Ruby master - Feature #12744

Add str.reverse_each_char and str.reverse_chars

09/09/2016 03:45 PM - bouk (Bouke van der Bijl)

Status:	Feedback	
Priority:	Normal	
Assignee:		
Target version:		

Description

This patch adds str.reverse_each and str.reverse_chars. It's currently not really possible to iterate a Ruby string in reverse while guaranteeing that you're not accidentally introducing an $O(N^{2)}$ bug, without encoding to a fixed-length encoding like UTF-32. This is because variable-length encodings like UTF-8 requiring iterating over the whole string if you want to address characters by index.

The patch uses rb_enc_left_char_head to iterate over the string in reverse, so you can do so without allocating more memory.

History

#1 - 09/09/2016 04:58 PM - bouk (Bouke van der Bijl)

- Subject changed from Add str.reverse_each and str.reverse_chars to Add str.reverse_each_char and str.reverse_chars

#2 - 09/10/2016 02:58 AM - duerst (Martin Dürst)

- Status changed from Open to Feedback

What about using

str.reverse.chars for string.reverse_chars?

It allocates some memory, but compared to all the memory allocated for the individual characters, the memory for the overall reversed string is not a big deal. Also, it's O(N).

What's the use case? It doesn't look like a very frequent operation needing a dedicated method if it can be done by method composition so easily.

Similar for
str.reverse_each, which you probably meant to be str.reverse_each_line, which should be something like string.lines.reverse.each

#3 - 09/12/2016 06:58 PM - bouk (Bouke van der Bijl)

Martin Dürst wrote:

> What about using
>
> str.reverse.chars for string.reverse_chars?
>
> It allocates some memory, but compared to all the memory allocated for the individual characters, the memory for the overall reversed string is not a big deal. Also, it's O(N).
>
> What's the use case? It doesn't look like a very frequent operation needing a dedicated method if it can be done by method composition so easily.
>
> Similar for
> str.reverse_each, which you probably meant to be str.reverse_each_line, which should be something like string.lines.reverse.each

I don't really have a use case for reverse_chars, but I added it for symmetry with the other methods. I meant str.reverse_each_char, I typo'd it in the issue but it's correct in the patch. The equivalent with doing allocation would be str.chars.reverse.each. I could use reverse_each_char in Sprockets, where we need to iterate over the string backwards to check that it ends with certain characters (and know what it ends with). This needs to be done many times when compiling assets, so having a native way to iterate characters without allocation is a beneficial optimization.

#4 - 09/13/2016 12:21 AM - shyouhei (Shyouhei Urabe)

I doubt if we can make a reverse_each_char which is faster than reverse.each_char. It Is not always clear where is a boundary between a character and another, especially when scanning backwards. We might end up scanning whole string from the beginning, splitting characters into separate substrings, then iterate over them.

#5 - 09/13/2016 05:12 PM - bouk (Bouke van der Bijl)

Shyouhei Urabe wrote:

> I doubt if we can make a reverse_each_char which is faster than reverse.each_char. It ls not always clear where is a boundary between a character and another, especially when scanning backwards. We might end up scanning whole string from the beginning, splitting characters into separate substrings, then iterate over them.

Not sure why you think we can't make it faster than reverse.each_char, I've already implemented it and attached the patch. It uses rb_enc_left_char_head, which is implemented by all the encodings to scan a string backwards.

For the most common encoding (UTF8) it is always possible to scan a string backwards from any point, and looking at the other encodings implemented in Ruby it seems only gb18030 has a stateful way to back up to previous characters, so iterating backwards over that one could end up being $O(N^{2)}$.

#6 - 09/16/2016 10:41 AM - duerst (Martin Dürst)

Bouke van der Bijl wrote:

> I don't really have a use case for reverse_chars, but I added it for symmetry with the other methods.

Other languages may do that, but Ruby doesn't add something just for symmetry.

> I meant str.reverse_each_char, I typo'd it in the issue but it's correct in the patch. The equivalent with doing allocation would be str.chars.reverse.each. I could use reverse_each_char in Sprockets, where we need to iterate over the string backwards to check that it ends with certain characters (and know what it ends with).

Wouldn't this usually be done with a Regexp? If using a Regexp directly isn't efficient, what about just applying the reverse of the Regexp to the reverse of the string (so that it gets applied from the start)?

> Not sure why you think we can't make it faster than reverse.each_char, I've already implemented it and attached the patch. It uses rb_enc_left_char_head, which is implemented by all the encodings to scan a string backwards.

Some of these implementations are not exactly trivial. Please look at enc/shift_jis.c or enc/gb18030.c. Please try your code on something like

```
"\x95\x95".force_encoding('Shift_JIS') * x
```

where you increase x and see whether the time increases linearly or not.

> For the most common encoding (UTF8) it is always possible to scan a string backwards from any point, and looking at the other encodings implemented in Ruby it seems only gb18030 has a stateful way to back up to previous characters, so iterating backwards over that one could end up being $O(N^{2)}$.

Yes indeed.

Files

add-reverse-string-iteration.patch	5.91 KB	09/09/2016	bouk (Bouke van der Bijl)