Fast and Reckless Combinatorics

Aaron Pierce

Combinatorics is for counting, frequently the number of ways we can do something.

There are two basic ways to count the number of ways to do things.

The number of ways you can dress yourself is an example of the first type. Imagine you have 3 shirts, 2 pairs of pants, and 2 pairs of shoes, and you want to know how many outfits you can make. You can think about this problem as a tree. You select a shirt, and you can do that in 3 ways because you have three shirts. Then each of those shirts branch off for their choice of pants, because for each of the 3 hats you can choose between two pairs of pants to wear with it. After you choose your pair of pants, you have two more things to choose from. Multiplying the number of choices has the effect of counting up how many possible paths you could take through the tree. Because each path through the tree represents an outfit we know that this will also be the number of possible outfits. This idea is called the **multiplication principle**.

The second type is a bit easier. If instead of doing related things, like selecting clothes for an outfit, you do unrelated things. For example, how many ways can you give a bone to your dog and some milk to your cat. There's only one way. You just give a bone to the dog and some milk to the cat, we don't care about when it happens. When you do unrelated things, you add the number of things you are doing. This is called the **addition principle**

Now we can complicate things a little. What if you are able to kill two birds with one stone? Maybe you need to name all cities that are capitals of a state or are the largest city in the state. How many cities will you name? If you say Oklahoma City, you have named both a capital and a largest city. If you had just added 50 capitals + 50 largest cities to get your answer you would count Oklahoma City twice, which is no good.

We can fix this by subtracting off the number of elements that get counted twice. This makes sense, because we are counting those elements twice, and then subtracting them once. $2 - 1 = 1$, so we get the correct count. This idea is called the **inclusion-exclusion principle**.

Another idea that is mentioned in textbooks is that if you have ten pidgeons and nine boxes, some box is going to have two pidgeons. That's called the

pidgeonhole principle. There's not a lot to cover here, but it's a useful observation to exploit in proofs later on.

Now we're going to move into permutation and combination land, which is way more useful than you might think.

A **permutation** is an ordered arrangement of elements from a set. The number of ways that you can do that is $n!$, where n is the number of elements in the set. This is for the same reason as choosing outfits. You have n numbers to choose to start your permutation with. After picking any of those, you have the rest of the set ($n-1$ items) left to choose from. This continues until you have one element left, and as you multiply n and $n-1$ and so on you end up doing all the computation for n factorial.

A **combination** is an unordered arrangement of elements from the set. If a permutation is lining things up, a combination is making a soup. If you've got a whole cabinet of soup stuff, maybe 5 different soup ingredients, but you only want to use 3 of them because using all 5 wouldn't make a better soup, how many different soups could you make?

There's a neat way to do this. If you consider every permutation of the ingredients (you line them all up in every possible way) and then choose the first 3 elements in the line, you will have selected all of the possible ways to choose 3 ingredients.

However, we're going to end up making a lot of the same soup. If we have five elements and line them up as (a, b, c, d, e), we will pick off the first 3 to make a soup from (a, b, c). But the next permutation in the list is (a, b, c, e, d). We then pick off (a, b, c) again!

We have 5 elements total and we are picking off the first 3. Therefore any specific 3 ingredients will be picked $(5-3)!$ times, because $5-3$ represents the number of elements that can be permuted to make a new unique permutation but with the same 3 elements as some other permutation, and taking $(5-3)!$ will count the number of times we can take those permutations that give us identical soups.

But we can duplicate that soup in another way, too! If we make a tomato-onion-garlic soup it'll taste the same as an onion-garlic-tomato soup. So any ordering of tomatoes, onions, and garlic will produce the same soup.

Because we are making soup from 3 ingredients, picking 3 fixed ingredients will make 3! identical soups, because there are 3! permutations of those 3 ingredients and order doesn't matter if you're boiling it all together. And don't forget that other duplication problem we had earlier. Each of those 3! identical soups will be duplicated themselves $(5-3)! = 2!$ times as well.

So for each unique soup, that unique soup could be made in $3! \cdot 2!$ identical ways.

We know that we have 5! soups, and we want to know unique soups. We can write this equation as $5! = \mathbf{U} \cdot (3! \cdot 2!)$, meaning that our 5! total soups is our

number of **U**nique soups times 12 duplicate soups that are also being made for each unique soup.

We can solve for **U** by dividing both sides by 12, so $\mathbf{U} = \frac{5!}{3!\cdot 2!} = 10$ unique soups!

Let's recap. We want to know unique soups. We can get every non-unique soup by permuting the list of soup ingredients and picking off however many ingredients we want. Let's call that number r. The number of non-unique soups is equal to the number of unique soups times their multiplicty (the number of times they are duplicated). We can then solve for unique soups by taking the number of nonunique soups over the number of duplicates each unique soup will have.

For n ingredients, this is represented as $\frac{n!}{r!(n-r)!}$. We take $n!$ total non-unique soups and get rid of the number of the duplicates. The duplicates are equal to the number of identical soups we'll make times the number of ways to make a soup that tastes the same as that identical soup.

What if soup order matters? Maybe the garlic will burn if you put it in before the onion, so a tomato-onion-garlic soup is now different from a garlic-onion-tomato soup, because one tastes burnt and the other doesn't. In this case, what we called similar tasting soups before will no longer taste similar. Because we care about those now, we do not want to remove the 3! soup taste-alikes, but we still don't want to count two identical soups. Recall that we permuted all of the ingredients and then grabbed the first 3 from each permutation. One problem was that we could freely swap the 2 ingredients we didn't take to create unique permutations with the same 3 ingredients.

Now that we only care to remove indentical duplicates of soups, we only need to remove a number of soups equal to the number of ways you can permute the ingredients we didn't use. So we can compute this by taking our combination formula and just dropping the part that removed tastealike soups, which gives us $\frac{n!}{(n-r)!}$. This is exactly the formula for an **r-permutation**, the number of permutations you can generate from r elements.

Now that we've made a whole bunch of soups, we need people to eat the soups, so we'll throw a dinner party. The problem is that your two different groups of friends are coming and they absolutely hate each other. You want to eventually find the optimal seating arrangement, but first you are curious as to how many ways you could possibly seat them.

If you sit at a circular table, there's an interesting problem with counting this. If everyone sits at the table, and then everyone gets up and moves one seat to the right, you are still sitting with the same people, the same people are still across from you, and so on. If you were in a featureless round room and you all just suddenly teleported to the next seat over, you would not know anything happened at all. Unless you felt the teleportation happen but that doesn't matter.

Because of this, if you rotate all the seats around the table you get the same seating arrangement, and you only want to count unique seating arrangements.

But here's the neat thing about this problem: if two people just swap seats, it's a totally new permutation. So if you all sit down and then two people switch seats, it's an instant new permutation.

But if you do a ton of swaps, it's possible that you make a full rotation of the table, and oops you have now double counted a permutation. So to avoid doing that, just glue someone to a seat. If someone does not move you cannot possibly make a full rotation. Therefore, if you glue someone's seat down, you can safely know that you cannot swap into a full rotation. So now you perform your swaps. At first, you have $n-1$ people to swap. So there are $n-1$ possible first swaps. And then after you select a swap, you cannot do that swap again otherwise you will get an identical table state again, so for each swap you reduce the number of next possible swaps by 1. That's just a factorial! So after fixing one person, you can make $(n-1)!$ unique swaps.

Another way to think about it is that after gluing your first person, you have $n-1$ seats left. We can get $(n-1)!$ unique permutations of people in those seats, and we know that no permutation is a pure rotation of another because our fixed person cannot possibly rotate, so these will all be unique. This is called a **circular arrangement/permutation**, the number of ways you can arrange people around a table or elements in a circular linked list!

Having had this dinner party, your friends are amped up on soup and they hate each other so much that they get in a fight. You hop into the fight and you have time to throw three punches. How many different ways can you deal some damage? You ask them to stop the fight while you think about some combinatorics, and they understand.

The catch with this problem is that the order in which you punch doesn't matter. If someone takes two punches they take two punches, no matter when those punches came.

So we want unique combinations where elements can be repeated. Order doesn't matter, but where we throw the punches does.

If we have 5 people in the fight and we throw 3 punches, taking 5 choose 3 would give us the number of ways we could throw a punch where we never hit the same guy twice. But there's no rule that says we can't punch the same guy 3 times; maybe some guy said the soup was bad. We need to change how we are computing our choice because a normal 5 choose 3 doesn't let us wail on that guy that said the soup was bad. We used factorials earlier because we weren't allowing repeats. If you choose an element, you have $n-1$ elements to choose next from. But in our fight we always have n elements to choose from. But we can be clever. Let's imagine we did 5 choose 3, but then every time we chose a number we threw it back in the set. We would need to throw our choice back in twice. No need to do it on the third try because we won't be drawing from

the pool again.

Now imagine that we're clairvoyant and we know what our first two punches are going to be. If we just pop those onto the end of the list, and take that list choose 3, we will get unique permutations of all possible punches allowing repeats on that guy we hate now or anyone else that makes us mad. But how can we be clairvoyant? We don't even need to know who we're going to punch. If we just throw in some wildcards that say to punch whoever you punched already again, we will have the same effect. The 1st wildcard means to repeat the first punch, and the second means to repeat whoever you punched second and so on. This way, if we pick 3 elements and get two wildcards, we can rearrange this and pretend like we punched some guy first and only then did we choose our two wildcards because order doesn't matter. Then the first wildcard equals our first punch, and our second wildcard equals our second punch, so it means we hit the same guy three times. Sorry, guy. Don't trash my soup next time.

I found this a little confusing because it seemed like you would double count some things. If the wildcards depend on order, how do you ensure the combinations are unique? Imagine you line up the 5 guys, and number them 1 through 5. Then you throw two wildcards into the lineup, so your set of possible punch locations looks like $\{1, 2, 3, 4, 5, \theta, \delta\}$, where θ means repeat your first punch and δ means repeat the second. There are 7 elements in that set. 7 choose 3 gives you the number of possible punches including repeating a punch on someone because we could choose a wildcard. How do we know that they all result in unique trips to the hospital? The choose operator ensures no element of that set duplicates, but we are worried about what happens when we evaluate θ and δ. How do we know they never create a duplicate?

Let's think about every possible combination of punches we got from 7 choose 3. If the combination doesn't contain a wildcard, we know it will be unique because wildcards repeat elements. No wildcards = no repeated punches, so there's no chance a wildcard could produce a combinations of punches with no repeats, so we can forget about those.

Now we need to show that any combination with a wildcard is unique.

When we added wildcards to our lineup we added one fewer wildcard than punches. This means that all combinations from 7 choose 3 will contain a number. If a combination has two wildcards the combination will evaluate to punching that guy 3 times, which will clearly be unique. But what if you draw $3, \delta, \theta$ and $3, \theta, \delta$? Aren't those the same? You can't draw both of those because we took 7 choose 3, and those are identical combinations of symbols, which 7 choose 3 cannot return.

So our only suspicious candidates left are combinations with two numbers and one wildcard. If you have two numbers and one wildcard, you can only get two possible different combinations, because the wildcard can only evaluate to the first or the second number. So if you've got two numbers a and b, you should get the combinations aab and bba. Who's to say you can't get aab two ways?

Well in order to get aab you must either draw a, b, θ or b, a, δ. If you draw both of those, you will actually double count that combination of punches. However, think about what we started with. Having a, b, and two wildcards should give two values, aab and bba. But we spent both of our wildcards to duplicate a. Therefore, we have no wildcards left to use on the numbers a and b in order to duplicate b to get bba. So we spent two wildcards to double count one value, but that didn't allow us to count our other value. So we count aab twice and bba zero times, so we still get a count of two combinations even though we just counted one twice and neglected the other. That's the ultimate key. You only get so many wildcards, and that many wildcards determines how many combinations you can produce. No matter whether you spend them 'correctly' or you duplicate the same element multiple times, you get the same count, which is all we care about.

This problem is an example of taking a **combination with repetition**. Where order doesn't matter but you are allowed to duplicate elements from your set. The way we did that for a set of n elements and r choices was to add $r-1$ wildcards and choose r elements from our new set. The so we would notate that as

$$\binom{n+(r-1)}{r}$$

Why not add a whole bunch of wildcards? Why limit ourselves to $r-1$? If we added any number of wildcards $\geq r$, we allow the possibility of only drawing wildcards, and then we never actually punched someone to repeat, so that wouldn't make any sense.

Throughout that explanation of the wildcards, and how we can make the soups, there was a common thread about overcounting. This seems to emerge frequently in the combinatorics problems I've seen. One general strategy to solve these problems is to overcount and fix. Let's look at another example of this pattern.

This example isn't as exciting as soup or fights, but how many different words can you create by rearranging letters in the word "pepper"? 'Word' is used loosely here. We'll consider any permutation of letters to be a word. So ppper is a word. But so is ppper, and so is ppper, and so is ppper. Because we're considering a word to be any permutation of letters, you can permute this word multiple different ways to get the same word. For example, one permutation is to swap the third and fourth letter, which results in "pepper" again.

In this case, just saying that we can make 6! permutations of the letters will overcount, so we just need to figure out what we're overcounting. The fundamental problem here is that we have repeated letters. Permutations of "abcde" will always be unique. This is because making permutations is like taking the scrabble tiles "abcde" and putting them in a line in front of you. When you move a letter from your hand to the table in front of you, you can't re-draw that letter because it's not in your hand anymore. However, if we look at "pepper"

again, after you put a p on the table you still have two more p tiles that you could have placed instead of what you chose and had the exact same outcome. So the number of exact same outcomes is just the number of ways we could permute all of the p tiles, which is 3! ways to draw those tiles. However, you could also do the same thing with the e tiles, but the e selections are totally independent of the p selections.

For each unique permutation, think about the number of ways we could duplicate it. We could duplicate it by choosing the p tiles in different orders, and you can do that 3! ways. You could also duplicate it by choosing the e tiles in one of the 2! different orders. But these are totally independent of each other. If the unique permutation we're looking at is *eppper*, you cannot choose a p first because e is the first letter. So now you can kind of re-order the tiles. We're still looking at the permutation *epper* but just imagine that you can format it as *eepppr* and then move the other e back where it should be. This clearly gives you $2! \cdot 3!$ posssible permutations. You order the e letters in some way, which forms two branches of the tree, and then each branch has 3! branches. So we take the total number of letters 6! and divide off the number of branches in the tree, 3!2!

That illuminates a more broad pattern about permutations with repetition. The way to arrange a collection of elements with repeats is the number of ways to permute all of the letters over the number of ways to permute the duplicate letters. So our example looked like $\frac{6!}{3!2!}$. In general you take $\frac{n!}{r_1!r_2!\ldots}$ where n is the number of letters, or total elements, and each r represents the number of times any given element repeats. So in our case $r_1 = 3$, the number of p tiles, and $r_2 = 2$, the number of e tiles. We could also include $r_3 = 1$, for our single r tile, but that won't change anything. This is the formula for a **permutation with repetition**.

I am not going to pretend like there is a nice segway into the next topic, nor am I going to pretend that I know exactly why it fits in, but it's on the test, so here we go. If you take a binomial, which is just two terms added together, like $(x + 1)$, and raise it to some power, we can use combinatorics to figure out the coefficients of each term and find out some cool stuff about combinations along the way.

The idea is that by taking $(x+1)^4$, you are really computing $(x+1)(x+1)(x+1)(x+1)$. That computation comes out to $x^4 + 4x^3 + 6x^2 + 4x + 1$. But where did those coefficients come from? We multiplied a whole bunch of $(x+1)$ terms together and got a 6 somewhere? What gives?

Well when you actually do that multiplication it looks a little something like

$$(x^2 + x + x + 1)(x + 1)(x + 1) \text{ After one foiling}$$
$$(x^3 + x^2 + x^2 + x + x^2 + x + x + 1)(x + 1) \text{ After the next}$$

And the last line is quite long, but notice that all of those terms have a coefficient of 1, and that pattern will continue. So the only way we got coefficients in that answer I gave above is by adding like terms. So each coefficient is the number of times we make that same term in that fully expanded multiplication.

Let's consider a slightly different binomial to find out what that number will be. Consider $(a+b)^4$. It evaluates to $a^4 + 4a^3b + 6a^2b^2 + 4ab^3 + b^4$. Each of the power of those terms adds up to 4. So the first term is a^4b^0, and $4+0=4$. The second is a^3b^1, and $3+1=4$ too.

So for each of these terms we have 4 'points' to spend (because we rose our binomial to the 4th power), and we can spend those points on either giving a or b a higher power. What's super cool about taking $(a+b)^4$ is that it finds every single way that you could spend those points. Why's that? Well let's pick a term to look at, $6a^2b^2$. You can create a^2b^2 by multiplying 2 as and 2 bs, so one way to get there is to take $aabb$, or $abab$, or $abba$ Who's to say $(a+b)^4$ will do all of those orders? Let's do the multiplication to figure it out.

We start with $(a+b)(a+b)$, giving us $aa+ab+ba+bb$. Now we hit that with another $(a+b)$, giving us $aaa+aba+baa+bba+aab+abb+bab+bbb$. Already looking promising. This is every way to fill 3 slots with any number of as and bs. Hitting that with our final $(a+b)$ gives us $aaaa+abaa+baaa+bbaa+aaba+abba+baba+bbba+aaab+abab+baab+bbab+aabb+abbb+babb+bbbb$. I left some terms that don't matter for this example. Let's leave those behind to finish with $bbaa+baba+baab+aabb+abab+abba$ We've added every single combination of 2 as and 2 bs. We took 4 elements $xyzw$ and chose two of them to be set to a, and the rest were set to b. This is exactly taking 4 choose 2, representing having 4 slots in our multiplication and filling two of them with an a. If we instead said we want all terms that look like a^3b^1, we would take 4 choose 3. Take our 4 points and spend them on 3 as in every way possible. Another way to say that is to take our 4 points and spend one of them on a b. This should produce the exact same set of outcomes. Finding when 3/4 are a and 1/4 are b, or when 1/4 are b and 3/4 are a should be the exact same. So this proves an identity that $\binom{n}{r} = \binom{n}{n-r}$. The idea is what we decided, that choosing r elements from a group to paint red is the same as choosing n-r (the other elements) to be blue. The algebra is also easy, as that syntax is computed by $\frac{n!}{(n-k)!k!}$, for n choose k. Because $(n-r)r!$ and $(n-(n-r))(n-r)!$ are equivalent, the algebra will work out too.

So if you look at n choose k, and vary k from 0 to n, you get something like 1, 5, 10, 10, 5, 1, which has the symmetry that we would expect. Choosing 3 of 5 or 2 of 5, should give you the same number of groupings, you just grab the opposite set of elements for each one.

This has all dealt with the binomial theorem, that the coefficients of some binomial is the number of ways you can multiply out the numbers that make up that term.

A little more generally, any binomial can be expanded as follows:

$$(a+b)^n = \sum_{i=0}^{n} \binom{n}{i} a^n b^{n-i}$$

Which says that every binomial will make each of its terms in every single way possible, and we can count those ways by taking n points and choosing i of them to spend on making as.