)，然后通过多项式乘法的快速运算在时限内解决问题

Polynomial Operations

- 利用FFT解决问题的思路在于将问题转化为多项式(生成函数)，然后通过多项式乘法的快速运算在时限内解决问题
- 那么，对于多项式乘法之外的操作呢？

Polynomial Operations

- 利用FFT解决问题的思路在于将问题转化为多项式(生成函数)，然后通过多项式乘法的快速运算在时限内解决问题
- 那么，对于多项式乘法之外的操作呢？
- 比如PE258中所要求的多项式除法？

Polynomial Operations

- 利用FFT解决问题的思路在于将问题转化为多项式(生成函数)，然后通过多项式乘法的快速运算在时限内解决问题
- 那么，对于多项式乘法之外的操作呢？
- 比如PE258中所要求的多项式除法？
- 以及多项式开根，多项式的幂次，多项式求指数，多项式取对数？

Polynomial Operations

- 利用FFT解决问题的思路在于将问题转化为多项式(生成函数)，然后通过多项式乘法的快速运算在时限内解决问题
- 那么，对于多项式乘法之外的操作呢？
- 比如PE258中所要求的多项式除法？
- 以及多项式开根，多项式的幂次，多项式求指数，多项式取对数？
- 这些方法在部分生成函数计数问题中非常有用

Polynomial Operations

- 利用FFT解决问题的思路在于将问题转化为多项式(生成函数)，然后通过多项式乘法的快速运算在时限内解决问题
- 那么，对于多项式乘法之外的操作呢？
- 比如PE258中所要求的多项式除法？
- 以及多项式开根，多项式的幂次，多项式求指数，多项式取对数？
- 这些方法在部分生成函数计数问题中非常有用
- 常数大得可怕(划掉

Basic Concepts

Basic Concepts

- 对于一个多项式$A(x)$，我们称其最高项的次数为这个多项式的度，记作$deg\ A$

Basic Concepts

- 对于一个多项式$A(x)$，我们称其最高项的次数为这个多项式的度，记作$deg\ A$
- 对于多项式$A(x), B(x)$,存在唯一的$Q(x), R(x)$满足$A(x) = B(x)Q(x) + R(x)$,其中$deg\ R < deg\ B$，我们就称$Q(x)$是$B(x)$除$A(x)$的商，$R(x)$是$B(x)$除$A(x)$的余数，记作$A(x) \equiv R(x) (\mathrm{mod}\ B(x))$

Polynomial Inverse

Polynomial Inverse

- 对于一个多项式$A(x)$，如果存在$B(x)$且满足$deg\ B \leq deg\ A$并且$A(x)B(x) \equiv 1 (\mathrm{mod}\ x^n)$,那么就称$B(x)$是$A(x)$在$\mathrm{mod}\ x^n$意义下的逆元，记作$A^{-1}(x)$

Polynomial Inverse

- 对于一个多项式$A(x)$，如果存在$B(x)$且满足$deg\ B \leq deg\ A$并且$A(x)B(x) \equiv 1 (\bmod\ x^n)$,那么就称$B(x)$是$A(x)$在$\bmod\ x^n$意义下的逆元，记作$A^{-1}(x)$
- 假设在$\bmod\ x^{\lceil \frac{n}{2} \rceil}$意义下$A(x)$的逆元是$B'(x)$,那么就有$A(x)B(x) \equiv 1 (\bmod\ x^{\lceil \frac{n}{2} \rceil})$以及$A(x)B'(x) \equiv 1 (\bmod\ x^{\lceil \frac{n}{2} \rceil})$

Polynomial Inverse

- 对于一个多项式$A(x)$，如果存在$B(x)$且满足$deg\ B \leq deg\ A$并且$A(x)B(x) \equiv 1(\bmod\ x^n)$,那么就称$B(x)$是$A(x)$在$\bmod\ x^n$意义下的逆元，记作$A^{-1}(x)$
- 假设在$\bmod\ x^{\lceil \frac{n}{2} \rceil}$意义下$A(x)$的逆元是$B'(x)$,那么就有$A(x)B(x) \equiv 1(\bmod\ x^{\lceil \frac{n}{2} \rceil})$以及$A(x)B'(x) \equiv 1(\bmod\ x^{\lceil \frac{n}{2} \rceil})$
- $B(x) - B'(x) \equiv 0(\bmod\ x^{\lceil \frac{n}{2} \rceil})$

Polynomial Inverse

- 对于一个多项式$A(x)$，如果存在$B(x)$且满足$deg\ B \leq deg\ A$并且$A(x)B(x) \equiv 1 (\mathrm{mod}\ x^n)$,那么就称$B(x)$是$A(x)$在$\mathrm{mod}\ x^n$意义下的逆元，记作$A^{-1}(x)$
- 假设在$\mathrm{mod}\ x^{\lceil \frac{n}{2} \rceil}$意义下$A(x)$的逆元是$B'(x)$,那么就有$A(x)B(x) \equiv 1 (\mathrm{mod}\ x^{\lceil \frac{n}{2} \rceil})$以及$A(x)B'(x) \equiv 1 (\mathrm{mod}\ x^{\lceil \frac{n}{2} \rceil})$
- $B(x) - B'(x) \equiv 0 (\mathrm{mod}\ x^{\lceil \frac{n}{2} \rceil})$
- $B(x) \equiv 2B'(x) - A(x)B'^2(x) (\mathrm{mod}\ x^n)$

Polynomial Inverse

- 对于一个多项式$A(x)$，如果存在$B(x)$且满足$deg\ B \le deg\ A$并且$A(x)B(x) \equiv 1(\bmod\ x^n)$,那么就称$B(x)$是$A(x)$在$\bmod\ x^n$意义下的逆元，记作$A^{-1}(x)$
- 假设在$\bmod\ x^{\lceil \frac{n}{2} \rceil}$意义下$A(x)$的逆元是$B'(x)$,那么就有$A(x)B(x) \equiv 1(\bmod\ x^{\lceil \frac{n}{2} \rceil})$以及$A(x)B'(x) \equiv 1(\bmod\ x^{\lceil \frac{n}{2} \rceil})$
- $B(x) - B'(x) \equiv 0(\bmod\ x^{\lceil \frac{n}{2} \rceil})$
- $B(x) \equiv 2B'(x) - A(x)B'^2(x)(\bmod\ x^n)$
- 使用FFT加速，时间复杂度为$O(n \log n)$

Newton's Method

Newton's Method

- 牛顿迭代法可以用来求解满足$G(F(x)) \equiv 0 (\bmod\ x^n)$的$F(x)$

Newton's Method

- 牛顿迭代法可以用来求解满足$G(F(x)) \equiv 0 \pmod{x^n}$的$F(x)$
- 假设我们已经求出了在$\bmod x^{\lceil \frac{n}{2} \rceil}$意义下满足$G(F_0(x)) \equiv 0 \pmod{x^{\lceil \frac{n}{2} \rceil}}$的$F_0(x)$

Newton's Method

- 牛顿迭代法可以用来求解满足$G(F(x)) \equiv 0 \pmod{x^n}$的$F(x)$
- 假设我们已经求出了在$\bmod\ x^{\lceil \frac{n}{2} \rceil}$意义下满足$G(F_0(x)) \equiv 0 \pmod{x^{\lceil \frac{n}{2} \rceil}}$的$F_0(x)$
- 在$\bmod\ x^n$意义下,将$G(F(x))$在$F_0(x)$处进行泰勒展开可以得到
- $G(F(x)) \equiv G(F_0(x)) + G'(F_0(x))(F(x) - F_0(x)) \pmod{x^n}$

Newton's Method

- 牛顿迭代法可以用来求解满足$G(F(x)) \equiv 0 (\bmod\ x^n)$的$F(x)$
- 假设我们已经求出了在$\bmod\ x^{\lceil \frac{n}{2} \rceil}$意义下满足$G(F_0(x)) \equiv 0 (\bmod\ x^{\lceil \frac{n}{2} \rceil})$的$F_0(x)$
- 在$\bmod\ x^n$意义下,将$G(F(x))$在$F_0(x)$处进行泰勒展开可以得到
- $G(F(x)) \equiv G(F_0(x)) + G'(F_0(x))(F(x) - F_0(x)) (\bmod\ x^n)$
- 所以有$F(x) \equiv F_0(x) - \frac{G(F_0(x))}{G'(F_0(x))} (\bmod\ x^n)$

Polynomial Square Root

Polynomial Square Root

- 对于一个多项式$A(x)$，要求求出$B(x)$满足$B^2(x) \equiv A(x) \pmod{x^n}$

Polynomial Square Root

- 对于一个多项式$A(x)$，要求求出$B(x)$满足$B^2(x) \equiv A(x) (\bmod\ x^n)$
- 代入之前推出的迭代方程可以得到$B(x) \equiv \frac{B_0^2(x)+A(x)}{2B_0(x)} (\bmod\ x^n)$

Polynomial Square Root

- 对于一个多项式$A(x)$，要求求出$B(x)$满足$B^2(x) \equiv A(x) \pmod{x^n}$
- 代入之前推出的迭代方程可以得到$B(x) \equiv \frac{B_0^2(x)+A(x)}{2B_0(x)} \pmod{x^n}$
- 利用多项式求逆元以及FFT计算，复杂度$O(n \log n)$

Polynomial Square Root

- 对于一个多项式$A(x)$，要求求出$B(x)$满足$B^2(x) \equiv A(x) \pmod{x^n}$
- 代入之前推出的迭代方程可以得到$B(x) \equiv \frac{B_0^2(x)+A(x)}{2B_0(x)} \pmod{x^n}$
- 利用多项式求逆元以及FFT计算，复杂度$O(n \log n)$
- 当系数在模意义下的时候，确定常数项时可能需要计算二次剩余

Codeforces #250 Div1 E The Child And Binary Tree

Codeforces #250 Div1 E The Child And Binary Tree

- 给定一个含有n个元素的正整数集合$S = \{c_1, c_2, ..., c_n\}$,我们称一个节点带权的有根二叉树是**好的**，当且仅当对于每个节点v,v的权值在S内，并且我们称这棵树的权值为所有节点的权值和

Codeforces #250 Div1 E The Child And Binary Tree

- 给定一个含有n个元素的正整数集合$S = \{c_1, c_2, ..., c_n\}$,我们称一个节点带权的有根二叉树是**好的**，当且仅当对于每个节点v,v的权值在S内，并且我们称这棵树的权值为所有节点的权值和
- 给定一个正整数m,对于所有$1 \leq i \leq m$,计算有多少个不同的**好的**二叉树，使得它的权值为s,结果对998244353取模。

Codeforces #250 Div1 E The Child And Binary Tree

- 给定一个含有n个元素的正整数集合$S = \{c_1, c_2, ..., c_n\}$,我们称一个节点带权的有根二叉树是**好的**，当且仅当对于每个节点v,v的权值在S内，并且我们称这棵树的权值为所有节点的权值和
- 给定一个正整数m,对于所有$1 \leq i \leq m$,计算有多少个不同的**好的**二叉树，使得它的权值为s,结果对998244353取模。
- $1 \leq n, m, c_i \leq 10^5$

Codeforces #250 Div1 E The Child And Binary Tree

Solution

Codeforces #250 Div1 E The Child And Binary Tree

Solution

- 用生成函数来考虑这个问题,对于一个节点来说,它的生成函数是$T(x) = \sum\limits_{i \in S} x^i$

Solution

- 用生成函数来考虑这个问题,对于一个节点来说,它的生成函数是$T(x) = \sum_{i \in S} x^i$
- 假设答案的生成函数是$F(x)$,根据二叉树的递归性质可以得到方程$F(x) = 1 + T(x)F^2(x)$

Codeforces #250 Div1 E The Child And Binary Tree

Solution

- 用生成函数来考虑这个问题,对于一个节点来说,它的生成函数是$T(x) = \sum_{i \in S} x^i$
- 假设答案的生成函数是$F(x)$,根据二叉树的递归性质可以得到方程$F(x) = 1 + T(x)F^2(x)$
- 于是解得$F(x) = \frac{1-\sqrt{1-4T(x)}}{2T(x)}$

Codeforces #250 Div1 E The Child And Binary Tree

Solution

- 用生成函数来考虑这个问题,对于一个节点来说,它的生成函数是$T(x)=\sum\limits_{i\in S}x^i$
- 假设答案的生成函数是$F(x)$,根据二叉树的递归性质可以得到方程$F(x)=1+T(x)F^2(x)$
- 于是解得$F(x)=\frac{1-\sqrt{1-4T(x)}}{2T(x)}$
- 利用多项式开方和多项式求逆即可在$O(n\log n)$的时间内求出答案

Polynomial Logarithm

Polynomial Logarithm

- 对于一个多项式$A(x) = 1 + \sum_{i \geq 1} a_i x^i$，我们可以用麦克劳林展开来定义多项式的对数:

Polynomial Logarithm

- 对于一个多项式$A(x) = 1 + \sum_{i \geq 1} a_i x^i$，我们可以用麦克劳林展开来定义多项式的对数:
- $ln(1 - A(x)) = -\sum_{i \geq 1} \frac{A^i(x)}{i}$

Polynomial Logarithm

- 对于一个多项式$A(x) = 1 + \sum_{i \geq 1} a_i x^i$，我们可以用麦克劳林展开来定义多项式的对数:
- $ln(1 - A(x)) = -\sum_{i \geq 1} \frac{A^i(x)}{i}$
- 直接计算似乎有些麻烦

Polynomial Logarithm

- 对于一个多项式$A(x) = 1 + \sum_{i \ge 1} a_i x^i$，我们可以用麦克劳林展开来定义多项式的对数:
- $ln(1 - A(x)) = -\sum_{i \ge 1} \frac{A^i(x)}{i}$
- 直接计算似乎有些麻烦
- $(ln(A(x)))' = \frac{A'(x)}{A(x)}$, $ln(A(x)) = \int \frac{A'(x)}{A(x)} \pmod{x^n}$

Polynomial Logarithm

- 对于一个多项式$A(x) = 1 + \sum_{i \geq 1} a_i x^i$，我们可以用麦克劳林展开来定义多项式的对数:
- $ln(1 - A(x)) = -\sum_{i \geq 1} \frac{A^i(x)}{i}$
- 直接计算似乎有些麻烦
- $(ln(A(x)))' = \frac{A'(x)}{A(x)}$, $ln(A(x)) = \int \frac{A'(x)}{A(x)} \pmod{x^n}$
- 多项式积分和求导时间复杂度$O(n)$，总的时间复杂度为$O(n \log n)$

Polynomial Exponentiation

Polynomial Exponentiation

- 对于一个多项式$A(x) = \sum_{i \geq 0} a_i x^i$，我们可以类似地定义多项式的指数:

Polynomial Exponentiation

- 对于一个多项式$A(x) = \sum_{i \geq 0} a_i x^i$，我们可以类似地定义多项式的指数:
- $e^{A(x)} = -\sum_{i \geq 0} \frac{A^i(x)}{i!}$

Polynomial Exponentiation

- 对于一个多项式$A(x)=\sum\limits_{i\geq 0} a_i x^i$，我们可以类似地定义多项式的指数:
- $e^{A(x)}=-\sum\limits_{i\geq 0}\frac{A^i(x)}{i!}$
- 这时不能够直接求导，需要运用Newton’s Method

Polynomial Exponentiation

- 对于一个多项式$A(x) = \sum_{i\geq 0} a_i x^i$，我们可以类似地定义多项式的指数:
- $e^{A(x)} = -\sum_{i\geq 0} \frac{A^i(x)}{i!}$
- 这时不能够直接求导，需要运用Newton’s Method
- $B(x) = e^{A(x)}, ln(B(x)) - A(x) = 0$

Polynomial Exponentiation

- 对于一个多项式$A(x) = \sum_{i\geq 0} a_i x^i$，我们可以类似地定义多项式的指数:
- $e^{A(x)} = -\sum_{i\geq 0} \frac{A^i(x)}{i!}$
- 这时不能够直接求导，需要运用Newton’s Method
- $B(x) = e^{A(x)}$, $ln(B(x)) - A(x) = 0$
- 根据公式计算出$B(x) \equiv B_0(x)(1 - ln(B_0(x)) + A(x))$

Polynomial Exponentiation

- 对于一个多项式$A(x) = \sum\limits_{i \geq 0} a_i x^i$，我们可以类似地定义多项式的指数:
- $e^{A(x)} = -\sum\limits_{i \geq 0} \frac{A^i(x)}{i!}$
- 这时不能够直接求导，需要运用Newton’s Method
- $B(x) = e^{A(x)}, ln(B(x)) - A(x) = 0$
- 根据公式计算出$B(x) \equiv B_0(x)(1 - ln(B_0(x)) + A(x))$
- 最后$B(x)$的常数项为1

Polynomial Exponentiation

- 对于一个多项式$A(x) = \sum_{i\geq 0} a_i x^i$，我们可以类似地定义多项式的指数:
- $e^{A(x)} = -\sum_{i\geq 0} \frac{A^i(x)}{i!}$
- 这时不能够直接求导，需要运用Newton’s Method
- $B(x) = e^{A(x)}, ln(B(x)) - A(x) = 0$
- 根据公式计算出$B(x) \equiv B_0(x)(1 - ln(B_0(x)) + A(x))$
- 最后$B(x)$的常数项为1
- 时间复杂度仍然是$O(n \log n)$

Polynomial Powers

Polynomial Powers

- 对于一个多项式$A(x)$，如何快速求出$A^k(x)$?

Polynomial Powers

- 对于一个多项式$A(x)$，如何快速求出$A^k(x)$?
- 分治FFT 时间复杂度$O(n \log n \log k)$

Polynomial Powers

- 对于一个多项式$A(x)$，如何快速求出$A^k(x)$?
- 分治FFT 时间复杂度$O(n\log n\log k)$
- 或者FFT+快速幂,时间复杂度也为$O(n\log n\log k)$

Polynomial Powers

- 对于一个多项式$A(x)$，如何快速求出$A^k(x)$?
- 分治FFT 时间复杂度$O(n\log n\log k)$
- 或者FFT+快速幂,时间复杂度也为$O(n\log n\log k)$
- $A^k(x)=e^{kln(A(x))}$,时间复杂度$O(n\log n)$

The End