NOI.PH Training: Math 2

More Modulo, and More Counting

Cisco Ortega

Contents

1 Modulo Tricks

1.1 Why Modulo Still Matters Outside Number Theory

Usually contests will have tight bounds in order to really stress test the efficiency of your algorithms. However, sometimes that ends up with numbers that blow up to astronomical levels. Combinatorics gives many examples—theoretically, $n!$ can be computed in $\mathcal{O}(n)$, but we can't actually use $(10^7)!$ itself due to the added overhead of performing arithmetic on integers with millions of digits.

Usually, contests will then ask you to give the answer mod something. $10^9 + 7$ is the most common modulus for this purpose, since it is the smallest prime number greater than 1 billion. With a modulus, as long as we remember to take the mod after each intermediary operation, we can perform addition and multiplication between 64-bit integers without having to worry about integer overflow.

> **Exercise 1.1.** The value $10^9 + 7$ fits in a 32-bit integer (such as `int`), and thus any remainder after dividing by it will also always fit in a 32-bit integer. Why do we use 64-bit integers like `long long` then?

Sometimes the modulus itself can offer some insights into what strategies are needed to solve a problem. Sometimes the modulus itself, instead of just being a background mechanism to prevent integer overflow, can be a part of the problem itself!

A smaller modulus like $10^6 + 3$ might indicate that there is some sort of solution that is linear or linearithmic in terms of the modulus. One example of an algorithm whose runtime is dependent on the modulus is Lucas' Theorem, for instance (which we will discuss later in this module).

> **Exercise 1.2.** Given an input n, which can be up to 10^{18}, output $n!$ mod $10^6 + 3$. What is the runtime of your solution?

You may also occasionally see the prime modulus 998244353 used. Since it has a $2^2$3rd root of unity, we can do more advanced math tricks with this modulus like Fast Fourier Transforms, but all of this is out of scope of IOI material.

Anyway, no need to overthink it. It's not a hard rule, just a thing to take note of. Sometimes the problem setter chose a weird modulus for no reason, or perhaps it's just obfuscation. For most common purposes, the only thing that matters is that the modulus is prime.

On that note, **never assume the modulus is prime**. It shouldn't take you more than a minute to type out a naive $\mathcal{O}(\sqrt{M})$ trial division primality test in Python **just** in case. They may not look it, but neither $10^7 + 9$ nor $10^6 + 7$ are prime! If one of your algorithms relied on the modulus being prime, you'd be dead for assuming they were.

1.2 Things that Do Not Hold under Modulo

So, we said that in the world of modulo, we can add and multiply numbers without too much trouble. We can turn a blind eye to the fact that the modulo is there and pretend like we are just performing normal addition and multiplication on these. It's a basic exercise in most introductions to number theory that $(a + b) \bmod m = (a \bmod m + b \bmod m) \bmod m$.

However, some things are not as straightforward. One shortcoming is that you can't exactly

have a sense of ordering anymore. For instance, it is true that $10 < 100$, but when taken mod 7, we see that $10 \bmod 7 = 3$ and $100 \bmod 7 = 2$, and obviously $3 \not< 2$. If you have insanely large numbers and need some way to compare them, you'll have to find a way to do it on the 'original value', not the modulo value.

One common trick is to compare the logarithms of two numbers instead of the numbers themselves. The logarithm is a strictly increasing function, so $a < b$ implies that $\log a < \log b$ and vice versa. Just be careful with floating point numbers, as always.

Exercise 1.3. Absolute value doesn't necessarily hold either. For instance, $-7 \equiv 2 \mod 3$, but $|-7| \equiv 1 \mod 3$. So, if you need to evaluate $|a-b| \mod m$, it is probably going to fail if you did $|a \mod m - b \mod m| \mod m$.

That said, how do you evaluate something like $|3^{200} - 5^{130}| \bmod 10^9 + 7$?

Properties like GCD and LCM are lost as well. Anything involving factors and prime factorization is going to get lost, even normal division. Consider that $(21 \div 3) \bmod 5 = 7 \bmod 5 = 2$, but if we 'apply mod at each step' like we do for addition and multiplication, we get

$$((21 \bmod 5) \div (3 \bmod 5)) \bmod 5 = (1 \div 3) \bmod 5,$$

and that's definitely nonsense—we don't even have an integer anymore!

1.3 "Dividing" under Modulo

Example 1.4. Evaluate the sum $1 + x + x^2 + \cdots + x^{n-1} \bmod 10^9 + 7$, where x and n are positive integers less than 10^{18}.

First, we set `x %= MOD` so that we don't immediately overflow when we multiply two x together. This is an important step! Don't forget it.

Next, we recall the formula for a geometric series, and with it note that this sum is equal to $\dfrac{x^n - 1}{x - 1}$. We can easily calculate $x^n - 1$ in logarithmic time using fast exponentiation. But, the number will be far too large to store in a 64-bit integer, so we have to take the modulo at each step.

However, that means we no longer have the original integer so division isn't guaranteed to work anymore. We still haven't divided by the $x - 1$, though, and we need to figure out how to do so somehow.

What does it actually mean to 'divide' by a number, even with normal integers? Solving for x in $10 \div 2 = x$ is equivalent to finding the x that satisfies $2x = 10$. In other words, asking, "What is a divided by b?" is equivalent to asking, "What number, when multiplied to b, is equal to a?". In fact, dividing seems to be all about 'cancelling out' multiplication—we can reframe 'dividing by b' as 'multiplying by b^{-1}', where b^{-1} refers to the *multiplicative inverse* of b, the unique real number such that $bb^{-1} = 1$.

You'll notice the notation for multiplicative inverse matches the notation for exponents, and that is no coincidence; defining b raised to the -1 as the multiplicative inverse of b keeps the laws of exponents consistent.

In the real numbers, we know that b^{-1} is equal to the fraction $\dfrac{1}{b}$, since for instance, $\dfrac{1}{2}$ is the unique real number such that $2 \cdot \dfrac{1}{2} = 1$. Because it has this property, we can say that dividing by 2 is equivalent to multiplying by the multiplicative inverse $2^{-1} = \frac{1}{2}$.

Now, in the modulo world, we may have lost information about factorization, but the notion of undoing multiplication is still something we can do. In order to 'divide' by x, we simply multiply by x^{-1}.

We don't have fractions any more in modulo world, but that doesn't mean multiplicative inverses don't exist. Going back to the definition, we know that a is a multiplicative inverse of x mod m if and only if $ax \equiv 1 \bmod m$. For instance, a multiplicative inverse of 7 mod 10 is 3, since $7 \cdot 3 \equiv 21 \equiv 1 \bmod 10$. Therefore we can multiply by 3 to simulate 'dividing' by 7.

Example 1.5. Find $\frac{154}{7} \bmod 10$.

We know that the answer is $\frac{154}{7} = 22 \equiv 2 \bmod 10$.

However, what if we immediately perform 154 mod 10, leaving us with 4? Is 'dividing' by 7 still possible? Well, we can use the multiplicative inverse 3 instead. Dividing by 7 is equivalent to multiplying by 3, and true enough, we can check that $(154 \bmod 10) \cdot 7^{-1} \equiv 4 \cdot 3 = 12 \equiv 2 \bmod 10$. We still get the same result of 2.

Example 1.6. Evaluate $1 + 3 + \cdots + 3^{39} \bmod 10^9 + 7$.

You can verify that this sum is exactly equal to 6078832729528464400, whose remainder after dividing by $10^9 + 7$ is 976635598.

However, let's use the geometric series formula from earlier. We know that this value is equal to $\frac{3^{40} - 1}{2} \bmod 10^9 + 7$. Using fast exponentiation, we quickly get that $3^{40} = 953271190 \bmod 10^9 + 7$. You can verify that $2^{-1} = 5 \cdot 10^8 + 4$ (why?). Therefore,

$$\begin{aligned} \frac{3^{40} - 1}{2} &= (3^{40} - 1)2^{-1} \\ &\equiv (953271189) \cdot (5 \cdot 10^8 + 4) \\ &\equiv 976635598 \bmod 10^9 + 7 \end{aligned}$$

which is the same result as earlier.

Exercise 1.7. You'll notice I said 'a' multiplicative inverse of 7, mod 10. That is because 3 is not unique; there are infinitely many integers such that $7a \equiv 1 \bmod 10$. Try to give at least 3 more, and spot the pattern for what they are.

1.4 Finding a Multiplicative Inverse

First of all, x is not always guaranteed to have a multiplicative inverse mod m. We present the following condition.

Theorem 1.8. An integer x has a multiplicative inverse mod m if and only if x and m are coprime, i.e. $\gcd(x, m) = 1$.

Proof. Recall that we are searching for an integer a such that $ax \equiv 1 \bmod m$. From the definition of modulo, we know that there exists some integer k that satisfies the equation $ax = 1 + km$, which we can rearrange to $ax + (-k)m = 1$.

But, suppose that $\gcd(x, m) = d > 1$, then we could factor out the d to get $d\left(\frac{a}{d}x + \left(\frac{-k}{d}\right)m\right) = 1$. But, if $d > 1$ and everything here is an integer, then this is a contradiction, because there is no way to multiply two integers and get 1 unless they are both 1 or both -1.

Therefore, for a solution to exist, x and m must be coprime. □

This gives us a direct way to find a multiplicative inverse of x. We can use the Extended Euclidean algorithm to find any a and k that satisfies $ax - km = 1$. By definition, this a is a valid multiplicative inverse of x, mod m.

Exercise 1.9. Here's an example of a snippet of code which finds the multiplicative inverse of a mod b.

```
ll inv(ll a, ll b) {
    a %= b;
    if (a == 0) return b == 1 ? 0 : -1;
     ll x = inv(b, a);
     return x == -1 ? -1 : ((1 - b*x)/a + b) % b;
 }
```

Can you explain how this code works? When does it return -1, and what does that indicate?

What if we wish to perform $\frac{a}{b} \bmod m$ but $\gcd(x, m) > 1$? Provided that b is non-zero, we can use the following trick,

Theorem 1.10. Let $b \mid a$. Then,

$$\frac{a}{b} \bmod m \equiv \frac{a \bmod (bm)}{b}$$

Here, "mod" is the operation.

Example 1.11. Evaluate $\dfrac{1200}{40} \bmod 25$.

We compute this as equal to $\dfrac{1200 \bmod 1000}{40} = \dfrac{200}{40} = 5$.

This matches a direct computation, which is that $\dfrac{1200}{40} \bmod 25 = 30 \bmod 25 = 5$.

Proof. Intuitively, we can see that if a is divisible by b, then $a \bmod (bm)$ should also be divisible by b, preserving 'structure' in some way.

To prove this, we go back to the division algorithm. Since b divides a, we know $\dfrac{a}{b}$ is an integer. We perform division algorithm on $\dfrac{a}{b}$ and m, so there exist unique integers q_1 and r_1

such that

$$\frac{a}{b} = mq_1 + r_1$$

where $0 \leq r_1 \leq m$. Note that this means $\frac{a}{b} \bmod m = r_1$. Multiplying through by b,

$$a = bmq_1 + br_1$$

where $0 \leq r_1 < bm$. Now, if we perform division algorithm on a and bm, there exist unique positive integers q_2 and r_2 such that

$$a = (bm)q_2 + r_2$$

and $0 \leq r_2 < bm$. Note that $a \bmod (bm) = r_2$. Now, since division algorithm guarantees uniqueness of the remainder, we conclude $r_2 = br_1$. Now it is clear to see that

$$\begin{aligned}\frac{a \bmod (bm)}{b} &= \frac{r_2}{b} \\ &= \frac{br_1}{b} \\ &= r_1 \\ &= \frac{a}{b} \bmod m\end{aligned}$$

□

The downside of this technique is that this makes the modulus a bit bigger, but if b can be bounded to be a small enough constant, then this works.

1.5 Fermat's Little Theorem and Euler's Theorem

In practice though, it would be rare to see the Extended Euclidean outside of Number Theory. For Combinatorics problems and the like, there is a much simpler and much easier-to-implement method of getting the multiplicative inverse. I said that it was important that $10^9 + 7$ was prime, because it lets us use the following theorem.

Theorem 1.12 (Fermat's Little Theorem)**.** For some prime p and an integer x coprime to p, we have that $x^{p-1} = 1 \bmod p$.

Proof omitted, but it is interesting, if you want to look it up.

For our purposes, this gives us the following neat trick.

Theorem 1.13. If x^{-1} is a multiplicative inverse of $x \bmod p$, then $x^{p-2} \equiv x^{-1} \bmod p$.

We've reduced finding the multiplicative inverse to an exponentiation problem, which is something we already know how to do quickly!

Example 1.14. Evaluate $\frac{20}{4} \bmod 10^9 + 7$. You can test it yourself that $\frac{20}{4} \equiv 20 \cdot 4^{10^9+5} \equiv 5 \bmod 10^9 + 7$

For problems where the modulus is always the same prime (like it is in most non-Number

Theory problems), we can use the following code to get multiplicative inverses,

```cpp
using ll = long long;
const ll MOD = 1e9+7;
ll mod_pow(ll base, ll exp) {
    if (exp == 0) return 1;
    if (exp & 1) return (base * mod_pow(base, exp-1)) % MOD;
    else return mod_pow((base*base) % MOD, exp/2);
}
ll mod_inv(ll x) {
    return mod_pow(x%MOD, MOD-2);
}
```

All it needs is the fast exponentiation code, which is a really easy 3 lines to memorize, and once you have that, the inverse is itself a one-liner. Very convenient! This is the fastest and easiest way to 'divide' under modulo operations, given that the modulus is prime.

Occasionally you might get non-primes as a modulus, in which case Fermat's Little Theorem doesn't apply anymore.

Theorem 1.15 (Euler's Theorem)**.** For any coprime integers x and n, we have that $x^{\phi(n)} = 1 \bmod n$, where $\phi(n)$, Euler's Totient Function, is equal to the number of integers from 1 to n which are coprime to n.

Proof is, again, omitted. Note that when n is prime, $\phi(n) = n - 1$, in which case the problem reduces to Fermat's Little Theorem. Then, to find the multiplicative inverse of some x, we raise it to the power of $\phi(m) - 1$ instead of $p - 2$.

Example 1.16. Verify that $\phi(30) = 8$ since it is coprime to $1, 7, 11, 13, 17, 19, 23, 29$. We compute for $7^{-1} \bmod 30$ as

$$\begin{aligned} 7^{\phi(30)-1} &\equiv 7^7 \mod 30 \\ &\equiv 13 \mod 30 \end{aligned}$$

and we can confirm that $7 \cdot 13 = 91 \equiv 1 \bmod 30$.

In practice, we usually compute $\phi(n)$ using either a sieve or an $\mathcal{O}(\sqrt{n})$ factorization algorithm.

Theorem 1.17. Let $\phi(n)$, Euler's Totient Function, be equal to the number of integers from 1 to n which are coprime to n. If n has the prime factorization $n = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$, then

$$\begin{aligned} \phi(n) &= n\left(1 - \frac{1}{p_1}\right)\left(1 - \frac{1}{p_2}\right)\dots\left(1 - \frac{1}{p_k}\right) \\ &= n\frac{p_1 - 1}{p_1}\frac{p_2 - 1}{p_2}\dots\frac{p_k - 1}{p_k} \end{aligned}$$

We leave its proof as a guided exercise.

Exercise 1.18. Prove that $\phi(n)$ is multiplicative. That is, if a and b are coprime, then $\phi(ab) = \phi(a)\phi(b)$.

Exercise 1.19. Find a formula for $\phi(p^e)$, for a prime p and a positive integer e.

From here, how do we then show Theorem 1.17?

1.5.1 Binomial and Multinomial Coefficients

Let's talk about one application of our modulo inverse tricks by examining some combinatorics problems.

Example 1.20. Suppose we have a string of length n that consists of k 0s and $n-k$ 1s. How many distinct strings can be formed with these digits if all digits of the same kind are indistinguishable from one another? Give your answer modulo $10^9 + 7$.

We know that there are $n!$ ways to permute the entire string. But, we divide by the $k!$ ways to permute all the 0s within themselves in order to avoid double-counting, since we said that all the k 0s are exactly the same. Similarly, we also divide by $(n-k)!$. Thus, the total number of distinct strings is $\dfrac{n!}{k!(n-k)!}$.

We recognize this value as the **binomial coefficient**, $\binom{n}{k}$. We already know how to pregenerate all the binomial coefficients in $\mathcal{O}(n^2)$ using DP, which allows us to query in $\mathcal{O}(1)$. However, with this factorial form, if we precompute all the relevant factorials, we can still answer binomial coefficient queries in $\mathcal{O}(1)$. The difference is that precomputing the factorials only takes $\mathcal{O}(n)$ time, as opposed to $\mathcal{O}(n^2)$.

Example 1.21. Evaluate $\binom{10}{5}$.

$$\begin{aligned}\binom{10}{5} &= \frac{10!}{5!5!} \\ &= \frac{3628800}{120 \cdot 120} \\ &= 252\end{aligned}$$

The only hiccup with this solution is that factorials rapidly blow up in size, so we already have to apply modulo at each step in precomputing the factorials. Naturally, naive division doesn't work anymore.

Thanks to Fermat's Little Theorem, however, we can get the multiplicative inverse of $k!$ and $(n-k)!$ in logarithmic time, so evaluating $n! \cdot k!^{-1} \cdot (n-k)!^{-1} \bmod p$ only takes $\mathcal{O}(\log p)$ time per query. In fact, if we precompute all the factorials' inverses ahead of time and store them in an array, too, we can go back to answering each query in $\mathcal{O}(1)$.

Here is some sample code:

```
ll fac[N+1];  // factorials
ll fac_inv[N+1];  // factorial inverses
```

```
ll C(ll n, ll k) {
    if (not (0 <= k && k <= n)) return 0;
    ll num = fac[n];
    ll den = (fac_inv[k]*fac_inv[n-k]) % MOD;
    return (num*den) % MOD;
}
```

Tip: When checking whether a number x is in some interval $[\ell, r]$, I recommend the following code: `l <= x && x <= r`, because of its conceptual clarity, and similarity with the Python syntax `l <= x <= r`. (But note that the latter doesn't work as you expect in C++!) For its negation, i.e., if x is not in the interval, I recommend simply negating it: `not (l <= x && x <= r)`.[1]

Exercise 1.22. We already know how to precompute all of the factorials in $\mathcal{O}(n)$ time. Then, to precompute all the factorials' inverses, we could use Fermat's Little Theorem on each one, resulting in $\mathcal{O}(n \log p)$ time to compute the inverses.

However, it is possible to precompute all the inverses in $\mathcal{O}(n + \log p)$ time. How?

The factorial definition of binomial coefficients is also easily generalizable to what we call the **multinomial** coefficients.

Example 1.23. Suppose that we have a string of length n, where there are n_0 0s, n_1 1s, n_2 2s, and etc., where $n_0 + n_1 + n_2 + \cdots = n$. How many distinct strings can be formed with these characters if all characters of the same kind are indistinguishable from one another? Give your answer modulo $10^9 + 7$.

The answer can be achieved with similar logic as the first example. There are $n!$ ways to permute all n digits in the string. Then, we divide by the $(n_0)!$ ways to permute the 0s among themselves, then we divide by the $(n_1)!$ ways to permute the 1s among themselves, and so on. Thus, the multinomial coefficient written as $\binom{n}{n_0, n_1, \ldots}$, can be computed as

$$\binom{n}{n_0, n_1, \cdots} = \frac{n!}{n_0! n_1! \cdots}$$

Just like binomial coefficients, we can efficiently compute this modulo some prime by precomputing all the necessary factorials.

1.5.2 Power Towers

Let's look at another relatively common thing that we could be asked to compute.

Example 1.24. Evaluate $5^{10^{10^9}} \bmod 10^9 + 7$.

Basically, the thing to be evaluated is of the form $a^{b^c} \bmod p$. We know how to evaluate this with fast exponentiation, but even with logarithmic time, this is going to take $\mathcal{O}(\log(b^c))$ time to compute, and in our example, since $c = 10^9$, it is still too large for our algorithm to be efficient.

[1] These are not necessarily the *shortest* ways to implement these, but the goal isn't to make your program short anyway.

If the modulus is prime, then we can easily apply Fermat's Little Theorem. Note the following.

$$\begin{aligned} a^b &= a^{b-(p-1)+(p-1)} \\ &= a^{b-(p-1)}a^{p-1} \\ &\equiv a^{b-(p-1)} \mod p. \end{aligned}$$

Since $a^b \equiv a^{b-(p-1)} \bmod p$, repeated application tells us also that $a^b \equiv a^{b \bmod (p-1)} \bmod p$. Therefore, $a^{b^c} \equiv a^{b^c \bmod (p-1)} \bmod p$. We can solve this with nested fast exponentiation, for example looking like `mod_pow(a, mod_pow(b,c,p-1), p)`. This time, we actually have to indicate the modulus as an argument since the modulus of the fast exponentiation actually changes. If we simply plug in the values, we quickly get that $5^{10^{10^9}} \equiv 769669527 \bmod 10^9 + 7$.

If the modulus is non-prime, then we simply use Euler's Theorem and swap out the $p - 1$ for the Totient Function, giving us $a^b \equiv a^{b \bmod \phi(n)} \bmod n$, for coprime a and n. In fact, one cool thing about using Euler's Theorem is that since this trick needs us to do another fast exponentiation, except in the exponent instead, we can recursively *chain* them together to answer tall "power towers".

Exercise 1.25. Let $a = 10^9 + 1$. Evaluate $a^{a^{a^a}} \bmod 10^9 + 7$.

Exercise 1.26. As an aside, the equation

$$a^b \equiv a^{b \bmod \phi(n)} \bmod n$$

isn't quite true when a and n are not coprime.

- Can you explain why? Can you give a bunch of counterexamples?
- Can you adjust it a bit so that it works? I'm looking for a correct version that looks something like

  $$a^b \equiv a^{f(b)} \bmod n$$

  for some f (that depends only on n, not on a).

  Your f doesn't have to be the tightest possible; indeed, $\phi(n)$ isn't even always tight. It doesn't have to be a single elegant formula either; there's nothing wrong with piecewise definitions.

2 Special Integer Sequences

2.1 Combinatorics' Relationship with DP

Quite a lot of counting problems can be solved using dynamic programming, which makes sense since many counting problems can be broken down and defined in terms of its smaller subproblems. This relationship that a problem has with smaller versions of itself is usually called the *recurrence relation*, in the context of combinatorics.

Usually finding such a recurrence relation will be ad hoc done per counting problem. We will examine some examples of famous sequences in combinatorics to: (a) give concrete examples behind the process of finding a recurrence relation, to help build intuition, and (b) expose you to some famous sequences so that you might recognize when they in particular apply in a contest setting.

2.2 Binomial Coefficients Revisited

Let's begin with a familiar example. Given a class of n students, how many ways can I select a committee of k of them, where the order in which I select them doesn't matter. In broader terms, we are counting the number of k-element subsets from a set of n elements, but I'll be sticking with the class-of-students analogy throughout the module, just to make things a little more colorful. We already know that this is $\frac{n!}{k!(n-k)!}$, but let's pretend for a moment that we didn't know that formula.

Let's denote this as a function $C(n, k)$, since it accepts both n and k as inputs and produces some output depending on those two values. Since this function is particularly common, it has its own special notation that we will be using, which we write as $\binom{n}{k}$. Writing your counting problem as a function with the input as parameters might be helpful in discovering some useful insights, especially in developing recurrence relations.

Let's focus on the nth student—let's call him Kevin. Do you want Kevin in your committee? Or perhaps you think that you should give chance to others. In any case, those are the only two possibilities for Kevin—he's either in the committee or he isn't. When we partition the thing we want to count into mutually exclusively scenarios based on some discriminating factor, we say that we **condition** on that discriminating factor[2]

In this case, we condition on whether Kevin is in the committee or not. If he is, then okay, from the remaining $n-1$ students, we need to choose the remaining $k-1$ committee members. If he isn't, then from the remaining $n-1$ students, we still need to choose all k committee members. Since these two possibilities are mutually exclusive and cover all possible scenarios from our original problem, we get the recurrence relation

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}, \tag{1}$$

when $n, k \geq 1$.

This line of reasoning is oftentimes in Math called a **combinatoric proof**. First, we establish some sort of counting problem. Then, we count it in two different ways, yielding one expression that goes on the left-hand side, and one expression that goes on the right-hand side. Since we are fundamentally answering the same problem, these two expressions must therefore be equal!

[2]Terminology borrowed from Benjamin and Quinn's "Art of the Combinatoric Proof".

I encourage you to get used to combinatoric proofs since they are incredibly powerful; many identities which are difficult or tedious to prove algebraically are rather straightforward and even intuitive using a combinatoric proof. In competitive programming, it hones the kind of thinking that is used to find many DP recurrence relations.

Finally, let's establish some base cases for our recursion. We can say that $\binom{n}{0} = 1$, since there is only one valid to form a committee of zero students (just doing nothing)[3]. Furthermore, $\binom{n}{n} = 1$ since the only way to form a committee of n students from a class of n is to simply take all of them. Finally, we say that $\binom{n}{k} = 0$ when $k < 0$ or $n < k$ (which is equivalent to saying that it's *not* true that $0 \leq k \leq n$).

With programming, we can thus define the following function,

```
using ll = long long;
ll C(ll n, ll k) {
    if (not (0 <= k && k <= n))
        return 0;
    else if (k == 0 || k == n)
        return 1;
    else
        return C(n-1,k) + C(n-1,k-1);
}
```

to get us exactly what we want.

Exercise 2.1. Of course, when implementing this in a contest, instead of just directly translating the math formula, there are several things you might want to add. For instance, to deal with overflow, you'd want to put a modulo operator in that last line. The biggest thing, though, is that this is missing **memoization**.

With memoization, we can solve each query in $\mathcal{O}(1)$ time with $\mathcal{O}(n^2)$ time and space for pre-processing or memoization. Show that without memoization, `C(n,k)` takes $\mathcal{O}(n^k)$ time to evaluate.

Here's a bunch of other well-known identities involving the binomial coefficient. I've used each of these in various combinatorics problems in different contests, but they have never been the main question being asked. Usually, these identities are just a tool to simplify your algebraic expressions and to get them down to manageable time complexities.

Most of these you can prove algebraically, or with induction, but I will attempt to prove them all combinatorially, just to really hammer in this method of thinking. The proofs might seem like magic at first, but given enough exposure to enough examples, you will be able to pick up the general patterns easily enough. Plus, thinking with recurrence relations also helps train your DP skills, so it's a win-win.

$$\binom{n}{k} = \binom{n}{n-k}. \tag{2}$$

This is rather obvious just by looking at the factorial definition of $\binom{n}{k}$. Combinatorially, the proof is just as simple.

[3] It's dipping into the philosophical, but this kind of "only one way to do nothing correctly" reasoning is rather common in combinatorics

Proof. **Question:** How many ways can you choose a committee of k students from a class of n?

For the left-hand side, this is just $\binom{n}{k}$—by straightforward application of the binomial coefficient, we choose the k students to go on the committee.

For the right-hand side, this is also $\binom{n}{n-k}$, since we could equivalently choose the $n-k$ students who are **not** going to be on the committee. Since these two count the same problem, they must be equal. □

$$\sum_{k=0}^{n} \binom{n}{k} = 2^n. \tag{3}$$

We could just evaluate $(1+x)^n$ using the Binomial Theorem and plug in $x = 1$ to get this result, but let's try to prove it combinatorially.

Proof. **Question:** How many ways can you choose a (potentially empty) committee of students from a class of n?

For the left-hand side, condition on k, the number of students in the committee. If we decide to make a committee with k students, then the number of committees that satisfies that is $\binom{n}{k}$. Then, we just sum over all valid k.

For the right-hand side, use the Fundamental Counting Principle. Each of the n students has 2 choices—to either be in the committee, or to not be. This choice is repeated n times, once for each student, thus there are 2^n possible committees. □

With this identity, we can now evaluate $\sum_{k=0}^{n} \binom{n}{k}$ in $\mathcal{O}(\lg(n))$ instead of $\mathcal{O}(n)$.

$$\sum_{k=1}^{n} k\binom{n}{k} = n2^{n-1}. \tag{4}$$

This one can also be proven algebraically, but it's tedious. Proving it with combinatorics is much cleaner and far more insightful as to why it's true.

Proof. **Question:** How many ways can we choose a committee of students from a class of n, such that one of the committee-members is declared chairperson?

For the left-hand side, condition on k, the number of people in the committee. If there are k people on the committee, then there are $\binom{n}{k}$ ways to choose the committee members, then k choices for who to make chairperson. Then, sum over all valid values of k.

For the right-hand side, choose the chairperson from the class **first**; we have n choices for the chairperson. Then, each of the remaining $n-1$ students decides whether they want to be on the committee or not. Since this choice between 2 options is repeated $n-1$ times, there are 2^{n-1} ways to choose the non-chairperson committee members, and therefore there are $n2^{n-1}$ ways total. □

I hope you can see how this argument can be extended and played around with in order to prove more things about similarly-structured sums.

$$\sum_{t=0}^{n} \binom{t}{k} = \binom{n+1}{k+1}. \tag{5}$$

This is often called the Hockey Stick identity, due to the shape that the summands form on Pascal's Triangle. In fact, from this image, there is a natural proof by induction which I

encourage you to try and do as an exercise. However, in the spirit of things, let's examine the combinatoric proof.

Proof. **Question:** How many ways can I form a committee of $k+1$ students from a class of $n+1$?

For the left-hand side, let's assume we sorted the students alphabetically and gave them each a class number from 1, 2, 3, $\cdots$, n, $n+1$. Condition on the biggest class number in our committee. Suppose we had to make a committee of 5 students, and we already decided the greatest class number in the group was 7. Then, of the class numbers from 1 to 6, we need to choose 4 of them to form the rest of the committee. In general, there are $\binom{t}{k}$ possible committees such that the largest class number is $t+1$; we pick the remaining k committee members from the remaining t available class numbers. Then, we sum this expression over all valid t.

For the right hand side, a direct application of the binomial coefficient tells us that the answer is $\binom{n+1}{k+1}$. □

$$\sum_{k \text{ is even}} \binom{n}{k} = \sum_{k \text{ is odd}} \binom{n}{k} = 2^{n-1}. \tag{6}$$

Again, we could just evaluate $(1+x)^n$ using the Binomial Theorem and plug in -1 to get this result, but once again let's try to prove this combinatorially.

We'll use a different flavor of combinatoric proof. Let's establish a **bijection** between two sets to show that they must contain the same number of elements. A bijection is a relationship between the two sets such that every element in the first set corresponds to exactly one element in the other set, and vice versa.

Proof. Given a class of n students, let the first set E be the set of all committees whose size is even, and let the second set O be the set of all committees whose size is odd. Obviously, their sizes are $|E| = \sum_{k \text{ is even}} \binom{n}{k}$ and $|O| = \sum_{k \text{ is odd}} \binom{n}{k}$, respectively. The empty set belongs in E.

Let's turn our attention to the nth student—Kevin again. Then, let's consider an operation I call the 'toggle' operation. Pick any committee from E—Kevin either is on that committee or he isn't. If he is on the chosen committee, remove him from it. If he isn't on the chosen committee, add him to it. Note that either removing Kevin from the committee or adding him to it changes the parity of the number of students in that committee—thus, we have found a method to transform every element of E into a unique element of O. Similar logic establishes that we have a method of transforming every element of O into an element of E.

Since there is a bijection between these two sets, they must be equal in size. Furthermore, every integer is either odd or even, so

$$\sum_{k \text{ is even}} \binom{n}{k} + \sum_{k \text{ is odd}} \binom{n}{k} = \sum_{k} \binom{n}{k} = 2^n.$$

But, since the sum of the odd k and the sum of the even k are equal, each of them must be equal to half of 2^n, which is 2^{n-1}. □

Exercise 2.2. Let m, n, k be positive integers. Prove that

$$\binom{m+n}{k} = \sum_{t=0}^{k} \binom{m}{t}\binom{n}{k-t}.$$

Hint: Suppose we have a class with m boys and n girls, and we need to form a committee of k students.

Exercise 2.3. Let F_n be the Fibonacci numbers, defined by the recurrence relation $F_n = F_{n-1} + F_{n-2}$ for $n \geq 2$, with base cases $F_0 = 0$ and $F_1 = 1$.

Let f_n be the number of ways to completely tile a $1 \times n$ board using 1×1 squares and $1times2$ dominoes such that there are no overlaps.

Show that $f_n = F_{n+1}$.

Exercise 2.4. Let F_n be the Fibonacci numbers, defined by the recurrence relation $F_n = F_{n-1} + F_{n-2}$ for $n \geq 2$, with base cases $F_0 = 0$ and $F_1 = 1$.

Let f_n be the number of ways to completely tile a $1 \times n$ board using 1×1 squares and 1×2 dominoes such that there are no overlaps.

Show that $f_n = F_{n+1}$.

Exercise 2.5. Consider the sum

$$\sum_{n-k \leq k} \binom{k}{n-k}$$

Try it for small values of n, and conjecture a 'simple' identity for it. Prove this identity by combinatorial techniques.

2.3 Catalan Numbers

One special integer sequence given in terms of the binomial coefficients is the sequence of *Catalan Numbers*, where we denoted the nth Catalan number as C_n.

We are interested in Catalan numbers because there are many seemingly unrelated counting problems whose solution are given by this sequence. We could pick any of the following to define them. Some well-known examples are as follows.

Example 1. C_n is the number of ways to arrange n pairs of open and closing parentheses such that they are 'balanced'. For instance, for $n = 3$,

$$()()(), \quad ()(()), \quad (()()), \quad ((())), \quad (())()$$

So, $C_3 = 5$.

Example 2. C_n is the number of monotonic lattice paths along the edges of a grid with $n \times n$ square cells, which do not pass above the diagonal. A monotonic path is one which starts in the lower left corner, finishes in the upper right corner, and consists entirely of edges pointing rightwards or upwards. For example, for $n = 4$ we have $C(4) = 14$ paths as in Fig. 1.

Example 3. C_n are the number of sequences $\langle a_1, a_2, \ldots, a_{2n} \rangle$ of +1's and -1's with the property that $a_1 + a_2 + \cdots + a_{2n} = 0$ and all their partial sums $(a_1), (a_1 + a_2), \ldots, (a_1 + a_2 + \cdots + a_{2n})$ are non-negative. To illustrate this, let $\nearrow$ denote +1 and $\searrow$ denote -1. Essentially, we want to draw a sequence of $2n$ $\nearrow$'s and $\searrow$'s such that the start and end are at the same

level (the ground level) and that we are not allowed to go below the ground level at any time. In Fig. 2 we see such a solution for $n = 3$, wherein we have $C_3 = 5$ different ways of making these so-called "mountain ranges".

Exercise 2.6. Describe bijections between the first three examples.

Example 4. $C(n)$ is the number of ways to insert n pairs of parentheses in a word of $n+1$ letters so that they match correctly. This can also be thought of as the number of different ways to *associate* (or group together; think associative property) n applications of some binary operator.

For example, the 4-letter word abcd can be inserted with 3 pairs of parentheses in $C_3 = 5$ ways:

$$((ab)(cd)), \quad (((ab)c)d), \quad ((a(bc))d), \quad (a((bc)d)), \quad (a(b(cd)))$$

Example 5. C_n is the number of full binary trees with $n+1$ leaves. A rooted binary tree is *full* if each node has exactly 2 or 0 children.

Exercise 2.7. Draw a bijection between Example 4 and Example 5.

If you're familiar with how arithmetic equations are parsed (like with a shunting yard), then the bijection between the first three and the last two examples may be easier to spot. However, we can show that they are equivalent by showing that they have the same recurrence relation.

We will build a recurrence relation for C_n using Example 5. Start with the root node. Then, we need to distribute the $n+1$ leaves among the left and right subtrees. Suppose there are k leaves in the left subtree, and thus $n+1-k$ leaves in the right subtree. Since this is a full binary tree, there has to be at least one node in each subtree, so $1 \leq k \leq n$.

How can the two subtrees be structured? Well the left subtree is itself a rooted binary tree, so there are C_{k-1} ways for them to be arranged there. Similarly, there are C_{n-k} ways to arrange the leaves in the right subtree. Since these are independent, we can just multiply the two values together. Summing up over all k, we get

$$C_n = \sum_{k=1}^{n} C_{k-1} C_{n-k}$$

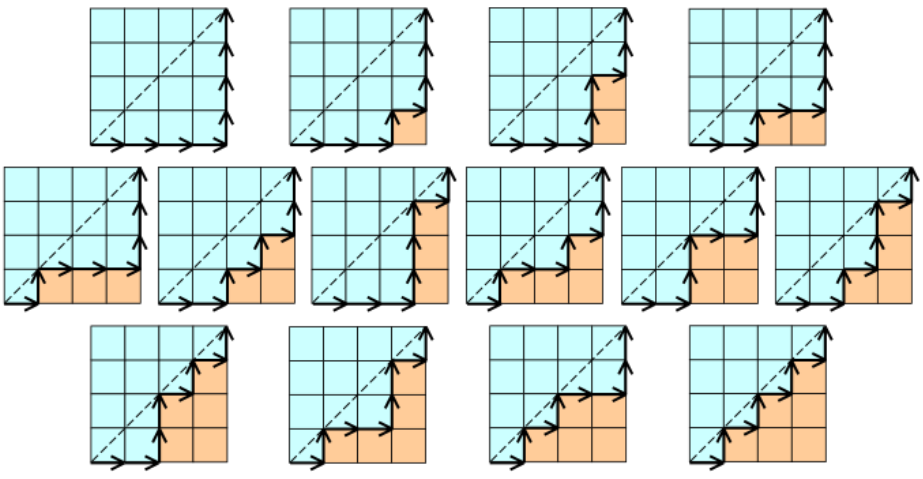


Figure 1: All the monotonic lattice paths in a 4×4 grid without passing above the diagonal.

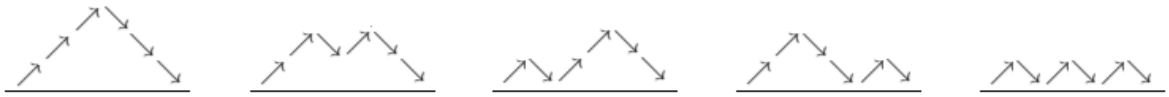


Figure 2: All the possible mountain ranges of width $2n$ where $n = 3$.

which is often stated more prettily as

$$C_{n+1} = \sum_{k=0}^{n} C_k C_{n-k} \tag{7}$$

with $C_0 = 1$. This also gives us a way to calculate the Catalan numbers in $\mathcal{O}(n^2)$.

Exercise 2.8. Show that any of the first three examples also follow this recurrence relation.

The first few Catalan numbers for $n = 0, 1, 2, 3, \ldots$ are listed as follows,

n	0	1	2	3	4	5	6	7	8	9	10
$C(n)$	1	1	2	5	14	42	132	429	1430	4862	16796

Although somewhat out of the scope of this material, the Catalan numbers also have a well-known closed form in terms of binomial coefficients, which are

$$C(n) = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n-1} = \frac{(2n)!}{(n+1)!\,n!} = \prod_{k=2}^{n} \frac{n+k}{k} \tag{8}$$

Note that this speeds up our calculation of the recurrence relation to $\mathcal{O}(n)$. A good combinatorial proof and many more examples can be found on the Catalan numbers' Wikipedia page, but I feel the best explanation for this closed form is told through generating functions, which are out of scope for this material.

Exercise 2.9. Through the closed form, show that the Catalan numbers also satisfy the recurrence relation,

$$\begin{aligned} C(0) &= 1 \\ C(n+1) &= \frac{2(2n+1)}{n+2} C(n) \quad \forall\, n \geq 0 \end{aligned} \tag{9}$$

2.4 Polygonal Numbers

First, we write down the third column of non-zero integers in Pascal's triangle as follows:

$$1,\ 3,\ 6,\ 10,\ 15,\ 21,\ 28, \ldots$$

We know that the nth integer in this sequence gives the binomial coefficient $\binom{n+1}{2}$, which counts the number of ways we can choose 2 elements from the set $\{1, 2, \ldots, n+1\}$. For example, the 3rd integer is 6 since $\binom{n+1}{2} = \binom{4}{2} = 6$. In combinatorics, these numbers are given a special name: *Triangular numbers*.

The name *triangular* is not due to the series being the 3rd column in Pascal's triangle. Rather, it is due to being the number of dots that form a triangle of side length n as in Fig. 3a.

Sequence	s	Formula	Examples
Triangular	3	$\frac{n(n+1)}{2}$	$1,\ 3,\ 6,\ 10,\ 15,\ 21,\ 28, \ldots$
Square	4	n^2	$1,\ 4,\ 9,\ 16,\ 25,\ 36,\ 49, \ldots$
Pentagonal	5	$\frac{n(3n-1)}{2}$	$1,\ 5,\ 12,\ 22,\ 35,\ 51,\ 70, \ldots$
Hexagonal	6	$n(2n-1)$	$1,\ 6,\ 15,\ 28,\ 45,\ 66,\ 91, \ldots$

Table 1: Sequences of polygonal numbers

Similarly, the square numbers, pentagonal numbers, hexagonal numbers, etc. are concepts extending from this definition. Collectively, they are called *Polygonal numbers*.

In Table 1, we list some examples of polygonal numbers and their formula given n. In general, if we denote $P(s, n)$ as the nth s-gonal number, then we have the closed form

$$P(s,n) = (s-2)\frac{n(n-1)}{2} + n = \frac{n^2(s-2) - n(s-4)}{2}. \tag{10}$$

We are interested in polygonal numbers because they also appear quite a lot in counting problems. For example, aside from the fact that the triangular numbers are the arithmetic sums

$$1 = \mathbf{1},\ 1+2 = \mathbf{3},\ 1+2+3 = \mathbf{6},\ 1+2+3+4 = \mathbf{10}, \ldots$$

they also arise in counting:

1. the number of ways to insert *one* pair of parentheses in a string of n letters, e.g. there are 6 ways to parenthesize NOI: (N)OI, (NO)I, (NOI), N(O)I, N(OI), NO(I).
2. the number of distinct handshakes in a room with $n+1$ people, e.g. Persons A, B, C can make 3 distinct handshakes: AB, AC, BC.

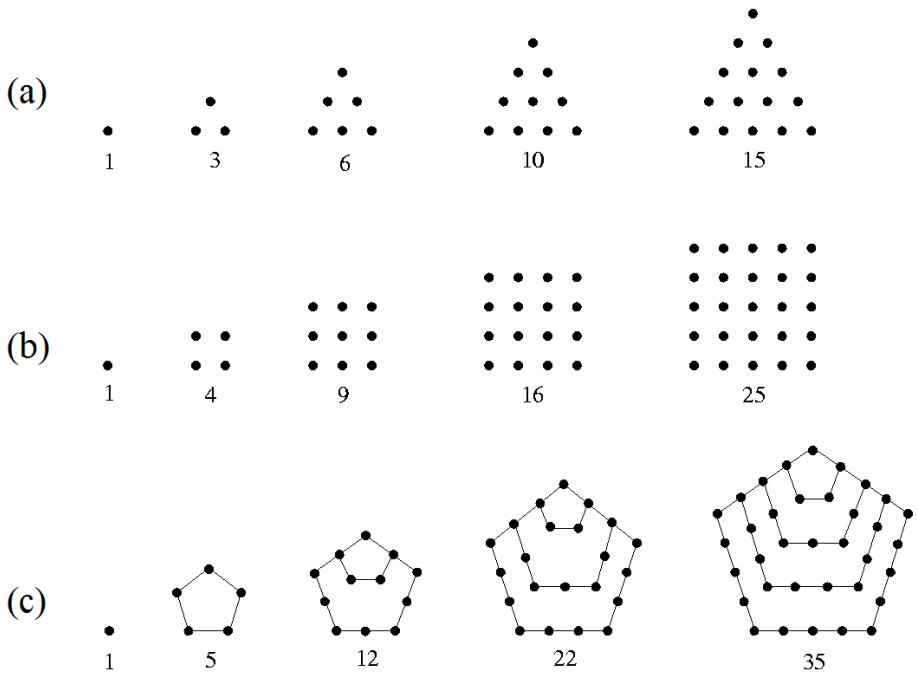


Figure 3: Polygonal numbers: (a) triangular, (b) square, and (c) pentagonal.

3. the number of non-negative integer solutions $\{x, y, z\}$ to $x + y + z = n - 1$ for $n \geq 1$, e.g. for $n = 3$, we have $\{0, 0, 2\}, \{0, 2, 0\}, \{2, 0, 0\}, \{0, 1, 1\}, \{1, 0, 1\}, \{1, 1, 0\}$.

We leave the proofs to these examples to the readers. All of these are equivalent to computing "$(n + 1)$ choose 2". Indeed, many other examples even for other polygonal numbers are abound during programming contests. So once you have spotted one of these sequences in your problem, you can use the closed-form Eq. (10) to compute them. Alternatively, we have recurrences:

$$T(1) = 1$$
$$T(n) = T(n-1) + n \tag{11}$$
$$P(s, n) = (s-2) \cdot T(n-1) + n \tag{12}$$

where $T(n)$ are the triangular numbers.

Finally, we note that some numbers belong to more than one polygonal set. For example, every triangular number $T(n)$ for which n is odd is also a hexagonal number. Interestingly, it was found that the number 1225 is 124-gonal, 60-gonal, 29-gonal, hexagonal, square, and triangular at the same time. Working backwards, given an s-gonal number denoted by x, we can find n such that $P(s, n) = x$ using (this can be easily derived from Eq. (10)):

$$n = \frac{\sqrt{8(s-2)x + (s-4)^2} + (s-4)}{2(s-2)}. \tag{13}$$

2.5 Stirling Numbers of the Second Kind

Say there is a group project, so you need to partition your class of n students into **exactly** k groups such that no group is empty. The order of the students in each group doesn't matter, merely which students are partnered with whom. Also, the groups can have different numbers of students, such that each group has at least one student. In broader terms, given a set of n objects, how many ways can we partition it into exactly k non-empty subsets?

For instance, suppose there is a class of only 4 students, whose class numbers are 1, 2, 3, and 4, and we need exactly 2 groups for this project. There are exactly 7 ways to partition the 4 students into 2 groups, which are

1. Alvin vs Barbie, CJ, and Daniella
2. Barbie vs Alvin, CJ, and Daniella
3. CJ vs Alvin, Barbie, and Daniella
4. Daniella vs Alvin, Barbie, and CJ
5. Alvin and Barbie vs CJ and Daniella
6. Alvin and CJ vs Barbie and Daniella
7. Alvin and Daniella vs Barbie and CJ

If they had class numbers $\{1, 2, 3, 4\}$, then the 7 ways to partition the 4 of them into 2 are, expressed with set notation,

$$\{1\} \cup \{2, 3, 4\}; \{2\} \cup \{1, 3, 4\}; \{3\} \cup \{1, 2, 3\}; \{4\} \cup \{1, 2, 3\}$$
$$\{1, 2\} \cup \{3, 4\}; \{1, 3\} \cup \{2, 4\}; \{1, 4\} \cup \{2, 3\}$$

Let's start off by once again defining a function $S(n, k)$ as "the number of ways to partition a class of n students into k non-empty groups". We will later see that this is also a rather famous combinatorics sequence, so it has its own special fancy notation $\left\{ {n \atop k} \right\}$.

Let's focus our attention on the nth student, Kevin, again. Let's ask a different question—is Kevin going to go solo for this project? Or maybe he trusts his classmates enough to delegate work to them, so he'll allow other people to join his group. Either way, we are again left with two mutually exclusive events—condition on whether Kevin is all by himself as a one-man group, or if he has other teammates in his group.

Let's think about this the other way—say we already have a class of $n-1$ students, then Kevin joins and we need to work him into the groups somehow. If the $n-1$ students are partitioned into $k-1$ groups, then if Kevin goes solo for this project, we get the requisite n students partitioned into k groups. If the $n-1$ students are already partitioned into k groups, then Kevin will have to decide which of the existing k groups he wants to join. To sum it up, we either have $n-1$ students partitioned into $k-1$ groups times the 1 way for Kevin to go solo, or the $n-1$ students partitioned into k groups times the k ways for Kevin to choose one of them to join. Thus, we get the recurrence relation

$$\left\{ {n \atop k} \right\} = \left\{ {n-1 \atop k-1} \right\} + k \left\{ {n-1 \atop k} \right\}, \tag{14}$$

where $n, k \geq 1$.

Finally, let's consider our base cases again. Again, rather philosophical, but let's say that $\left\{ {0 \atop 0} \right\} = 1$ because there is only one valid way to partition an empty class into 0 groups, and that $\left\{ {n \atop 0} \right\} = 0$ for $n > 0$ because if the class has at least one student, then there has to be at least one non-empty group, so the way of partitioning them into 0 groups must be 0. If you're not comfortable with these abstract arguments, perhaps you could at least accept the pragmatic argument that these are the starting conditions that make the recurrence relation properly count what we want it to count.

The number of ways to partition a set of n objects into k non-empty subsets is called the **Stirling Numbers of the Second Kind**. With DP, we can use their recurrence relation to pregenerate all numbers in $\mathcal{O}(n^2)$ and answer queries in $\mathcal{O}(1)$.

There exists another identity for the Stirling Numbers of the Second Kind, which is that

$$k! \left\{ {n \atop k} \right\} = \sum_{j=1}^{k} (-1)^{k-j} \binom{k}{j} j^n, \tag{15}$$

which allows you to compute a single $\left\{ {n \atop k} \right\}$ in $\mathcal{O}(k)$ time per query. This expression looks daunting, so instead of memorizing it, I re-derive it every time I need it to make sure I get all the details right.

Exercise 2.10. Combinatorially prove (14).

Consider the following problem: How many ways can you place n different balls, labeled 1 to n, into k different baskets, labeled 1 to k, such that none of the baskets are empty. Note that the order in which the balls are placed in the baskets doesn't matter; ultimately, we only care about which balls go in which baskets.

First, answer this question using Stirling Numbers of the Second Kind to get the left-hand side. Then, answer this question using the Inclusion-Exclusion Principle to get the right-hand side.

2.5.1 Bell Numbers

Furthermore, suppose we ditch that k requirement. Given a class of n students, how many ways can we partition them into **any** number of groups? We shall denote this sequence, often called the Bell Numbers, as B_n.

They have a rather obvious relationship with the Stirling Numbers of the Second Kind, which is that

$$B_n = \sum_{k=0}^{n} \left\{ {n \atop k} \right\}, \tag{16}$$

which follows from the fact that we are simply summing over all possible k, the number of non-empty groups the class was partitioned into. The Bell Numbers themselves have their own recurrence relation, however, which is that

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k, \tag{17}$$

whose combinatoric proof has also been left as a guided exercise. Regardless, both formulas allow you to compute for B_n in $\mathcal{O}(n^2)$ time.

Exercise 2.11. Combinatorially prove Equation 17.

Hint: You may look to the proof of the Hockey Stick identity for inspiration.

2.6 Stirling Numbers of the First Kind

Given a set of n objects, $\{1, 2, \ldots, n\}$, how many ways can we partition them into k *cycles* (instead of subsets)?

To explain further, we define a **cycle** as a cyclic arrangement of objects (such as in a necklace). For example, the cycle $[1, 2, 3, 4]$ is identical to $[2, 3, 4, 1]$, $[3, 4, 1, 2]$, and $[4, 1, 2, 3]$. However, it is not the same as $[4, 3, 2, 1]$ or $[1, 2, 4, 3]$. We provide an illustration in Fig. 4. By convention, a singleton is also considered a cycle of length 1.

As an example, consider the set $\{1, 2, 3, 4\}$. We can partition this into 2 cycles in 11 different ways:

$$\begin{array}{cccc} [1,2,3],[4] & [1,2,4],[3] & [1,3,4],[2] & [2,3,4],[1] \\ [1,3,2],[4] & [1,4,2],[3] & [1,4,3],[2] & [2,4,3],[1] \\ \multicolumn{4}{c}{[1,2],[3,4] \quad [1,3],[2,4] \quad [1,4],[2,3]} \end{array}$$

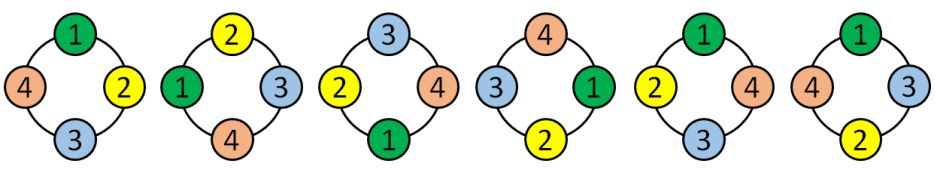


Figure 4: Examples of cycles: (a) to (d) are identical to each other, but (e) and (f) are each different from the rest.

n	$\begin{bmatrix} n \\ 0 \end{bmatrix}$	$\begin{bmatrix} n \\ 1 \end{bmatrix}$	$\begin{bmatrix} n \\ 2 \end{bmatrix}$	$\begin{bmatrix} n \\ 3 \end{bmatrix}$	$\begin{bmatrix} n \\ 4 \end{bmatrix}$	$\begin{bmatrix} n \\ 5 \end{bmatrix}$	$\begin{bmatrix} n \\ 6 \end{bmatrix}$	$\begin{bmatrix} n \\ 7 \end{bmatrix}$	$\begin{bmatrix} n \\ 8 \end{bmatrix}$	$\begin{bmatrix} n \\ 9 \end{bmatrix}$
0	1									
1	0	1								
2	0	1	1							
3	0	2	3	1						
4	0	6	11	6	1					
5	0	24	50	35	10	1				
6	0	120	274	225	85	15	1			
7	0	720	1764	1624	735	175	21	1		
8	0	5040	13068	13132	6769	1960	322	28	1	
9	0	40320	109584	118124	67284	22449	4536	546	36	1

Figure 5: Stirling's triangle for cycles.

The answer to our question is given by the *Stirling numbers of the first kind*, denoted by $\begin{bmatrix} n \\ k \end{bmatrix}$. From our example, it follows that $\begin{bmatrix} 4 \\ 2 \end{bmatrix} = 11$.

Just as before, we can write a recurrence relation for $\begin{bmatrix} n \\ k \end{bmatrix}$:

$$\begin{bmatrix} n \\ k \end{bmatrix} = (n-1)\begin{bmatrix} n-1 \\ k \end{bmatrix} + \begin{bmatrix} n-1 \\ k-1 \end{bmatrix} \tag{18}$$

We leave to you the derivation of this relation as an exercise.

Similarly, Stirling's triangle for cycles is given in Fig. 5. Some observations are worth noting. First, it is obvious that $\begin{bmatrix} n \\ k \end{bmatrix} \geq \begin{Bmatrix} n \\ k \end{Bmatrix}$ since every partition into non-empty subsets leads to at least one arrangement of cycles. Second, when cycles are necessarily singletons or doubletons, the cycles are equivalent to subsets such that

$$\begin{bmatrix} n \\ n \end{bmatrix} = \begin{Bmatrix} n \\ n \end{Bmatrix} = 1,$$
$$\begin{bmatrix} n \\ n-1 \end{bmatrix} = \begin{Bmatrix} n \\ n-1 \end{Bmatrix} = \binom{n}{2}.$$

Lastly, notice that by enumerating all possible cycle partitions for all k, we are essentially listing down the permutations of the set $\{1, 2, \ldots, n\}$. Thus, the sum of each row in Stirling's triangle of cycles is the total number of permutations:

$$\sum_{k=0}^{n} \begin{bmatrix} n \\ k \end{bmatrix} = n!. \tag{19}$$

To give you an example, let's count the number of permutations of $\{1, 2, 3\}$ by enumerating $\begin{bmatrix} 3 \\ k \end{bmatrix}$ for $k = 1, 2, 3$ as shown in Table 2. In total, there are $3! = 6$ permutations.

2.7 Eulerian Numbers

Given a set of n objects, $\{1, 2, \ldots, n\}$, how many permutations of this set have exactly k *ascents*?

$k=1$	$k=2$	$k=3$
$[1,2,3]$	$[1,2],[3]$	$[1],[2],[3]$
$[1,3,2]$	$[1,3],[2]$	
	$[1],[2,3]$	

Table 2: Listing the permutations of $\{1,2,3\}$ using $\begin{bmatrix}3\\k\end{bmatrix}$ for $k=1,2,3$

n	$\left\langle {n \atop 0} \right\rangle$	$\left\langle {n \atop 1} \right\rangle$	$\left\langle {n \atop 2} \right\rangle$	$\left\langle {n \atop 3} \right\rangle$	$\left\langle {n \atop 4} \right\rangle$	$\left\langle {n \atop 5} \right\rangle$	$\left\langle {n \atop 6} \right\rangle$	$\left\langle {n \atop 7} \right\rangle$	$\left\langle {n \atop 8} \right\rangle$	$\left\langle {n \atop 9} \right\rangle$
0	1									
1	1	0								
2	1	1	0							
3	1	4	1	0						
4	1	11	11	1	0					
5	1	26	66	26	1	0				
6	1	57	302	302	57	1	0			
7	1	120	1191	2416	1191	120	1	0		
8	1	247	4293	15619	15619	4293	247	1	0	
9	1	502	14608	88234	156190	88234	14608	502	1	0

Figure 6: Euler's triangle for $n=0,1,\ldots,9$ and $k=0,1,\ldots,9$.

For example, consider $\{1,2,3,4\}$. Of the $4!=24$ permutations of this set, 11 of them have exactly 2 ascents:

$$1324 \quad 1423 \quad 2314 \quad 2413 \quad 3412$$
$$1243 \quad 1342 \quad 2341 \quad 2134 \quad 3124 \quad 4123$$

If we denote a permutation as $\pi_1\pi_2\ldots\pi_n$, then the first 5 in the above list have the form $\pi_1<\pi_2>\pi_3<\pi_4$, the next 3 have the form $\pi_1<\pi_2<\pi_3>\pi_4$, and the last 3 have the form $\pi_1>\pi_2<\pi_3<\pi_4$. In all these forms, we always see exactly 2 "less than" signs, hence, they all have exactly 2 ascents.

The answers to this counting problem are given the name *Eulerian Numbers*, denoted by $\left\langle {n \atop k} \right\rangle$. In our example, we can write $\left\langle {4 \atop 2} \right\rangle = 11$.

As with the previous integer sequences, the following recurrence relation is given for Eulerian numbers:

$$\left\langle {n \atop k} \right\rangle = (k+1)\left\langle {n-1 \atop k} \right\rangle + (n-k)\left\langle {n-1 \atop k-1} \right\rangle. \tag{20}$$

The base cases are: $\left\langle {0 \atop 0} \right\rangle = 1$ since we have 1 way to permute an empty set; as well $\left\langle {n \atop 0} \right\rangle = 1$ since we have only 1 way to permute an n-element set with no ascents (decrease the elements). From here, try to derive the recurrence in Eq. (20). Due to this, we can also build Euler's triangle as shown in Fig. 6. Obviously, $\sum_{k=0}^{n}\left\langle {n \atop k} \right\rangle = n!$ since we have a list of permutations considering all possible number of ascents. For programming contests, it is useful to memorize the entries in rows 3-5 of Euler's triangle, in case you run into them for small test cases.

Eulerian numbers are primarily useful because of their connection to ordinary powers and consecutive binomial coefficients (called "Worpitzky's identity"):

$$x^n = \sum_k \left\langle {n \atop k} \right\rangle \binom{x+k}{n} \quad \forall\, n \geq 0 \tag{21}$$

For example, we can write the following powers as:

$$\begin{aligned}
x^2 &= \binom{x}{2} + \binom{x+1}{2}, \\
x^3 &= \binom{x}{3} + 4\binom{x+1}{3} + \binom{x+2}{3}, \\
x^4 &= \binom{x}{4} + 11\binom{x+1}{4} + 11\binom{x+2}{4} + \binom{x+3}{4}
\end{aligned}$$

and so on. Try to prove Eq. (21) by mathematical induction.

3 Partition Numbers

In this section, we explore some counting problems in number theory, specifically in counting the number of ways to partition integers.

3.1 Partition-p function

Here, we ask: How many ways can we write a non-negative integer n as a sum of positive integers?

As an example, for $n = 4$, we can have 5 distinct partitions:

$$\begin{aligned} \mathbf{4} &= 4 \\ &= 3 + 1 \\ &= 2 + 2 \\ &= 2 + 1 + 1 \\ &= 1 + 1 + 1 + 1 \end{aligned}$$

Note that since addition is commutative, the order is irrelevant, e.g. $2 + 1 + 1$ is identical to $1 + 2 + 1$ and $1 + 1 + 2$. In number theory, the answer to our question is given a notation $p(n)$, called the *Partition-p function*[4]. In our example, it follows that $p(4) = 5$. For the first few values of n, we have a sequence of *partition numbers* listed as follows:

n	0	1	2	3	4	5	6	7	8	9	10	11
$p(n)$	1	1	2	3	5	7	11	15	22	30	42	56

By convention, $p(0) = 1$ and $p(n) = 0$ for $n < 0$.

Before we discuss how to compute $p(n)$, let us first restrict ourselves with the question: How many ways can we write a non-negative integer n as a sum of exactly k positive integers? (Q1)

It turns out that this question is equivalent to asking: How many ways can we write a non-negative integer n as a sum of positive integers, the largest of which is exactly k? (Q2)

Let's denote the answer to these questions as $p(n, k)$. As an example, $p(8, 3) = 5$ since we have 5 ways to partition 8 into 3 terms (list A), and we also have 5 ways to partition 8 with 3 as the largest term (list B).

List A	List B
$6 + 1 + 1$	$3 + 3 + 2$
$5 + 2 + 1$	$3 + 3 + 1 + 1$
$4 + 3 + 1$	$3 + 2 + 2 + 1$
$4 + 2 + 2$	$3 + 2 + 1 + 1 + 1$
$3 + 3 + 2$	$3 + 1 + 1 + 1 + 1 + 1$

Focusing on Q1, the base cases are as follows: $p(0, 0) = 1$ since we have 1 way to partition $n = 0$, namely $0 = 0$, and this actually makes zero positive terms; also $p(n, 0) = 0$ for $n > 0$ since we can't have zero positive terms in partitioning a positive integer. Now to derive a recurrence, we categorize all the $p(n, k)$ partitions of n into two: those containing at least one $+1$ term, and those that do not. In the former, we can count the number of partitions $p(n - 1, k - 1)$ from which we can add a $+1$ term to all of them. In the latter, we can count the number of partitions $p(n - k, k)$, from which we can add 1 to each of the k terms. Thus, we have the recurrence relation:

$$p(n, k) = p(n - k, k) + p(n - 1, k - 1) \tag{22}$$

[4]Don't be confused between P and p, which stands for the polygonal and the partition numbers, respectively.

n\k	0	1	2	3	4	5	6	7	8	9
0	1									
1	0	1								
2	0	1	1							
3	0	1	1	1						
4	0	1	2	1	1					
5	0	1	2	2	1	1				
6	0	1	3	3	2	1	1			
7	0	1	3	4	3	2	1	1		
8	0	1	4	5	5	3	2	1	1	
9	0	1	4	7	6	5	3	2	1	1

Figure 7: Triangle of partition-p numbers for $n = 0, 1, \ldots, 9$ and $k = 0, 1, \ldots, 9$.

Naturally, we can make a table of partition numbers as shown in Fig. 7. From here, we can recover the original partition function $p(n)$ by taking the sum of the entries in the nth row of the partition number triangle:

$$p(n) = \sum_{k=1}^{n} p(n, k) \tag{23}$$

In programming contests, we recommend that you compute $p(n, k)$ or $p(n)$ by using the recurrence relation in Eq. (22). Try to derive this relation from the perspective of Q2 in our earlier discussion.

Another useful recurrence relation is given as

$$p(n) = \frac{1}{n} \sum_{k=0}^{n-1} \sigma(n-k)p(k) \tag{24}$$

where $\sigma(x)$ is the sum of the positive divisors of x.

The kth columns of Fig. 7 also have some closed forms for small k:

$$p(n, 2) = \lfloor n/2 \rfloor \tag{25}$$

$$p(n, 3) = [n^2/12] \tag{26}$$

where $\lfloor x \rfloor$ is the floor function and $[x]$ is the nearest integer function.

3.2 Partition-q function

Now, we ask: How many ways can we write a non-negative integer n as a sum of *distinct* positive integers?

As an example, for $n = 10$, we have 10 different ways:

$$\begin{aligned}\mathbf{10} &= 10\\ &= 9+1\\ &= 8+2\\ &= 7+3\\ &= 7+2+1\\ &= 6+4\\ &= 6+3+1\\ &= 5+4+1\\ &= 5+3+2\\ &= 4+3+2+1\end{aligned}$$

Again, note that the order is irrelevant. The answers to this question are given by $q(n)$, known as the *Partition-q function*. The first few values of $q(n)$ are listed as follows:

n	0	1	2	3	4	5	6	7	8	9	10	11
$q(n)$	1	1	1	2	2	3	4	5	6	8	10	12

As with the Partition-p, we will first concern ourselves with the question: How many ways can we partition a non-negative integer n into exactly k distinct positive integers?

We can denote $q(n, k)$ as the answer to this question, and from the enumeration above, we can take the example $q(10, 3) = 4$.

In order to find a recurrence, the base cases of $q(n, k)$ are noted as follows: $q(0, 0) = 1$ since $0 = 0$ is a valid partition of zero distinct positive integers; and, $q(n, 0) = 0$ for $n > 0$ since, similar to $p(n, 0)$, we can't have zero distinct terms in partitioning n. From here, we will leave it to you to derive the following recurrence:

$$q(n, k) = q(n-k, k) + q(n-k, k-1) \tag{27}$$

This gives a different triangle of partition numbers, shown in Fig. 8, which is not exactly a triangle. The nth row in this table contains only $1 + \lfloor \frac{1}{2}(\sqrt{1+8n} - 1) \rfloor$ entries. In other words, the number of entries increments by 1 at each triangular-number-valued row[5] (see Section 2.4). Nonetheless, we can compute $q(n)$ from this table as:

$$q(n) = \sum_{k=1}^{n} q(n, k) \tag{28}$$

Some useful relationships between the Partition-p and Partition-q functions are given:

$$q(n, k) = p\left(n - \binom{k}{2}, k\right) \tag{29}$$

$$p(n) = \sum_{k=0}^{\lfloor n/2 \rfloor} q(n-2k)p(k) \tag{30}$$

Due to Euler, it is known that $q(n)$ also answers the question: How many ways can we write a non-negative integer n as a sum of only *odd* integers (allowing repetition)?

In the contest, the solution to other restrictions to integer partitioning may be required of you. Using the techniques above, we hope that you can also establish recurrences and base cases specific to the problem.

[5] Try to think why this is so.

n\k	0	1	2	3	4	5	6	7	8	9
0	1									
1	0	1								
2	0	1								
3	0	1	1							
4	0	1	1							
5	0	1	2							
6	0	1	2	1						
7	0	1	3	1						
8	0	1	3	2						
9	0	1	4	3						
10	0	1	4	4	1					
11	0	1	5	5	1					
12	0	1	5	7	2					
13	0	1	6	8	3					
14	0	1	6	10	5					
15	0	1	7	12	6	1				

Figure 8: Triangle of partition-q numbers for $n = 0, 1, \ldots, 15$.

4 Fibonacci Numbers

Because this sequence is very well-known, it deserves a separate section.

Let's go ahead and establish the recurrence for Fibonacci Numbers without a background problem. Denoting $F(n)$ as the nth Fibonacci number, we have:

$$\begin{aligned} F(0) &= 0 \\ F(1) &= 1 \\ F(n) &= F(n-1) + F(n-2) \end{aligned} \tag{31}$$

The first few Fibonacci numbers are then:

n	0	1	2	3	4	5	6	7	8	9	10
$F(n)$	0	1	1	2	3	5	8	13	21	34	55

Interestingly, we have a closed-form for $F(n)$, known as Binet's formula, given by:

$$F(n) = \frac{\varphi^n - (1-\varphi)^n}{\sqrt{5}} \tag{32}$$

where $\varphi = \frac{1+\sqrt{5}}{2} = 1.618\ldots$ is the golden ratio. Even though an irrational number $\sqrt{5}$ appears heavily Eq. (32), these things always cancel out, regardless of the value of n, so that $F(n)$ is always integer. A quicker way to compute $F(n)$ via a closed form is given by

$$F(n) = \left\lfloor \frac{\varphi^n}{\sqrt{5}} + \frac{1}{2} \right\rfloor \quad n \geq 0 \tag{33}$$

where $\lfloor x \rfloor$ is the floor function, which has one less exponentiation. However, we still recommend that you build $F(n)$ using the recurrence in Eq. (31) as a first try. We can also work backwards and try to find n given a Fibonacci number F using:

$$n(F) = \left\lfloor \log_\varphi \left(F \cdot \sqrt{5} + \frac{1}{2} \right) \right\rfloor \tag{34}$$

where you can use change-of-base to compute $\log_\varphi(x) = \ln(x)/\ln(\varphi)$.

In the following subsections, we list other interesting properties of the Fibonacci numbers that you may find useful in programming contests.

4.1 Fibonacci by Matrices

Recall: An $r \times c$ matrix is any rectangular array of numbers of r rows and c columns. For example, a 2×2 square matrix of numbers a, b, c, d can be written as $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$.

A column vector, on the other hand, is a particular case of a matrix when $c = 1$. For example, a 2×1 column vector of numbers e and f can be written as $\begin{bmatrix} e \\ f \end{bmatrix}$. Here, we recall that the product of a 2×2 matrix and a 2×1 column vector is also a 2×1 column vector computed as:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} ae + bf \\ ce + df \end{bmatrix} \tag{35}$$

As well, we recall matrix multiplication as the process given by:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix} \tag{36}$$

Now, it turns out that we can compute successive Fibonacci numbers using the matrix-vector multiplication procedure above:

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F(n+1) \\ F(n) \end{bmatrix} = \begin{bmatrix} F(n+2) \\ F(n+1) \end{bmatrix} \tag{37}$$

where we can call $\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$ a special *Fibonacci matrix*.

To enlighten you, let's have some examples:

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \times 1 + 1 \times 0 \\ 1 \times 1 + 0 \times 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 13 \\ 8 \end{bmatrix} = \begin{bmatrix} 1 \times 13 + 1 \times 8 \\ 1 \times 13 + 0 \times 8 \end{bmatrix} = \begin{bmatrix} 21 \\ 13 \end{bmatrix}$$

In the first example, we have produced $\begin{bmatrix} F(2) \\ F(1) \end{bmatrix}$ from $\begin{bmatrix} F(1) \\ F(0) \end{bmatrix}$, while in the second, we have produced $\begin{bmatrix} F(8) \\ F(7) \end{bmatrix}$ from $\begin{bmatrix} F(7) \\ F(6) \end{bmatrix}$, simply by pre-multiplying matrix $\mathbf{A}$ to the previous Fibonacci column vector.

We can do better by taking 2 or more successive steps using matrix multiplication. For example:

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 13 \\ 8 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^2 \begin{bmatrix} 13 \\ 8 \end{bmatrix} = \begin{bmatrix} 34 \\ 21 \end{bmatrix}$$

What does this mean?

This means we have yet another way to compute $F(n)$ using matrix exponentiation, which is arguably faster than building them from the recurrence in Eq. (31). Namely, we have the important result:

$$\mathbf{A}^n = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \begin{bmatrix} F(n+1) & F(n) \\ F(n) & F(n-1) \end{bmatrix} \tag{38}$$

The recurrence in Eq. (31) answers the query $F(n)$ in $\mathcal{O}(n)$ time. But using matrix exponentiation, $\mathbf{A}^n$, we can now answer it in $\mathcal{O}(\log_2 n)$ (read about binary exponentiation).

Exercise 4.1. Use matrices to prove the following identity about Fibonacci numbers.

$$F(m+n) = F(m)F(n+1) + F(m-1)F(n) \tag{39}$$

4.2 Fibonacci by Tiling

Here is one of the most common physical combinatorial interpretations of the Fibonacci numbers. How many ways can you completely tile a $1 \times n$ board with only 1 squares and 1×2 dominos such that no tiles overlap?

Let the number of such ways be f_n. By convention, we say that $f_0 = 1$ because there is only one way to tile the empty board (doing nothing), and $f_1 = 1$ because the only way to tile a single square is to just place a single square then. For the general n case, what are our options? Well, how do we tile the first cell? We do so by either beginning with a square or with a domino. If we begin with a square, then we need to tile the remaining $n-1$ cells in some fashion, and if we begin with a domino, we need to tile the remaining $n-2$ cells in some fashion. So, by conditioning on the first type of tile used, we get

$$f_n = f_{n-1} + f_{n-2}.$$

This shows us the connection it has with the Fibonacci numbers!

$n \to$	$\pi(m)$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$m = 1$	1	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$m = 2$	3	**0**	**1**	**1**	0	1	1	0	1	1	0	1	1	0	1	1	0
$m = 3$	8	**0**	**1**	**1**	**2**	**0**	**2**	**2**	**1**	0	1	1	2	0	2	2	1
$m = 4$	6	**0**	**1**	**1**	**2**	**3**	**1**	0	1	1	2	3	1	0	1	1	2

Table 3: Sequences of $F(n) \mod m$ and their periods $\pi(m)$ for $m = 1, 2, 3, 4$.

Theorem 4.2. If $F(n)$ is the nth Fibonacci number and f_n is the number of ways to tile a $1 \times n$ board, then $f_n = F(n+1)$.

Exercise 4.3. How many ways can a $2 \times n$ board be tiled using only 1×2 dominos?

Exercise 4.4. Use this notion of Fibonacci numbers as an alternate proof for the following identity.

$$F(m+n) = F(m)F(n+1) + F(m-1)F(n) \tag{40}$$

4.3 Pisano Period

As you may have noticed, $F(n)$ can grow quickly.

In fact, $F(93)$ (which has 20 digits) already exceeds the long long data type which has a limit of $2^{63} - 1 = 9.22 \times 10^{18}$. Some problems may require you to work with larger numbers such as $F(1000)$, etc. More often, since these numbers are very large, you will also be required to report $F(n) \mod m$ instead (recall modular arithmetic).

Fortunately, $F(n) \mod m$ is a repeating sequence! More formally, it is called a *periodic* sequence for $n = 0, 1, 2, \ldots$.

To give you some examples, we take some values of m and list $F(n) \mod m$ for $n = 0, 1, 2, \ldots, 15$ in Table 3. We will denote $\pi(m)$ as the *period* of the sequence $F(n) \mod m$. Notice that $F(n) \mod 2$ repeats every $\pi(2) = 3$ terms, $F(n) \mod 3$ repeats every $\pi(3) = 8$ terms, and $F(n) \mod 4$ repeats every $\pi(4) = 6$ terms.

In number theory, the function $\pi(m)$ is given the name *Pisano Period*. The sequence of Pisano periods can be listed as

m	1	2	3	4	5	6	7	8	9	10
$\pi(m)$	1	3	8	6	20	24	16	12	24	60

Computing $\pi(m)$ is hard (the closed form is still an open problem). In the contest, you can use any cycle detection algorithm for obtaining $\pi(m)$. Nonetheless, we are assured that the value of $\pi(m)$ is at most $6m$. Other useful results for $k > 1$ are:

$$\pi(2^k) = 3 \cdot 2^{k-1}$$
$$\pi(5^k) = 4 \cdot 5^k$$
$$\pi(2 \cdot 5^k) = 6 \cdot 2 \cdot 5^k$$

Also, if a and b are coprime, then $\pi(ab) = \text{lcm}(\pi(a))(\pi(b))$, where $\text{lcm}(\cdot, \cdot)$ is the least common multiple.

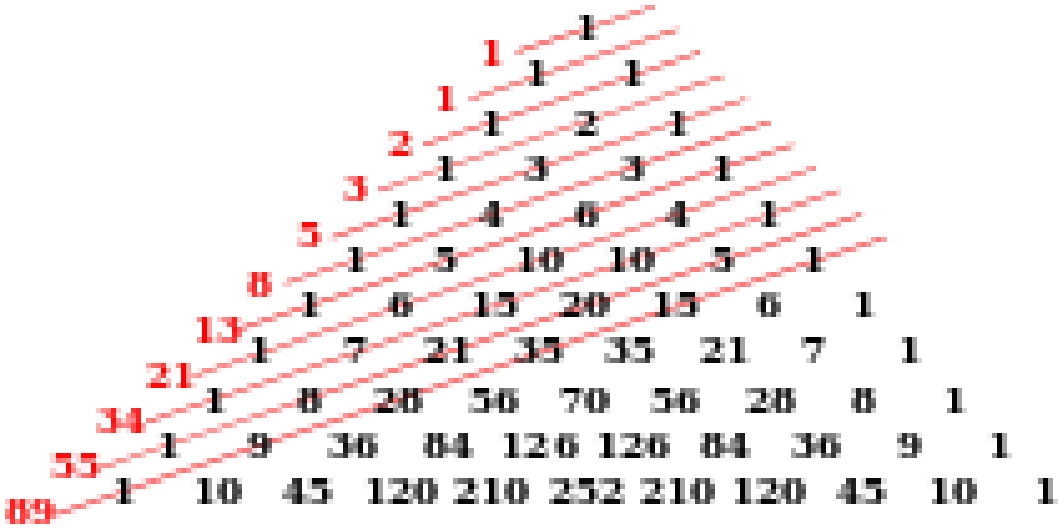


Figure 9: Connection between Fibonacci numbers and Pascal's triangle.

Once you have computed $\pi(m)$, it becomes easier to determine $F(n) \mod m$ for very large values of n, i.e. you only need to compute $F(0)$ to $F(\pi(m))$ and obtain any information from that sequence. This works if m is relatively small.

4.4 Connection to Pascal's Triangle

The sum of entries in the "shallow" diagonals of Pascal's triangle gives rise to Fibonacci numbers (see Fig. 9). In writing, we have:

$$F(n) = \sum_{k=0}^{\lfloor \frac{n-1}{2} \rfloor} \binom{n-k-1}{k} \tag{41}$$

where $\binom{n}{k}$ is the binomial coefficient.

Exercise 4.5. Prove the sum (41) combinatorially.

Hint: Think of the Fibonacci numbers as a tiling.

4.5 Zeckendorf Representation

Here, we discuss how to represent integers as a sum of only Fibonacci numbers.

Zeckendorf's theorem (due to Belgian mathematician, Edouard Zeckendorf) states that:

Theorem 4.6 (Zeckendorf's theorem)**.**

- (Existence) Every positive integer n can be written as a sum of one or more distinct non-consecutive Fibonacci numbers. We call such a sum as the *Zeckendorf representation* of n.

- (Uniqueness) No positive integer has 2 different Zeckendorf representations.

For example, here are the Zeckendorf representations, $Z(n)$, of selected positive integers:

n	$Z(n)$	1	2	3	5	8	13	21	34	55	89	144
1	1	1	0	0	0	0	0	0	0	0	0	0
2	2	0	1	0	0	0	0	0	0	0	0	0
4	3+1	1	0	1	0	0	0	0	0	0	0	0
33	21+8+3+1	1	0	1	0	1	0	1	0	0	0	0
71	55+13+3	0	0	1	0	0	1	0	0	1	0	0
100	89+8+3	0	0	1	0	1	0	0	0	0	1	0

Mathematically, we can write the Zeckendorf representation of $n > 0$ as:

$$n = \sum_{k=2}^{L} \epsilon_k F(k) \tag{42}$$

where L is the index of the largest Fibonacci number less than or equal to n, and ϵ_k is 0 or 1, with the restriction $\epsilon_k \epsilon_{k+1} = 0$. In the examples above, the 0's and 1's are the values of ϵ_k for $k = 2, 3, \ldots, 12$.

The proof of Zeckendorf's theorem is available elsewhere. Here, we are interested in generating the representation using a *greedy* algorithm. It turns out that to generate $Z(n)$: you can always take the largest $F \leq n$, then take $n := n - F$, and then repeat until $n = 0$. Using this method, you are guaranteed that no consecutive Fibonacci numbers will be taken at any time.

As an exercise, you can implement this algorithm of generating $Z(n)$ for any $n > 0$.

4.6 Selected Extensions

In this subsection, we discuss some extensions of Fibonacci numbers that might also appear in some problems.

4.6.1 Negafibonacci

If we retain the base cases but rearrange the recurrence in Eq. (31) like so:

$$F(n-2) = F(n) - F(n-1) \tag{43}$$

then we can generate the Fibonacci numbers backwards for $n < 0$. This sequence is given the name *Negafibonacci* numbers.

A sample list of the negafibonacci numbers are given:

n	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8
$F(n)$	-21	13	-8	5	-3	2	-1	1	0	1	1	2	3	5	8	13	21

The negafibonacci numbers satisfy the relation:

$$F(-n) = (-1)^{n+1} F(n) \tag{44}$$

It is said that a Zeckendorf representation for all integers (negative and positive) can be achieved with both Fibonacci and negafibonacci numbers. Can you write a code that can generate this representation?

4.6.2 Lucas Numbers

Lucas numbers, denoted by $L(n)$, are closely related to the Fibonacci numbers. They are defined as:

$$\begin{aligned} L(0) &= 2 \\ L(1) &= 1 \\ L(n) &= L(n-1) + L(n-2) \end{aligned} \tag{45}$$

The first few Lucas numbers are:

n	0	1	2	3	4	5	6	7	8	9	10
$L(n)$	2	1	3	4	7	11	18	29	47	76	123

Similarly, a closed formula for $L(n)$ is given by:

$$L(n) = \varphi^n + (1-\varphi)^n \tag{46}$$

where φ is the golden ratio. This is also guaranteed to always yield integers regardless of $n \in \mathbb{N}$, even if φ is an irrational number.

Useful relationships between $L(n)$ and $F(n)$ are known:

$$F(2n) = L(n)F(n) \tag{47}$$

$$L(n) = F(n-1) + F(n+1) = F(n) + 2F(n-1) = F(n+2) - F(n-2) \tag{48}$$

$$F(n) = \frac{L(n-1) + L(n+1)}{5} \tag{49}$$

4.6.3 Generalized Fibonacci Series

Consider the generalized series defined as:

$$\begin{aligned} G(0) &= a \\ G(1) &= b \\ G(n) &= G(n-1) + G(n-2) \end{aligned} \tag{50}$$

With a slight abuse of notation, we re-denote this generalized series as $G(a, b, n)$. Thus, $G(0, 1, n) = F(n)$ and $G(2, 1, n) = L(n)$. Let's list the first few elements of $G(a, b, n)$ for any a, b:

n	0	1	2	3	4	5
$G(a,b,n)$	a	b	$a+b$	$a+2b$	$2a+3b$	$3a+5b$

If we continue to do this, we can list down the successive coefficients of a: $1, 0, 1, 1, 2, 3, \ldots$. Likewise, the successive coefficients of b are $0, 1, 1, 2, 3, 5, \ldots$. It turns out that these are just $F(n-1)$ and $F(n)$, respectively! Hence, we can write the following relationship:

$$G(a, b, n) = aF(n-1) + bF(n) \tag{51}$$

This is called the *Reduction Rule* since it basically tells us that the generalized series are just a sum of two multiples of Fibonacci series. Therefore, the generalized series may as well be called the *Generalized Fibonacci Series*.

We can also derive a closed-form for $G(a, b, n)$, a matrix-based computation just as in Section 4.1, and other similar properties. We will leave this to you as exercises.

k	name of sequence	first few terms of the sequence
2	Fibonacci	$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$
3	Tribonacci	$0, 0, 1, 1, 2, 4, 7, 13, 24, 44, 81, \ldots$
4	Tetranacci	$0, 0, 0, 1, 1, 2, 4, 8, 15, 29, 56, 108, \ldots$
5	Pentanacci	$0, 0, 0, 0, 1, 1, 2, 4, 8, 16, 31, 61, 120, \ldots$

Figure 10: First few k-generalized Fibonacci numbers for $k = 2, 3, 4, 5$

4.6.4 *k*-generalized Fibonacci numbers

Many other generalizations to the Fibonacci numbers exist. Here, we consider one last generalization. Consider the extended definition:

$$F^{(k)}(n) = \begin{cases} 0, & 0 \leq n < k-1 \\ 1, & n = k-1 \\ \sum_{i=1}^{k} F^{(k)}(n-1) & n > k-1 \end{cases} \tag{52}$$

This sequence is given the name k-generalized Fibonacci numbers. Others have named them as the tribonacci, tetranacci, pentanacci numbers for $k = 3, 4, 5$, for example.

The first few terms in these sequences are given[6] in Fig. 10. You might find these useful in some applications.

[6] Bacani and Rabago (2015). *On Generalized Fibonacci Numbers*. Applied Mathematical Sciences.

5 Combinatorics on Graphs

There are a few important counting results concerning graphs as well.

5.1 Counting trees: Cayley's formula

Given n, how many unrooted, labelled trees are there?

This is well-known, and *Cayley's formula* gives the answer: n^{n-2}. A really nice proof using double counting can be found in this link: https://en.wikipedia.org/wiki/Double_counting_(proof_technique)#Counting_trees. I suggest that you read that proof and understand it before proceeding.

However, it should be noted that although the above proof is quite elegant, there are other proofs, and some have certain advantages over this one. For example, one such proof involves creating a bijection (one-to-one mapping) between the set of labelled trees with n nodes and the set of sequences of integers in $\{1, 2, \ldots, n\}$ of length $n-2$. The latter set clearly contains n^{n-2} elements, so this also proves Cayley's formula. The nontrivial part is the bijection itself.

If you find the Prüfer sequence method of enumeration hard to grasp/memorize, just to be sure, there are other ways of enumerating unrooted labelled trees. For example, one may choose to just enumerate *rooted* trees and then forget the root of each one. Enumerating rooted trees is easier since they can be done recursively (how?), although one has to be careful with this method since it enumerates unrooted trees multiple times. (Can you think of a way to fix this?)

Counting *unlabelled* trees is significantly harder; however, enumerating them for small n is still possible. This will be left as an exercise.

5.2 Counting graphs

Let's get more general: given n, how many simple undirected graphs of n nodes are there?

Now, it turns out that this question is very easy since, unlike trees, there are no structural restrictions on the graphs, hence we can independently select each edge. The final formula is very simple so it will be left to the reader to write down. Let's denote this number by $g(n)$.

A more interesting question is the following: given n, how many simple undirected *connected* graphs of n nodes are there? Let's denote the answer to this by $g_{\text{conn}}(n)$. This is tougher now since connectedness is a bit hard to summarize in a counting formula.

It would be nice if we can find a formula for $g_{\text{conn}}(n)$ in terms of $g(n)$ since we already know $g(n)$ and they seem to be somewhat related. However, interestingly, a better approach turns out to be the reverse: to find a formula for $g(n)$ in terms of $g_{\text{conn}}(n)$. This may seem counterintuitive; such a formula doesn't seem useful if we already know $g(n)$ and also don't know $g_{\text{conn}}(n)$. But we shall soon see why this helps us.

To find such a formula, let's find another way of counting graphs by piecing together several connected graphs. To do so, let's look at a particular node, say 1, and find its connected component. Let's assume it belongs to a connected component of size k. How many such graphs are there? Well, to build one,

- We must select which nodes will be in the component. There are $\binom{n-1}{k-1}$ ways to do this. (Question: Why not $\binom{n}{k}$?)
- Next, we must form a connected graph out of these k nodes. There are $g_{\text{conn}}(k)$ ways to do this by definition.

- Finally, we need to build a graph out of the remaining $n-k$ nodes. There are $g(n-k)$ ways to do this by definition.

Since these choices can be made independently, we get $\binom{n-1}{k-1} \cdot g_{\text{conn}}(k) \cdot g(n-k)$ graphs where node 1 belongs to a component of size k. Since k can be anything between 1 and n, by adding up all such possibilities, we get the following formula:

$$g(n) = \sum_{k=1}^{n} \binom{n-1}{k-1} g_{\text{conn}}(k) g(n-k)$$

Note that this equation is valid since this exhausts all possibilities and no possibility is counted twice.

Now, how does this help us compute $g_{\text{conn}}(k)$? Let's extract the $k = n$ term of the sum and rearrange:

$$\begin{aligned}
g(n) &= \sum_{k=1}^{n} \binom{n-1}{k-1} g_{\text{conn}}(k) g(n-k) \\
g(n) &= \sum_{k=1}^{n-1} \binom{n-1}{k-1} g_{\text{conn}}(k) g(n-k) + \binom{n-1}{n-1} g_{\text{conn}}(n) g(n-n) \\
g(n) &= \sum_{k=1}^{n-1} \binom{n-1}{k-1} g_{\text{conn}}(k) g(n-k) + g_{\text{conn}}(n) \\
g_{\text{conn}}(n) &= g(n) - \sum_{k=1}^{n-1} \binom{n-1}{k-1} g_{\text{conn}}(k) g(n-k)
\end{aligned}$$

This formula now allows us to compute $g_{\text{conn}}(n)$ using *dynamic programming*!

The lesson here is to not always look for a direct formula for the answer to the question in terms of related but known results; rather, one should try to build relationships between known and unknown results, and see whether the derived formulas can somehow be converted into an algorithm.

5.3 Graph coloring

Let us now turn to graph coloring. Given some graph G and k colors, our task is to determine if the nodes can be colored such that no two adjacent nodes are given the same color. We assume that the graph has no self-loops.

The *chromatic number* of the graph is the fewest number of colors required to color the graph in a valid way. If we can determine that a graph is k-colorable for any k in polynomial time, then the chromatic number can be computed in polynomial time by simply looking for it in a linear fashion (or with binary search if you're feeling fancy). However, it is well-known that checking for k-colorability is hard and that there's no known polynomial-time algorithm that solves this, even for $k = 3$.[7] Nonetheless, we will deal with a harder version of the problem which is the *counting* version: How many ways can you color the graph if there are k colors available?

[7] In fact, solving this in polynomial time, or proving that it's impossible, even for $k = 3$, is equivalent to the famous P vs NP problem, one of the seven Millennium Prize Problems.

5.3.1 Chromatic polynomial

Let's denote the answer to the counting problem by $P_G(k)$. P_G is called the *chromatic polynomial* of G, so named because $P_G(k)$ is indeed a polynomial in k; this should become possible to show later. Note that the chromatic number is simply the least k such that $P_G(k) \neq 0$.

In general, the polynomial P_G is hard to compute given G; indeed, computing it in polynomial time allows us to solve the non-counting version by simply computing $P_G(k)$ and checking whether it is zero. However, for some special graphs, it turns out to be easy to compute. Let's analyze a few of them.

Empty graphs. Let's define an empty graph as a graph with no edges. Such graphs are very easy to color since there are no restrictions at all! The answer is clearly k^n. Notice that it is indeed a polynomial in k.

Complete graphs. Similarly, complete graphs are easy to color; once we use up a color, it can never be used again. This gives us k choices for the first node, then $k-1$ choices for the second one, and $k-2$ for the next one, etc. Thus, the chromatic polynomial of a complete graph is

$$k(k-1)(k-2)\cdots(k-(n-1)).$$

Again, it's a polynomial in k.

Trees. Trees are easy to color because of two useful facts about them: that there is always a leaf, and that removing a leaf results in a smaller tree. Thus, to color a tree with n nodes, we can first look for a leaf, remove it, and (recursively) color the remaining tree. Now, adding the leaf back in, this gives us $(k-1)$ choices for its color. We can repeat the argument in the smaller tree (i.e., removing another leaf) until we're left with a singleton, which obviously has k ways of coloring. This shows that the chromatic polynomial of a tree with n nodes is

$$k(k-1)^{n-1}.$$

Alternatively, imagine rooting the tree and coloring it in a top-down fashion. The root has k choices, but every other node has $(k-1)$ choices since it has to be different from its parent. This gives the same result.

Note that we didn't really specify which tree we're dealing with, but the above argument holds nonetheless, so it actually shows that the chromatic polynomials of all trees with n nodes are the same!

Also, notice again that it's a polynomial in k.

Disconnected Graphs. Consider two disjoint graphs G_1 and G_2, and let their union be G. Can we somehow relate P_{G_1}, P_{G_2} and P_G? Indeed, we can, and it's not that hard; if we want to color G, we can simply color G_1 and G_2 independently, thus by the product rule, $P_G(k) = P_{G_1}(k) \cdot P_{G_2}(k)$.

5.3.2 The deletion-contraction algorithm

There's a general way to compute P_G for a given G. To begin, consider two nodes u and v that are not connected by an edge. If you can't find such a pair, then G must be a complete graph, and thus we already know P_G. Otherwise, consider u and v. In a valid coloring of G, u and v can have different colors or they can have the same color. Let's try to count those possibilities separately.

- If they have different colors, then we can add an edge between u and v and the coloring remains valid. If we denote by $G + uv$ the graph with the edge (u, v) added, then the

number of ways to color G with u and v having different colors is $P_{G+uv}(k)$ since the new edge ensures that we only count colorings where u and v are colored differently.

- If they have the same color, then we can treat u and v as the same node altogether. So denote by G/uv the graph with nodes u and v identified with each other. Then any coloring G/uv extends naturally to a coloring of G with u and v having the same color. Hence, the number of ways to color G with u and v having the same color is $P_{G/uv}(k)$.

Since these possibilities are disjoint and cover all cases, we arrive at the following formula:

$$P_G(k) = P_{G+uv}(k) + P_{G/uv}(k)$$

We can then recursively repeat the procedure for $G + uv$ and G/uv. Note that this process will halt since we're either adding an edge or decreasing the number of nodes by one, and each of those brings the graph strictly closer to becoming a complete graph.

Unfortunately, there's no guarantee that this process will halt in *polynomial* time since there may be exponentially many distinct graphs encountered in the process, even if we memoize/hash on graphs.

This algorithm has a mirrored version. Consider now two nodes u and v that are connected by an edge, and let $G - uv$ be the graph formed by removing the edge (u, v), and G/uv is defined similarly as before, except that no self-loop will be formed. Then the formula above can be rearranged to obtain the following:

$$P_G(k) = P_{G-uv}(k) - P_{G/uv}(k)$$

(It is instructive to interpret this using the inclusion-exclusion principle.) Similarly to the above, we can repeat the procedure on the resulting graphs until all edges disappear, in which case we get an empty graph which we also already know how to color. Hence, this will halt, although again not necessarily in polynomial time. This version of the algorithm is called the *deletion-contraction algorithm*.

It should be noted that while this algorithm is not guaranteed to halt in polynomial time, for some special types of graphs it may yield a polynomial-time solution in case the resulting smaller/larger graphs are special as well. For example, consider cycle graphs. It turns out that deleting and contracting edges are quite special and in fact it allows us to compute the chromatic polynomials of cycles. This is left as an exercise.

Finally, the deletion-contraction formula (or its mirror) actually allows us to show by induction that P_G is indeed a polynomial for any graph G. This will be left as an exercise.

6 More Modulo Stuff

6.1 Chinese Remainder Theorem

Consider the following problem. Find an integer x that satisfies both the following recurrences:

$$x \equiv a_1 \pmod{m_1}$$
$$x \equiv a_2 \pmod{m_2}$$

What can we say about the existence of a solution to this problem?

The *Chinese remainder theorem* guarantees that if m_1 and m_2 are coprime, then there's a solution to this system, and furthermore, it is unique modulo $m_1 m_2$. This is quite an elegant result, and it can be proven using a simple counting argument as follows.

Let's consider the sequence $0, 1, 2, 3, \ldots$ and consider their remainders modulo m_1 and m_2. For example, for $m_1 = 3$ and $m_2 = 4$, the list will look like this:

x	$(x \bmod 3, x \bmod 4)$
0	$(0, 0)$
1	$(1, 1)$
2	$(2, 2)$
3	$(0, 3)$
4	$(1, 0)$
5	$(2, 1)$
6	$(0, 2)$
7	$(1, 3)$
8	$(2, 0)$
$\vdots$	$\vdots$

Now, we ask, when does the pair of remainders first repeat? Well, let's say the smallest pair of repeated remainders correspond to x and x' with $0 \leq x < x'$. Then:

- x must be zero, otherwise, $x - 1$ and $x' - 1$ are another repeated pair that comes earlier, contradicting the fact that x and x' are the smallest pair.
- $x' \equiv x \equiv 0 \pmod{m_1}$, hence, x' must be divisible by m_1.
- $x' \equiv x \equiv 0 \pmod{m_2}$, hence, x' must be divisible by m_2.
- Thus, x' is a common multiple of m_1 and m_2 and thus must be divisible by the LCM of m_1 and m_2, but since they are coprime, the LCM is simply $m_1 m_2$. Thus, x' must be at least $m_1 m_2$.
- But it's also easy to see that $x' = m_1 m_2$ satisfies $x' \equiv 0 \pmod{m_1}$ and $x' \equiv 0 \pmod{m_2}$, hence, we have $x' = m_1 m_2$.
- Since $x' = m_1 m_2$ is the first repeat, it follows that all the numbers in the set $\{0, 1, 2, \ldots, m_1 m_2 - 1\}$ have distinct remainder pairs modulo m_1 and m_2. Since there are exactly $m_1 m_2$ such remainder pairs, this guarantees the existence and uniqueness of the solution for any given remainder pair (a_1, a_2). This completes the proof.

Basically, the above argument establishes a bijection between the numbers in the set of solutions modulo $m_1 m_2$, $\{0, 1, \ldots, m_1 m_2 - 1\}$ and the set of problems, $\{0, 1, \ldots, m_1 - 1\} \times \{0, 1, \ldots, m_2 - 1\}$.

Now that the existence and uniqueness (modulo m_1m_2) is established, let's now figure out how to compute it. Clearly, a brute-force $\mathcal{O}(m_1m_2)$ (or even $\mathcal{O}(\min(m_1, m_2))$) approach is too slow.

We will discuss two methods. By the way, note that this theorem (and the following algorithms) is easily generalizable to more than two congruences, and the (generalized) Chinese remainder theorem will apply as long as the moduli are *pairwise* coprime.

6.1.1 Garner's algorithm

For our first method, we note that $x \equiv a_1 \pmod{m_1}$ is equivalent to saying that $x = a_1 + m_1y$ for some integer y. Substituting this to the second congruence gives:

$$\begin{aligned} a_1 + m_1y &\equiv a_2 && \pmod{m_2} \\ m_1y &\equiv a_2 - a_1 \\ y &\equiv (a_2 - a_1)m_1^{-1} && \pmod{m_2} \end{aligned}$$

Remember that m_1 and m_2 are coprime, so m_1 is invertible modulo m_2. This gives us the integer y which we can substitute to $x = a_1 + m_1y$ to get x, the solution!

This naturally generalizes to more than two congruences. Suppose that there's a third congruence $x \equiv a_3 \pmod{m_3}$ with m_3 being coprime to m_1 and m_2. From the above, we know that $x \equiv a_1 + m_1y \pmod{m_1m_2}$, so $x = a_1 + m_1y + m_1m_2z$ for some integer z. Substituting to the new congruence gives

$$\begin{aligned} a_1 + m_1y + m_1m_2z &\equiv a_3 && \pmod{m_3} \\ m_1m_2z &\equiv a_3 - a_1 - m_1y \\ z &\equiv (a_3 - a_1 - m_1y)m_1^{-1}m_2^{-1} && \pmod{m_3} \end{aligned}$$

This now gives us the new integer z that allows us to construct the answer!

This time, m_1^{-1} and m_2^{-1} denote the inverses modulo m_3, so in particular, m_1^{-1} will be different from the previous m_1^{-1}, so be careful when interpreting these formulas.

With more congruences, this idea allows us to incrementally compute x modulo m_1, then m_1m_2, then $m_1m_2m_3$, etc.

6.1.2 Linear combinations

Let's look at the second method, starting again with just two recurrences.

This method uses the fact that if y is a solution to the system of congruences

$$\begin{aligned} y &\equiv a_1 \pmod{m_1} \\ y &\equiv a_2 \pmod{m_2} \end{aligned}$$

and z is a solution to the system of congruences

$$\begin{aligned} z &\equiv b_1 \pmod{m_1} \\ z &\equiv b_2 \pmod{m_2} \end{aligned}$$

then $\alpha y + \beta z$ is the solution x to the following system:

$$\begin{aligned} x &\equiv \alpha a_1 + \beta b_1 \pmod{m_1} \\ x &\equiv \alpha a_2 + \beta b_2 \pmod{m_2} \end{aligned}$$

This should be very easy to see if you're familiar with modular arithmetic. But now, suppose we're able to solve the following two systems:

$$\begin{aligned} y &\equiv 1 \pmod{m_1} \\ y &\equiv 0 \pmod{m_2} \end{aligned}$$

and

$$\begin{aligned} z &\equiv 0 \pmod{m_1} \\ z &\equiv 1 \pmod{m_2} \end{aligned}$$

Then we get the original answer to our problem as $x = a_1 y + a_2 z$. (Verify.)

Hence, let's analyze the following simpler system:

$$\begin{aligned} y &\equiv 1 \pmod{m_1} \\ y &\equiv 0 \pmod{m_2} \end{aligned}$$

The second one says that m_2 divides y, so $y = m_2 r$ for some integer r. Substituting this to the first congruence, we get

$$m_2 r \equiv 1 \pmod{m_1},$$

but this is simply saying that r is the inverse of m_2 modulo m_1. Hence, we get the solution

$$y = m_2(m_2^{-1} \bmod m_1).$$

Similarly, we get

$$z = m_1(m_1^{-1} \bmod m_2).$$

Combining the two, we get the solution to the original system as

$$x = a_1 m_2(m_2^{-1} \bmod m_1) + a_2 m_1(m_1^{-1} \bmod m_2),$$

which you may readily verify to be correct!

The generalization to multiple congruences is straightforward. Let M_i be the product of all moduli except m_i. Then the solution to the congruences $x \equiv a_i \pmod{m_i}$ for all $1 \leq i \leq k$ is

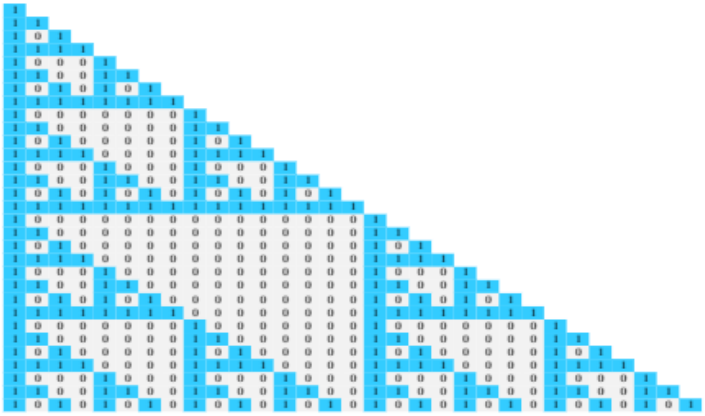
$$x \equiv \sum_{i=1}^{k} a_i M_i (M_i^{-1} \bmod m_i).$$

Indeed, you may verify that it works for any i by reducing it modulo m_i, and noting that $m_i \mid M_j$ if $j \neq i$.

For the purposes of implementation though, one may simply implement the two-congruence version and then repeatedly call that function until all congruences are included.

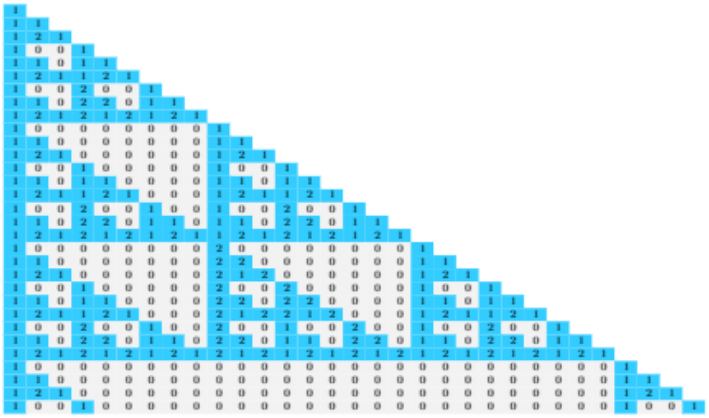
6.2 Lucas's theorem

Something interesting happens when we reduce Pascal's triangle modulo 2:

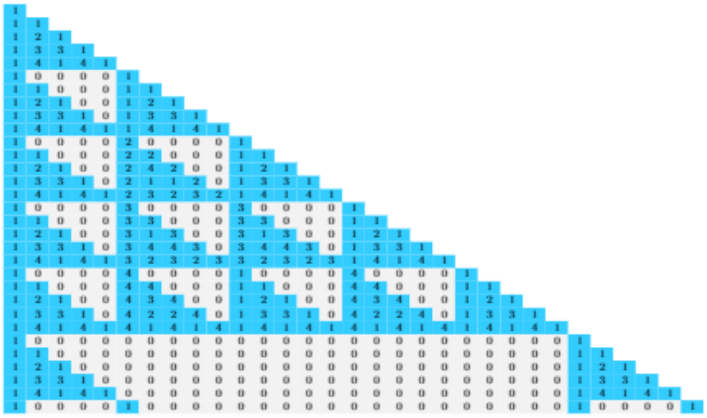


Those of you who are familiar will recognize this as the Sierpinski triangle! Similar Sierpinski-like fractals appear if we reduce the triangle modulo 3, 5, 7, etc., as long as we reduce it modulo a prime.

Modulo 3:



Modulo 5:



For non-prime moduli, it doesn't yield nice patterns (try plotting them!), but we shall learn how to deal with at least some of them.

The pattern can be exploited to allow you to compute $\binom{n}{r} \bmod 2$ (or, in general, mod a prime p) quicker than the previously-discussed methods such as dynamic programming and the factorial formulas. I suggest you try to implement it before proceeding, i.e., given n and r, find $\binom{n}{r} \bmod 2$ quickly.

In fact, there's a very nice way of summarizing the fractal nature of the pattern:

$$\binom{n}{r} \equiv \binom{\lfloor n/p \rfloor}{\lfloor r/p \rfloor}\binom{n \bmod p}{r \bmod p} \pmod{p}$$

This is called *Lucas's theorem* and immediately provides a recursive procedure to compute $\binom{n}{r} \bmod p$. The base case would be when $n < p$ and $r < p$; in those cases, we may assume that p is small enough that the previous methods will work. I recommend precomputing all factorials and their inverses modulo p and simply using

$$\binom{n}{r} \equiv n!(n-r)!^{-1}r!^{-1} \pmod{p}.$$

An alternative way of stating the above involves writing n and r into their base-p forms. Notice that $n \bmod p$ is simply the last digit of n in base p, and $\lfloor n/p \rfloor$ denotes the rest of the digits. Thus, by expanding the latter repeatedly, we get an alternative version of Lucas's theorem in terms of base p representations.

Let n and r be written in base p as $n_{k-1}n_{k-2}\ldots n_1n_0$ and $r_{k-1}r_{k-2}\ldots r_1r_0$, respectively, where we pad with leading zeroes so they become the same length. Then Lucas's theorem is equivalent to:

$$\binom{n}{r} \equiv \binom{n_{k-1}}{r_{k-1}}\binom{n_{k-2}}{r_{k-2}}\cdots\binom{n_1}{r_1}\binom{n_0}{r_0} \pmod{p}$$

Clearly, each base p digit is $< p$, so each term on the right can be easily computed. For $p = 2$, this is especially simple since each digit is either 0 or 1, thus, $\binom{n_i}{r_i} \equiv 0$ iff $n_i = 0$ and $r_i = 1$. Thus, we find that $\binom{n}{r} \equiv 1$ if every digit of n is at least as large as the corresponding digit in r, and $\equiv 0$ otherwise. A similar fact is true for other primes.

Now, what about non-prime moduli? It turns out that it's a bit tricky to handle the general case immediately; even the case of a modulus that's a square of a prime (p^2) is already difficult. But it turns out that we can already deal with some special cases with the help of the Chinese remainder theorem.

To see this, suppose we already know $\binom{n}{r} \bmod m_1$ and $\binom{n}{r} \bmod m_2$, and we want to find $\binom{n}{r} \bmod (m_1m_2)$. If m_1 and m_2 are coprime, then the Chinese remainder theorem allows us to compute the answer! Thus, since we already know how to compute $\binom{n}{r}$ modulo (small-ish) primes, we can combine them to be able to compute $\binom{n}{r}$ modulo *products of distinct primes*, also known as *squarefree numbers*.

As a demonstration, let's say we want to compute the last digit of $\binom{420}{160}$. This is equivalent to computing it modulo 10, and $10 = 2 \cdot 5$. Thus, we compute it modulo 2 and 5 separately, and combine them with the Chinese remainder theorem.

Computing $\binom{420}{160} \bmod 2$. We write 420 and 160 in base 2:

$$\begin{aligned}\binom{420}{160} &\equiv \binom{110100100_2}{010100000_2} \\ &\equiv \binom{1}{0}\binom{1}{1}\binom{0}{0}\binom{1}{1}\binom{0}{0}\binom{0}{0}\binom{1}{0}\binom{0}{0}\binom{0}{0} \\ &\equiv 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \equiv 1\end{aligned}$$

Thus, we get $\binom{420}{160} \equiv 1 \pmod{2}$. Note that we padded 160 with a leading zero so that they have the same number of digits.

Computing $\binom{420}{160} \bmod 5$. We write 420 and 160 in base 5:

$$\begin{aligned}\binom{420}{160} &\equiv \binom{3140_5}{1120_5} \\ &\equiv \binom{3}{1}\binom{1}{1}\binom{4}{2}\binom{0}{0} \\ &\equiv 3 \cdot 1 \cdot 6 \cdot 1 \equiv 18 \equiv 3\end{aligned}$$

Thus, we get $\binom{420}{160} \equiv 3 \pmod 5$.

Computing $\binom{420}{160} \bmod 10$. This is equivalent to solving the following system of congruences:

$$\begin{aligned}x &\equiv 1 \pmod 2 \\ x &\equiv 3 \pmod 5\end{aligned}$$

One can now just use any of the earlier-discussed methods to find that $x \equiv 3 \pmod{10}$. Hence, we find that $\binom{420}{160} \equiv 3 \pmod{10}$, so its last digit must be a 3. You may verify this easily by explicitly computing $\binom{420}{160}$ and looking at the last digit.[8] So the method worked!

[8] In this case, we're able to do this check since 420 is relatively small, but note that we may not always be able to check the answer, particularly if the arguments to the binomial coefficient are large.

7 Problems

Solve as many as you can! Ask me if anything is unclear.[9] In general, the harder problems will be worth more points, although I won't be saying which ones are harder.

7.1 Warmup Problems

W1 Basic seating arrangements: https://www.hackerrank.com/contests/noi-ph-2019-preselection/challenges/basic-seating-arrangements

W2 Basic grid paths 1: https://www.hackerrank.com/contests/noi-ph-2019-preselection/challenges/basic-grid-paths-1

W3 Basic grid paths 2: https://www.hackerrank.com/contests/noi-ph-2019-preselection/challenges/basic-grid-paths-2

W4 Basic grid paths 3: https://www.hackerrank.com/contests/noi-ph-2019-preselection/challenges/basic-grid-paths-3

7.2 Non-coding problems

No need to be overly formal in your answers; as long as you're able to convince me, it's fine!

N1 [4★] Exercise 1.22

N2 [5★] Exercise 2.10

N3 [3★] Exercise 2.11

N4 [5★] Exercise 4.5

N5 Find a 'nice' expression for each of the following sums. I will consider it "nice" if you can remove the summation sign somehow; you'll just **know** when it's nice enough. When in doubt, you can just ask me if what you have is "nice". As a hint, all of these sums can be associated with some sort of counting problem, so I highly encourage the use of combinatoric proofs. However, if you have a solution by other means, that is equally valid as well. Just make sure you **prove** your answers :))

(a) [5★]

$$\sum_{k=1}^{n} k^2 \binom{n}{k}$$

(b) [5★]

$$\sum_{k=0}^{r} \binom{m}{k} \binom{n}{r-k}$$

(c) [5★]

$$\sum_{k=0}^{n} \binom{n}{k}^2$$

[9] Especially for ambiguities! Otherwise, you might risk getting fewer points even if you *technically* answered the question correctly.

(d) [8★]

$$\sum_{k=1}^{m-n+1} \binom{m-k}{n-1}$$

(e) [5★]

$$\sum_{k=0}^{\min(m,n-m)} \binom{m}{k}\binom{n-m}{k}$$

N6 Let's attempt to solve Fibonacci GCD[10] by proving a remarkable property about the Fibonacci numbers. We will show that $\gcd(F(m), F(n)) = F(\gcd(m, n))$. This shall be done in the form of a guided proof. The Lemmas and general outline of the proof will be given, but it is up to you to fill in the gaps and justify each step. We presume that you already know how the normal Euclidean algorithm works.

(a) [3★] Do Exercise 5.1 or 5.4

(b) [2★] Prove that adjacent Fibonacci numbers are coprime, i.e. $\gcd(F(m), F(m+1)) = 1$.

(c) [2★] Prove that if $\gcd(a, b) = 1$, then $\gcd(a, bc) = \gcd(a, c)$.

(d) [5★] Now, let's evaluate $\gcd(F(m), F(n))$. Let $m = nq+r$, where q and r are integers such that $0 \le r < n$. Show that $\gcd(F(m), F(n)) = \gcd(F(n), F(r))$. How does this relate to the normal Euclidean Algorithm?

N7 [4★] Show that, given the value of $\binom{x}{y} \bmod p$, one can compute $\binom{x+1}{y} \bmod p$, $\binom{x}{y+1} \bmod p$, $\binom{x-1}{y} \bmod p$ and $\binom{x}{y-1} \bmod p$ in $\mathcal{O}(\log p)$ time each. Assume that p is prime.

N8 [3★] Show that, given the value of $\binom{x}{y} \bmod p$, one can compute $\binom{x'}{y'} \bmod p$ in $\mathcal{O}(d \log p)$ time, where $d = |x - x'| + |y - y'|$.

N9 [5★] Two arrays A and B are *isomorphic* if they have the same length n, and for every two indices i, j such that $1 \le i, j \le n$, $A_i = A_j$ if and only if $B_i = B_j$. How many nonisomorphic arrays of length n are there?

N10 [4★] Show that $\begin{bmatrix} n \\ k \end{bmatrix}$ is the coefficient of x^k in the product

$$x(x+1)(x+2)\cdots(x+(n-1)).$$

Hint: Use its recurrence relation.

N11 [3★] Show that a string of parenthesis characters is balanced if and only if it can be formed by starting with the letter `X` and repeatedly replacing any `X` with either the empty string or `(X)X`, until there are no more `X`s. For example, `(()())()` can be formed as follows:

```
X
(X)X
(X)(X)X
(X)()X
((X)X)()X
((X)(X)X)()X
((X)(X))()X
((X)(X))()
(()(X))()
(()())()
```

[10] **Fibonacci GCD:** https://www.hackerrank.com/contests/noi-ph-2019-preselection/challenges/fibonacci-gcd

N12 [5★] Use the previous fact to find a recurrence relation on the Catalan numbers $C(n)$ in terms of itself. Show that this leads to an $\mathcal{O}(n^2)$-time algorithm for computing $C(n)$ mod m.

N13 [5★] Show that the number of full binary trees with $n+1$ leaves is $C(n)$. *Hint:* There are many ways to do this, but one way would be to show that the number of full binary trees also satisfy the recurrence in the previous question.

N14 [5★] Prove Binet's formula (Equation 32) via induction. *Hint:* ϕ and $1-\phi$ are the two roots of the equation $x^2 = x+1$, so by multiplying by x^{n-2}, they also satisfy $x^n = x^{n-1} + x^{n-2}$.

N15 [5★] Let F_n be a sequence defined recursively as $F_n = a \cdot F_{n-1} + b \cdot F_{n-2}$ for fixed constants a and b. Show that F_n mod m can be computed in $\mathcal{O}(\log n)$ time given F_0, F_1, a, b, n, m.

N16 [5★] Show that if m_1 and m_2 are coprime, then $\pi(m_1 m_2) = \operatorname{lcm}(\pi(m_1), \pi(m_2))$.

N17 [4★] Show that if $b \mid a$, then

$$\frac{a}{b} \bmod m = \frac{a \bmod bm}{b},$$

demonstrating that we can still divide even if the divisor is not coprime with m, at the cost of making the modulus larger. *Hint:* Let $a = bc$ for some integer c, and replace a with bc to get the equivalent statement $b(c \bmod m) = (bc \bmod bm)$, which is simply saying that multiplication distributes over mod. *Hint 2:* Use $x = \lfloor \frac{x}{y} \rfloor y + (x \bmod y)$.

N18 The goal of this task is to prove Lucas's theorem.

(a) [2★] Prove that

$$\binom{n}{r} = \sum_{i=0}^{k} \binom{k}{i} \binom{n-k}{r-i}.$$

Note the similarity to Non-Coding exercise 3b.

For $k = 1$, this is just Pascal's identity. Other special instances:

$$\begin{aligned}
\binom{n}{r} &= \binom{n}{r} \\
\binom{n}{r} &= \binom{n-1}{r} + \binom{n-1}{r-1} \\
\binom{n}{r} &= \binom{n-2}{r} + 2\binom{n-2}{r-1} + \binom{n-2}{r-2} \\
\binom{n}{r} &= \binom{n-3}{r} + 3\binom{n-3}{r-1} + 3\binom{n-3}{r-2} + \binom{n-3}{r-3} \\
&\dots
\end{aligned}$$

(b) [4★] Prove that for prime p,

$$\binom{n}{r} \equiv \binom{n-p}{r} + \binom{n-p}{r-p} \pmod{p}.$$

Hint: Use the previous result for $k = p$.

(c) [3★] Prove that

$$\binom{n_1 p + n_0}{r_1 p + r_0} \equiv \binom{(n_1 - 1)p + n_0}{r_1 p + r_0} + \binom{(n_1 - 1)p + n_0}{(r_1 - 1)p + r_0} \pmod{p}.$$

(d) [5★] Prove that if $0 \le n_0, r_0 < p$, then for $r_1 < 0$ or $r_1 > n_1$,

$$\binom{n_1 p + n_0}{r_1 p + r_0} \equiv 0 \pmod{p}.$$

(e) [5★] Prove that if $0 \le n_0, r_0 < p$, then for $r_1 = 0$ or $r_1 = n_1$,

$$\binom{n_1 \cdot p + n_0}{r_1 \cdot p + r_0} \equiv \binom{n_0}{r_0} \pmod{p}.$$

(f) [6★] Hence, prove Lucas's theorem: for $0 \le n_0, r_0 < p$,

$$\binom{n_1 p + n_0}{r_1 p + r_0} \equiv \binom{n_1}{r_1}\binom{n_0}{r_0} \pmod{p}.$$

Hint: Fix n_0 and r_0, and use induction and the previous results.

N19 [2★] Find $g(n)$, the number of simple undirected graphs with n labelled nodes.

N20 [5★] Show that the number of rooted labelled forests with n nodes is $(n+1)^{n-1}$. (This formula looks very familiar...)

N21 [5★] Show that $P_G(k)$ is a polynomial in k by using the deletion-contraction formula.

N22 Furthermore, prove the following properties of $P_G(k)$:

- [3★] $P_G(x)$ has degree exactly n, where n is the number of nodes.
- [3★] $P_G(x)$ is monic, i.e., the coefficient of x^n is 1.
- [5★] If the coefficient of x^{n-1} is a_{n-1}, then $-a_{n-1}$ is the number of edges of G.
- [5★] The coefficients of $P_G(k)$ alternate in sign.

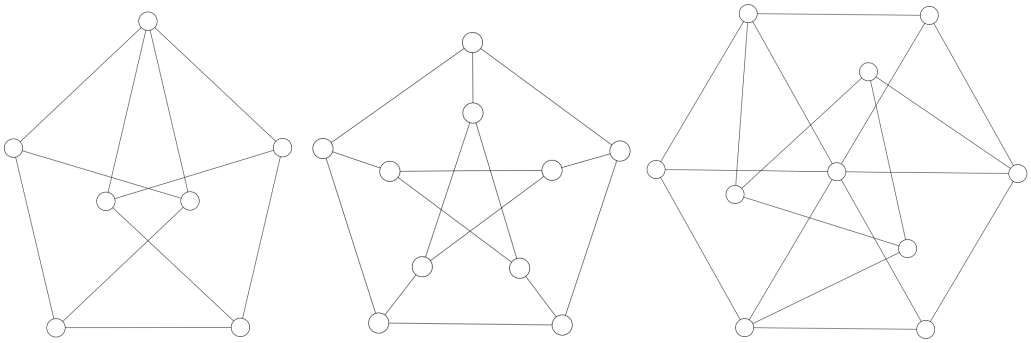
N23 Furthermore, prove the following properties of $P_G(k)$:

- [4★] If $n \ge 1$, then the constant term is 0.
- [3★] If $n \ge 1$, then the coefficient of x^1 is nonzero iff the graph is connected.
- [5★] If i is the smallest index such that the coefficient of x^i is nonzero, then G has exactly i connected components.

N24 [6★] Use deletion-contraction to prove that the chromatic polynomial of a cycle graph with n nodes is $(k-1)^n + (-1)^n(k-1)$.

N25 [4★] What does the previous formula say about the number of colorings of a cycle graph with $n = 1$ node? Shouldn't the answer be k for a singleton graph? Explain what's going on here.

N26 Find the chromatic number of each of the following graphs ([5★] each):



Explain your answers, please.

N27 [7★] Recall the following formula relating $g(n)$ and $g_{\text{conn}}(n)$:

$$g(n) = \sum_{k=1}^{n} \binom{n-1}{k-1} \cdot g_{\text{conn}}(k) \cdot g(n-k)$$

Compare it with the following formulas:

$$n! = \sum_{k=1}^{n} \binom{n-1}{k-1} \cdot (k-1)! \cdot (n-k)!$$

$$B_n = \sum_{k=1}^{n} \binom{n-1}{k-1} \cdot 1 \cdot B_{n-k}$$

In the second one, B_n refers to the nth Bell number, i.e., the number of ways to partition an n-element set. In other words, it is the sum of the nth row of the Stirling numbers of the second kind.

In the first one, $n!$ denotes the number of permutations of n elements, which can also be understood as the number of cycle partitions of n elements. In other words, it is the sum of the nth row of the Stirling numbers of the first kind.

Explain what each of the above formulas are saying combinatorially, and explain why they are so similar.

N28 [6★] Use the previous scheme to solve the following problem in $\mathcal{O}(n^2)$: Given n, find the number of simple undirected graphs of n nodes where each node has degree exactly 2.

N29 [6★] Derive a similar recurrence to count the number of unrooted labelled forests.

N30 [7★] Derive a similar recurrence to count the number of rooted labelled forests. Then, recall that the number of rooted labelled forests is $(n+1)^{n-1}$, from above. Similarly, the number of rooted labelled trees is easy to find. Substituting those gives you some kind of funny-looking equation that is nonetheless true. What's that equation?

N31 [5★] Let $d(n)$ be the number of positive divisors of n. Show that

$$\sum_{i=1}^{n} d(i) = \sum_{i=1}^{n} \left\lfloor \frac{n}{i} \right\rfloor.$$

N32 [3★] Show that $\sum_{i=1}^{n} \lfloor \frac{n}{i} \rfloor$ is the number of lattice points[11] in Quadrant I bounded by the hyperbola $xy = n$ and the coordinate axes. Are the points on the hyperbola itself included in this count? Are the points on the axes included?

[11] points with integer coordinates

N33 [8★] By exploiting the symmetry of the hyperbola $xy = n$, show how to compute $\sum_{i=1}^{n} d(i)$ in $\mathcal{O}(\sqrt{n})$ time. Check this against a brute force program and make sure you take care of all the edge cases!

N34 Recall that the totient of n, denoted $\phi(n)$, is the number of integers coprime to n in the range $[1, n]$.

Our goal in this task is to prove the equation

$$\sum_{d|n} \phi(d) = n.$$

- Remember (show) that $\gcd(x, n)$ is a divisor of n.
- [5★] For a fixed divisor g, find the number of integers x in $[1, n]$ such that $\gcd(x, n) = g$. The answer will be in terms of ϕ.
- [7★] Show why this implies that $\sum_{d|n} \phi(d) = n$.

N35 [4★] Show how the previous formula leads to a dynamic programming algorithm to compute $\phi(n)$. *Bonus:* Implement it!

N36 [3★] Suppose we plant a tree on every lattice point in the grid except the origin $(0, 0)$. We say that point (x, y) is *visible* if there is no tree in the straight line of sight from $(0, 0)$ to (x, y). Assume that trees are infinitely thin and perfectly vertical.

Show that (x, y) is visible if and only if $\gcd(x, y) = 1$.

N37 Define $f(n)$ to be the number of visible lattice points in $[1, n] \times [1, n]$.

- [4★] Let $f_g(n)$ be the number of lattice points (x, y) in $[1, n] \times [1, n]$ such that $\gcd(x, y) = g$. Show that $f_g(n) = f(\lfloor n/g \rfloor)$.
- [6★] Show that
$$n^2 = \sum_{g=1}^{n} f(\lfloor n/g \rfloor)$$
- [5★] Use the latter formula to find an algorithm that computes $f(n)$ using dynamic programming. *Bonus:* Implement it!
- [6★] Define $h(n)$ to be the number of visible lattice points in $[0, n] \times [0, n]$, excluding the origin. Show that
$$(n + 1)^2 - 1 = \sum_{g=1}^{n} h(\lfloor n/g \rfloor)$$

N38 [8★] Using techniques similar to those in problem **N33**, show how to compute $f(n)$ or $h(n)$ in $o(n)$ time (i.e., sublinear time).[12] *Hint:* Compress like terms. *Bonus:* Implement it!

N39 Determine the exact running time of the previous algorithm.

N40 [4★] Show how to compute $f(n)$ or $h(n)$ in $\mathcal{O}(n^{2/3}(\log \log n)^{1/3})$ time. *Bonus:* Implement it!

N41 [10★] We've previously asked about how to find the inverses of 0! until $n!$ in $\mathcal{O}(n + \log p)$ time. How do you find the inverses of all integers from 1 to n, mod p, in $\mathcal{O}(n)$ time?

[12] We say $f(n) = o(g(n))$ if for all $\varepsilon > 0$, $f(n) \leq \varepsilon g(n)$ for all sufficiently large n. Alternatively, $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$.

7.3 Coding problems

First-timers, solve as many as you can! Returning trainees, try to get [80★].

C1 [6★] Your program takes in a labelled, rooted tree with n vertices as input. We will enumerate each of its vertices exactly once in sequence, and we will say that this sequence is in topologically sorted order if each node (besides the root) occurs *after* its parents in the sequence. Write a program that counts the number of ways to topologically sort the given tree, modulo $10^9 + 7$, in $\mathcal{O}(n)$.

C2 [4★] Find $(10^{18} - 10^7 - 987)! \bmod (10^{18} + 10^7 + 987)$. You may use Wilson's theorem. You may use big integers/`__int128` because the numbers involved here are a bit too large.

C3 [6★] Find $\binom{115537603770}{11368469256} \bmod 77777777$.

C4 [3★] It can be shown that

$$\frac{69^{9876543210} - 69^{420}}{11}$$

is an integer. Find its last nine digits.

C5 [8★] Implement a program that takes an integer n and enumerates all rooted unlabelled trees of size n. It should run relatively quickly (i.e., waitable time) up to $n \leq 20$, hence an algorithm that enumerates labelled trees first and forgetting the labels (i.e., removing isomorphic trees) afterwards will be too slow since 20^{18} is huge.

C6 [8★] It is an amusing fact that every Gaussian integer (complex number with integer coordinates) can be uniquely represented in "base $i - 1$", where i denotes the imaginary number: $i^2 = -1$, and every digit is either 0 or 1. For example, $24 - 11i$ can be represented as 110010110011 in base $i - 1$ since

$$24 - 11i = (i-1)^{11} + (i-1)^{10} + (i-1)^7 + (i-1)^5 + (i-1)^4 + (i-1)^1 + (i-1)^0.$$

Write a program that takes a Gaussian integer $a + bi$ and outputs its (unique) base $i - 1$ representation.

C7 [8★] Implement an $\mathcal{O}(n \log n)$ algorithm that converts a tree into its corresponding Prüfer sequence and vice versa.

C8 [8★] Implement a program that takes an integer n and enumerates all unrooted labelled trees of size n by enumerating rooted trees and forgetting the root. Ensure that each distinct tree is enumerated exactly once.

C9 [2★] **Counting summations:** https://projecteuler.net/problem=76

C10 [3★] **Prime summations:** https://projecteuler.net/problem=77

C11 [6★] **Pandigital Fibonacci ends:** https://projecteuler.net/problem=104

C12 [5★] **Counting block combinations I:** https://projecteuler.net/problem=114

C13 [6★] **Counting block combinations II:** https://projecteuler.net/problem=115

C14 [6★] **Red, green or blue tiles:** https://projecteuler.net/problem=116

C15 [6★] **Red, green, and blue tiles:** https://projecteuler.net/problem=117

C16 [10★] **Exploring Pascal's triangle:** https://projecteuler.net/problem=148

C17 [12★] **Exploring Pascal's pyramid:** https://projecteuler.net/problem=154

C18 [14★] **Factorial trailing digits:** https://projecteuler.net/problem=160

C19 [10★] **Exploring the number of different ways a number can be expressed as a sum of powers of 2:** https://projecteuler.net/problem=169

C20 [20★] **Coloured Configurations:** https://projecteuler.net/problem=194

C21 [6★] **Squarefree Binomial Coefficients:** https://projecteuler.net/problem=203

C22 [9★] **The prime factorisation of binomial coefficients:** https://projecteuler.net/problem=231

C23 [7★] **Zeckendorf Representation:** https://projecteuler.net/problem=297

C24 [9★] **Primonacci:** https://projecteuler.net/problem=304

C25 [20★] **Binomial coefficients divisible by 10:** https://projecteuler.net/problem=322

C26 [10★] **A huge binomial coefficient:** https://projecteuler.net/problem=365

C27 [10★] **Maximum Integer Partition Product:** https://projecteuler.net/problem=374

C28 [6★] **(prime-k) factorial:** https://projecteuler.net/problem=381

C29 [7★] **Idempotents:** https://projecteuler.net/problem=407

C30 [13★] **Admissible paths through a grid:** https://projecteuler.net/problem=408

C31 [10★] **Consecutive die throws:** https://projecteuler.net/problem=423

C32 [28★] **Rigid graphs:** https://projecteuler.net/problem=434

C33 [40★] **Integers in base i-1:** https://projecteuler.net/problem=508

S1 [4★] **Ayoub and Lost Array:** https://codeforces.com/problemset/problem/1105/C

Note: The presented problem can actually be solved in $\mathcal{O}(\lg n)$ time, rather than the intended $\mathcal{O}(n)$ solution! However, finding this solution is beyond the scope of the concepts covered in this module, and so to be fair I will not give points for finding it. You're free to discuss it with me if you have any ideas, though, but it's really beyond what's covered here.

S2 [12★] **Animesh does not gift Malvika on her birthday:** https://www.codechef.com/problems/CHN10

S3 [12★] **An unavoidable detour for home:** https://codeforces.com/problemset/problem/814/E

S4 [2★] **Basic modular inverse:** https://www.hackerrank.com/contests/noi-ph-2019-preselection/challenges/basic-modular-inverse

S5 [5★] **Basic string counting:** https://www.hackerrank.com/contests/noi-ph-2019-preselection/challenges/basic-string-counting

S6 [4★] **Basic partition numbers:** https://www.hackerrank.com/contests/noi-ph-2019-preselection/challenges/basic-partition-numbers

S7 [7★] **Counting Perfect Subsequences:** https://www.hackerrank.com/contests/noi-ph-2019-preselection/challenges/p-string

S8 [7★] **Solve the Queries!:** https://www.hackerrank.com/contests/noi-ph-2019-preselection/challenges/solve-the-queries

S9 [9★] **Religious War Prevention:** https://www.hackerrank.com/contests/noi-ph-2019-preselection/challenges/2018-religious-war-prevention

S10 [7★] **Bananagrams:** https://www.hackerrank.com/contests/noi-ph-2019-preselection/challenges/2018-bananagrams

S11 [8★] **Clash of the Brainf***s:** https://www.hackerrank.com/contests/noi-ph-2019-preselection/challenges/clash-of-the-brainf---s

S12 [10★] **Carpet Game:** https://www.hackerrank.com/contests/noi-ph-2019-preselection/challenges/carpet-game

S13 [12★] **Dishes: How We Do It?:** https://www.hackerrank.com/contests/noi-ph-2019-preselection/challenges/dishes-how-we-do-it

S14 [12★] **Bisayaka Grid:** https://www.hackerrank.com/contests/noi-ph-2019-preselection/challenges/corrupted-grid

S15 [18★] **Help Your Friend Come Out:** https://www.hackerrank.com/contests/noi-ph-2019-preselection/challenges/help-your-friend-come-out

Acknowledgment

Thanks to Karl Pilario and Kevin Atienza, since a lot of this material was taken from their previous work!