Always use [closed, open) intervals

Sat Oct 08 2022 • programming

Intervals or ranges pop-up everywhere in the programming world. The classic example is picking a start and end date, like you would when booking an AirBnB or a flight. But that's just one example: from slicing a JS Array, to Java's List#sublist and even SQL's `LIMIT` operator, ranges are everywhere.

Have you ever wondered why they are always implemented as `[closed, open)` as opposed to `[closed, closed]`?

Wait, what's a [closed, open) interval?

A closed-open interval, usually denoted as `[a, b)` is just a short form for expressing `a <= x < b` or the set of all values starting from and including `a` up to, but excluding `b`. For example if we're using integers then `[0,5) == 0, 1, 2, 3, 4`.

A closed-closed interval, usually denoted as `[a, b]` is one that *includes* the last value e.g. `[0,5] == 0, 1, 2, 3, 4, 5`

Never, ever, ever use [closed, closed] intervals

A couple of years ago I had the (mis)fortune of working on a system that used [closed, closed] ranges extensively. The system worked well for the most part, but had tons of clunky code to handle a few cases. Here's a few of the cases we had to battle with:

The empty interval

Imagine you want to describe a zero length time interval (i.e. an empty interval) that start's at time `T=1`. With closed-open intervals this is a trivial task, simply `[T,T)`, with closed-closed... not so much. You could try

`[T, T-1]`, but that's a bit clunky and it won't work if `T` is a decimal number.

Splitting by time

Imagine that you want to group users by the hour in the day when they registered to your site. This is essentially the problem of "splitting" the 24 hours of the day in 1-hour intervals.

With `[closed, open)` you get a very nice looking sequence `[0,1), [1,2), [2,3), [3,4), ..., [23,24)`.

With `[closed, closed]`, you get this monstruosity `[0,0:59:59], [1, 1:59:59], ... [23,23:59,59]`. Note that this essentially forces a precicion into your entire system. What happened to users that registered between `0:59:59` and `1`? They will be lost.

Calculating the length of an interval

Another common task, whenever intervals are involved, is to calculate their length.

With `[a, b)` it's trivial, simply `b-a`.

With `[a, b]` you get a few edge cases, for one the length of `[a, a-1]` could be 0 as we saw earlier, or it could be negative if `a < 1`.

Another property that you would expect from a properly implemented range is that splitting a range by half should yield two ranges whose length add up to the original range's length.

For example, splitting the hours in a day into two ranges, should result in 2 ranges with length 12, so that 12+12 = 24. This property is lost when using `[a, b]` ranges.

Final thoughts

When I was writing this article I found a short 1982 note by good ol' Edsger W. Dijkstra on why he preferred `[closed, open)` intervals. I won't bore you with the details, but let's just say that the brilliant people at Xerox PARC tried them, and found that `[a, b]` ranges lead to buggier and more complex code.

I hope this short read has convinced you of the perils and pitfals of using `**[closed, closed]**` intervals. My guess is that the reason people sometimes fall for them is because they look nice and symmetric, and in fact work *most of the time*.

It's only in the edge cases that they start to break down. But that is exactly how you should evaluate how good a design is: by testing it against the edge cases.

If you liked this article, follow me on twitter. I post about once a month on software engineering practices, startup engineering and programming in general.

2022 © Fernando Hurtado Cardenas.