

An Intro to Small String Optimization

wadec@andrew.cmu.edu

Outline

- What are strings/why optimize?
- Technical overview of strings
- How we can optimize

Hey guys! I'm going to be going over an introduction to *small string optimization.* We'll start with a conceptual overview of what strings are, and why we would want to optimize them. Then, we'll have a technical overview of strings that will motivate some really cool optimizations we can make.

An Intro to Small String Optimization: What are strings/why optimize?



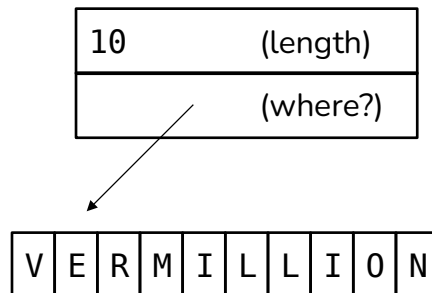
When programmers say “string,” we’re referring to strings of characters.

- Strings are used to store the pages that make up the websites you browse every day.
- They’re used to store your name and passwords so your bank knows who you are when you make a transaction,
- They’re used to store the character you’re playing in a video game.


Strings are just a really really important part of data storage and transfer.

But why optimize? We’ll use databases as our motivating use case. Modern companies are storing millions on millions of database entries, so squeezing out performance over staggering amount of data is key. (Polars, on the handout has some benchmarks that show doing so makes big changes). So let’s say a client wants to dig into their database and run some statistics on users’ favorite color. Maybe vermilion enjoyers pay more for shoes.

An Intro to Small String Optimization: Technical overview of strings

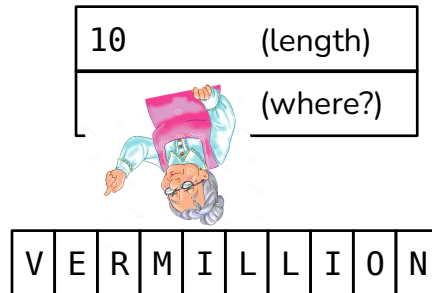


KEY

 one byte, to scale

So, how do we represent the string ``vermillion``? Let's call one of these rectangles one byte. A common string representation stores the length of the string, which might take eight bytes, and the location of the string's contents in the computer (also eight bytes). You can think of this location as a pointer.

An Intro to Small String Optimization: Technical overview of strings



KEY

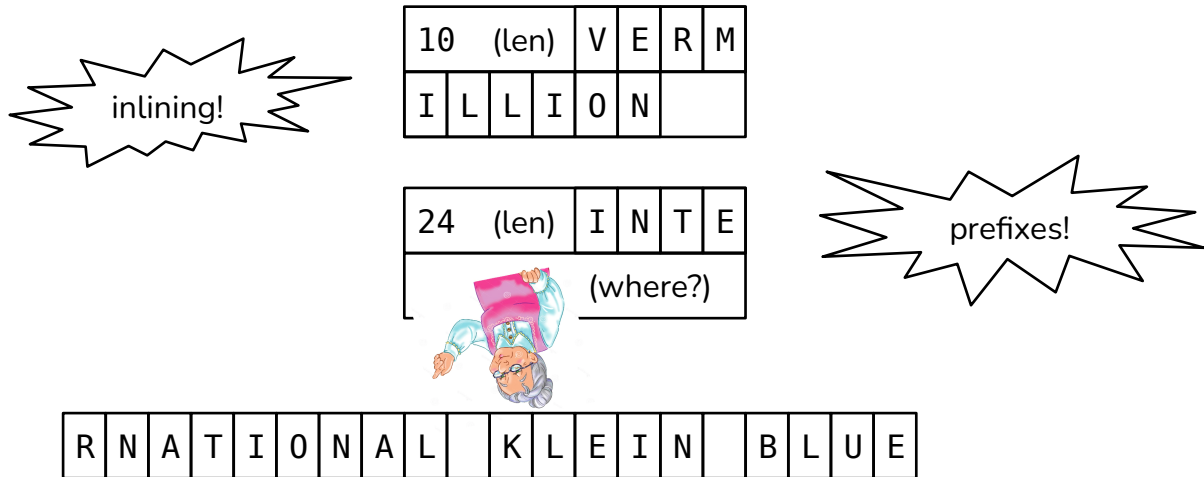


one byte, to scale

But pointers are relatively slow. You might think of ole Granny CPU thinking, oh, where did I put that word? Oh, right, vermillion is here! If we can avoid following these pointers, we'll have much faster code.

How can we do this? The solutions are pretty simple.

An Intro to Small String Optimization: How we can optimize

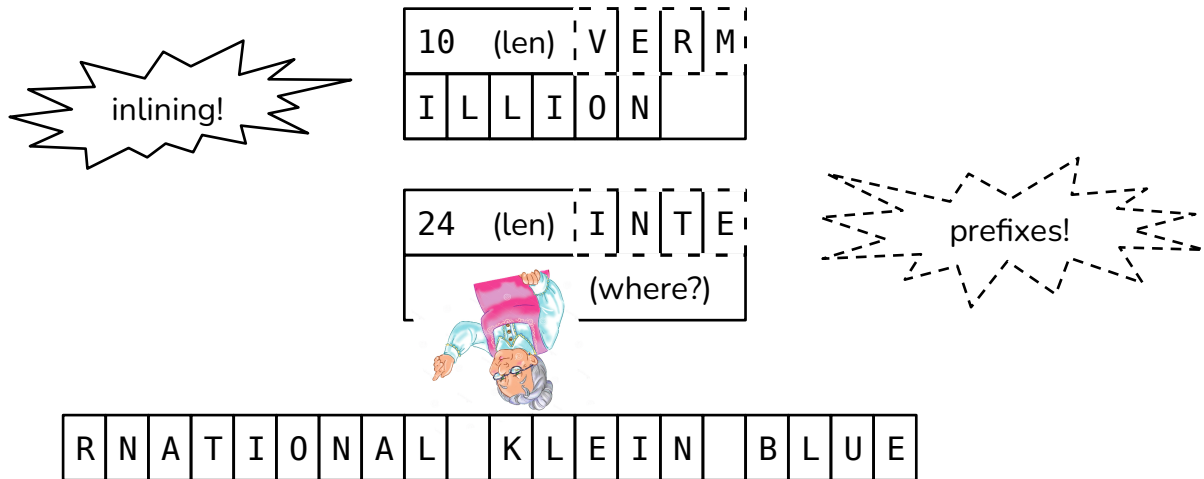


Inlining: just compress the word into the amount of space we were already using! We can shrink the length field down to four bytes, which makes any string up to twelve characters possible to inline.

If they're greater than twelve, we default back to our previous model, where we have a pointer to the string content.

Also:

An Intro to Small String Optimization: How we can optimize



We have prefixes! Whether we have short or long strings, we keep track of the first four characters. This way, many database operations can be done without depending on Granny CPU. Going back to our example, I want to check if this string is the string `vermillion`. Well, `v` and `i` are different, so we're done! We find that the strings are different, without needing to follow a pointer.

So with these two tricks, we put information where we need it in the string data structures to make them much more efficient. Ty <3

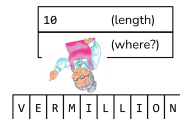
Questions?

Outline

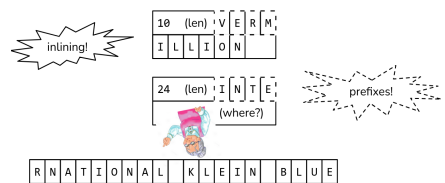
- What are strings/why optimize?



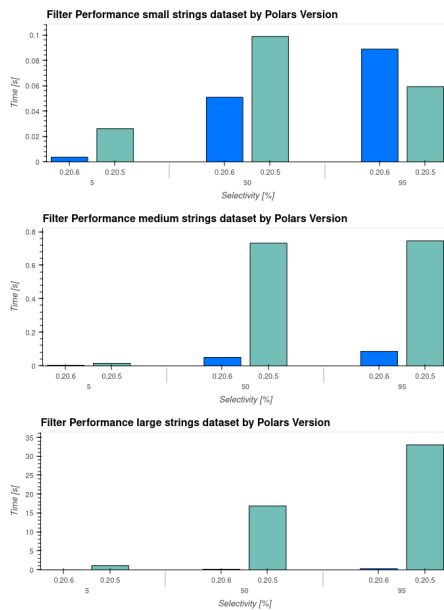
- Technical overview of strings



- How we can optimize



An Intro to Small String Optimization: Unused assets/production notes



<https://pola.rs/posts/polars-string-type/>

H	E	L	L	O	!	
---	---	---	---	---	---	--

H	E	L	L	O	!	:)	
---	---	---	---	---	---	---	---	--

c style strings

Slidedeck produced with Google Slides

Fonts used:

- Nunito (body/titles)
- Consolas (monospace)

Amaranth Deep Purple is a nice sounding color