Adding `same_all?` for checking whether all items in an Array are same

Today, I needed to write a code to judge whether all items in an Array are the same. I wanted to make the following expression, which I've written first, more efficient.

```ruby
ary.all? { |e| e.foo == ary[0].foo }
```

I considered about the following simpler case, too.

```ruby
ary.all? { |e| e == ary[0] }
```

As I discussed with some CRuby committers and my colleagues at Speee, Inc., I found that both cases can be written more efficiently as follows:

```ruby
# for the 1st case
ary.empty? || ary[0].foo.yield_self { |e0| ary[1..-1].all? { |e| e.foo == e0 } } %

# for the 2nd case
ary.empty? || ary[0..-2] == ary[1..-1]
```

However, it is not easy to understand the intent of either expression, which is to check whether all items are the same.

I want to give this feature a clear name to make code readable. And I think it should be provided as a core feature because it can be made more efficient by being implemented in C language.

The benchmark script is: https://gist.github.com/mrkn/26a0fcfc431a45fe809fbbe95aceaf5

I used it to find the efficient expressions. The example result of this benchmark on my MacBook Pro is here.

```
$ ruby -v benchmark.rb
ruby 2.6.0dev (2018-02-14 trunk 62402) [x86_64-darwin16]
```

### Benchmark case: shuffle

<table>
<thead>
<tr>
<th>user</th>
<th>system</th>
<th>total</th>
<th>real</th>
</tr>
</thead>
<tbody>
<tr>
<td>all?-method-1 0.001525</td>
<td>0.000088</td>
<td>0.001613 ( 0.001632)</td>
<td></td>
</tr>
<tr>
<td>all?-method-2 0.000489</td>
<td>0.000031</td>
<td>0.000520 ( 0.000565)</td>
<td></td>
</tr>
<tr>
<td>all?-method-3 0.000444</td>
<td>0.000039</td>
<td>0.000483 ( 0.000482)</td>
<td></td>
</tr>
<tr>
<td>all?-item 0.000325</td>
<td>0.000078</td>
<td>0.000403 ( 0.000402)</td>
<td></td>
</tr>
<tr>
<td>opt-method 0.655959</td>
<td>0.033814</td>
<td>0.689773 ( 0.708515)</td>
<td></td>
</tr>
<tr>
<td>opt-item 0.000316</td>
<td>0.000001</td>
<td>0.000317 ( 0.000317)</td>
<td></td>
</tr>
</tbody>
</table>

### Benchmark case: tail0

<table>
<thead>
<tr>
<th>user</th>
<th>system</th>
<th>total</th>
<th>real</th>
</tr>
</thead>
<tbody>
<tr>
<td>all?-method-1 9.412810</td>
<td>0.231126</td>
<td>9.643936 ( 9.681118)</td>
<td></td>
</tr>
<tr>
<td>all?-method-2 5.375075</td>
<td>0.137908</td>
<td>5.512983 ( 5.754550)</td>
<td></td>
</tr>
<tr>
<td>all?-method-3 5.226132</td>
<td>0.167640</td>
<td>5.393772 ( 5.507031)</td>
<td></td>
</tr>
<tr>
<td>all?-item 0.873700</td>
<td>0.007545</td>
<td>0.881245 ( 0.917210)</td>
<td></td>
</tr>
<tr>
<td>opt-method 5.319648</td>
<td>0.172547</td>
<td>5.492195 ( 5.633140)</td>
<td></td>
</tr>
<tr>
<td>opt-item 0.174349</td>
<td>0.001974</td>
<td>0.176323 ( 0.183002)</td>
<td></td>
</tr>
</tbody>
</table>

### Benchmark case: head0

<table>
<thead>
<tr>
<th>user</th>
<th>system</th>
<th>total</th>
<th>real</th>
</tr>
</thead>
</table>

03/09/2022
History

#1 - 02/14/2018 04:32 PM - dsferreira (Daniel Ferreira)

Usually I use the following code to achieve the same purpose:

```ruby
ary.uniq.size < 2
```

I didn't test but if you need to use that complex code I expect this example to be less performant.

Instead of Array#same_all? why not Array#uniq?

#2 - 02/14/2018 04:37 PM - Eregon (Benoit Daloze)

Why would it be more efficient in C?

A sufficiently smart JIT (or of course it can be done manually) could turn

```ruby
ary.all? {|e| e.foo == ary[0].foo }
```

into (assuming foo has no side effects):

```ruby
e0 = ary[0].foo
eray.all? {|e| e.foo == e0 }
```

and then the only performance difference is the extra comparison of ary[0].foo with itself, which could be avoided by:

```ruby
e0 = ary[0].foo
{1...ary.size}.all? {|i| ary[i].foo == e0 }
```

#3 - 02/14/2018 04:45 PM - dsferreira (Daniel Ferreira)

I believe we could even do:

```ruby
ary.uniq?(&:foo)
```

#4 - 02/14/2018 05:15 PM - mrkn (Kenta Murata)

uniq scans all elements, whereas all? and == don't.
And uniq allocates a new array, so uniq is always slower than all?-method-3 and opt-item in all cases of my benchmark.

Eregon, if we write it in C, we can avoid using all?. It reduces the number of block calls.

#5 - 02/14/2018 05:45 PM - dsferreira (Daniel Ferreira)

mrkn (Kenta Murata) wrote:

uniq scans all elements, whereas all? and == don't.
And uniq allocates a new array, so uniq is always slower than all?-method-3 and opt-item in all cases of my benchmark.

That makes sense.

I suppose that Array#uniq? could work as Array#all? returning once false?
I suppose that `Array#uniq?` could work as `Array#all?` returning once false?

We don't have `Array#uniq?`. Do you suggest adding such a new method?

That is what I meant with:

"Instead of `Array#same_all?` why not `Array#uniq?`"

Array#uniq? makes sense to me and maps well with the logic of `ary.uniq.size < 2` that I mentioned earlier.

If we return earlier from the method at the first false we will have what you are looking for isn't that right, or am I missing something?

I also think it is a method that will simplify some code and people will use it.
I will use it for sure.
Developing it in C is even better because it will guarantee the best performance results.

Four points.
First, re. uniq?:

mrkn (Kenta Murata) wrote:

We don't have `Array#uniq?`. Do you suggest adding such a new method?

I remember having suggested a method named uniq?. Unfortunately, I didn't find the issue (yet). But the meaning was different. The method checked whether all elements were unique, not whether all elements were the same. In other words, it was:

```ruby
def uniq?
  uniq.length == length
end
```

In an efficient implementation, it would return false as soon as it detected two elements that were the same, and true otherwise.

Second, about naming:

`same_all?` doesn't sound good to me. The two words 'same' and 'all' don't appear in this order in English. What about `all_same?`, or better even, `all_equal??`

Third, about need:

It's clear that when you implement something in C, it should get faster. If that were the only criterion for accepting new methods into Ruby, we'd have many more methods. The way you describe it, my understanding is that you only just today had a need for such functionality, not before. I think there should be some indication that this is needed regularly in order to accept it.

Fourth, about alternative ways to write it. What about

```ruby
ary.each_cons(2).all? { |a, b| a == b }
```

Array#uniq? sounds like that all elements are unique, `ary.uniq.size == ary.size`.

Array#uniq? sounds like that all elements are unique, `ary.uniq.size == ary.size`.

I agree it makes more sense for that situation.
I agree with Martin. Perhaps alternatives such as all_same? or even better, all_equal? could be explored/used.

Nobu wrote:

Array#uniq? sounds like that all elements are unique, ary.uniq.size == ary.size.

Agreed.

Perhaps someone can ask matz in the dev meeting about the names. :)

#12 - 03/15/2018 05:42 AM - matz (Yukihiro Matsumoto)
- Status changed from Assigned to Rejected

Rejected. Unfortunately, the incompatibility this proposal would bring is too big. Besides that, we have performance concern too.

Matz.

#13 - 03/15/2018 05:44 AM - matz (Yukihiro Matsumoto)
- Assignee changed from matz (Yukihiro Matsumoto) to mrkn (Kenta Murata)
- Status changed from Rejected to Assigned

Ah, sorry. Posted to the wrong proposal.

Regarding this issue, we have the naming issue. I agree to add the functionality.

Matz.

#14 - 03/15/2018 07:19 AM - matz (Yukihiro Matsumoto)
I am not satisfied with any of the candidates.

- same_all? - weird word order
- all_same? - word order is OK, but there's an ambiguity that "same" means equality or identical.
- all_equal? - the name suggests comparison is done by equal?
- uniform? - I like this best, but can uniform mean equality?

Matz.

#15 - 03/15/2018 07:42 AM - duerst (Martin Dürst)
matz (Yukihiro Matsumoto) wrote:

- uniform? - I like this best, but can uniform mean equality?

When seeing uniform?, one thing I think about is that it means "Are all elements of the same type?".

Example:

```
[1, 2, 3].uniform? % => true
[a, b, c].uniform? % => true
[1, b, 'c'].uniform? % => true
```

#16 - 03/15/2018 07:54 AM - duerst (Martin Dürst)
duerst (Martin Dürst) wrote:

Example:

```
[1, 2, 3].uniform? % => true
[a, b, c].uniform? % => true
[1, 'b', 'c'].uniform? % => true
```

Sorry, this should have been:

```
[1, 2, 3].uniform? % => true
[a, b, c].uniform? % => true
[1, 'b', 'c'].uniform? % => false
```
How about all_equivalent? or all_duplicates??

I hope this new feature never makes it.

enum.each_cons(2).all? { _1 == _2 } is simply superior and more versatile.

Moreover, I'm not convinced that the frequency of this use warrants a new feature, especially that good alternatives exist and all have similar performance.

I wish we could write enum.each_cons(2).all?(&:==), but I'll add that to issues with yielding arity that nobody seems interested in discussing.