Ruby master - Feature #16039

Array#contains? to check if one array contains another array

08/02/2019 06:09 PM - cha1tanya (Prathamesh Sonpatki)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

I woud like to propose Array#contains? which will check if the one array is part of another array.
Implementation can be as follows:

```
def contains?(other)
  (other - self).empty?
end
```

Some test cases:

```
[1, 2, 3].contains?([2, 3]) => true
[1, 2, 3].contains?([]) => true
[1, 2, 3].contains?([1, 2, 3, 4]) => false
[].contains?([]) => true
[].contains?([1, 2, 3, 4]) => false
```

History

#1 - 08/02/2019 06:10 PM - cha1tanya (Prathamesh Sonpatki)

- Description updated

#2 - 08/02/2019 07:36 PM - Eregon (Benoit Daloze)

contains? sounds like it would check if an element is part of the Array, i.e., like include?.
So I think the name is problematic.

superset? would be a better name, and that's defined on Set.
So I think in this case it's better to simply use a Set like Set[1,2,3].superset?(Set[2,3]).

FWIW, Array#- already uses a Hash or Set internally, so an efficient implementation has to use a Set for such functionality anyway.
I don't it's valuable to hide this behavior by having a method on Array, I think it's better to be explicit and use Set directly in this case.

#3 - 08/03/2019 05:50 AM - cha1tanya (Prathamesh Sonpatki)

Agree that superset is better name. Here is the actual use case:

```
Project.pluck(:id).contains?([1,2,3])
```

Where Project.pluck returns an array of integers.

To use set, I have to convert the array to a set.

```
Project.pluck(:id).to_set.superset?(Set[1,2,3])
```

I wanted to avoid creating Set objects just for the purpose of this check so my motivation was to have such method on Array.

#4 - 08/03/2019 08:49 AM - shevegen (Robert A. Heiler)

I think the use case is ok - you want to find out whether an Array or
an Array-like object, is contained in another object (container in a
container in a container ...).

My biggest problem with this is that #contains? is similar in meaning
to #include?, even though they do slightly different things. (By the way
I think for consistency, it would have to be #contain? rather than
#contains?, similar to #include? rather than #includes?).

I am not sure if superset is a better name; to me it conveys a different
meaning than #contains?; but one advantage that superset may have is

that it would not conflict with e. g. #include?, whereas I feel that #contains? would do so more.

> I wanted to avoid creating Set objects just for the purpose of this check so my motivation was to have such method on Array.

#contains? would be simpler to read in your example indeed :) - but I think it could lead to ruby users wondering when to use #include? and when to use e. g. #contain?, and I am not sure this would be good. We already have people wondering when to use strings and when to use symbols in a Hash. Keeping things simple(r) would be good, IMO. ;)

#5 - 08/03/2019 12:58 PM - Eregon (Benoit Daloze)

cha1tanya (Prathamesh Sonpatki) wrote:

> I wanted to avoid creating Set objects just for the purpose of this check so my motivation was to have such method on Array.

Array#- and any efficient (O(n+m) and not O(n*m), n the size of the LHS, m the size of the RHS) implementation of a superset check needs to use some kind of Hash internally.
So you might save a Set allocation, but internally it has to allocate a Hash anyway, so I don't think there is much of a difference, performance-wise.

I would recommend defining a helper method like you did above if you use this frequently in your code base.

#6 - 08/03/2019 08:40 PM - ahvetm (Erik Madsen)

I think this is a great proposal in terms of having one of those nice, useful methods easily available directly on the class you're interacting with, similar to Array#last which can quite verbose in other languages.

I would propose the name #include_all? or something similar to make it obvious that you're comparing it with another array.

#7 - 08/04/2019 01:25 AM - sawa (Tsuyoshi Sawada)

I am not a fan of this feature, but by analogy from Range, cover? may be a better name.

#8 - 12/14/2019 01:33 AM - Dan0042 (Daniel DeLorme)

There's some similarity with #15198, to the point that I can re-use my suggestion from there:

It might make sense to use ary1.to_set.superset?(ary2). That way it makes explicit the fact that ary1 must be converted to a set. But Set#superset? would have to support any Enumerable.

#9 - 12/18/2019 07:11 PM - JustJosh (Joshua Stowers)

#cover?

I do not think we should use the name cover? because the types of arguments accepted by Range#cover? would be incompatible with this use case.

For example:

```
(1..3).cover?(2) # true
```

But if Array's implementation worked similarly, we would have the following issue:

```
[1, 2, 3].cover?(2) # true by design of Range#cover?
[1, 2, 3].cover?([2]) # true because all values in argument are also in self
[1, [2], 3].cover?([2]) # ?
```

#superset?

It is worth noting that the unique nature of sets would affect the expected behavior of this method:

```
[1, 2, 3].contains?([1, 2, 2]) # false because self contains only one 2
[1, 2, 3].superset?([1, 2, 2]) # true because duplicates are ignored
```

In my opinion, the unambiguous behavior of superset? is preferable.

Array/Set

Although I personally like array.superset?() more than array.to_set.superset?(), I think Set would benefit from more compatibility with Enumerable. So

I agree with @Dan0042.

I recommend that we update Set#superset?, proper_superset?, subset?, and proper_subset? to accept any Enumerable.

#10 - 12/19/2019 12:28 AM - sawa (Tsuyoshi Sawada)

JustJosh (Joshua Stowers) wrote:

> I do not think we should use the name cover? because the types of arguments accepted by Range#cover? would be incompatible with this use case.
>
> For example:
>
> ```
> (1..3).cover?(2) # true
> ```
>
> But if Array's implementation worked similarly, we would have the following issue:
>
> ```
> [1, 2, 3].cover?(2) # true by design of Range#cover?
> [1, 2, 3].cover?([2]) # true because all values in argument are also in self
> [1, [2], 3].cover?([2]) # ?
> ```

When the argument is an array, it should be understood as the usual case; i.e., it should be interpreted as the $\subset$ relation. Otherwise, it should be considered as the abbreviated form; in such case, it should be interpreted as the $\in$ relation. So

```
[1, [2], 3].cover?([2])
```

should be unambiguously false. To achieve the interpretation that leads to the true output, you need to write:

```
[1, [2], 3].cover?([[2]])
```

That is exactly analogous to how Range#cover? works, and there hasn't been a problem there.

#11 - 12/19/2019 04:52 AM - JustJosh (Joshua Stowers)

sawa (Tsuyoshi Sawada) - I 100% agree with what you are saying. I did a poor job expressing my concerns.

I am nervous that singling out array arguments in the abbreviated form could result in confusion and misuse. This is not a problem for Range#cover?, because a range cannot be composed of other ranges.

Since Set#superset? does not have an abbreviated form, there is less opportunity for misuse.

#12 - 12/19/2019 06:47 AM - sawa (Tsuyoshi Sawada)

JustJosh (Joshua Stowers) wrote:

> sawa (Tsuyoshi Sawada) - I 100% agree with what you are saying. I did a poor job expressing my concerns.
>
> I am nervous that singling out array arguments in the abbreviated form could result in confusion and misuse. This is not a problem for Range#cover?, because a range cannot be composed of other ranges.
>
> Since Set#superset? does not have an abbreviated form, there is less opportunity for misuse.

I understand your point. If that is a concern, then we can simply not allow the abbreviated form for Array#cover?; raise an argument error when the argument is not an array, which is probably what the original proposal in this thread assumed. This somewhat weakly brakes the analogy from Range#cover?, but it should not be a big deal.

(And still, there is an alternative view to not worry about that too much, and just allow the abbreviated form.)