zscoder's blog

[Tutorial] Generating Functions in Competitive Programming (Part 2)

By **zscoder**, history, 5 years ago,

Welcome to Part 2 of my tutorial on generating functions. The first part focused on introducing generating functions to those without any background in generating functions. In this post, I will demonstrate a few applications of generating functions in CP problems. Let us start with some relatively straightforward examples.

Note: Unless stated otherwise, all computations are done modulo a convenient prime (usually 998244353). Also, $[n]$ denotes the set $\{1, 2, \ldots, n\}$.

Blatant Applications in Counting Problems

Problem. AGC 005 Problem F You have a tree T with n vertices. For a subset S of vertices, let $f(S)$ denote the minimum number of vertices in a subtree of T which contains all vertices in S. For all $1 \leq k \leq n$, find the sum of $f(S)$ over all subsets S with $|S| = k$.

Constraints: $n \leq 2 \cdot 10^5$.
Solution

Convolutions of this type appear in countless FFT problems and usually the hard part is to reduce your sums into a form that can be seen as the convolution of two polynomials. However, generating functions are much more powerful than this, as we shall see the next examples.

Problem. The Child and Binary Tree You have a set of positive integers $C = \{c_1, c_2, \ldots, c_n\}$. A vertex-weighted rooted binary tree is called good if all the vertex weights are in C. The weight of such a tree is the sum of weights of all vertices. Count the number of distinct good vertex-weighted rooted binary trees with weight s for all $1 \leq s \leq m$. Note that the left child is distinguished from the right child in this problem (see the samples for more info).

Constraints: $n, m \leq 10^5$
Solution

Problem. Descents of Permutations (from Enumerative Combinatorics 1) Call a permutation $p_1, p_2, \ldots, p_n$ of $[n]$ k-good if $p_i > p_{i+1}$ iff $k \mid i$. Find the number of k-good permutations of length n.

Constraints: $n, k \leq 5 \cdot 10^5, n \geq 3, k \geq 2$
Solution

Expected Value of Stopping Time

I think this is also a beautiful application of generating functions. The recent problem Slime and Biscuits can be solved using the trick I will demonstrate here (there is a short editorial using this method here). Let's look at a different example.

Problem. Switches There are n switches, each of which can be on or off initially. Every second, there is a probability of $\frac{p_i}{S}$ that you will flip the state of the i-th switch. The game ends when all switches are off. What is the expected number of seconds the game will last?

Constraints: $n \leq 100, \sum p_i = S, p_i > 0, S \leq 50000$
Solution

In general, the trick of introducing A and B can be used to solve other problems that asks for the **first stopping time** of some system if you have multiple possible ending states and the time taken to reach from one ending state to another is equal, and the probability to reach an ending state in a fixed amount of moves is easy to compute.

Roots of Unity Filter

Next, we introduce a trick that is more rarely used in competitive programming but nevertheless interesting to learn. The motivation is the following classic problem.

Problem. Roots of Unity Filter Find the sum $\sum_{i \equiv r \pmod{m}} \binom{n}{i}$ modulo an arbitrary MOD.

Constraints: $1 \le n \le 10^{18}, 2 \le m \le 2000, 0 \le r \le n-1, 10^8 \le MOD \le 10^9$

Solution

Next, we look at a nice harder example.

Problem. Rhyme Compute the sum of $\frac{n!}{x_1!x_2!\ldots x_k!}$ over all tuples of positive integers $(x_1, x_2, \ldots, x_k)$ such that $d|x_i$ and $\sum x_i = n$, modulo 19491001 (a prime).

Constraints: $n \le 10^9, k \le 2000, d \in \{4, 6\}$.

Solution

Generating Functions that you can't compute "naively"

In some problems, the constraints might be too large to even compute the generating functions you found with FFT or algorithms involving polynomial operations. In this case, you usually need to analyze some special properties of the generating function you are dealing with (and thus it is helpful to recognize some common generating functions).

We start with a relatively easy example.

Problem. Perfect Permutations Find the number of permutations of length n with exactly k inversions, modulo $10^9 + 7$.

Constraints: $1 \le n \le 10^9$, $0 \le k \le 10^5$, $n \ge k$. There are 100 testcases per input file.

Solution

Next, we look at a problem that has a natural statement in generating functions, but it turns out that the computation in generating functions is quite tricky. The official editorial has a nice and simpler solution using a magical observation, but to demonstrate the power of generating functions I will show an alternative method (which seems more straightforward and generalizable).

Problem. Sum of Fibonacci Sequence Let $d_{n,k}$ be defined by the recurrence $d_{1,1} = d_{1,2} = 1$, $d_{1,k} = d_{1,k-1} + d_{1,k-2}$ for $k \ge 3$, and $d_{n,k} = \sum_{i=1}^{k} d_{n-1,i}$ for $n \ge 2$, $k \ge 1$.

Compute $d_{n,m}$ modulo 998244353.

Constraints: $1 \le n \le 200000$, $1 \le m \le 10^{18}$

Solution

To end this section, we conclude with a problem that heavily relies on linear recurrences. Actually, it might be a stretch to call this a generating function problem but I just want to demonstrate the trick of using generating functions to compute linear recurrences which is basically the same as the one shown here.

Problem. Sum Modulo You have a number x which is initially K. Every second, for $1 \le i \le n$, there is a probability p_i that you will replace x with $(x - i) \pmod{M}$. Find the expected number of moves before the counter goes to 0. p_i are given as $\frac{A_i}{\sum A_i}$ for some positive integers $A_1, A_2, \ldots, A_n$ and your output should be modulo 998244353 (and it is guaranteed that you don't have to worry about division by 0).

Constraints: $1 \le n \le \min(500, M-1)$, $2 \le M \le 10^{18}$, $1 \le A_i \le 100$

Solution

Lagrange Inversion Formula

Finally, inspired by this Div. 1 F problem, I looked up on some applications on Lagrange Inversion Formula and found a few examples on it. I would like to end this article by demonstrating a few applications of it.

Some examples here are from Enumerative Combinatorics Volume 2 and some are from **jcvb**'s Chinese paper on generating functions.

The idea of the Lagrange Inversion Formula is that sometimes we want to find the compositional inverse of a function but it is difficult to find. However, the coefficients of this inverse function might have a simpler formula, which we can obtain from Lagrange Inversion Formula.

There are many variants of stating the Lagrange Inversion Formula, so I will show what I think is the most helpful version of it (also given in this comment).

Theorem. Let $F(x), G(x)$ be formal power series which are compositional inverses (i.e. $F(G(x)) = x$). Suppose $F(0) = G(0) = 0$, $[x^1]F(x) \neq 0$, $[x^1]G(x) \neq 0$, then

$$[x^n]G(x) = \frac{1}{n}[x^{-1}]\frac{1}{F(x)^n}$$

Also, for any power (or Laurent) series $H(x)$, we have

$$[x^n]H(G(x)) = \frac{1}{n}[x^{-1}]H'(x)\frac{1}{F(x)^n}$$

Note: Laurent Series can be intuitively seen as the generalization of power series where the powers can go negative.

Intuitively, if you "know" how to compute $F(x)$, then you can also get the coefficients of the compositional inverse of $F(x)$. Let's go through a few examples.

Tree Enumeration

Problem. Count the number of labelled trees on n vertices (number of trees where vertices are labelled).

Solution

Number of 2-edge connected graphs

Problem. Find the number of labelled 2-edge connected graphs on n vertices. A graph is 2-edge connected graphs if it has no bridges, i.e. removing any edge does not disconnect the graph.

Constraints: $n \leq 3 \cdot 10^5$

Solution

Coefficient of fixed x^k in $f(x)^i$

This is a more of a trick than a specific problem. Let $f(x)$ be a power series with a compositional inverse ($[x^0]f(x) = 0$, $[x^1]f(x) \neq 0$). We can find the coefficient of x^k (assume $k \geq 1$) in $f(x)^i$ for all $1 \leq i \leq n$ in $O(n \log n)$ time (assume $k = O(n)$).

Let $ans(i)$ denote the answer for fixed i. Instead of looking at $ans(i)$ as a sequence, let's introduce a new variable u and consider the OGF

$$A(u) = ans(0) + ans(1)u + ans(2)u^2 + \ldots = \sum_{n \geq 0}[x^k]f(x)^n u^n = [x^k]\sum_{n \geq 0}(f(x)u)^n = [x^k]\frac{1}{1 - uf(x)}.$$

Since $f(x)$ has a compositional inverse (say $g(f(x)) = x$), by Lagrange Inversion formula (with $H(x) = \frac{1}{1-ux}$), we obtain

$$[x^k]\frac{1}{1-uf(x)} = \frac{1}{k}[x^{-1}]\left(\frac{1}{1-ux}\right)'\left(\frac{1}{g(x)^k}\right) = \frac{1}{k}[x^{k-1}]\left(\frac{1}{1-ux}\right)'\left(\frac{1}{\left(\frac{g(x)}{x}\right)^k}\right).$$

Note that by Quotient Rule, $\left(\frac{1}{1-ux}\right)' = \frac{u}{(1-ux)^2}$.

Our goal is to rewrite our sum in a clear manner so that we can "read off" the coefficients of u^i. We try to change the problem of finding the coefficients of u^i into a problem about purely finding the coefficients of x^j of some function.

The idea is to expand the series

$\frac{u}{(1-ux)^2} = u(1-ux)^{-2} = u\sum_{i\geq 0}\binom{i+1}{1}(ux)^i$ (recall how to expand $(1-x)^{-2}$)

$= u^{i+1}\sum_{i\geq 0}(n+1)x^i$, thus

$$[x^k]\frac{1}{1-uf(x)} = \frac{1}{k}[x^{k-1}]\sum_{i\geq 0}(i+1)x^iu^{i+1}\left(\frac{1}{\left(\frac{g(x)}{x}\right)^k}\right)$$

Let's look at $ans(i+1)$, the coefficient of u^{i+1}. We have

$$ans(i+1) = [u^{i+1}]\frac{1}{k}[x^{k-1}]\sum_{i\geq 0}(i+1)x^iu^{i+1}\left(\frac{1}{\left(\frac{g(x)}{x}\right)^k}\right) = \frac{i+1}{k}[x^{k-i-1}]\frac{1}{\left(\frac{g(x)}{x}\right)^k}.$$

Now, our problem reduces to computing the coefficients of one **fixed** function $P(x) = \frac{1}{\left(\frac{g(x)}{x}\right)^k}$, which we can compute the first n terms of using the usual polynomial operations! Thus, we can compute $ans(i)$ for all i in $O(n\log n)$ time!

If $f(x)$ does not have a compositional inverse, it is possible to "adjust" our function f (create a new function related to f) so that it has a compositional inverse. I leave this as an exercise.

Final Boss: Div 1 F — Slime and Sequences

As the grand finale of this 2-part article, I would like to discuss the recent very difficult Div. 1 F problem which was the inspiration of the entire blog in the first place.

Problem. Slime and Sequences A sequence of positive integers s is called good if for each $k > 1$ that is present in s, the first occurrence of $k-1$ (which must exist) must occur before the last occurrence of k. Count the number of times each integer $1 \leq i \leq n$ appears over all good sequences of length n.

Constraints: $1 \leq n \leq 100000$

Solution

If you have any questions or spotted any errors, please tell me in the comments. There are probably more cool applications of generating functions in CP that I am not aware of, so feel free to share them in the comments too. :)

mathforces, advanced math, math and programming, generating function