Ruby trunk - Feature #13110

Byte-based operations for String

01/06/2017 04:02 PM - shugo (Shugo Maeda)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

How about to add byte-based operations for String?

```
s = "□□□□□□□□□□"
p s.byteindex(/□□/, 4) #=> 18
x, y = Regexp.last_match.byteoffset(0) #=> [18, 24]
s.bytesplice(x...y, "□□□")
p s #=> "□□□□□□□□□□"
```

History

#1 - 01/06/2017 04:04 PM - shugo (Shugo Maeda)

- File byteindex.diff added

#2 - 01/06/2017 04:24 PM - shugo (Shugo Maeda)

Let me share my use case.

I'm implementing a text editor, in which the point of a buffer is represented by a byte-based offset for speed.
We can get substrings of a buffer by byteslice, but we need force_encoding(Encoding::ASCII_8BIT) to change
the contents of a buffer because we don't have bytesplice:

https://github.com/shugo/textbringer/blob/d69d4daddc268a95b1a51dbe397bff8fc2d587f9/lib/textbringer/buffer.rb#L32
https://github.com/shugo/textbringer/blob/d69d4daddc268a95b1a51dbe397bff8fc2d587f9/lib/textbringer/buffer.rb#L135

We also need force_encoding(Encoding::ASCII_8BIT) for text search:

https://github.com/shugo/textbringer/blob/d69d4daddc268a95b1a51dbe397bff8fc2d587f9/lib/textbringer/buffer.rb#L472

In this case, searching by a regular expression like /[□□□□□]/ fails.

String#bytesplice, String#byteindex, and MatchData#byteoffset are useful for these purposes.

#3 - 01/06/2017 09:31 PM - normalperson (Eric Wong)

Kirk Haines wyhaines@gmail.com wrote:

> My software makes use of buffers of network data where the character
> encodings are irrelevant. They are just streams of bytes being moved
> around, and sometimes manipulated along the way, and the byte based
> operations that you propose would be extremely useful.

Agreed to some extent. I think we could also use a byteslice!
bang method to efficiently deal with partial writes. But then,
! probably isn't safe on user-supplied data with Rack or most
frameworks; only a custom vertically-integrated network apps...

For reading and parsing operations, I'm not sure they're needed
because IO#read/read_nonblock/etc all return binary strings when
passed explicit length arg; and //n exists for Regexp. (And any
socket server reading without a length arg would be dangerous)

#4 - 01/06/2017 11:54 PM - shugo (Shugo Maeda)

Eric Wong wrote:

> For reading and parsing operations, I'm not sure they're needed
> because IO#read/read_nonblock/etc all return binary strings when
> passed explicit length arg; and //n exists for Regexp. (And any

> socket server reading without a length arg would be dangerous)

Let me clarify my intention.

I'd like to handle not only singlebyte characters but multibyte characters efficiently by byte-based operations.

Once a string is scanned, we have a byte offset, so we don't need scan the string from the beginning, but we are forced to do it by the current API.

In the following example, the byteindex version is much faster than the index version.

```
lexington:ruby$ cat bench.rb
require "benchmark"

s = File.read("README.ja.md") * 10

Benchmark.bmbm do |x|
  x.report("index") do
    pos = 0
    n = 0
    loop {
      break unless s.index(/\p{Han}/, pos)
      n += 1
      _, pos = Regexp.last_match.offset(0)
    }
  end
  x.report("byteindex") do
    pos = 0
    n = 0
    loop {
      break unless s.byteindex(/\p{Han}/, pos)
      n += 1
      _, pos = Regexp.last_match.byteoffset(0)
    }
  end
end
lexington:ruby$ ./ruby bench.rb
Rehearsal -----------------------------------------
index       1.060000   0.010000   1.070000 (  1.116932)
byteindex   0.000000   0.010000   0.010000 (  0.004501)
-------------------------------- total: 1.080000sec

                user     system      total        real
index       1.050000   0.000000   1.050000 (  1.080099)
byteindex   0.000000   0.000000   0.000000 (  0.003814)
```

#5 - 01/07/2017 12:55 AM - shugo (Shugo Maeda)

- Tracker changed from Bug to Feature

#6 - 01/07/2017 03:01 AM - nobu (Nobuyoshi Nakada)

- Description updated

How about to let s.index(/□□/n, 4) return 18?

#7 - 01/07/2017 06:40 AM - shugo (Shugo Maeda)

Nobuyoshi Nakada wrote:

> How about to let s.index(/□□/n, 4) return 18?

What does /□□/n represent in this context?

As I stated in the previous comment, I'd like to handle multibyte characters. For example, . should match with a single or multibyte character, not a byte.

#8 - 01/09/2017 09:35 AM - duerst (Martin Dürst)

Shugo Maeda wrote:

> Let me clarify my intention.
>
> I'd like to handle not only singlebyte characters but multibyte characters efficiently by byte-based operations.

What about using UTF-32? It will use some additional memory, but give you the speed you want.

> Once a string is scanned, we have a byte offset, so we don't need scan the string from the beginning, but we are forced to do it by the current API.

One way to improve this is to somehow cache the last used character and byte index for a string. I think Perl does something like this.

This could be expanded to a string with several character index/byte index pairs cached, which could be searched by binary search. All this could (should!) be totally opaque to the Ruby programmer (except for the speedup).

Another way would be to return an Index object that keeps the character and byte indices opaque, but can be used in a general way where speedups are needed.

> In the following example, the byteindex version is much faster than the index version.

Of course it is. (Usually programs in C are faster than programs in Ruby, and this is just moving closer to C, and thus getting faster.)

But what I'm wondering is that using a single string for the data in an editor buffer may still be quite inefficient. Adding or deleting a character in the middle of the buffer will be slow, even if you know the exact position in bytes. Changing the representation e.g. to an array of lines will make the efficiency mostly go away. (After all, editors need only be as fast as humans can type :-).

More generally, what I'm afraid of is that with this, we start to more and more expose String internals. That can easily lead to problems.

Some people may copy a Ruby snippet using byteindex, then add 1 to that index because they think that's how to get to the next character. Others may start to use byteindex everywhere, even if it's absolutely not necessary. Others may demand byte- versions of more and more operations on strings. We have seen all of this in other contexts.

#9 - 01/09/2017 12:41 PM - shugo (Shugo Maeda)

Martin Dürst wrote:

> Shugo Maeda wrote:
>
> > Let me clarify my intention.
> >
> > I'd like to handle not only singlebyte characters but multibyte characters efficiently by byte-based operations.
>
> What about using UTF-32? It will use some additional memory, but give you the speed you want.

UTF-32 is not useful because it's a dummy encoding.

> > Once a string is scanned, we have a byte offset, so we don't need scan the string from the beginning, but we are forced to do it by the current API.
>
> One way to improve this is to somehow cache the last used character and byte index for a string. I think Perl does something like this.
>
> This could be expanded to a string with several character index/byte index pairs cached, which could be searched by binary search. All this could (should!) be totally opaque to the Ruby programmer (except for the speedup).
>
> Another way would be to return an Index object that keeps the character and byte indices opaque, but can be used in a general way where speedups are needed.

Theses ways seem worth considering.

> > In the following example, the byteindex version is much faster than the index version.
>
> Of course it is. (Usually programs in C are faster than programs in Ruby, and this is just moving closer to C, and thus getting faster.)

I don't think it's a language issue but a data structure issue.

> But what I'm wondering is that using a single string for the data in an editor buffer may still be quite inefficient. Adding or deleting a character in the middle of the buffer will be slow, even if you know the exact position in bytes. Changing the representation e.g. to an array of lines will make the efficiency mostly go away. (After all, editors need only be as fast as humans can type :-).

I use a technique called buffer gap described in "The Craft of Text Editing" to improve performance.

https://www.finseth.com/craft/

See Chapter 6 of the book for details.

> More generally, what I'm afraid of is that with this, we start to more and more expose String internals. That can easily lead to problems.
>
> Some people may copy a Ruby snippet using byteindex, then add 1 to that index because they think that's how to get to the next character. Others may start to use byteindex everywhere, even if it's absolutely not necessary. Others may demand byte- versions of more and more operations on strings. We have seen all of this in other contexts.

Doesn't this concern apply to byteslice?

#10 - 01/09/2017 03:03 PM - Eregon (Benoit Daloze)

Shugo Maeda wrote:

> Martin Dürst wrote:
>
> > What about using UTF-32? It will use some additional memory, but give you the speed you want.
>
> UTF-32 is not useful because it's a dummy encoding.

What about UTF-32BE or UTF-32LE?

#11 - 01/09/2017 10:50 PM - shugo (Shugo Maeda)

Benoit Daloze wrote:

> Shugo Maeda wrote:
>
> > Martin Dürst wrote:
> >
> > > What about using UTF-32? It will use some additional memory, but give you the speed you want.
> >
> > UTF-32 is not useful because it's a dummy encoding.
>
> What about UTF-32BE or UTF-32LE?

They are better, but I prefer UTF-8 because it can be used as source encoding and has
better interoperability with other libraries.

#12 - 01/20/2017 10:21 AM - duerst (Martin Dürst)

Benoit Daloze wrote:

> Shugo Maeda wrote:
>
> > UTF-32 is not useful because it's a dummy encoding.
>
> What about UTF-32BE or UTF-32LE?

Yes, that's what I meant.

Shugo Maeda wrote:

> Martin Dürst wrote:

> Shugo Maeda wrote:
>
> But what I'm wondering is that using a single string for the data in an editor buffer may still be quite inefficient. Adding or deleting a character in the middle of the buffer will be slow, even if you know the exact position in bytes. Changing the representation e.g. to an array of lines will make the efficiency mostly go away. (After all, editors need only be as fast as humans can type :-).

I use a technique called buffer gap described in "The Craft of Text Editing" to improve performance.
https://www.finseth.com/craft/
See Chapter 6 of the book for details.

The "buffer gap" technique is very well known, I'm familiar with it since the early 90ies. I was thinking about it, but I think it won't work with UTF-8. If you have figured out how you would make it work with UTF-8, then please tell us.

Here is why I think it won't work with UTF-8. The problem is that you can't move characters from before the gap to after or the other way round and change them when there are edits. If some characters are changed, they might change their byte length. But if you want to keep the string as valid UTF-8, you have to constantly fix the content of the gap. One could imagine using two separate String objects, one for before the gap and one for after. For before the gap, it actually might work quite well (as long as Ruby doesn't shorten the memory allocated to a string when the string contents is truncated), but for after the gap, it won't work, because every insertion or deletion at the end of the gap will make the string contents shift around.

> More generally, what I'm afraid of is that with this, we start to more and more expose String internals. That can easily lead to problems.
>
> Some people may copy a Ruby snippet using byteindex, then add 1 to that index because they think that's how to get to the next character. Others may start to use byteindex everywhere, even if it's absolutely not necessary. Others may demand byte- versions of more and more operations on strings. We have seen all of this in other contexts.

Doesn't this concern apply to byteslice?

Yes, it does. The less we have of such kinds of methods, the better.

Anyway, one more question: Are you really having performance problems, or are you just worried about performance? Compared to today's hardware speed, human editing is extremely slow, and for most operations, there should be on delay whatever.

Regards, Martin.

#13 - 01/20/2017 12:26 PM - shugo (Shugo Maeda)

> The "buffer gap" technique is very well known, I'm familiar with it since the early 90ies. I was thinking about it, but I think it won't work with UTF-8. If you have figured out how you would make it work with UTF-8, then please tell us.
>
> Here is why I think it won't work with UTF-8. The problem is that you can't move characters from before the gap to after or the other way round and change them when there are edits. If some characters are changed, they might change their byte length. But if you want to keep the string as valid UTF-8, you have to constantly fix the content of the gap. One could imagine using two separate String objects, one for before the gap and one for after. For before the gap, it actually might work quite well (as long as Ruby doesn't shorten the memory allocated to a string when the string contents is truncated), but for after the gap, it won't work, because every insertion or deletion at the end of the gap will make the string contents shift around.

In my implementation, the gap is kept filled with NUL, and moved to the end of the buffer
when regular expression search is needed.

> > More generally, what I'm afraid of is that with this, we start to more and more expose String internals. That can easily lead to problems.
> >
> > Some people may copy a Ruby snippet using byteindex, then add 1 to that index because they think that's how to get to the next character. Others may start to use byteindex everywhere, even if it's absolutely not necessary. Others may demand byte- versions of more and more operations on strings. We have seen all of this in other contexts.
>
> Doesn't this concern apply to byteslice?
>
> Yes, it does. The less we have of such kinds of methods, the better.
>
> Anyway, one more question: Are you really having performance problems, or are you just worried about performance? Compared to today's hardware speed, human editing is extremely slow, and for most operations, there should be on delay whatever.

Using character indices was slow, but my current implementation uses ASCII-8BIT strings
whose contents are is encoded in UTF-8, so there's no performance problem while editing
Japanese text whose size is over 10MB.

However, the implementation has the following terrible method:

```
    def byteindex(forward, re, pos)
      @match_offsets = []
      method = forward ? :index : :rindex
      adjust_gap(0, point_max)
      if @binary
        offset = pos
      else
        offset = @contents[0...pos].force_encoding(Encoding::UTF_8).size
        @contents.force_encoding(Encoding::UTF_8)
      end
      begin
        i = @contents.send(method, re, offset)
        if i
          m = Regexp.last_match
          if m.nil?
            # A bug of rindex
            @match_offsets.push([pos, pos])
            pos
          else
            b = m.pre_match.bytesize
            e = b + m.to_s.bytesize
            if e <= bytesize
              @match_offsets.push([b, e])
              match_beg = m.begin(0)
              match_str = m.to_s
              (1 .. m.size - 1).each do |j|
                cb, ce = m.offset(j)
                if cb.nil?
                  @match_offsets.push([nil, nil])
                else
                  bb = b + match_str[0, cb - match_beg].bytesize
                  be = b + match_str[0, ce - match_beg].bytesize
                  @match_offsets.push([bb, be])
                end
              end
              b
            else
              nil
            end
          end
        else
          nil
        end
      ensure
        @contents.force_encoding(Encoding::ASCII_8BIT)
      end
    end
```

As long as copy-on-write works, the performance of the code would not be so bad, but it looks terrible.

A text editor is just an example, and my take is that ways to get byte offsets should be provided because we already have byteslice. Otherwise, byteslice is not so useful.

#14 - 03/01/2017 04:57 AM - matz (Yukihiro Matsumoto)

Are byteindex and byteoffset good enough for your use-case?
Should byteoffset be byteoffsets since it returns both edge?

Matz.

#15 - 03/01/2017 07:39 AM - shugo (Shugo Maeda)

Yukihiro Matsumoto wrote:

> Are byteindex and byteoffset good enough for your use-case?

I also want bytesplice, but its priority is lower.

> Should byteoffset be byteoffsets since it returns both edge?

byteoffsets may be better, but we should consider consistency with MatchData#offset.

#16 - 03/13/2017 02:15 PM - shyouhei (Shyouhei Urabe)

We looked at this issue at today's developer meeting.

I remember no one there had strong pro- or contra against byteoffset and byteindex. But what about bytesplice? Seems a programmer can pass arbitrary byte index to it. That can be troublesome because the passed index might be a middle of some code point.

#17 - 03/13/2017 03:05 PM - shugo (Shugo Maeda)

shyouhei (Shyouhei Urabe) wrote:

> We looked at this issue at today's developer meeting.
>
> I remember no one there had strong pro- or contra against byteoffset and byteindex. But what about bytesplice? Seems a programmer can pass arbitrary byte index to it. That can be troublesome because the passed index might be a middle of some code point.

It's relatively easy to implement bytesplice using force_encoding at my own risk, so I withdraw the proposal to introduce bytesplice.

Files

byteindex.diff	2.83 KB	01/06/2017	shugo (Shugo Maeda)