Maths and computer science

Articles and notes that are more about the conceptual side of maths and computer science, rather than anything specific to a particular programming language.

- Prefix codes (explained simply)

Prefix codes (explained simply)

> This article was originally published at https://gist.github.com/joepie91/26579e2f73ad903144dd5d75e2f03d83.

A "prefix code" is a type of encoding mechanism ("code"). For something to be a prefix code, the entire set of possible encoded values ("codewords") must not contain *any* values that start with any *other* value in the set.

For example: `[3, 11, 22]` is a prefix code, because none of the values start with ("have a prefix of") any of the other values. However, `[1, 12, 33]` is *not* a prefix code, because one of the values (12) starts with another of the values (1).

Prefix codes are useful because, if you have a complete and accurate sequence of values, you can pick out each value without needing to know where one value starts and ends.

For example, let's say we have the following codewords: `[1, 2, 33, 34, 50, 61]`. And let's say that the sequence of numbers we've received looks like this:

> 1611333425012

We can simply start from the left, until we have the first value:

> **1** 611333425012

It couldn't have been any value other than `1`, because by definition of what a prefix code is, if we have a `1` codeword, none of the other codewords can start with a `1`.

Next, we just do the same thing again, with the numbers that are left:

> 1 **61** 1333425012

Again, it could *only* have been `61` - because in a prefix code, none of the other codewords would have been allowed to start with `61`.

Let's try it again for the next number:

> 1 61 **1** 333425012

Same story, it could only have been a `1`. And again:

> 1 61 1 **33** 3425012

Remember, our set of possible codewords is `[1, 2, 33, 34, 50, 61]`.

In this case, it could only have been a `33`, because again, nothing else in the set of codewords was allowed to start with `33`. It couldn't have been `34` either - even though it *also* starts with a `3` (like `33` does), the lack of a `4` as the second digit excludes it as an option.

You can simply keep repeating this until there are no numbers left:

> 1 61 1 33 34 2 50 1 2

... and now we've 'decoded' the sequence of numbers, even though the sequence didn't contain any information on where one number starts and the next number ends.

Note how the fact that both `33` and `34` start with a `3` didn't matter; shared prefixes are fine, so long as one value isn't *in its entirety* used as a prefix of another value. So while `[33, 34]` is fine (it only shares the `3`, neither of the numbers in its entirety is a prefix of the other), `[33, 334]` would *not* be fine, since `33` is a prefix of `334` in its entirety (`33` followed by `4`).

This only works if you can be certain that you got the *entire* message accurately, though; for example, consider the following sequence of numbers:

> 11333425012

(Note how this is just the last part of 16 **11333425012**)

Now, let's look at the first number - it starts with a `1`. However, we don't know what came before, so is it part of a `61`, or is it just a single, independent `1`? There's no way to know for sure, so we

can't split up this message.

It doesn't work if you violate the "can't start with another value" rule, either; for example, let's say that our codewords are `[1, 3, 12, 23]`, and we want to decode the following sequence:

> 12323

Let's start with the first number. It starts with a `1`, so it could be either `1` or `12`. We have no way to know! In this particular example we can't figure it out from the numbers after it, either, as there are two different ways to decode this sequence:

> 1 23 23

> 12 3 23

And that's why a prefix code is useful, if you want to distinguish values in a sequence that doesn't have explicit 'markers' of where a value starts and ends.