NOI.PH Training: Math 1

Divisibility and Basic Counting

Carl Joshua Quines

Contents

1 Introduction

Almost all problems in competitive programming involve math in one way or another. Hence the common complaint that a problem is a "math problem". But by convention, when we talk about **math problems**, we're referring to a very specific subset of problems:

- **number theory** problems, which involve the integers, divisibility, factorization, modulo, Diophantine equations, multiplicative number theory, etc., and
- **combinatorics** problems, which involve counting, combinations, permutations, binomial coefficients, probability, game theory, etc.

In particular, *geometry* problems are usually considered separate from math problems, and *graph theory* problems are on a class of their own. Yes, these fields are very much math even with a strict definition of math, but in competitive programming, it's more useful to study these separately.

Unfortunately, it's these areas of math—number theory and combinatorics—that are most often neglected in a high school curriculum. And through some weird convention,[1] these are both discussed together, even if they should be thought of completely different things. But we'll follow this convention anyway.

Tip. I strongly recommend doing every exercise! All exercises, unless otherwise marked, are allowed to be discussed in Discord. This means that you can ask questions about them, ask for hints, and give hints to others. If you are stuck on an exercise; the right thing to do *might* be thinking about the problem more, but sometimes it could be getting help.

[1] Through some weird convention of NOI.PH, that is.

2 Proof

What is a proof? Why bother proving things? How do you prove theorems? To many students, mathematics is only finding numerical answers to problems. But mathematics is far richer than this—being able to *prove* that your answer is correct is often more important than the actual answer.

You probably already have an intuitive grasp of what a proof is and how it is done. We'll assume some familiarity with concepts in logic: the difference between "if", "only if", and "if and only if", what a converse or contrapositive is, and proofs by contradiction. If you need a refresher, feel free to look it up online or ask on Discord!

You'll hear the term **rigor** a lot when we talk about proofs: it refers to how formal we are when proving things. A completely rigorous proof is one that justifies *all* its steps and manipulations, making sure each step has its own justification. On the opposite end is an informal proof, which can sometimes just consist of giving the idea to the proof.

We won't be focusing too much on making our proofs fully rigorous in this module. But we *will* make sure that our proofs are actually correct. You may ask what's the point of proving something if we don't do it rigorously, or why we're even discussing proofs at all. Well...

2.1 Why learn proofs?

There are many good reasons to learn the proofs of standard results, as well as learning how to prove things yourself. While it's one thing to know that a result is true, it's another thing to know *why* a result is true, and it's often in this understanding where we get better at solving problems.

Here's a specific example. Why do we need to know how the sieve of Eratosthenes works, when we can just memorize the code and write it from memory every time we need it? Two big reasons:

1. Memorizing the algorithm is fault-prone. It's easy to make a mistake when typing something from memory, and I point out in the text several parts that are easy to mess up. In contrast, understanding how it works will make it easier to catch mistakes.
2. Knowing why the algorithm works will allow you to modify it for other uses. For example, a good understanding of how the sieve of Eratosthenes works can help you modify it to solve the related problem of identifying squarefree integers.

Of course, we won't try to prove *everything* that we discuss; otherwise it would take us forever. I've organized the text so as to present the proofs that will be most useful for competitive programming; other proofs that I think are less useful in this respect appear in the appendix.

2.2 Induction

A useful kind of proof technique is called **mathematical induction**. Suppose we wanted to prove a statement that's true for all positive integers, something like

> **Example 2.1.** The sum of the first n odd numbers is n^2.

We need to show that it's true for $n = 1$, then $n = 2$, then $n = 3$, and so on. The idea of induction is that we can rely on the proof for n to show that it's true for $n + 1$.

Specifically, an inductive proof consists of two parts:

- The *base case*, where you prove it for $n = 1$. For our example, the first odd number is 1 so the sum is indeed 1^2.
- The *inductive step*, where you show that if it's true for n, then it's true for $n + 1$. So assume that the sum of the first n numbers is n^2:

$$1 + 3 + \cdots + (2n - 1) = n^2.$$

But the $(n + 1)$st odd number is $2n + 1$, so adding $2n + 1$ to the sum would give us

$$1 + 3 + \cdots + (2n - 1) + (2n + 1) = n^2 + 2n + 1 = (n + 1)^2.$$

But this is the statement for $n + 1$! So we're done with our proof.

Why does this work? From the base case, we know it's true for $n = 1$. Using the inductive step, it's true for $n = 2$. Using the inductive step again, it's true for $n = 3$, and continuing like this, we can see it must be true for all n.

Often, the main difficulty in an inductive proof is the inductive step. In fact, if we're reasoning informally, we often write something like "the base case is obvious", because many times it really is obvious. **It's always important to mention the base case**, even if it's just with a comment like this; otherwise, the induction isn't complete.

Exercise 2.2. Prove that, for any n, $1 + 2 + \cdots + n = \frac{1}{2}n(n + 1)$.

Exercise 2.3. Prove that, for any n, $(1 + 2 + \cdots + n)^2 = 1^3 + 2^3 + \cdots + n^3$.

3 The integers

3.1 Divisibility

We say that a **divides** b (or b is **divisible** by a) if $a \neq 0$ and the ratio b/a is an integer. We write this as $a \mid b$, so

$$a \mid b \quad \iff \quad a \neq 0 \quad \text{and} \quad b = ak \text{ for some integer } k.$$

If a does not divide b we write $a \nmid b$. Some texts will require that $a > 0$, but we'll use the more general definition. Note some technicalities of this definition: 0 does not divide any integer, but yet every integer divides 0.

The numbers which divide an integer are called its **divisors**: for example, the divisors of 12 are 1, 2, 3, 4, 6, and 12, as well as -12, -6, -4, -3, -2, and -1. If we want to refer to only the positive divisors, we'll say **positive divisors**.

Exercise 3.1. List down the divisors of 20.

Exercise 3.2. Prove or disprove: If $a \mid b$, then $a \leq b$.

Exercise 3.3. For each $1 \leq k \leq 6$, what is the smallest positive integer with exactly k positive divisors?

3.1.1 Sensible division

We all know how to divide, find the remainder, and use the modulo function. In fact, when we're working with positive integers, we all agree with how these are defined. But weird things happen when we start considering negative integers:

Example 3.4. The way division and modulo are handled in C++ irritates me. Quick: what's `7 % 3`, `7 % -3`, and `-7 % 3`?

In C++, the definition of `a % b` is `a - (a/b)*b`. And the definition of `a / b` is to do the division with real numbers, and then drop everything after the decimal point. So `7 / 3` is `2`, `7 / -3` is `-2`, and `-7 / 3` is also `-2`.

Which looks okay. You get `7 % 3` as `7 - (7/3)*3`, which is `1`. Perfectly fine. And you get `7 % -3` as `7 - (7/-3)*-3`, which is `1`. But then you get `-7 % 3` as `-7 - (-7/3)*3`, which is `-1`, which is weird!

Why is -1 a value modulo 3? Shouldn't the modulo function give us the *remainder*, and isn't the remainder always positive? If this doesn't weird you out, hopefully this section will make it clear why this should be weird.

We all agree with how to find the quotient and remainder when we divide positive integers by positive integers. To extend to negative integers, we can do what C++ does, and simply *truncate towards zero*, or drop everything after the decimal point. But as illustrated, this doesn't really make sense.

Instead, there are two sensible ways to extend division to negative integers.

The first is **floor division**, which is what nice languages like Python do. This kind of division is what some computer scientists will prefer. For two integers a and b, the quotient is $\left\lfloor \frac{a}{b} \right\rfloor$, and the remainder is $a \bmod b = a - b \left\lfloor \frac{a}{b} \right\rfloor$. Basically, it does division with real numbers, and then *rounds down*. This is different from what C++ does, which simply truncates; in floor division, we always round *down*.[2]

The second is **Euclidean division**, or what is commonly called the division algorithm.[3] This kind of division is what number theorists and most mathematicians will prefer. It relies on the following theorem which we'll call the **division theorem**: for two integers a and b, where $b \neq 0$, there exists integers q and r such that

$$a = bq + r$$

and $0 \leq r < |b|$. Then the quotient is q and the remainder is r. This definition makes sense because the remainder is always positive, even if b is negative.

There are some good things about both approaches, but we'll mostly focus on Euclidean division. Unless otherwise stated, we'll work with Euclidean division because this is what most people are familiar with.

Exercise 3.5. What is the remainder when 7 is divided by -3? What is the remainder when -7 is divided by 3?

Exercise 3.6. Solve **Preface:** https://www.urionlinejudge.com.br/judge/en/problems/view/1837

3.2 Greatest common divisor

In around fourth grade, we learn about reducing fractions to lowest terms. The factor that we take out is what we call the **greatest common divisor**. We know that $12/18 = 2/3$ because we can divide the numerator and denominator by 6. The number 6 here is called the greatest common divisor of 12 and 18.

In more formal terms, we define the greatest common divisor of two integers to be the largest integer that divides them both:

$$\gcd(a, b) = \max \{k \mid k \mid a \quad \text{and} \quad k \mid b\}.$$

As a consequence of the above definition, for $n \neq 0$, we have $\gcd(0, n) = n$. We'll take the value of $\gcd(0, 0)$ as undefined.

Two integers with a gcd of 1 are called **relatively prime** or coprime. For example, 28 and 15 have a gcd of 1, so they're relatively prime.

Example 3.7. When we reduce a fraction to lowest terms, we take out the greatest common divisor of the numerator and denominator. So if a fraction is a/b, we replace its numerator with $a/\gcd(a, b)$ and its denominator with $b/\gcd(a, b)$, and then there are no more common factors. Thus, in general, $a/\gcd(a, b)$ and $b/\gcd(a, b)$ are relatively prime. This follows from a more

[2] For example, $k(a \bmod b) = (ka) \bmod (kb)$, with an almost direct proof.

[3] Which isn't actually an algorithm, so we'll call it Euclidean division, which is a much better name. But most books will call this theorem the division algorithm.

general law, $\gcd(ka, kb) = k \gcd(a, b)$, that you'll be asked to prove in an exercise.

Another familiar thing should be the **least common multiple**, which you again learn about in around fourth grade when you add fractions. For example, when adding $1/12$ and $1/18$, we first make them similar fractions and add them: $3/36 + 2/36 = 5/36$. The number 36 is called the least common multiple of 12 and 18.

In more formal terms, the least common multiple of two integers is the smallest *positive* integer that is divisible by both:

$$\operatorname{lcm}(a, b) = \min \{k \mid k > 0, \quad a \mid k \quad \text{and} \quad b \mid k\}.$$

We'll only leave this defined when a and b are both positive. It won't be of much consequence anyway, since we won't care about the lcm as much.

Exercise 3.8. What is $\gcd(96, 36)$? What about $\operatorname{lcm}(96, 36)$? What can you say about their product and the product of 96 and 36? Can you generalize?

3.2.1 Euclidean algorithm

You're probably familiar with the school method of computing greatest common divisors. For example, to compute $\gcd(84, 112)$, we factor $84 = 2^2 \cdot 3 \cdot 7$ and $112 = 2^4 \cdot 7$. Then we take the prime factors in common: in this case, two 2s and one 7. So the gcd is $2^2 \cdot 7 = 28$.

Unfortunately, this method of computing gcds does not scale; computing prime factors like this turns out to be quite a hassle, as we'll see later. It turns out to be much easier to compute the gcd using the following recurrence:

$$\gcd(a, b) = \gcd(b \bmod a, a), \qquad \text{for } a > 0.$$

Here $b \bmod a$ is the remainder when b is divided by a. The base case of this recurrence is the case that $a = 0$, but as we mentioned earlier, $\gcd(0, n) = n$. So for example,

$$\gcd(84, 112) = \gcd(28, 84) = \gcd(0, 28) = 28.$$

It turns out it is much easier to program this, like any other recurrence:

```
int gcd(int a, int b) {
  if (a == 0) return b;
  return gcd(b % a, a);
}
```

This is called the **Euclidean algorithm**. You can also write it iteratively, which is slightly faster:

```
int gcd(int a, int b) {
  while (a != 0) {
    int r = b % a; b = a; a = r;
  }
  return b;
}
```

Either way, it runs in $\mathcal{O}(\log \min \{m, n\})$, which you'll be asked to prove in an exercise.

Exercise 3.9. Note that $\gcd(-9, -12) = 3$. But what is the result when we call the above function with `gcd(-9, -12)`?

Most of the time we won't need to take gcds of negative numbers. But if you are, and this is an issue, an easy way to fix this is to just take the absolute value of `a` and `b`.

Exercise 3.10. Why do we know the Euclidean algorithm always stops?

Why does the recurrence work? The key is that any common divisor of m and n must also be a common divisor of $n \bmod m$ and m. Let's go back to Euclidean division; we have

$$n = mq + n \bmod m.$$

Suppose $d \mid n$ and $d \mid m$. Then d divides the left-hand side, so it must also divide the right-hand side. But you know that d divides mq, because $d \mid m$. So we can conclude that $d \mid n \bmod m$. We have showed that if n and m has a common divisor, it also divides $n \bmod m$.

Similarly, if we assume that $d \mid m$ and $d \mid n \bmod m$, we get that d divides the right-hand side, so it must also divide the left-hand side, which is n. This shows that if m and $n \bmod m$ have a common divisor, it must also divide n.

These two show that the common divisors of m and n must also be the same common divisors of $n \bmod m$ and m! So that gives us the recurrence.

Exercise 3.11. In Exercise 3.8, you should have observed that $\gcd(m, n) \operatorname{lcm}(m, n) = mn$. We won't ask you to prove this yet, but use it to express $\operatorname{lcm}(m, n)$ in terms of $\operatorname{lcm}(n \bmod m, m)$.

3.3 Linear Diophantine equations

Here we'll discuss how to solve the equation

$$ax + by = c$$

for all integers x and y, given integers a, b, and c. In general, an equation where we're looking only for integer solutions is called a **Diophantine equation**, and this particular case is known as the linear Diophantine equation.

Let's rule out some obvious conditions. We can quickly see that something like

$$4x + 6y = 3$$

isn't going to have any solutions x and y. We know this because 2 always divides the left-hand side, but $2 \nmid 3$. In particular, one *necessary* condition for x and y to exist in the first place is that

$$\gcd(a, b) \mid c.$$

And it turns out that is a *sufficient* condition. In other words, as long as we have the somewhat "obvious", somewhat "light" requirement that $\gcd(a, b) \mid c$, then there's a solution x and y![4]

[4]This happens quite a lot in math, where the "obvious" necessary condition turns out to be the only one we need. The existence of Eulerian cycles, or Hall's marriage theorem, and so on. I'm sure if you ask in Discord you'll get more examples.

Let's start with the fundamental such equation, or solving for x and y such that

$$ax + by = \gcd(a, b).$$

By slightly modifying the Euclidean algorithm, we can find such x and y. The idea is also recurrence, but turned into equation form: we use the solutions to a simpler equation to find the solutions of this one. In the same way that

$$\gcd(a, b) = \gcd(b \bmod a, b), \qquad \text{for } a > 0.$$

relates a, b, and $b \bmod a$, we want to relate the solutions of these two equations:

$$\begin{aligned} ax + by &= \gcd(a, b) \\ (b \bmod a)x' + ay' &= \gcd(b \bmod a, a). \end{aligned}$$

Consider the second equation, and recall how $b \bmod a$ is defined. By Euclidean division, we have $b = aq + b \bmod a$. Also, by the gcd recurrence, the right-hand side here is also $\gcd(a, b)$. So the second equation is

$$\begin{aligned} (b \bmod a)x' + ay' &= \gcd(b \bmod a, a) \\ (b - aq)x' + ay' &= \gcd(a, b) \\ a(y' - qx') + bx' &= \gcd(a, b). \end{aligned}$$

This is how the two equations are related! The main idea here is how the aq part gets absorbed in a, and how the b part becomes its own turn. Pay close attention to how the a gets pulled out, leaving y' and $-qx'$ behind, while bx' becomes its own term.

In particular, if you have a solution (x', y') to the second equation, you have a solution to the first equation, by taking $x = y' - qx'$ and $y = x'$. The base case of this is also the same as the base case of gcd, which is when $a = 0$.

Example 3.12. That was pretty abstract, so let's do an example. Suppose we're solving $5x + 17y = 1$ for x and y. Then, by applying the above recursion:

$$\begin{aligned} 5x + 17y &= 1 \\ 2x_1 + 5y_1 &= 1 \\ 1x_2 + 2y_2 &= 1 \\ 0x_3 + 1y_3 &= 1. \end{aligned}$$

And we know the last equation has the solution $x_3 = 0$ and $y_3 = 1$. We got 0 here from Euclidean division. In particular we divided 2 by 1:

$$2 = 1 \cdot 2 + 0 \implies 0 = 2 - 1 \cdot 2$$

where 2 is the quotient and 0 is the remainder. So we can replace 0 in the last equation with $2 - 1 \cdot 2$. This gives us

$$\begin{aligned} 0 \cdot 0 + 1 \cdot 1 &= 1 \\ (2 - 1 \cdot 2) \cdot 0 + 1 \cdot 1 &= 1 \\ 2 \cdot 0 - 1 \cdot 2 \cdot 0 + 1 \cdot 1 &= 1 \\ 1(-2 \cdot 0 + 1) + 2 \cdot 0 &= 1 \\ 1 \cdot 1 + 2 \cdot 0 &= 1. \end{aligned}$$

(If the above manipulations are a little confusing, keep going; it will be clearer later.) So this gives us a solution to the third equation. Now we replace 1, which we got from dividing 5 by 2:

$$5 = 2 \cdot 2 + 1 \implies 1 = 5 - 2 \cdot 2,$$

and making this replacement we get

$$\begin{aligned} 1 \cdot 1 + 2 \cdot 0 &= 1 \\ (5 - 2 \cdot 2) \cdot 1 + 2 \cdot 0 &= 1 \\ 5 \cdot 1 - 2 \cdot 2 \cdot 1 + 2 \cdot 0 &= 1 \\ 2(-2 \cdot 1 + 0) + 5 \cdot 1 &= 1 \\ 2 \cdot -2 + 5 \cdot 1 &= 1. \end{aligned}$$

Finally, we make one last replacement:

$$17 = 5 \cdot 3 + 2 \implies 2 = 17 - 5 \cdot 3,$$

so

$$\begin{aligned} 2 \cdot -2 + 5 \cdot 1 &= 1 \\ (17 - 5 \cdot 3) \cdot -2 + 5 \cdot 1 &= 1 \\ 5 \cdot 7 + 17 \cdot -2 &= 1. \end{aligned}$$

Exercise 3.13. Find a solution to $18x + 34y = 2$ using this algorithm.

So this shows us that the equation

$$ax + by = \gcd(a, b)$$

can always be solved for integers x and y. This result is known as **Bézout's lemma**,[5] and it turns out to be an incredibly useful result.

For example, it follows from Bézout's lemma that any common divisor of a and b also divides $\gcd(a, b)$. Our definition only tells us that the common divisor is *less than* $\gcd(a, b)$, but we get something stronger: it must also *divide* $\gcd(a, b)$.

3.3.1 Extended Euclidean algorithm

It's a bit of a hassle, however, to compute solutions in this manner. The difficulty stems from the fact that we have to go all the way forward to find the coefficients of each equation, then go all the way backward to find the original solutions. The **extended Euclidean algorithm** avoids this problem.

Rather than jumping into the code, it's easier to explain and remember it visually. We focus on the quotients in each step of the Euclidean algorithm. For example, with 5 and 17, the Euclidean algorithm looks like

$$\gcd(5, 17) = \gcd(2, 5) = \gcd(1, 2) = \gcd(0, 1).$$

[5] If you're unfamiliar with the word *lemma*, just think of it as synonymous to *theorem*. Strictly speaking though, lemmas are theorems which are used mainly to prove other, perhaps more important, theorems. But lemmas can still be interesting on their own right.

The quotients in each step are 3, 2, and 2, respectively. You then write them in a table like this:

		3	2	2
0	1			
1	0			

where the two leftmost columns are fixed. Then what we do is fill in the entries of the table inductively. In particular:

		$\cdots$	q	$\cdots$
$\cdots$	x'	x	$x' - qx$	$\cdots$
$\cdots$	y'	y	$y' - qy$	$\cdots$

is how you want to fill up the column under q. So for example, since $0 - 1 \cdot 3 = -3$ and $1 - 0 \cdot 3 = 1$, the first column should be

		3	2	2
0	1	-3		
1	0	1		

Then as $1 - (-3) \cdot 2 = 7$ and $0 - 1 \cdot 2 = -2$, the next column is

		3	2	2
0	1	-3	7	
1	0	1	-2	

Finally, the last column is $-3 - 7 \cdot 2 = -17$ and $1 - (-2) \cdot 2 = 5$, so

		3	2	2
0	1	-3	7	-17
1	0	1	-2	5

To read off the answer, then, you look at the second-to-the last column. It has 7 and -2. This means $5 \cdot 7 + 17 \cdot -2 = 1$.

Exercise 3.14. Find a solution to $18x + 34y = 2$ using the extended Euclidean algorithm.

This should feel like black magic. Sure, we're doing a bunch of algebraic manipulations, which is what we did in the naive way to solve this Diophantine equation. But the difference is that last time, our manipulations relied on going backwards. Here, we're finding the answer without ever needing to go backwards, which should feel very magical, or at least very cool.

There are a couple of more things to be observed in the above algorithm:

- The last column has -17 and 5. It turns out that, up to sign, you'll always end up with the original two numbers in the last column after dividing by their gcd. (Verify this for Exercise 3.14!) This is a useful sanity check when doing the algorithm by hand.
- Consider the 2×2 matrix formed by the entries in two adjacent columns. For example, taking the third and fourth columns gives

$$\begin{bmatrix} -3 & 7 \\ 1 & -2 \end{bmatrix}.$$

  Then the products of the diagonals differ by 1. Here, the products of the diagonals are 6 and 7. If you take the fourth and fifth columns, the products of the diagonals are 35 and 34. Another way to say this is that the determinant is ± 1.

- More than that, observe what happens when we plug in the entry (x, y) in each column to $ax + by$. We get

$$\begin{aligned} 5 \cdot 0 + 17 \cdot 1 &= 17 \\ 5 \cdot 1 + 17 \cdot 0 &= 5 \\ 5 \cdot -3 + 17 \cdot 1 &= 2 \\ 5 \cdot 7 + 17 \cdot -2 &= 1 \\ 5 \cdot -17 + 17 \cdot 5 &= 0. \end{aligned}$$

  These are the remainders in each step in the Euclidean algorithm! It turns out that this is the insight used to prove that the extended Euclidean algorithm works. In my opinion, the proof is kind of boring, so I'll use my author card and leave it to the reader:

Exercise 3.15. Prove that the extended Euclidean algorithm works using the above observation. You want to show, by induction, that the column (x, y) produced at each step gives $ax+by$ as the remainder in that step.

There are several ways to implement this; we'll show two. You don't want to memorize either, but once you understand how the above works, you should be able to rewrite it from memory. First, the extended Euclidean algorithm naturally lends itself to a recursion:

```
pair<int, int> xgcd(int a, int b, int xp = 1, int yp = 0, int x = 0, int y =
↪  1) {
  if (a == 0) return {xp, yp};
  int q = b / a;
  int r = b % a;
  return xgcd(r, a, x, y, xp - q*x, yp - q*y)
}

```

You'd call this by something like `tie(x, y) = xgcd(5, 17);` and after that `x` would be `7` and `y` would be `-2`. Yeah, this is how you use the function `tie`, and this is the morally correct C++ way to return two outputs. You can also convert this to something iterative, which would again be slightly faster:

```
pair<int, int> xgcd(int a, int b) {
  int xp = 0, x = 1;
  int yp = 1, y = 0;
  while (a != 0) {
    int q = b / a;
    int r = b % a; b = a; a = r;
    int nx = xp - q*x; xp = x; x = nx;
    int ny = yp - q*y; yp = y; y = ny;
  }
  return {xp, yp};
}
```

Exercise 3.16. Convince yourself that both of the above programs works. In both programs, the variables `xp`, `x`, `yp` and `y` represent the same thing. What do they represent? In the iterative version, what does line 6 do?

Exercise 3.17. In either program, what happens if we change the return statement to `return b` instead? What would the function return in this case?

3.3.2 General case

We now deal with the linear Diophantine equation

$$ax + by = c$$

in full generality. We previous established that $\gcd(a, b) \mid c$ is necessary. In other words, if $\gcd(a, b) \nmid c$, then we instantly know that there aren't any solutions.

We now want to show that if $\gcd(a, b) \mid c$, then there are solutions. In fact, by Bézout's lemma, we know

$$ax + by = \gcd(a, b)$$

has a solution (x, y). Now take $d = \frac{c}{\gcd(a,b)}$, and multiply both sides by d to get

$$a(dx) + b(dy) = c.$$

We've now created a solution (dx, dy) to the equation! So we've shown that the condition $\gcd(a, b) \mid c$ is equivalent to the equation having solutions.

Now, we found one particular solution, but there are many more. Recall these two equations (or really, the first one):

$$5 \cdot 7 + 17 \cdot -2 = 1$$
$$5 \cdot -17 + 17 \cdot 5 = 0.$$

Now by adding the two equations together, we get

$$5 \cdot -10 + 17 \cdot 3 = 1.$$

So the solution $(7, -2)$ gave us the solution $(-10, 3)$. In fact, we can add the second equation again to give us the solution $(-27, 8)$. We can also go backwards and *subtract* the second equation from the first, which gives us the solution $(24, -7)$, and so on.

Example 3.18. Consider $5x + 17y = 49$. From the solution $(7, -2)$ to $5x + 17y = 1$, we get the solution $(343, -98)$ to this equation. By the above reasoning, we see that $(343 - 17 \cdot 20, -98 + 5 \cdot 20)$ or $(3, 2)$ is also a solution.

Exercise 3.19. Suppose (x', y') is a solution to $ax + by = c$. Describe *all* the solutions to this Diophantine equation. Be careful with the case $\gcd(a, b) \neq 1$! (It takes quite a bit of cleverness to prove that there aren't any other solutions; you don't need to do this.)

Exercise 3.20. Find all *positive* integer solutions to $5x + 17y = 387$. There are five of them. No, don't just loop over all possible pairs (x, y) with a computer; use the previous exercise.

Exercise 3.21. How many positive integer solutions are there to $69x + 1337y = 420420420$?

3.4 Primes

A positive integer p is called **prime** if it has two positive divisors, namely 1 and p. For the rest of this module, I'll use p to refer only to primes.[6] By the definition, 1 isn't a prime, so the list of primes looks like

$$2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, \ldots$$

Numbers which have more than two divisors are called **composite**. For example, 8 is composite, and so is $91 = 7 \cdot 13$. Every positive integer except for 1 is either prime or composite, but not both. The number 1 is neither prime nor composite; it's a special case.

3.4.1 Prime factorization

The reasons primes are so important is something called the **fundamental theorem of arithmetic**. You're probably familiar with what it states: every positive integer can be written as a *unique* product of primes, up to ordering. For example, $12 = 2 \cdot 2 \cdot 13$ or $11111 = 41 \cdot 271$.

Proving that this product exists is pretty simple. If n is not a prime then it has a divisor d such that $1 < d < n$, so write $n = de$. By induction, we know that d and e can be written as products of primes, and putting them together, we see that n can be too. But proving that this product is unique takes a little bit more effort, and we won't go into that here.

Example 3.22. You may have taken unique prime factorization for granted since your first introduction to it in third grade or so. The statement actually has substance: it's something that not all "integer-like" sets of numbers have.

A simple example is the set of all numbers $a + b\sqrt{10}$ where a and b are integers. Clearly we can add numbers, and a little bit of algebra should convince you we can multiply numbers here too. And you can call a number "prime" if it has anything other than the obvious divisors. But then

$$6 = 2 \cdot 3 = (4 + \sqrt{10})(4 - \sqrt{10}),$$

and you can check that in this system, 2, 3, $4 + \sqrt{10}$, and $4 - \sqrt{10}$ are all "prime". So unique prime factorization in the integers is actually a pretty deep fact.

For now, we'll accept unique prime factorization without question as something that exists. As we observed, if a number n is not a prime, then there's a divisor d such that $1 < d < n$ and $n = de$. In fact, if we require $d < e$, then we must know that $d \leq \sqrt{n}$.

Exercise 3.23. Convince yourself that this is true: if a positive integer n is not a prime, then

[6] If you see me use p for something else, tell me!

it has a divisor between 2 and $\sqrt{n}$. So taking its contrapositive, we know that if n doesn't have a divisor between 2 and $\sqrt{n}$, it must be prime.

This suggests the following algorithm, called **trial division**:

```
bool isprime(int n) {
  if (n < 2) return false;
  for (int x = 2; x*x <= n; x++) {
    if (n % x == 0) return false;
  }
  return true;
}
```

Exercise 3.24. There's a sizable constant factor from the test `x*x <= n`. Modify the code (or at least see *how* to modify it) to replace this part with something faster.

This is $\mathcal{O}(\sqrt{n})$. With some care, we can slightly modify the above algorithm to factorize numbers. Two important points to remember, which are easy to forget in my experience:

- Primes can divide a number multiple times, so you want to make sure to keep dividing until it doesn't divide it any more.
- If, after checking up to $\sqrt{n}$, the remaining number is greater than 1, it must be a prime and we should add it as a prime factor.

Keeping these in mind, we can write something like this:

```
vector<int> f;
void factor(int n) {
  for (int x = 2; x*x <= n; x++) {
    while (n % x == 0) {
      n /= x;
      f.push_back(x);
    }
  }
  if (n > 1) f.push_back(x);
}
```

Exercise 3.25. In the above code, we're still testing `x*x <= n`, but note that we're also making the value of `n` smaller while we're doing this. How do we know that we're not missing any prime factors due to this?

3.4.2 Sieve of Eratosthenes

I'm not sure how well-known this is, but when I was in third grade, I learned about the **sieve of Eratosthenes** from school. It's a very old algorithm for finding all primes less than a specified number, and the idea is similar to trial division: If a number n has a divisor between 2 and $n-1$, we know it must be composite. If it *doesn't*, then we know it must be prime.

The difference is that the sieve of Eratosthenes takes the reverse perspective. Instead of considering the divisors of each number, we're considering the multiples instead. *This is a general trick:* by stepping back and considering the whole picture, we can often reduce redundant computation.

It goes like this. First you start with a list of the integers, say from 2 to 30:

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

We're going to process from left-to-right, and we're going to cross out integers that we know are composite. We start with 2. It's not crossed out, so we know it's a prime. Then we cross out all the multiples of 2:

	2	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~
11	~~12~~	13	~~14~~	15	~~16~~	17	~~18~~	19	~~20~~
21	~~22~~	23	~~24~~	25	~~26~~	27	~~28~~	29	~~30~~

We move to 3. It's not crossed out, so we know it's a prime. So we cross out all the multiples of 3:

	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
~~21~~	~~22~~	23	~~24~~	25	~~26~~	~~27~~	~~28~~	29	~~30~~

We move to 4. It's crossed out, so it's composite. Then we move to 5. It's not crossed out, so we know it's a prime. We cross out all the multiples of 5:

	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
~~21~~	~~22~~	23	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~	29	~~30~~

We move to 6. It's crossed out, so it's composite. Then we move to 7. It's not crossed out, so we know it's a prime. We then cross out all the multiples of 7, and so on. By repeating this, we're left with the prime numbers as not crossed out. A quick implementation:

```
1  const int n = 100;
2  bool isprime[n];
3  for (int i = 2; i < n; i++) isprime[i] = true;
4
5  for (int i = 2; i < n; i++) {
6    if (!isprime[i]) continue;
7    for (int j = 2*i; j < N; j += i)
8      isprime[j] = false;
9  }
```

At the end of the algorithm, `isprime[i]` contains whether i is a prime or not, for $i \geq 2$. We can easily see that the algorithm is $\mathcal{O}(n^2)$. In fact, the inner loop in line 7 here is executed only n/i times instead of n times, and so the running time is upper bounded by

$$\frac{n}{2} + \frac{n}{3} + \cdots + \frac{n}{n}.$$

One of the exercises is to prove that this sum is actually in $\mathcal{O}(n \log n)$, and not just $\mathcal{O}(n^2)$. In fact, since the inner loop is only run when i is prime, the complexity is even $\mathcal{O}(n \log \log n)$; the proof is quite advanced and we're not going to do so here.

Exercise 3.26. One variant for line 7 is `for (int j = 2; j*i < N; j++)`. Convince yourself that this is slower than the loop presented above.

Exercise 3.27. As we noted in trial division, to check if n is a prime we only need to check its divisors up to $\sqrt{n}$. Make a suitable modification to the algorithm to get something closer to $\mathcal{O}(n + \sqrt{n} \log n)$ instead.

In fact, we can use the sieve of Eratosthenes to get something better than trial division. To test if n is a prime, we only need to divide by the *primes* up to $\sqrt{n}$. Similarly, we can speed up prime factorization a bit with this trick as well. But if you have $\mathcal{O}(n)$ space, you can use a modification of the sieve to make this faster:

Exercise 3.28. A common modification to the sieve of Eratosthenes is, instead of just storing whether a number is prime, also store its largest prime factor. Modify the above code to do that. With such an array, we can find the prime factorization of a number n in $\mathcal{O}(\log n)$.

Exercise 3.29. Here are some modifications to the sieve of Eratosthenes. For each one, describe the class of numbers that it produces:

1. Replace line 7 with `for (int j = i*i; j < N; j += i)`.
2. Replace line 7 with `for (int j = i*i; j < N; j *= i)`.
3. Replace line 8 with `isprime[j] ^= true;`.
4. Replace line 8 with `if (j/i % i != 0) isprime[j] = false;`.
5. Replace `isprime` with an array of integers, initially set to 0. Replace line 6 with `if (isprime[i] > 0) continue;`. Then replace line 8 with `isprime[j] += 1;`. What are the numbers i with `isprime[i]` equal to 2?
6. Replace `isprime` with an array of integers, initially set to 1. Replace line 6 with `if (isprime[i] != 1) continue;`. Then replace line 8 with `isprime[j] *= i;`. What are the numbers i with `isprime[i]` equal to 6?

3.5 Modulo

Many programming problems will ask you to find the final answer **modulo** n. This operation should be familiar intuitively; it's the `%` operation in C++ and Python that we discussed earlier, and what we denote using mod.

This is often done because the final answer is too large to store in an `int` or `long long`. Instead of actually computing the answer, which would cause integer overflow, we can instead do all our operations modulo n and output that instead.

From a problem-setting perspective, the chance that a contestant gets the correct answer modulo n, assuming they guess randomly, is $1/n$. For large n, this is small. It's an easy way to confirm that the contestant knows how to compute the correct answer, without asking them to compute the correct answer themselves.

We'll use a convenient notation for using mod with entire equations:

$$a \equiv b \pmod{m} \qquad \iff \qquad a \bmod m = b \bmod m.$$

For example, $12 \equiv -3 \pmod 5$ because $12 \bmod 5 = 2$ and $-3 \bmod 5 = 2$ as well. Another way of defining this notation is using the difference of the two numbers:

$$a \equiv b \pmod{m} \qquad \iff \qquad m \mid (a - b).$$

This definition is often more convenient to use when writing proofs. We won't be doing that many proofs in this section since most of them are kind of straightforward, so it doesn't really matter which one you use because:

Exercise 3.30. Convince yourself that these two definitions are equivalent.

As an aside, the more *morally correct* way of thinking of modulo is as a quotient of groups. For example, when we're considering this modulo 5, we're basically "treating all the multiples of 5 as zero, and then wrapping around." Or, you're "ignoring a difference of 5." So you're setting

$$\cdots \equiv -10 \equiv -5 \equiv 0 \equiv 5 \equiv 10 \equiv \cdots \pmod 5.$$

You want addition to work the same way, so for example, if we add 1 to everything, we get

$$\cdots \equiv -9 \equiv -4 \equiv 1 \equiv 6 \equiv 11 \equiv \cdots \pmod 5,$$

as we expect. This more *spiritual* perspective is what people refer to when they use modulo in a sentence like, "I would date you, modulo the fact that I don't like you." Here, the word *modulo* is used to mean something like "ignoring".

3.5.1 Everybody's favorite prime

A typical presentation of modulo would give you rules like

$$a \equiv b \quad \text{and} \quad c \equiv d \implies a + c \equiv b + d \pmod m.$$

But I want to be very practical here. Let's talk about doing math modulo everybody's favorite prime, $10^9 + 7$, the most important prime for programming competitions, and how to do stuff with it.

Your solution starts with a line like `#declare MOD 1'000'000'007` or preferably `const long long MOD = 1'000'000'007;`. Be very careful with the number of zeroes here. Adding the apostrophes helps: it's a feature in C++ beginning with C++14.

First, we change all our additions and subtractions to be done modulo `MOD` instead. Instead of writing something like `(a + b) % MOD`, we can write `(a % MOD + b % MOD) % MOD` instead. The idea is that we can take each number modulo `MOD` *before* adding them, and this is useful to prevent overflow. Similarly, we can write `(a % MOD - b % MOD) % MOD` as well. Why is this true? Well, here's a similar question:

Exercise 3.31. What's the last digit of $7854778548 + 6523665419 + 7945612356$?

You don't need to compute the whole sum: all you need to do is take the last digit of each addend and add them together. It's $8 + 9 + 6$, so you know the last digit must be 3. Well, "taking the last digit" is the same as finding something modulo 10. But the base doesn't really

matter here, so you'd expect that the same trick will work no matter what modulo you use, and you'd be right.

More importantly, `((a % MOD) * (b % MOD)) % MOD`. Taking the modulo *before* multiplying is very important, since otherwise, we can get overflow. It's also important that a **and** b **here should be** `long long` and not just `int`: even if the final answer fits an `int`, it's possible that the result of `(a % MOD) * (b % MOD)` doesn't fit in an `int`.

Finally, you want to find the final answer: finding the value of x modulo `MOD`. Now you might think this is as simple as `x % MOD`, but as we discussed earlier, the `%` operation in C++ has some quirks when dealing with negative numbers. In particular, if x is negative, then `x % MOD` is also negative, which we don't want – we always want to **force a positive modulus**.

So before outputting your final answer x, do something like `((x % MOD) + x) % MOD`. Always remember to force positive modulus!

3.5.2 Fast exponentiation

And now, we rant about `pow`, another horrible thing in C++.

Example 3.32. In C++ and Python, the power operation is not `^`. This is bitwise XOR, so if you do `2^3`, you get the result of `1`. Unlike C++, Python *does* have a good power operation, which is `**`. So `2**3` gives the result of `8`.

You might think, *oh, but C++ does have a power function,* `pow`! Well, `pow` actually works in *floating point*, and if you should know anything about floating point, you know to avoid it whenever possible. Because although stuff like `int x = pow(5, 3);` gives you the sensible `125`, something like `long long x = pow(5, 25);` gives you

$$298023223876953152,$$

which is clearly *not* a power of 5, it doesn't even end with 5. The solution? Write our own power functions.

The naive way to write exponentiation would be a loop. We're going to assume you're taking modulo `MOD`, because these numbers get really big really quickly:

```
1  using ll = long long;
2
3  ll modpow(ll b, ll e) {
4    ll res = 1;
5    for (ll i = 1; i <= e; i++) {
6      res = (b * res) % MOD;
7    }
8    return res;
9  }
```

You call this with something like `modpow(5, 25)`. Note that in line 6, we don't have to write `res % MOD` because `res` is always reduced after each loop. Of course, it's possible that b isn't, so you might need to call `modpow(b % MOD, e)` instead.

Exercise 3.33. Why can't we replace line 6 with `res *= b % MOD`?

The problem with this approach is that it's $\mathcal{O}(e)$, which can be rather slow; often we want to find it faster than that. We need to do something called **fast exponentiation**. The trick is *repeated squaring*. Observe that

$$b^{2e} = b^e \cdot b^e \quad \text{and} \quad b^{2e+1} = b^e \cdot b^e \cdot b.$$

Here, we can *reuse* the value of b^e instead of having to compute it twice. That way, we can compute it in $\mathcal{O}(\log e)$ instead, by doing something recursive:

```
using ll = long long;

ll modpow(ll b, ll e) {
  if (e == 0) return 1;
  ll res = modpow(b, e/2);
  res = (res * res) % MOD;
  if (e % 2 == 1) res = (b * res) % MOD;
  return res;
}
```

We can also write it iteratively to make it slightly faster:

```
using ll = long long;

ll modpow(ll b, ll e) {
  ll res = 1;
  for (; e > 0; e /= 2) {
    if (e % 2 == 1) res = (b * res) % MOD;
    res = (res * res) % MOD;
  }
  return res;
}
```

By the way, in Python, the function `modpow(b, e)` is a default function: it's `pow(b, e, MOD)`. And yes, it uses fast exponentiation; you can view the source at https://github.com/python/cpython/blob/master/Objects/longobject.c#L4316.

Exercise 3.34. Convince yourself that `modpow` runs in $\mathcal{O}(\log e)$.

3.6 Big integers

I'll write a few words on implementing big integers, and leave the actual implementation as an exercise.

Some problems require supporting **arbitrary-precision arithmetic**. This means that we want our integers to be of arbitrary size, and not just be limited by `int` or `long long`. Java and Python support this, but sometimes these languages aren't available, or we need to use C++ for speed. In the rare occasion that this happens, it would be helpful to know how exactly to implement this.

From a high-level perspective, you represent numbers with an array of digits, and then pro-

gram the algorithms you were taught in first and second grades.[7] To make the implementation easier, we often represent the digits of a number from right to left. For example, if we're working in base 10, we can represent the number 6429 with the array `{9, 2, 4, 6}`, and leave the rest of the entries zero.

Here's pseudocode for an addition algorithm in base 10:

```
def add(x, y):
    # digits(x) returns the number of digits of x
    d = max(digits(x), digits(y))
    w = [0, 0, ..., 0] # of length d + 1
    for i in 0, 1, 2, .., d - 1:
        w[i] = x[i] + y[i]
        if w[i] > 10:
            w[i] -= 10
            w[i+1] += 1
    return w
```

Addition, subtraction, and comparison should be relatively straightforward to implement. Multiplication and modulo are harder, but still doable. You'll be asked to implement these five as an exercise. Division is much harder, but again, the easiest way really is in analogy to the pencil-and-paper algorithm we were taught at school.

[7] Revisiting mathematics taught to us in elementary seems to be a theme of this module.

4 Counting

In this section we learn how to count.

Example 4.1. But I already know how to count: $1, 2, 3, 4, 5, \ldots$

Great! We already know how to count. See you next week.

Levity aside, you probably have some idea of the kinds of questions we're considering. How many positive integers less than 100 are divisible by 3 or 5? How many ways are there to arrange six objects in a row? How many different ways are there to rearrange the letters of the word `BANANA`?

In the end, **counting is just arithmetic**. There will be lots of different and fancy tricks that we're going to discuss, but the underlying ideas will be the same. There are only a handful of universal principles, and applying these principles just consists of doing some arithmetic.

4.1 Counting is bijection

Let's start counting!

Exercise 4.2. How many numbers are in the list $1, 2, 3, \ldots, 27$?

There are 27 numbers. That was pretty easy!

Exercise 4.3. How many numbers are in the list $5, 6, 7, \ldots, 31$?

No, there aren't $31 - 5 = 26$ numbers. We construct a new list with **the same amount of numbers** by subtracting 4 from each one:

$$\begin{array}{ccccc} 5 & 6 & 7 & \cdots & 31 \\ -4 & -4 & -4 & \cdots & -4 \\ \hline 1 & 2 & 3 & \cdots & 27 \end{array}$$

And this is the previous list, so there are 27 numbers.

Exercise 4.4. How many numbers are in the list $15, 18, 21, \ldots, 69$?

We use the same trick! First, let's divide everything by 3, to make a new list with **the same amount of numbers**: $5, 6, 7, \ldots, 23$. But we know how to count this list! There are 19 numbers here.

Exercise 4.5. How many four-digit positive integers are perfect squares?

The list is $1024, 1089, 1156, \ldots, 9801$. But note that we can also write this list as $32^2, 33^2, 34^2, \ldots, 99^2$. And there are the same amount of numbers in the list $32, 33, 34, \ldots, 99$, which has 68 numbers.

What we've been doing is taking a **bijection**. Given something we want to count, we find something else that has the same amount of items, and count that instead. We'll explore this more later on, but first, some problems:

Exercise 4.6. How many numbers are in the list $147, 144, 141, \ldots, 39$?

Exercise 4.7. How many three-digit positive integers are divisible by 7?

Exercise 4.8. How many sets of four consecutive integers have product less than 100 000?

4.2 Counting is addition and subtraction

Now we move on to adding and subtracting. You're likely familiar with drawing Venn diagrams and using them to solve problems like these:

Exercise 4.9. In your classroom, 20 people like chocolates, 15 people like sharing, and 10 people like both. How many people in your classroom like sharing or chocolates (or both)?

Here, the answer is $20+15-10 = 25$. This is known as PIE, or the **Principle of Inclusion–Exclusion**. It's often stated in this form:

$$|A \cup B| = |A| + |B| - |A \cap B|.$$

Here, A and B are sets, and $|A|$ is the number of elements in set A. In this problem, our two sets are "people who like sharing" and "people who like chocolates". Written out in words, this is just

$$|\text{like sharing or chocolates}| = |\text{like sharing}| + |\text{like chocolates}| - |\text{like both}|.$$

Exercise 4.10. How many integers between 100 and 200, inclusive, are divisible by 3 or 5?

By PIE, we need to count the number of integers that are divisible by 3, the number of integers that are divisible by 5, and the number of integers that are divisible by both. The ones divisible by 3 are

$$102, 105, 108, \ldots, 198,$$

and there are 33 of these. (Check that!) The ones divisible by 5 are

$$100, 105, 110, \ldots, 200,$$

and there are 21 of these. (Check that too!) The ones divisible by 15 are

$$105, 120, 135, \ldots, 195,$$

and there are 7 of these. (Check that as well!) So our final answer is, by PIE, $33 + 21 - 7 = 47$ integers.

Exercise 4.11. Tim has 27 dogs. Of these, 14 have short hair, 11 are puppies, and 5 are long-haired adult dogs. How many of the puppies have short hair?

Exercise 4.12. Solve **Multiples of 3 and 5:** https://projecteuler.net/problem=1. Yes, you can use a computer, but can you solve it by hand?

Exercise 4.13. How many integers between 1 and 30 000, inclusive, are divisible by 4, 6, or 10?

4.3 Counting is multiplication

The next principle is multiplication. We use multiplication to count a series of independent events.

Exercise 4.14. Suppose I have 3 different shirts and 5 different pairs of pants. How many outfits can I make using one shirt and one pair of pants?

Suppose the shirts are labeled S_1, S_2, S_3, and that the pants are labeled P_1, P_2, P_3, P_4, P_5. Then we can represent all possible choices using a table:

	P_1	P_2	P_3	P_4	P_5
S_1	S_1P_1	S_1P_2	S_1P_3	S_1P_4	S_1P_5
S_2	S_2P_1	S_2P_2	S_2P_3	S_2P_4	S_2P_5
S_3	S_3P_1	S_3P_2	S_3P_3	S_3P_4	S_3P_5

We see there are 15 choices. We have 3 choices of shirts (corresponding to choosing a row), and for each choice of shirt, we have 5 choices of pants (corresponding to choosing a column). This gives $3 \cdot 5 = 15$ outfits.

Exercise 4.15. Suppose I have 3 shirts, 5 pairs of pants, 4 pairs of socks, and 6 pairs of shoes. How many outfits can I make using one shirt, one pair of pants, one pair of socks, and one pair of shoes?

We tackle a problem like this in steps:

- There are 3 choices of shirts to wear.
- For each of the 3 choices of shirts, we can pick a pair of pants in 5 ways, making $3 \cdot 5 = 15$ ways to pick both.
- For each of the 15 outfits so far, we can choose a pair of socks in 4 ways, so there are $15 \cdot 4 = 60$ ways. The reasoning here is the same as the previous problem.
- Finally, for each of the 60 outfits so far, we can a pair of shoes in 6 ways, so there are $60 \cdot 6 = 360$ ways in total.

In other words, there are $3 \cdot 5 \cdot 5 \cdot 6 = 360$ different ways we can do this. We can think of this as "3 choices for the first slot, 5 choices for the second slot, 5 choices for the third slot, and 6 choices for the last slot."

Note that the events are **independent**: that means that each decision doesn't depend on the other decisions. Whatever shirt we wear doesn't restrain what kind of pants we wear, or anything else. None of the choices affects any other choice, so they're all independent.

Exercise 4.16. We have 4 cubes in a row, and we want to color each one of 5 colors: navy, orange, indigo, purple, and hot pink. How many ways are there to color them such that no two adjacent cubes have the same color?

There are 5 choices of colors for each cube, so it's $5 \cdot 5 \cdot 5 \cdot 5 = 625$. Right? Not really: this doesn't take into account the possibility that two cubes can have the same color.

Instead, suppose we pick one of the 5 choices of colors for the first cube. Then there are only 4 possible choices of colors for the second cube, since we can't choose the same color we picked for the first cube. Then for the third cube, there are another 4 possible choices, and for the fourth cube, there are another 4 possible choices.

In total, that's $5 \cdot 4 \cdot 4 \cdot 4 = 320$. Note that although the choices themselves aren't independent, the *number* of choices is, and that's what matters.

Exercise 4.17. In how many ways can I arrange 5 books on a shelf?

Again, it isn't just $5 \cdot 5 \cdot 5 \cdot 5 \cdot 5$, since once we use a book, we can't use it any more.

So suppose we pick one of the 5 books for the first spot. Then there are 4 remaining books, so there are 4 remaining choices for the second spot. Then there are 3 remaining choices for the third slot, 2 remaining choices for the last slot, and 1 remaining choice for the slot.

The answer is $5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$. Again, although the choices aren't independent, the *number* of choices is.

Exercise 4.18. How many license plates consist of 3 letters, followed by 2 even digits, followed by 2 odd digits?

Exercise 4.19. Suppose I have 6 different books, 2 of which are programming books. In how many ways can I arrange them in a row if I want a programming book at both ends of the row?

4.3.1 Permutations

In fact, this comes up often enough that we have a name for it:

Exercise 4.20. In how many ways can I arrange n books on a shelf?

It's $n \cdot (n-1) \cdot (n-2) \cdots 2 \cdot 1$. This occurs so frequently in combinatorics that we denote this by the notation $n!$. We also define $0! = 1$, because there is one way to arrange zero things on a shelf: the empty arrangement. (It also makes sense if you consider $n! = n \cdot (n-1)!$.)

This is an example of a problem we call a **permutation**. It occurs whenever we're arranging several items out of a group of items. Here's another problem involving permutations:

Exercise 4.21. There are 20 people in your classroom. How many ways are there to select a president, a vice president, and a secretary, if no person can be more than one role?

There are 20 choices for the president, 19 choices for the vice president (since we can't choose the president), and 18 choices for the secretary (since we can't choose either the president or the vice president). So this gives $20 \cdot 19 \cdot 18 = 6840$ different choices. More generally:

Exercise 4.22. There are n people in your classroom. How many ways are there to select k different officers, such that no person can be more than one officer?

There are n choices for the first officer, $n-1$ choices for the second officer, $n-2$ choices for the third officer, and so on. When we get to the kth officer, there are $n-(k-1)$ choices. This means the answer is

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

Is there an easier way to write this? Well, it looks a lot like $n!$, but some of the terms are missing. Let's add in the missing terms:

$$\frac{n \cdot (n-1) \cdot (n-2) \cdots (n-k+1) \cdot (n-k) \cdot (n-k-1) \cdots 2 \cdot 1}{(n-k) \cdot (n-k-1) \cdots 2 \cdot 1}.$$

But the bottom fraction is $(n-k)!$. So this is just $\frac{n!}{(n-k)!}$. This is sometimes denoted by $P(n,k)$, the letter P meaning "permutation".

Exercise 4.23. I place 7 slips of paper in a box. In how many ways can I draw 3 sheets of paper in order, if I keep each slip out of the box after drawing it?

Applying the previous formula, this is

$$P(7,3) = \frac{7!}{(7-3)!} = \frac{5040}{24} = 210.$$

It turns out that in cases like these, it's actually easier to cancel out the individual factors. This is where we use the fact that $n! = n \cdot (n-1)!$:

$$\begin{aligned} 7! &= 7 \cdot 6! \\ &= 7 \cdot 6 \cdot 5! \\ &= 7 \cdot 6 \cdot 5 \cdot 4!. \end{aligned}$$

So the fraction is

$$\frac{7 \cdot 6 \cdot 5 \cdot \cancel{4!}}{\cancel{4!}} = 7 \cdot 6 \cdot 5 = 210.$$

Exercise 4.24. I place 7 slips of paper in a box. In how many ways can I draw 3 sheets of paper in order, if I place each slip back in the box after drawing it?

Exercise 4.25. For all n, prove that $1 \cdot 1! + 2 \cdot 2! + \cdots + n \cdot n! = (n+1)! - 1$.

Exercise 4.26. Simplify $P(n,k)P(n-k,j)$. What does this represent?

4.3.2 Break down

Often, we have to split into cases, and do **casework** on each one separately:

Exercise 4.27. How many strings with no more than 3 letters can we make from the letters A, B, C, and D, if we are allowed to repeat letters? Count the empty string.

Here, we have four cases:

- 1 empty string.
- 4 strings of length 1: you have 4 choices for the letter.
- $4 \cdot 4$ strings of length 2: you have 4 choices for the first letter, and 4 choices for the second letter.
- $4 \cdot 4 \cdot 4$ strings of length 3, by similar reasoning.

In total, there are $1 + 4 + 16 + 64 = 85$ different strings. Note the difference between the addition and the multiplication here.

Exercise 4.28. How many 3-letter strings can we make from the letters A, B, C, and D, if we are allowed to repeat letters and the letter A must be used at least once?

Let's split this into three cases, based on the number of As there are:

- Consider strings that have exactly one A. We have three cases again:
  - Strings of the form Axx, where $x \neq A$. There are $3 \cdot 3$ strings for this case.
  - Strings of the form xAx, where $x \neq A$. There are $3 \cdot 3$ strings for this case.
  - Strings of the form xxA, where $x \neq A$. There are $3 \cdot 3$ strings for this case.

  In total, there are $3 \cdot 3 \cdot 3 = 27$ strings for this case. We could also have counted this by considering "there are 3 different choices to place the A, 3 choices for the first non-A letter, and 3 choices for the second non-A letter." We'll use this for the next cases.
- Consider strings that have exactly two As. There are 3 possible ways the two As can be position: AAx, AxA, and xAA. Then there are 3 choices for the non-A letter, giving $3 \cdot 3 = 9$ strings.
- Finally, there's strings with three As, which is just 1 string.

In total, there are $27 + 9 + 1 = 37$ such strings.

Exercise 4.29. There are four cities in a country, labeled A, B, C, and D. They are connected with a series of one-way roads. The only roads are 5 one-way roads from A to B, 4 one-way roads from A to C, 3 one way roads from B to D, and 6 one-way roads from C to D. How many ways are there to go from A to D through a series of one-way roads?

4.3.3 The wrong thing

Let's revisit this problem:

Exercise 4.30. How many 3-letter strings can we make from the letters A, B, C, and D, if we are allowed to repeat letters and the letter A must be used at least once?

Let's use a technique called **complementary counting**. A strong signal to use this is if we're being asked to count *at least one*, where it's often easier to count how many have *none*.

How many 3-letter strings are there in total? We know that there are $4 \cdot 4 \cdot 4$ of these: you have 4 choices for each letter. Now how many don't use the letter A? It's $3 \cdot 3 \cdot 3$: you have 3 choices for each letter. So the answer is $64 - 27 = 37$, as expected.

Exercise 4.31. In how many ways can 6 people be seated in a row if two of the people, Kevin and Charles, had a fight and refuse to sit next to each other?

Here, we count the number of ways there are to seat them in total, and then subtract the number of ways where Kevin and Charles sit next to each other. There are 6! ways to seat them in total.

How many ways are there where Kevin and Charles sit next to each other? Well, we can merge them as the same person, and then there are 5! ways to arrange the four people and one merged person in a row. And for each way, you have 2! ways to order Kevin and Charles, so that makes $2 \cdot 5!$ ways to seat such that they're next to each other.

So the final answer is $6! - 2 \cdot 5! = 5!(6 - 2) = 480$.

Exercise 4.32. Four boys and three girls sit in a row. How many ways are there to do this such that at least two of the boys are next to each other?

Exercise 4.33. How many 5-digit numbers have at least two zeroes?

4.4 Counting is division

Here's a familiar kind of problem:

Exercise 4.34. How many ways are there to arrange the letters in the word $KEVIN$?

This is easy, there are $5! = 120$ different ways. No sweat, right? Now what about something like

Exercise 4.35. How many ways are there to arrange the letters in the word $CISCO$?

Here, the answer simply isn't 5!. The same reasoning doesn't work, because two of the letters are the same. Because of that, the answer of 5! is too big compared to the actual answer. We're **overcounting**.

How so? Let's pretend that the Cs are different for now, and label them C_1 and C_2. Then C_1ISC_2O and C_2ISC_1O are two different arrangements counted in 5!. But these both correspond to the word $CISCO$ when we remove the numbers and return to the original setup. So the word $CISCO$ is counted *twice* with 5!.

Similarly, any actual arrangement will be counted twice. For example, $SICCO$ is counted

as both SIC_1C_2O and SIC_2C_1O. So the number of arrangements of C_1ISC_2O, which is 5!, is exactly twice the number of arrangements of $CISCO$.

To get the answer, then, we divide it by 2. So $5!/2 = 60$, which is the number of arrangements in the word $CISCO$.

Exercise 4.36. How many ways are there to arrange the letters in the word $RATATA$?

Let's try using the same trick, and label the different letters: $RA_1T_1A_2T_2A_3$. There are 6! ways to arrange these letters. Now by how much do overcount?

Well, we want to figure out the number of ways to change the different-labeled letters such that the actual word doesn't change. If you move the As around, without changing the positions of the other letters, you have 3! different ways to arrange them. And if you move the Ts around, without changing the positions of the other letters, you have 2! different ways.

These choices are independent of each other, and they all produce the same word. So there are $3! \cdot 2! = 12$ different ways:

$$\begin{array}{lll} RA_1T_1A_2T_2A_3 & RA_1T_1A_3T_2A_2 & RA_2T_1A_1T_2A_3 \\ RA_2T_1A_3T_2A_1 & RA_3T_1A_1T_2A_2 & RA_3T_1A_2T_2A_1 \\ RA_1T_2A_2T_1A_3 & RA_1T_2A_3T_1A_2 & RA_2T_2A_1T_1A_3 \\ RA_2T_2A_3T_1A_1 & RA_3T_2A_1T_1A_2 & RA_3T_2A_2T_1A_1 \end{array}$$

By similar reasoning, we realize that we count each actual word 12 times. So the answer of 6! overcounts by a factor of 12, and the answer is $6!/12 = 60$.

That's the general trick in this section. If, due to symmetries, the same thing is counted multiple times, you want to divide by the number of symmetries in order to find the correct count. We'll talk more about this in the next part.

Exercise 4.37. How many ways are there to arrange the letters in $HOLLYWOOD$?

Exercise 4.38. I have 8 boxes in a row, and I want to color 4 of them navy, 2 of them orange, and 2 of them indigo. How many ways can I do this?

4.4.1 Symmetries

What do I mean by a **symmetry**? Basically, this is like different ways that we count the same thing. Previously, for example, arrangements of the word $RATATA$ had 12 symmetries: there are 12 ways to arrange its letters such that you get the same thing, so we had to divide by 12. Consider the following problem:

Exercise 4.39. In an 10-person meeting, each pair of persons shakes hands exactly once. How many handshakes are held?

There are 10 people, and each one shakes hands with 9 persons. That means that there are $10 \cdot 9$ handshakes right?

Not really: each handshake is counted twice. Consider two people, Kevin and Charles. For Kevin, part of the 9 count was his handshake with Charles. And for Charles, part of the 9 count was his handshake with Kevin. So the handshake between Kevin and Charles was counted twice, when it should only be counted once.

Thus, since $10 \cdot 9$ counts each handshake twice, dividing by 2 gives the correct answer of 45.

Here, the symmetry is with the handshakes. In this case, the handshake has 2 "symmetries", so we had to divide by 2 to get the correct count. Here's another example where the symmetries are clearer:

Exercise 4.40. How many different ways can 6 people be seated around a circle, if rotations of the same table are considered the same?

The 6! arrangements of them in a row overcounts. For example, if we went around the circle and see $ABCDEF$, and then in another arrangement see $BCDEFA$, these are the same.

So arrangements here have 6 symmetries: one for each of its rotations. For example, $ABCDEF$ is actually counted six times instead of just once: with $ABCDEF$, $BCDEFA$, $CDEFAB$, $DEFABC$, $EFABCD$, $FABCDE$.

Dividing by the number of symmetries gives the answer of 120.

Exercise 4.41. In how many ways can 5 keys be placed on a keychain, if rotations and reflections of the keychain are considered the same?

Again, there are 5! ways to arrange them in a row, but this overcounts. One of the symmetries is rotation: there are 5 different ways to rotate a keychain. But another symmetry is reflection: each arrangement has 2 reflections. So in total, an arrangement is counted $5 \cdot 2 = 10$ different times, so we must divide 5! by 10 to get the final answer of 12.

Exercise 4.42. How many ways can 8 people sit around a table if Kevin and Charles don't want to sit next to each other?

Exercise 4.43. How many ways can we color the vertices of a regular 7-gon red and blue if rotations are considered the same? (Careful with the all red and all blue configurations!)

4.4.2 Combinations

Let's revisit an old problem:

Exercise 4.44. There are 20 people in your classroom. How many ways are there to select a president, a vice president, and a secretary, if no person can be more than one role?

The answer here is $P(20, 3) = 20 \cdot 19 \cdot 18$. But what if we change it to this:

Exercise 4.45. There are 20 people in your classroom. How many ways are there to select a three-person committee (where the order of the people in the committee doesn't matter)?

This is definitely less than $20 \cdot 19 \cdot 18$, since the order of the people in the committee doesn't matter. How many symmetries are there? With $20 \cdot 19 \cdot 18$, each possible configuration of people is counted six times:

$$ABC, \quad ACB, \quad BAC, \quad BCA, \quad CAB, \quad CBA.$$

So there are 6 symmetries, and we have to divide by 6 to get the final answer. This gives 1140. More generally:

Exercise 4.46. There are n people in your classroom. How many ways are there to select k people to form a committee (where the order of the people in the committee doesn't matter)?

We know there are $P(n,k) = \frac{n!}{(n-k)!}$ ways to select people, if order did matter. But order doesn't matter here. So there are $k!$ symmetries, coming from the $k!$ different ways to arrange the k people. We then have to divide the final answer by $k!$ to get

$$\frac{n!}{k!(n-k)!}.$$

This appears often enough that it gets its own notation: $\binom{n}{k}$. This is read as "n choose k", because we're choosing k out of n things. These are called **combinations**, where order doesn't matter, as opposed to permutations where it does.

Exercise 4.47. How many ways can I form a team of 3 people out of 11 competitive programmers?

We see that this is

$$\binom{11}{3} = \frac{11!}{3!\,8!}.$$

Of course, it would be a hassle to compute 3! and 8! and 11! and do all the multiplications and divisions. Instead, we can rewrite 11! in terms of 8! by repeatedly using $n! = n \cdot (n-1)!$:

$$\begin{aligned} 11! &= 11 \cdot 10! \\ &= 11 \cdot 10 \cdot 9! \\ &= 11 \cdot 10 \cdot 9 \cdot 8! \\ \frac{11!}{3!\,8!} &= \frac{11 \cdot 10 \cdot 9 \cdot \cancel{8!}}{3!\,\cancel{8!}} \\ &= \frac{11 \cdot 10 \cdot 9}{3 \cdot 2 \cdot 1} = 165. \end{aligned}$$

Exercise 4.48. How many diagonals does a regular decagon have? (A diagonal is a segment joining two non-adjacent vertices.)

How is this related to combinations? Well, note that a diagonal is formed by choosing two of the vertices of a regular decagon. So this is $\binom{10}{2}$. But we have to subtract the "diagonals" we counted that are actually edges, of which there are 10. So the final answer is 35.

Exercise 4.49. What is $\binom{n}{0}$, $\binom{n}{1}$, and $\binom{n}{n}$? Does the formula here make sense?

Exercise 4.50. How many triangles can be formed among the vertices of a regular decagon?

Exercise 4.51. In my classroom, there are 11 girls and 10 boys. How many ways can I choose a 6-person committee consisting of 3 girls and 3 boys?

4.4.3 Binomial coefficients

In fact, the numbers $\binom{n}{k}$ are called **binomial coefficients**, and here's why.

Exercise 4.52. Expand $(x+y)^4$. And then list down the numbers $\binom{4}{0}, \binom{4}{1}, \ldots, \binom{4}{4}$.

Let's expand $(x+y)^4$. I know that $(x+y)^2 = x^2 + 2xy + y^2$, so I need to find its square. I can do this using something like long multiplication:

$$\begin{array}{cccccc} & & x^2 & 2xy & y^2 \\ & \times & x^2 & 2xy & y^2 \\ \hline & & x^2y^2 & 2xy^3 & y^4 \\ & 2x^3y & 4x^2y^2 & 2xy^3 & \\ x^4 & 2x^3y & x^2y^2 & & \\ \hline x^4 & 4x^3y & 6x^2y^2 & 4xy^3 & y^4 \end{array}$$

And interestingly, if we list down those numbers, we get

$$\binom{4}{0} = 1, \quad \binom{4}{1} = 4, \quad \binom{4}{2} = 6, \binom{4}{3} = 4, \binom{4}{4} = 1.$$

What is happening here? How are these numbers appearing as the coefficients of the powers of a binomial?

Exercise 4.53. Expand $(x+y)^5$ by multiplying our previous expansion by $x+y$. Convince yourself that you get the same coefficients as $\binom{5}{0}, \binom{5}{1}, \ldots, \binom{5}{5}$.

Here's a good explanation of why this is happening. Consider something like

$$(x+y)^3 = (x+y)(x+y)(x+y).$$

Each term in the expansion comes from a choice of either x or y in each pair of parentheses. For example, if we choose x, x, then y, we get the term x^2y. And if we choose y, x, then x, we get x^2y as well:

$$(x+y)(x+y)(x+y) = xxx + xxy + xyx + xyy + yxx + yxy + yyx + yyy.$$

But how many are there of each term?

Exercise 4.54. Consider the $x^{n-k}y^k$ term in $(x+y)^n$. Convince yourself that there are $\binom{n}{k}$ ways to choose x or y to make this term.

In this case, we see there are three xy^2 terms. This is become such a term is produced by choosing two ys out of three possible factors. So there are $\binom{3}{2}$ possible ways to pick this term, and this must be its coefficient.

More generally, we can write it out the expansion of $(x+y)^n$ as

$$\binom{n}{0}x^n + \binom{n}{1}x^{n-1}y + \binom{n}{2}x^{n-2}y^2 + \cdots + \binom{n}{n-1}xy^{n-1} + \binom{n}{n}y^n.$$

Exercise 4.55. Expand $(2x-1)^5$.

Exercise 4.56. What is the x^5y^2 term of $(x+2y)^7$?

Exercise 4.57. What is the constant term (the one without any variable) in the expansion of $\left(x^2-\frac{2}{x}\right)^9$?

Finally, a quick note. Consider this:

$$\begin{aligned}(x+y)^0 &= 1\\ (x+y)^1 &= x+y\\ (x+y)^2 &= x^2+2xy+y^2\\ (x+y)^3 &= x^3+3x^2y+3xy^2+y^3\\ (x+y)^4 &= x^4+4x^3y+6x^2y^2+4xy^3+y^4.\end{aligned}$$

If we consider only the binomial coefficients, we can line them up in a triangle like this:

$$\begin{array}{ccccccccccc}
 & & & & & 1 & & & & & \\
 & & & & 1 & & 1 & & & & \\
 & & & 1 & & 2 & & 1 & & & \\
 & & 1 & & 3 & & 3 & & 1 & & \\
 & 1 & & 4 & & 6 & & 4 & & 1 & \\
\end{array}$$

This is called **Pascal's triangle**. Observe that each number is the sum of the two numbers directly above it. Is this just a coincidence?

Exercise 4.58. By using the property that each number is the sum of the two numbers above it, list down the next row of Pascal's triangle. It should have 6 numbers.

Exercise 4.59. Using our observation from Pascal's triangle, convince yourself that $\binom{n}{k-1}+\binom{n}{k}=\binom{n+1}{k}$.

Much more about binomial coefficients can be said, but we'll leave this for Math 2.

4.5 Counting is bijection, again

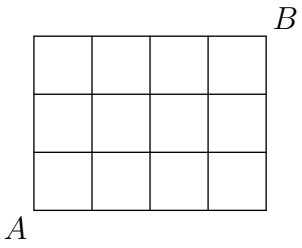
We end this section with some important applications of binomial coefficients.

4.5.1 Paths in a grid

Let's consider counting the number of paths in a grid:

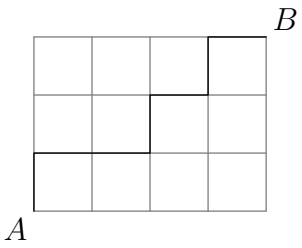
Exercise 4.60. In a 4×3 grid, how many ways are there to go from the lower-left corner to the upper-right corner by walking either right or up on the grid lines?

Let's draw a figure. We need to find a path from A to B that only goes right or up along the grid lines:



Each "step" is either one unit to the right, or one unit up. In total, we need to make 4 steps right, and 3 steps up. There are 7 steps in total, so in order to specify a path, we need to *choose* 4 of these steps to be right, and then the remaining steps will be up. So the number of paths is $\binom{7}{4} = 35$.

More specifically, we can represent any such path using a sequence of Rs (for right steps) and Us (for up steps). For example, $URRURUR$ represents the following path:



So each sequence of steps represents a rearrangement of the letters $RRRRUUU$. Similarly, each rearrangement of the letters corresponds to a sequence of steps. So the number of paths and the number of rearrangements of the letters are in **bijection**—meaning to count one of them, all we have to do is count the other.

And to count rearrangements of $RRRRUUU$, it's just $\frac{7!}{4!3!}$, or $\binom{7}{4}$, as desired.

Exercise 4.61. How many ways are there to go from $(0, 0)$ to $(5, 4)$ if our only moves are going one unit up or one unit to the right?

Exercise 4.62. How many ways are there to go from $(0, 0)$ to $(5, 4)$ if our only moves are going one unit up or one unit to the right, and we must pass through the point $(4, 2)$?

4.5.2 Twelvefold way

Let's talk about **distinguishability**. We call objects **distinguishable** if we can tell them apart, and **indistinguishable** if we can't. Many of the counting problems we can hope to solve can be modeled with the balls and boxes model. It goes like this:

Example 4.63. How many ways are there to place b (in/distinguishable) balls into x (in/distinguishable) boxes (such that (at most 1/at least 1) ball is in each box)?

We can represent the $2 \cdot 2 \cdot 3 = 12$ possibilities with a table. This is often called the twelvefold way:

b balls	x boxes	no restriction	at most 1	at least 1
distinguishable	distinguishable			(hard)
indistinguishable	distinguishable			
distinguishable	indistinguishable	(hard)		(hard)
indistinguishable	indistinguishable	(hard)		(hard)

Try copying this table and filling out the entries as we talk about them. Five possibilities in the twelvefold way are more complicated, and will be discussed in Math 2. Two of the possibilities should be clear:

- How many ways are there to place b *different* boxes into x *identical* boxes *such that at most* 1 *ball is in each box*? This is just 1 if $b \leq x$, and 0 otherwise.
- How many ways are there to place b *identical* balls into x *identical* boxes *such that at most* 1 *ball is in each box*? This is again just 1 if $b \leq x$, and 0 otherwise.

Three of the possibilities we've discussed:

- How many ways are there to place b *different* balls into x *different* boxes? We have x choices for the first ball, x choices for the second ball, and so on, making $x \cdot x \cdots x = x^b$ ways.
- How many ways are there to place b *different* balls into x *different* boxes *such that at most* 1 *ball is in each box*? This is $P(x, b)$: you have x choices for the first ball, $x - 1$ choices for the second ball, and so on.
- How many ways are there to place b *identical* balls into x *different* boxes *such that at most* 1 *ball is in each box*?

  Since the balls are identical and each box can only get one ball, we only care about which boxes receive a ball and which ones don't. Out of the x boxes, we want to pick b of them to get a ball. And there are $\binom{x}{b}$ ways to do this.

We're left with considering the final two possibilities. Let's consider a specific example of the first one:

Exercise 4.64. How many ways are there to place 4 *identical* balls into 3 *different* boxes?

Here, since the balls are identical, we only care about the number of balls in each box. We can represent the number of balls in each box with an ordered triple (a, b, c). We can enumerate them to get

$$\begin{array}{lllll} (0,0,4) & (0,1,3) & (0,2,2) & (0,3,1) & (0,4,0) \\ (1,0,3) & (1,1,2) & (1,2,1) & (1,3,0) & \\ (2,0,2) & (2,1,1) & (2,2,0) & & \\ (3,0,1) & (3,1,0) & & & \\ (4,0,0) & & & & \end{array}$$

You can verify that we didn't miss any cases. From this, we see there are 15 different ways to do this.

There are several more clever way to count this. The main idea, like the whole main idea of this section, is to find a clever bijection. We're going to represent each triple using something that is easier to count. Here, we're going to use an argument called **stars and bars**.

Let's convert each such representation of the number of balls like this:

$$(1,1,2) \to \star \,|\, \star \,|\, \star\star \qquad (3,0,1) \to \star\star\star \,||\, \star \qquad (0,4,0) \to |\, \star\star\star\star \,|$$

Basically, we're representing the number of balls in each box using stars, and we're inserting dividers between the two different boxes. So each configuration corresponds to an arrangement of stars and bars. Here, there are 4 stars, and there are 2 bars.

But more importantly, **this is also a bijection**! Any arrangement of 4 stars and 2 bars also corresponds to a placement of balls into the boxes! And it's easy to count the number of arrangements of stars and bars: there are 6 symbols, we pick 2 of them to be bars, so this is $\binom{6}{2}$. Which is, as expected, 15!

Exercise 4.65. Consider the general problem: How many ways are there to place b *identical* balls into x *different* boxes? Convince yourself, using a similar argument, that the answer is $\binom{b+x-1}{x-1}$.

We can deal with the last possible case using a somewhat similar argument:

Exercise 4.66. How many ways are there to place b *identical* balls into x *different* boxes *such that at least* 1 *ball is in each box*?

Again, we only care about the number of balls in each box. One way we can do it is to do something similar to the previous case:

Exercise 4.67. Argue that the problem is equivalent to placing $b - x$ identical balls into x different boxes, with no other restriction. Then use the previous argument to show that the answer is $\binom{b-1}{x-1}$.

But we can also do this directly by constructing another clever bijection. Consider the stars in a row, and placing dividers with them. Note that we can only insert dividers *between* two stars, and we can only insert at most one such divider:

$$\star\_\star\_\star\_\star\_\star \qquad \star \,|\, \star\_\star \,|\, \star\_\star \to (1,2,2) \qquad \star \,|\, \star\_\star\_\star \,|\, \star \to (1,3,1).$$

But there are $b-1$ slots in between the b stars. And we need to choose $x-1$ of them to be dividers. So the answer is $\binom{b-1}{x-1}$.

Exercise 4.68. How many ways are there to give three identical candies to five children?

Exercise 4.69. How many solutions, in non-negative integers, does $a + b + c + d = 8$ have?

Exercise 4.70. How many solutions, in odd positive integers, does $a + b + c = 19$ have?

5 Problems

We'll distinguish between two kinds of first-timers, based on their math background:

- **Mathy first-timers** will refer to first-timers who have a math background. For example, if you've qualified for PMO Areas or PMO Nationals, you're probably a mathy first-timer. For this problem set, **mathy first-timers will be grouped with returning students**.
- **Non-mathy first-timers** will refer to first-timers who don't have a background in competition math. If this module is your first introduction to these concepts, consider yourself a non-mathy first-timer.

If you're a mathy person who's doing the non-mathy set, it's your loss—not only will you be less challenged, but you're also going to get bored. And if you're a non-mathy person, don't force yourself to try the mathy set, because it really is intended for people with a math background.

Follow this general principle: if you find the problems too easy, choose the harder set; if you find the problems too hard, choose the easier set.

This being a math module, there are substantially more non-coding problems than usual. I'm recommending you **prioritize finishing the coding requirement before the non-coding requirement,** *especially* if you're a mathy first-timer or a returning student.

Finally, **ask for hints if you need it!** Here's the discussion policy: if it's worth [2★] or [3★], you can ask for hints, give hints to others, generally discuss the problem on Discord. If it's worth more, you can ask for hints on Discord, but don't give other people hints.

5.1 Warmup problems

W1 Preface: https://www.urionlinejudge.com.br/judge/en/problems/view/1837

W2 Basic Factorization: https://www.hackerrank.com/contests/noi-ph-2019-preselection/challenges/basic-factorization

W3 Teams: UVa 11609

W4 Diagonal: UVa 10784

5.2 Non-coding problems

No need to be overly formal in your answers; as long as you can convince me, it's fine. For problems that do not ask you to prove your answer, simply giving the answer, without any solution, is fine.

Do not use a computer or code to solve any of these problems. (Except maybe the debugging problems.) That would be against the spirit of the module. You can use a four-function calculator to help with checking your arithmetic.

5.2.1 Debugging

Let's do some debugging! Each of the following programs has a mistake, possibly several mistakes. For each problem, identify what is wrong with the code, and how to fix it.

All students are required to solve [15★].

N1.1 [5★] The following code attempts to compute $\gcd(a, b)$.

```cpp
using ll = long long;

ll gcd(ll a, ll b) {
    return b ? gcd(b, a - b) : a;
}
```

N1.2 [5★] The following code attempts to compute the value $a^k \bmod n$.

```cpp
using ll = long long;

ll modpow(ll a, ll k, ll n) {
    if (n == 0) return 1;
    if (n == 1) return a;
    if (n & 1) return a * modpow(a, k - 1, n) % n;
    return modpow(a, k >> 1, n) * modpow(a, k >> 1, n) % n;
}
```

N1.3 [5★] The following code attempts to solve the following problem: *Given a positive integer $n \leq 10^6$, what is the smallest prime greater than n? (There are multiple test cases.)*

```cpp
#include <bits/stdc++.h>
using namespace std;

const int N = 1000000;
bool is_prime[N+1];

void prime_sieve() {
    for (int n = 3; n <= N; n += 2) is_prime[n] = true;
    for (int p = 3; p * p <= N; p += 2) {
        if (!is_prime[p]) continue;
        for (int n = p * p; n <= N; n += 2 * p) is_prime[n] = false;
    }
}

int main() {
    prime_sieve();
    int cases;
    for (cin >> cases; cases--;) {
        int n;
        cin >> n;
        while (!is_prime[++n]);
        cout << n << '\n';
    }
}
```

5.2.2 Non-mathy set

If you're a non-mathy first-timer, solve at least [80★] from this set. Red denotes required problems.

Tip. Remember that you don't have to submit any solution or explanation for problems that just ask for an answer!

N2.1 [3★] Exercise 2.3

N2.2 [2★] Exercise 3.3

N2.3 [5★] Exercise 4.13

N2.4 [2★] Exercise 4.19

N2.5 [5★] Exercise 4.25

N2.6 [2★] Exercise 4.29

N2.7 [3★] Exercise 4.32

N2.8 [3★] Exercise 4.38

N2.9 [3★] Exercise 4.42

N2.10 [2★] Exercise 4.51

N2.11 [3★] Exercise 4.57

N2.12 [3★] Exercise 4.62

N2.13 [3★] Exercise 4.70

N2.14 [8★] Here's a "proof" by induction that a set of n horses all have the same color. The base case is clear. Now for the inductive step, assume that all sets of n horses have the same color. We want to show that all sets of $n+1$ horses have the same color.

Remove the last horse, and consider the first n horses. By inductive hypothesis, all these horses have the same color. Now remove the first horse, and consider the last n horses; again by induction they have the same color.

So the first horse is the same color as the horses in the middle, which has the same color as the last horse. Therefore, all the horses have the same color. What's wrong with this proof?

N2.15 [5★] Prove that $\gcd(m,n)\operatorname{lcm}(m,n) = mn$.

N2.16 [5★] Simplify the following: $\dfrac{\operatorname{lcm}(b, a-b)}{\operatorname{lcm}(a,b)} + \dfrac{b}{a}$.

N2.17 The Euclidean algorithm isn't just useful in programming, but it's also useful for solving math problems. Use it to solve these two problems:

(a) [5★] (IMO 1959) Prove that, for all positive integers n, the fraction $\dfrac{21n+4}{14n+3}$ is in lowest terms.[8]

(b) [5★] (AIME 1985) Let $a_n = n^2 + 100$, and for each positive integer n, let $d_n = \gcd(a_n, a_{n+1})$. What is the maximum value of d_n?

N2.18 [2★] Let $\pi(x)$ denote the number of primes less than or equal to x. Is it true that $\pi(x) - \pi(x-1) = 1$ if and only if x is prime? Why or why not?

[8] If you've solved this, congratulations! You've solved an IMO problem!

N2.19 [2★] What does $a \equiv b \pmod 0$ mean?

N2.20 [5★] Prove that the square of any integer is 0, 1, or 4 modulo 8.

N2.21 Consider the problem of counting the number of positive divisors of an integer.

(a) [2★] Consider 540. We know that $540 = 2^2 3^3 5$. Why must any divisor of 540 be of the form $2^a 3^b 5^c$?

(b) [3★] Explain why, if $2^a 3^b 5^c$ is a divisor of 540, we can't have $a \geq 3$. Thus, a can only be 0, 1, or 2. In particular, we have 3 different choices for a.

(c) [3★] Similarly, we know that b can only be 0, 1, 2, or 3, and c can only be 0 or 1. Using this information, how many positive divisors does 540 have?

(d) [8★] Prove that, if the prime factorization of n is $p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$, then the number of its positive divisors is $(e_1+1)(e_2+1)\cdots(e_k+1)$.

(e) [2★] What are the positive integers with an odd number of positive divisors? No need to prove this.

N2.22 [5★] Let x and y be integers such that $x \geq y \geq 0$. Find a simpler formula for

$$\frac{\binom{x}{y+1}}{\binom{x}{y}} \qquad \text{and} \qquad \frac{\binom{x+1}{y}}{\binom{x}{y}}$$

in terms of x and y.

N2.23 We can use the binomial theorem to prove some cool identities about summations.

(a) [2★] Write out the expansion of $(x+y)^4$. Now replace $x = y = 1$. On the left-hand side, you get 2^4, and on the right-hand side, you get the sum of the coefficients. What does this tell you?

(b) [3★] Prove that $\binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{n} = 2^n$.

(c) [5★] Prove that $\binom{n}{0} - \binom{n}{1} + \binom{n}{2} - \cdots + (-1)^n \binom{n}{n} = 0$.

(d) [5★] Find a formula for $\binom{n}{0} + 2\binom{n}{1} + 4\binom{n}{2} + \cdots + 2^n \binom{n}{n}$.

N2.24 Consider the problem of counting the number of subsets of a set. For example, consider finding the subsets of {N, O, I}. There are 8:

$$\begin{array}{cccc} \varnothing & \{\text{N}\} & \{\text{O}\} & \{\text{I}\} \\ \{\text{N, O}\} & \{\text{N, I}\} & \{\text{O, I}\} & \{\text{N, O, I}\} \end{array}$$

(a) [8★] In general, a set with n elements has 2^n subsets. Prove this. Treat this is a counting problem: how many ways can you form a subset? In particular, consider whether each element is in the subset or not.

(b) [3★] Use this to give an alternate proof that

$$\binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{n} = 2^n.$$

N2.25 [8★] (PMO 2016) A circular railway passes through four towns A, B, C, and D, in that order. A trip between two adjacent towns costs one ticket. How many different ways are there to start from A, use exactly eight tickets, and end at A? (Passing through A during the middle of the trip is allowed.)

Hint. Represent a trip to the right town with R and a trip to the left town with L. What are the trips that start and end with A?

5.2.3 Mathy set

If you're a mathy first timer or a returning student, solve at least [75★] from this set. Red problems are highly recommended.

N3.1 [5★] Exercise 3.21

N3.2 [18★] Exercise 3.29. This is [3★] for each; please include an explanation with your answer.

N3.3 Let $d(n)$ be the number of positive divisors of n. In this problem, we prove that $d(1) + d(2) + \cdots + d(n)$ is in $\mathcal{O}(n \log n)$.

(a) [5★] Show that $d(1) + d(2) + \ldots + d(n) = \left\lfloor \frac{n}{1} \right\rfloor + \left\lfloor \frac{n}{2} \right\rfloor + \ldots + \left\lfloor \frac{n}{n} \right\rfloor$.
Hint: How many integers $\leq n$ are divisible by k?

(b) [2★] Show that $d(1) + d(2) + \ldots + d(n) \leq nH_n$, where H_n is the nth Harmonic number, given by $H_n = \frac{1}{1} + \frac{1}{2} + \ldots + \frac{1}{n}$.

(c) [3★] Show that $H_{2^k} \leq 1 + k$.
Hint: $\frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8} \leq \frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4} = 1$.

(d) [5★] Show that $d(1) + d(2) + \ldots + d(n) \leq n\left(\lceil \log_2 n \rceil + 1\right)$.

N3.4 Some facts about the gcd and lcm functions:

(a) [3★] Prove that $\gcd(ka, kb) = k \gcd(a, b)$.

(b) [2★] Use this to prove that $\gcd\left(\frac{a}{\gcd(a,b)}, \frac{b}{\gcd(a,b)}\right) = 1$.

(c) [2★] Prove that $\operatorname{lcm}(ka, kb) = k \operatorname{lcm}(a, b)$.

(d) [3★] Prove that $\gcd(a, b, c) \operatorname{lcm}(ab, bc, ca) = abc$.

(e) [3★] Prove that $\gcd(a, \operatorname{lcm}(b, c)) = \operatorname{lcm}(\gcd(a, b), \gcd(a, c))$.

(f) [5★] Use this to prove that $\operatorname{lcm}(a, b, c) = \frac{abc \cdot \gcd(a, b, c)}{\gcd(a, b) \cdot \gcd(b, c) \cdot \gcd(c, a)}$.

(g) [2★] Prove that $\operatorname{lcm}(a, \gcd(b, c)) = \gcd(\operatorname{lcm}(a, b), \operatorname{lcm}(a, c))$.

(h) [3★] Use this to prove that $\gcd(a, b, c) = \frac{abc \cdot \operatorname{lcm}(a, b, c)}{\operatorname{lcm}(a, b) \cdot \operatorname{lcm}(b, c) \cdot \operatorname{lcm}(c, a)}$.

N3.5 [3★] Suppose that a/b and c/d are in lowest terms. Prove that, when $a/b + c/d$ is reduced in lowest terms, its denominator is bd if and only if $\gcd(b, d) = 1$.

N3.6 [3★] For positive integers n, define a_n and b_n as the integers such that $a_n + b_n\sqrt{2} = \left(1 + \sqrt{2}\right)^n$. Prove that a_n and b_n are relatively prime.

N3.7 These problems have the same feel, in that they involve the Euclidean algorithm and exponents:

(a) [8★] Prove that if $\gcd(a, b) = 1$ and $a > b$ are positive integers, then

$$\gcd(a^m - b^m, a^n - b^n) = a^{\gcd(m,n)} - b^{\gcd(m,n)}.$$

(b) [3★] (PUMAC 2013) Find the integer n such that $\gcd\left(2^{30^{10}} - 2, 2^{30^{45}} - 2\right) = 2^n - 2$.

(c) [8★] (Japan 1996) Let m and n be relatively prime integers. Find $\gcd(5^m + 7^m, 5^n + 7^n)$.

N3.8 In this exercise we argue, informally, that the complexity of the Euclidean algorithm is $\mathcal{O}(\log \min\{m, n\})$.

(a) [2★] The "worst-case" that can happen is if the numbers don't decrease a lot after each step. So you want $n \bmod m$ to not be drastically small compared to n. This happens when the quotient is zero. If the quotient when n divided by m is zero, how are n, m, and $n \bmod m$ related?

(b) [2★] Which numbers m and n are there such that $\gcd(m, n)$ calls $\gcd(1, 0)$, and such that the quotient when m divided by n is zero? Similarly, which m' and n' are there such that $\gcd(m', n')$ calls $\gcd(m, n)$, where the quotient of m' divided by n' is zero?

(c) [3★] Keep continuing this backwards. The numbers should be familiar: do you recognize them? Argue that these numbers grow exponentially large in terms of the number of steps.

In other words, the number of steps is logarithmic in the size of the numbers. Since this is the "worst-case", then the complexity is logarithmic in the size of the inputs.

N3.9 This is known as the **binary gcd algorithm**. Multiplication and division are relatively more expensive, so the following algorithm only divides and finds numbers modulo 2:

```
int gcd(int x, int y) {
  int g = 1;
  while (x % 2 == 0 and y % 2 == 0) {
    x /= 2; y /= 2; g *= 2;
  }
  while (x != 0) {
    while (x % 2 == 0) x /= 2;
    while (y % 2 == 0) y /= 2;
    int t = abs(x - y)/2;
    if (x >= y) x = t;
    else y = t;
  }
  return g*y;
}
```

(a) [8★] Prove that this algorithm works.

(b) [5★] What is its worst-case complexity? In particular, is it the same as the Euclidean algorithm?

N3.10 (Putnam 2001) Let $n > m \geq 1$ be integers. Prove $\dfrac{\gcd(m, n)}{n}\dbinom{m}{n}$ is an integer.

(a) [2★] Prove that $\dfrac{m}{n}\dbinom{m}{n} = \dbinom{m-1}{n-1}$.

(b) [3★] Use Bézout's lemma to replace $\gcd(m, n)$. Then use the above to solve the problem.

N3.11 [5★] Find $100! \bmod 10^{25}$, and show your work.

N3.12 This problem deals with arbitrary-precision arithmetic. As mentioned previous, integers are represented as digit arrays in some base $B \geq 2$.

(a) [3★] Explain why we generally want B to fit in a 32-bit integer.

(b) [3★] With pen-and-paper arithmetic, we set $B = 10$ because we need the output to be in decimal. Explain why in this case, $B = 10^9$ is a better choice. Similarly, explain why $B = 2^{31}$ is a better choice than $B = 2$ in case we need the integer to be in binary.

N3.13 In this problem, we prove a formula for $\phi(n)$, which is the number of positive integers at most n and relatively prime to n.

(a) [3★] Let $p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$ be the prime factorization of n. How many positive integers less than n are divisible by p_i?

(b) [8★] Use the Principle of Inclusion–Exclusion with the above fact to prove that

$$\phi(n) = n\left(1 - \frac{1}{p_2}\right)\left(1 - \frac{1}{p_2}\right)\cdots\left(1 - \frac{1}{p_k}\right).$$

N3.14 [5★] How many pairs of positive integers (m, n) satisfy $\operatorname{lcm}(m, n) = 21\,600$?

N3.15 [8★] In a group of 7 people, each person chooses whom among the other 6 people they love the most. We call such a group *happy* if there is a pair of people who love each other the most. In how many ways can each of the 7 people choose such that their group is happy?

N3.16 In this problem we prove Fermat's little theorem.

(a) [3★] Solve Exercise 4.43.

(b) [5★] Let p be a prime. We color a necklace of p beads with $a \geq 2$ different colors. If rotations of a necklace are considered the same, prove that the number of different necklaces is $\frac{a^p - a}{p} + a$.
Since this number is an integer, then $p \mid a^p - a$, which proves Fermat's little theorem.

N3.17 [8★] How many ways can we choose 4 vertices of a regular n-gon to form a quadrilateral, such that at least one side of the quadrilateral is a side of the n-gon?

N3.18 [3★] How many 9-digit integers have digits sorted in non-decreasing order?

N3.19 [5★] How many k-element subsets of $\{1, 2, \ldots, n\}$ have no pair of integers with difference 1?

N3.20 In this problem, we prove the **multinomial theorem**, a theorem analogous to the binomial theorem, but for general polynomials.

(a) [2★] Consider n distinct objects, and m distinct bits. Suppose we want to distribute the objects into the bins such that the ith bin has k_i items. Prove that the number of ways to do this is

$$\binom{n}{k_1, k_2, \ldots, k_m} := \frac{n!}{k_1! k_2! \cdots k_m!}.$$

This is called the **multinomial coefficient**.

(b) [2★] Consider a string of length n over an alphabet with m letters. Suppose that the string has k_i of the ith letter. Prove that the number of ways to rearrange its letters is $\binom{n}{k_1, k_2, \ldots, k_m}$.

(c) [5★] Prove that

$$\binom{n}{k_1, k_2, \ldots, k_m} = \binom{k_1 + k_2 + \cdots + k_m}{k_1} \binom{k_2 + k_3 + \cdots + k_m}{k_2} \cdots \binom{k_m}{k_m}.$$

Provide both an algebraic proof and a combinatorial interpretation.

(d) [5★] The multinomial theorem states that

$$(x_1 + x_2 + \cdots + x_m)^n = \sum_{k_1+k_2+\cdots+k_m=n} \binom{n}{k_1, k_2, \ldots, k_m} x_1^{k_1} x_2^{k_2} \cdots x_m^{k_m}.$$

Prove this in a similar way we proved the binomial theorem.

(e) [2★] What is the coefficient of the $x^2y^2z^2$ term in $(x + y + z)^6$?

Like the binomial coefficients, the multinomial coefficients with $m = 3$ can be arranged in a pyramid. In this pyramid, each element would be the sum of the three terms above it. We leave the exploration of this pyramid, and its properties, to the reader.

5.3 Coding problems

Non-mathy first-timers, solve at least [75★]. Mathy first-timers and returning students, solve at least [120★]. Red problems are recommended. No credit for problems solved before this problem set was released.

For each **S** category except the last, you'll get a [8★] bonus for the first problem that you solve (after this problem set was released). **Focus on finishing one problem per category** before solving additional **S** problems.

C1 [5★] How many ways can you express $2^{64} - 1$ as the sum

$$\sum_{k=0}^{12345} a_k 2^k$$

where $0 \le a_k < 2^8$?

Note: Provide the answer and your code that computes it, which should run in at most 1 second.

C2 [5★] How many primes p are there such that $(p - 10^{15})^2 \le 10^{15}$?

Note: Provide the answer and your code that computes it, which should run in at most 20 seconds, even if we replace 10^{15} with some other number (of similar magnitude).

C3 [8★] Let p be a permutation of $\{0, 1, \ldots n-1\}$. We can think of p as a *function* from $\{0, 1, \ldots, n-1\}$ to $\{0, 1, \ldots, n-1\}$, one which sends i to $p[i]$.

Let $p^k(i)$ be the result of applying p k times starting from i. In other words, $p^k(i) = p[p[p[\ldots p[i]\ldots]]]$, where there are k ps there. In particular, $p^1(i) = p[i]$. Implement an algorithm that computes the following: *What is the smallest positive k such that for all i, $p^k(i) = i$?*

C4 [8★] Implement an algorithm that lists all squarefree integers up to n. It should run in $\mathcal{O}(n)$ time.

C5 [13★] Implement an algorithm that, given n, produces the list of divisors of each integer i, $1 \le i \le n$. It should run in $\mathcal{O}(n \log n)$ time.

C6 [17★] Implement arbitrary-precision arithmetic in C++ that supports addition, subtraction, multiplication, comparison, and modulo with a modulus that fits in a 32-bit integer. Represent your integers as digit arrays in some base $B \geq 2$ defined as a `const int` at the top of the code. Partial points will be given based on which features are implemented.

C7 [5★] **Rigid graphs:** https://projecteuler.net/problem=434

C8 [8★] **Factorials divisible by a huge integer:** https://projecteuler.net/problem=320

C9 [13★] **An amazing Prime-generating Automaton:** https://projecteuler.net/problem=308

C10 [13★] **Crazy Function:** https://projecteuler.net/problem=340

C11 [13★] **Primonacci:** https://projecteuler.net/problem=304

C12 [17★] **Problem 500!!!:** https://projecteuler.net/problem=500

S1 [2★] **Prime Gap:** UVa 1644

S2 [2★] **Chef and Prime Divisors:** https://www.codechef.com/problems/CHAPD

S3 [2★] **Little Nephew:** UVa 12463

S4 [2★] **Chess Queen:** UVa 11538

S5 [3★] **Finite or not?:** https://codeforces.com/problemset/problem/983/A

S6 [2★] **N-Factorful:** https://www.spoj.com/problems/NFACTOR

S7 [8★] **Winter is here:** https://codeforces.com/problemset/problem/839/D

S8 [3★] **Breaking Sticks:** https://www.hackerrank.com/contests/noi-ph-2019-preselection/challenges/breaking-sticks

S9 [5★] **Chef and Square Set:** https://www.codechef.com/problems/CHEFSSET

S10 [8★] **Jealous Numbers:** https://www.codechef.com/problems/NUMSET

S11 [3★] **Another Packing Problem:** https://www.hackerrank.com/contests/noi-ph-2019-preselection/challenges/another-packing-problem

S12 [5★] **Nastya Studies Informatics:** https://codeforces.com/problemset/problem/992/B

S13 [5★] **Sharing Candies:** https://www.codechef.com/problems/GIVCANDY

S14 [8★] **Kirito in labyrinth:** https://www.codechef.com/problems/KIRLAB

S15 [5★] **The Modular Wizard:** https://www.hackerrank.com/contests/noi-ph-2019-preselection/challenges/the-modular-wizard

S16 [5★] **Divisible by Seven:** https://codeforces.com/problemset/problem/375/A

S17 [8★] **Print F:** https://www.hackerrank.com/contests/noi-ph-2019-preselection/challenges/2016-print-f

---

S18 [3★] **Cartesian Country:** https://www.hackerrank.com/contests/noi-ph-2019-preselection/challenges/cartesian-country

S19 [3★] **Jimmy's Balls:** UVa 11480

S20 [3★] **Hapless Hedonism:** UVa 11554

S21 [5★] **Karen and Test:** https://codeforces.com/problemset/problem/815/B

S22 [3★] **Cheerleaders:** UVa 11806

S23 [5★] **Adding Time:** https://www.hackerrank.com/contests/noi-ph-2019-preselection/challenges/2017-adding-time

S24 [5★] **Maximum Palindromes:** https://www.hackerrank.com/contests/noi-ph-2019-preselection/challenges/maximum-palindromes

S25 [8★] **Pabebe Girl Cuts Cake:** https://www.hackerrank.com/contests/noi-ph-2019-preselection/challenges/pabebe-cake

S26 [8★] **Kyota and Permutation:** https://codeforces.com/problemset/problem/553/B

S27 [8★] **On the Bench:** https://codeforces.com/problemset/problem/840/C

S28 [13★] **Oprah's Favorite Things:** https://www.hackerrank.com/contests/noi-ph-2019-preselection/challenges/oprahs-favorite-things

S29 [13★] **Replace All:** https://codeforces.com/problemset/problem/794/G

S30 [13★] **Parol Banawa:** https://www.hackerrank.com/contests/noi-ph-2019-preselection/challenges/parol-banawa

S31 [13★] **Bisayaka Grid:** https://www.hackerrank.com/contests/noi-ph-2019-preselection/challenges/corrupted-grid

S32 [17★] **Surveillance:** https://www.hackerrank.com/contests/noi-ph-2019-preselection/challenges/surveillance-x

Acknowledgements

Lots of the material is based on Introduction to Counting and Probability, The Art of Problem Solving, and Concrete Mathematics. Thanks to Jared who wrote the previous math week module, and to Kevin for contributing to the problem set and appendix.

A Extra proofs

In this section, we give rigorous proofs for some of the theorems mentioned in the main text.

A.1 Division theorem

We mentioned the following theorem when we talked about Euclidean division:

Theorem A.1 (Division theorem)**.** For any two integers a and b where $b \neq 0$, there exist unique integers q and r such that

$$a = bq + r$$

and $0 \leq r < |b|$.

We omitted the proof of this because it feels very intuitive, but in this appendix, we will show you how to prove it. This also gives you a taste of the more rigorous side of math. In a college course, this will go even further; *everything* will be proven starting only with a few basic assumptions (called **axioms**), but we will not go that far here.

Proof. We will first prove it for the case $b > 0$. So let's assume first that $b > 0$.

Note that the theorem says that there *exist unique* integers q and r. We begin by showing that they exist, and then later on show that they are unique.

To begin, note that there are many integers of the form $a - bQ$ for integer Q. Our goal is to show that there is *exactly one* such integer in the range $[0, b)$.

Since $b > 0$, we can make $a - bQ$ nonnegative by selecting Q to be a very very negative number, regardless of a. Therefore, the set

$$\{a - bQ : Q \in \mathbb{Z}, a - bQ \geq 0\}$$

is not empty.[a]

Let R be the smallest element of the set above. Thus, $R = a - bQ$ for some Q, and $R \geq 0$ (by definition of the set). We will now show that $R < b$, hence, we can choose $r := R$ and $q := Q$, showing the existence of a solution $a = bq + r$ with $0 \leq r < b$.

Suppose $R \geq b$, for the purpose of contradiction. then $0 \leq R - b < R$. But $R - b = a - b(Q + 1)$, so $R - b$ is an element of the set as well, and which is smaller than R, contradicting the fact that R is the *smallest* element of the set. Therefore, our assumption is false, and indeed $R < b$.

To show that this is unique, suppose there are two solutions (q_1, r_1) and (q_2, r_2), that is,

$$a = bq_1 + r_1$$
$$a = bq_2 + r_2$$

with $0 \leq r_1, r_2 < b$.

Subtracting them, we get

$$0 = b(q_1 - q_2) + (r_1 - r_2)$$
$$b(q_2 - q_1) = r_1 - r_2.$$

This means that $r_1 - r_2$ is a multiple of b. But $-b < r_1 - r_2 < b$, and there is only one multiple of b between $-b$ and b, namely, 0. Therefore $r_1 - r_2$ must be 0, or $r_1 = r_2$. This also means that $q_2 - q_1 = \frac{r_1 - r_2}{b}$ is also 0, so $q_1 = q_2$. So the two solutions that we started with are the same, so the solution must be unique.

Finally, the case $b < 0$ can now be proven easily. First, any solution (q, r) to $a = bq + r$ with $0 \le r < |b|$ can be paired one-to-one with a solution (q', r') to $a = (-b)q' + r'$ with $0 \le r' < |-b|$, namely, $(q', r') = (-q, r)$. But since $-b > 0$, we have already shown that there is a unique solution to the latter, so there must also be a unique solution to the former. □

[a] If you want a stricter proof on why this is nonempty, simply set $Q = -|a|$; then $a - bQ = a + b|a| \ge a + 1|a| \ge a + (-a) = 0$.

A.2 Well-ordering principle

Although the previous proof feels *rigorous* enough, in fact there's a somewhat nontrivial step there that we did, namely, declaring that the set S has a *smallest* element. Although this feels intuitively obvious, in fact, it still has to be proven, since not all nonempty sets have a smallest element, for example, the set $\mathbb{Z}$ (the set of integers) doesn't have one! And it's not enough to find a lower bound, since the set of all *positive rational numbers* also doesn't have a minimum!

In fact, we implicitly invoked something called the *well-ordering principle*, which is a fundamental fact about $\mathbb{N}$ (the set of natural numbers).[9]

Theorem A.2 (Well-ordering principle)**.** Any nonempty subset of $\mathbb{N}$ has a smallest element.

Proof. The theorem can be restated as:

Let S be a subset of $\mathbb{N}$. If S is nonempty, then S has a smallest element.

We will prove the contrapositive (which is equivalent):

Let S be a subset of $\mathbb{N}$. If S doesn't have a smallest element, then S is empty.

To show the latter, suppose S is a subset of $\mathbb{N}$ that doesn't have a smallest element. We will show, by induction, the following statement, which we'll denote $\phi(n)$:

"S doesn't contain any integers $\le n$."

- The *base case*: Proving that $\phi(0)$. It's obvious; if S contains 0, then 0 will be the smallest element since all elements are nonnegative, which is a contradiction.
- The *inductive case*: Proving that $\phi(n) \implies \phi(n+1)$. Suppose $\phi(n)$ is true. Then to show $\phi(n+1)$, we only need to show that $n+1$ is not in S. But that is true, since we already established (from $\phi(n)$) that S doesn't contain all smaller integers $\le n$, so if $n+1$ were in S, then it will be the smallest element of S, contradicting the statement that S doesn't have a smallest element. Therefore, $\phi(n+1)$ is true.

This completes the induction. But look more closely at the statement we just proved: It states that S doesn't contain any natural number at all! Therefore, S is a subset of $\mathbb{N}$ that doesn't contain any element of $\mathbb{N}$, so S is empty. □

The proof we presented here is unusually low-level. The only "theorem" we used here was induction. This is because there's something deeper going on here. In fact, we can prove

[9] We will use the convention that $\mathbb{N}$ contains 0, which is common practice in higher math.

that the well-ordering principle is equivalent to the principle of induction! We will prove this equivalence shortly.

But before that, let's talk about what this "equivalence" really means. The well-ordering principle is a statement about subsets of $\mathbb{N}$, while induction is a proof technique, so how are they equivalent?

Well, in fact, induction can be viewed as a statement about subsets of $\mathbb{N}$ as well:

Theorem A.3 (The principle of induction)**.** Let S be a subset of $\mathbb{N}$. If $0 \in S$, and $n \in S \implies n+1 \in S$ for any n, then $S = \mathbb{N}$.

Now, let's think about why this is equivalent to induction. In induction, you usually want to prove that some statement, say $\phi(n)$, holds for all natural numbers n. But this is equivalent to proving that the set $\{n \in \mathbb{N} : \phi(n)\}$ is the same as $\mathbb{N}$ itself.

The idea here is that proving $0 \in S$ is equivalent to proving the base case, and proving that $n \in S \implies n+1 \in S$ is equivalent to proving the inductive case, and after proving both cases, Theorem A.3 allows us to conclude that $S = \mathbb{N}$, that is, the statement is true for all natural numbers. Therefore, Theorem A.3 is equivalent to induction!

Using this, we can now state (and prove) the equivalence between the well-ordering principle and the principle of induction:

Theorem A.4. The principle of induction is equivalent to the well-ordering principle.

Proof. To prove the equivalence, we need to show that either implies the other. But the proof of Theorem A.2 above is essentially the proof that "induction implies well-ordering". Therefore, we only have to prove that "well-ordering implies induction".

Let S be a subset of $\mathbb{N}$ such that $0 \in S$, and $n \in S \implies n+1 \in S$. We want to show that $S = \mathbb{N}$, assuming the well-ordering principle.

Suppose, for the purpose of contradiction, that $S \neq \mathbb{N}$. Therefore, the set S' of all natural numbers not in S is nonempty. By the well-ordering principle, it has a smallest element, say $n \in S'$. Note that $n \notin S$.

Now, what can n be? Can it be 0? No, because $0 \in S$. Therefore, $n > 0$. Now, $n-1 < n$ and n is the *smallest* element not in S, thus $n-1$ must be in S. But $n-1 \in S \implies n \in S$, contradicting the fact that n is not in S. Therefore, our assumption must be false, and $S = \mathbb{N}$, and the induction follows from well-ordering. □

One final note about these proofs (and we're getting incredibly low-level here): There are still a few hidden assumptions that we had in them! For example, how can we be sure that there are no natural numbers between n and $n+1$?

In fact, these hidden assumptions also have to be *proven*, strictly speaking, but to do that, we need to have a precise definition of $\mathbb{N}$ and the set of axioms we start with, which we didn't really give in this module. Well, we won't do that here; as I said, we are already getting too low-level. Instead, if you still wish to see a rigorous treatment of the theory of natural numbers, we will refer you to read about the Peano axioms.

A.3 Bézout's lemma

This is another one of the theorems that we didn't really use as a *theorem* in the main text. However, it will play particular importance in our proof of the fundamental theorem of

arithmetic:

Theorem A.5 (Bézout's lemma)**.** Let a and b be integers, not both equal to 0. Then, there exists integers x and y such that

$$ax + by = \gcd(a, b).$$

In the main text, we gave a constructive proof that found the integers using an algorithm. While satisfying, this is pretty hard to make fully rigorous. Instead, the typical way this is proved in a number theory course goes like this:

Proof. Let $\mathcal{S}$ be the set of all $ax + by$ where x and y are integers and $ax + by > 0$. Then $\mathcal{S}$ is a set of positive integers. It is also nonempty, because $a^2 + b^2 > 0$ is in the set. Thus, by the well-ordering principle, it has some smallest element d. We need to prove that $d = \gcd(a, b)$.

Since d is in $\mathcal{S}$, there are x and y such that $d = ax + by$. By Euclidean division, we can write $a = dq + r$ for some integers q and $0 \le r < d$. Now

$$\begin{aligned} r &= a - dq \\ &= a - (ax + by)q \\ &= a(1 - xq) + b(yq). \end{aligned}$$

Since $r < d$, and d is the smallest element of $\mathcal{S}$, r cannot be in $\mathcal{S}$. In particular, this means that $r = 0$, so $a = dq$, and $d \mid a$. A similar proof gives $d \mid b$, and so d is a common divisor of a and b.

It now remains to show that any other common divisor is at most d. Suppose that $c \mid a$ and $c \mid b$. Then

$$c \mid a, \quad c \mid b \quad \implies \quad c \mid ax + by \quad \implies \quad c \mid d \quad \implies \quad c \le d,$$

as desired. □

One good example of an application of Bézout's lemma is the following theorem:

Theorem A.6. Let a and b be relatively prime integers. Suppose that c is an integer such that $a \mid bc$. Then $a \mid c$.

Again, this is one of those theorems that looks deceptively intuitive, until you actually begin proving it. While your first instinct might be to consider the prime factors, *we don't even know that prime factorizations exist yet*! So we can't use that. But using Bézout's lemma makes quick work of this:

Proof. As a and b are relatively prime, by Bézout's lemma, there exists x and y such that $ax + by = 1$. Multiply through by c to get $acx + bcy = c$. But as $a \mid bc$, we can write $bc = ak$ for some k. So this is just $acx + aky = c$, or $a(cx + ky) = c$, hence $a \mid c$. □

This particular theorem is incredibly useful.[10] For example, it quickly follows from this that:

Theorem A.7 (Euclid's lemma)**.** Let p be a prime, and a and b be integers. Suppose that $p \mid ab$. Then $p \mid a$ or $p \mid b$.

[10] When I was in PROMYS, we actually called *this* theorem the fundamental theorem of arithmetic, over unique prime factorization. So that's at least one author who doesn't call unique prime factorization as the fundamental theorem of arithmetic.

Proof. If $p \mid a$, then we are done. If $p \nmid a$, then as p's only positive divisors are 1 and itself, we must have $\gcd(p, a) = 1$. So by the previous theorem, $p \mid b$. □

This seemingly simple fact, that took us a lot of work to prove, is the key to finally proving the fundamental theorem of arithmetic.

But before that, we first prove an obvious generalization/corollary[11] of Theorem A.7.

Corollary A.1. Let p be a prime and $a_1, a_2, \ldots, a_k$ be integers. Suppose that $p \mid a_1 a_2 \cdots a_k$. Then $p \mid a_i$ for some $1 \leq i \leq k$.

Proof. This is easily proven by induction on k and Theorem A.7.

For $k = 1$, the statement is obvious.

Now, suppose it is true for some $k \geq 1$. We would like to show it now for $k+1$. Suppose that $p \mid a_1 a_2 \cdots a_k a_{k+1}$. Then $p \mid (a_1 a_2 \cdots a_k) a_{k+1}$, so by Theorem A.7, $p \mid a_1 a_2 \cdots a_k$ or $p \mid a_{k+1}$. But the former implies $p \mid a_i$ for some $1 \leq i \leq k$ (using the inductive hypothesis), so we're done, and in the latter, we're also done. Thus, the inductive step is complete. □

A.4 Fundamental theorem of arithmetic

We now have enough to tackle the fundamental theorem of arithmetic. In fact, like the division theorem, this consists of two different parts:

- the *existence* of a prime factorization, meaning that given any positive integer n, some sequence of primes has n as their product;
- and the *uniqueness* of a prime factorization, meaning that given any positive integer n, the sequence of primes that multiply to n is unique up to reordering.

We'll tackle these two parts separately:

Proof. First, let's prove the existence. Actually, the existence proof is already given in the main text; we will just state it here a bit more formally.

To be precise, we will prove the following statement for $n \geq 1$ by induction:

"Every positive integer $\leq n$ has a prime factorization."

- The *base case*. This is obvious since 1 is a product of zero primes, or the empty product.
- The *inductive case*. We want to show that $n + 1$ has a prime factorization. If $n + 1$ is prime, then we're done. Otherwise, $n + 1$ is not prime, so it has a factor d other than 1 and itself, so $1 < d < n + 1$. By definition of divisor, there is an e such that $de = n$. But e is also neither 1 nor $n + 1$ since that would imply that d is $n + 1$ or 1, respectively. Therefore, $1 < e < n + 1$.

  But this means that $d \leq n$ and $e \leq n$, so by the inductive hypothesis, d and e can be written as the product of primes. Therefore $n + 1 = de$ itself can be written as a product of primes.

This proves the existence.

[11] A *corollary* is just a theorem that follows easily from another.

To prove the uniqueness (up to reordering), suppose n has two prime factorizations $n = p_1 p_2 \cdots p_k = q_1 q_2 \cdots q_\ell$.

First, note that $p_1 \mid q_1 q_2 \cdots q_\ell$. By Corollary A.1, $p_1 \mid q_i$ for some i. But since q_i is prime, it only has two prime factors, 1 and itself, and since $p_1 \neq 1$, we must have $p_1 = q_i$. Let's now divide both sides of the prime factorizations by p_1, so we have $p_2 p_3 \cdots p_k = q_1 q_2 \cdots q_\ell$ (with q_i understood to be removed).

Next, note that p_2 divides the right hand side again, so using a similar argument, $p_2 = q_{i_2}$ for some $i_2 \neq i$. Then we again divide p_2 from both sides to get $p_3 p_4 \cdots p_k = q_1 q_2 \cdots q_\ell$ (with q_i and q_{i_2} understood to be removed).

In fact, we can continue going like this, producing equal pairs $p_3 = q_{i_3}$, $p_4 = q_{i_4}$, etc., until either the p_js or the q_js run out. But in fact, they have to run out at the same time, otherwise, one side of the equality is 1 while the other is greater than 1. Therefore, the number of p_js and q_js are the same, and the equal pairs we generated show that they are actually just the same factorization, just reordered. Therefore, the prime factorization is unique. □