Ruby master - Feature #14473

Add Range#subrange?

02/13/2018 08:58 PM - greggzst (Grzegorz Jakubiak)

Status:	Closed	
Priority:	Normal	
Assignee:	tarui (Masaya Tarui)	
Target version:		

Description

Hi there,

I'd like to propose a method that returns true when a range that the method gets called on is a subrange of a range passed in as an argument.

Example:

```
(2..4).subrange?(1...4)
=> true
(-2..2).subrange?(-1..3)
=> false
```

Associated revisions

Revision 9ca73892 - 09/05/2018 07:06 PM - tarui (Masaya Tarui)

range.c: Range#cover? accepts Range object. [Feature #14473]

```
    * range.c (range_cover): add code for range argument.
      If the argument is a Range, check it is or is not
      covered by the reciver. If it can be treated as a
      sequence, this method treats it that way.
    * test/ruby/test_range.rb (class TestRange): add tests
      for this feature.

      This patch is written by Owen Stephens. thank you!
```

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@64640 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 64640 - 09/05/2018 07:06 PM - tarui (Masaya Tarui)

range.c: Range#cover? accepts Range object. [Feature #14473]

```
* range.c (range_cover): add code for range argument.
      If the argument is a Range, check it is or is not
  covered by the reciver. If it can be treated as a
  sequence, this method treats it that way.
* test/ruby/test_range.rb (class TestRange): add tests
  for this feature.

  This patch is written by Owen Stephens. thank you!
```

Revision 64640 - 09/05/2018 07:06 PM - tarui (Masaya Tarui)

range.c: Range#cover? accepts Range object. [Feature #14473]

```
* range.c (range_cover): add code for range argument.
      If the argument is a Range, check it is or is not
  covered by the reciver. If it can be treated as a
  sequence, this method treats it that way.
* test/ruby/test_range.rb (class TestRange): add tests
  for this feature.

  This patch is written by Owen Stephens. thank you!
```

History

#1 - 02/14/2018 01:37 AM - duerst (Martin Dürst)

- Backport deleted (2.3: UNKNOWN, 2.4: UNKNOWN, 2.5: UNKNOWN)

- Tracker changed from Bug to Feature

#2 - 02/16/2018 06:33 AM - greggzst (Grzegorz Jakubiak)

- Subject changed from Range#subrange? to Add Range#subrange?

#3 - 02/21/2018 04:48 PM - owst (Owen Stephens)

+1 for this suggestion - we have a similar method in our code base, implemented approximately as follows:

```
Range.class_exec do
  def subset?(other)
    raise ArgumentError unless other.is_a?(Range)

    first >= other.first && last <= other.last
  end
end
```

As a Range represents a set of elements, should there also be proper_subset?, superset? and proper_superset? methods (and perhaps their operator aliases) similar to those on Set (http://ruby-doc.org/stdlib-2.4.2/libdoc/set/rdoc/Set.html)?

#4 - 02/21/2018 04:54 PM - owst (Owen Stephens)

So, something like (naive implementation in Ruby):

```
Range.class_exec do
  def subset?(other)
    raise ArgumentError unless other.is_a?(Range)

    first >= other.first && last <= other.last
  end

  def proper_subset?(other)
    raise ArgumentError unless other.is_a?(Range)

    self != other && subset?(other)
  end

  def superset?(other)
    raise ArgumentError unless other.is_a?(Range)

    first <= other.first && last >= other.last
  end

  def proper_superset?(other)
    raise ArgumentError unless other.is_a?(Range)

    self != other && superset?(other)
  end
end

require 'minitest/autorun'
class TestRange < MiniTest::Test
  def test_subset
    assert_equal true, (1..10).subset?(1..10)
    assert_equal true, (2..9).subset?(1..10)
    assert_equal true, (1..9).subset?(1..10)
    assert_equal true, (2..10).subset?(1..10)
    assert_equal false, (0..11).subset?(1..10)
    assert_equal false, (0..10).subset?(1..10)
    assert_equal false, (1..11).subset?(1..10)
    assert_raises(ArgumentError) { (1..10).subset?('not a range') }
  end

  def test_proper_subset
    assert_equal false, (1..10).proper_subset?(1..10)
    assert_equal true, (2..9).proper_subset?(1..10)
    assert_equal true, (1..9).proper_subset?(1..10)
    assert_equal true, (2..10).proper_subset?(1..10)
    assert_equal false, (0..11).proper_subset?(1..10)
    assert_equal false, (0..10).proper_subset?(1..10)
    assert_equal false, (1..11).proper_subset?(1..10)
    assert_raises(ArgumentError) { (1..10).proper_subset?('not a range') }
  end
```

```
  def test_superset
    assert_equal true, (1..10).superset?(1..10)
    assert_equal false, (2..9).superset?(1..10)
    assert_equal false, (1..9).superset?(1..10)
    assert_equal false, (2..10).superset?(1..10)
    assert_equal true, (0..11).superset?(1..10)
    assert_equal true, (0..10).superset?(1..10)
    assert_equal true, (1..11).superset?(1..10)
    assert_raises(ArgumentError) { (1..10).superset?('not a range') }
  end

  def test_proper_superset
    assert_equal false, (1..10).proper_superset?(1..10)
    assert_equal false, (2..9).proper_superset?(1..10)
    assert_equal false, (1..9).proper_superset?(1..10)
    assert_equal false, (2..10).proper_superset?(1..10)
    assert_equal true, (0..11).proper_superset?(1..10)
    assert_equal true, (0..10).proper_superset?(1..10)
    assert_equal true, (1..11).proper_superset?(1..10)
    assert_raises(ArgumentError) { (1..10).proper_superset?('not a range') }
  end
end
```

For the record, I'd be interested in taking a stab at implementing this in MRI, if the proposal is accepted.

#5 - 02/22/2018 07:21 AM - nobu (Nobuyoshi Nakada)

It should consider exclude_end? too.

#6 - 02/22/2018 06:10 PM - owst (Owen Stephens)

- File 0001-range.c-add-subset-superset-methods.patch added

Thank you nobu, I came to the same realisation when writing the tests for my attempt at an implementation (patch attached).

Please let me know what you think, when you have time. This is my first "proper" MRI patch, so all comments gratefully received!

#7 - 02/23/2018 01:39 AM - duerst (Martin Dürst)

As long as we only consider ranges starting and ending with integers, concepts such as subset and superset make sense. But ranges can also be constructed from other numbers, and other objects in general.

What e.g. should be the result of (5..10).subset?(5.5..7.9)?

Or what if we introduce something like Range.new(1, 10, step: 2), which would produce [1, 3, 5, 7, 9] when converted to an Array?

#8 - 02/23/2018 12:30 PM - owst (Owen Stephens)

Good questions duerst. I wonder if I should have stuck with the subrange naming rather than subset, due to the Numeric behaviour.

> What e.g. should be the result of (5..10).subset?(5.5..7.9)?

The implementation I have used in my patch is equivalent to:

```
(5.5..7.9).cover?((5..10).min) && (5.5..7.9).cover?((5..10).max)
```

which is false, but means

```
(6..7).subset?(5.5..8.5)
```

is true as both (5.5..8.5).cover?(6) and (5.5..8.5).cover?(7) are true.

I think this result is what I would expect - what do you think?

> Or what if we introduce something like Range.new(1, 10, step: 2), which would produce [1, 3, 5, 7, 9] when converted to an Array?

Assuming the implementation in terms of cover? (and using the subrange? naming) I think this would still make sense:

```
(2..4).subrange?(Range.new(1, 5, step: 2)) # => true
```

#9 - 02/23/2018 12:43 PM - owst (Owen Stephens)

- File v2-0001-range.c-add-subset-superset-methods.patch added

Attached updated patch, the former had a syntax error (a missing)).

#10 - 02/23/2018 04:30 PM - al2o3cr (Matt Jones)

Minor point: IMO a "Range with steps" isn't a Range, it's something else. It doesn't seem relevant to the discussion.

Bigger point: the tests on improper ranges in the patch seem to imply some odd consequences:

```
range_1 = "g".."f"
range_2 = "b".."e"
range_1.subset?(range_2) # => true
range_2.any? { |el| range_1.include? el } # => false
```

So range_1 claims to be a subset of range_2, but no elements of range_2 are included in range_1.

```
range_3 = "d".."a"
range_1.subset?(range_3) # => true
range_3.subset?(range_1) # => true
```

Generally, a <= b and b <= a implies a == b. In this case, however, the early return in range_subset means that any improper range passed to subset? will return true.

min and max also have bad performance characteristics for exclusive ranges that linear_object_p returns false for, as they need to generate all the intermediate elements using succ. For instance, attempting to evaluate ("aaaaaaaaaa"..."zzzzzzzzzz").max ran for over a minute before I gave up and aborted it. There's also explicit code (see discrete_object_p and callsites) that makes exclusive ranges of Time objects raise TypeError if you call max on them.

#11 - 02/23/2018 05:09 PM - vo.x (Vit Ondruch)

Why not modify Range#include? to accept Range object. Anyway, there is missing any justification for the RFE.

#12 - 02/23/2018 05:43 PM - owst (Owen Stephens)

Thanks for your thoughts al2o3cr, your "bigger point" is interesting:

> So range_1 claims to be a subset of range_2, but no elements of range_2 are included in range_1.

My thinking here was that the empty set is a subset of all sets, so similar behaviour seemed reasonable for empty/invalid Ranges, i.e. phrase the subset "definition" as: "all elements of range_1 are covered by range_2", which is vacuously true.

You're right about the implication for <= so maybe it's a bad idea to treat "empty"/invalid ranges as if they were empty sets (and thus all equivalent). However, the same thing is true for exclusive/inclusive ranges that contain the same elements (when considered as sets):

```
> (1..2).subset?(1...3)
=> true
> (1...3).subset?(1..2)
=> true
> (1..2) == (1...3)
=> false
> Set.new((1..2)) == Set.new((1...3))
=> true
```

With .. vs ... we can have non-equal Range representations of the same set of elements. I'm not sure what to suggest here. It doesn't quite feel right to restrict subset? to only allow an argument with the same exclude_end? value.

> min and max also have bad performance characteristics for exclusive ranges that linear_object_p returns false for,

Yes, I didn't like this myself, I'm glad you've brought it up. An option would be to prevent such calls from occurring, which doesn't seem nice, but is probably preferable to performance issues?

#13 - 02/23/2018 05:53 PM - owst (Owen Stephens)

Good point v.ox, I did consider implementing subset? as an overloading of include? (or possibly cover? ?), as one can't have a range-of-ranges (so there is no ambiguity).

How would you distinguish between strict/non-strict - a strict: kwarg, perhaps? Something like:

```
(1..3).cover?((1..3)) # => true
(1..3).cover?((1..3), strict: false) # => true
(1..3).cover?((1..3), strict: true) # => false
(1..3).cover?((1..2), strict: true) # => true
```

I'm not sure the level of justification required for a method to be included in the stdlib, but I can provide an example where we would (and do) use Range#subset?: we have products that have a range of allowed values, and in some circumstances we restrict products to a dynamically-generated range of allowed values, and want to make sure that range is compatible with the product range, something like:

```
raise ArgumentError unless dynamic_value_range.subset?(product_value_range)
```

The other methods (strict_subset?, and the *superset? methods) were only added for symmetry.

#14 - 03/17/2018 10:32 PM - owst (Owen Stephens)

Are there any further thoughts on my latest patch? Based on the discussion above, I think I should rename the added methods: s/set/range/ and perhaps also prevent the "bad performance" cases where either range has exclude_end? true and linear_object_p false?

#15 - 03/25/2018 03:56 PM - owst (Owen Stephens)

- File v3-0001-range.c-add-subrange-superrange-methods.patch added

I've attached an updated patch; it includes the renamed (by s/set/range/) methods and a performance improvement for strict_subset? (before, we would calculate max on the receiver twice, which is potentially slow when the receiver is an exclude_end Range of non-Numeric objects, now it's only calculated once).

Any comments would be gratefully received!

#16 - 05/17/2018 07:38 AM - matz (Yukihiro Matsumoto)

- Status changed from Open to Feedback

Any (real-world) use-case?

Matz.

#17 - 05/17/2018 02:02 PM - owst (Owen Stephens)

Hi Matz!

A slightly contrived example - imagine we want to filter candidates for a new job position and harshly reject any who have a salary requirement that is not covered by our predetermined position salary range:

```
Candidate = Struct.new(:name, :desired_salary_range)

candidates = [
  Candidate.new('Andrew', (15_000..20_500)),
  Candidate.new('John', (25_000..35_000)),
  Candidate.new('Owen', (15_000..17_500)),
  Candidate.new('Zack', (5_000..9_000)),
]

position_salary_range = (10_000..20_000)

acceptable, unacceptable = candidates.partition { |c|
  c.desired_salary_range.subrange?(position_salary_range)
}

puts "Unacceptable: #{unacceptable.map(&:name).join(',')}" # => Unacceptable: Andrew, John, Jack
puts "Acceptable: #{acceptable.map(&:name).join(',')}" # => Acceptable: Owen
```

Looks like I've got the job, as no-one else is acceptable! :-)

As a more serious example, in my job, we offer loans of between £1000 and £8000. We allow certain customers to top-up their loans with an additional advance (chosen from a range of possible advances), if the new loan (the top-up is considered as a new loan) will pay-off their existing loan and cover all possible additional advances. Therefore, to check if someone can top-up their loan we do something similar to:

```
class Range
  def offset(n)
    (first + n..last + n)
  end
end

VALID_LOAN_VALUE_RANGE = (1000..8000)
TOP_UP_RANGE = (1..5) # For example; very unrealistic numbers

def top_up_message(existing_loan_settlement_value)
  top_up_value_range = TOP_UP_RANGE.offset(existing_loan_settlement_value)
```

```
  if top_up_value_range.subrange?(VALID_LOAN_VALUE_RANGE)
    puts "you may top-up your loan with between £#{TOP_UP_RANGE.first} and £#{TOP_UP_RANGE.last}!"
  else
    puts "sorry, at this stage we can't top-up your loan"
  end
end

puts top_up_message(7995) # => you may top-up...
puts top_up_message(7999) # => sorry...
```

N.b. this adds another useful Range method, Range#offset (which probably only makes sense for numeric ranges), would you accept a new Feature to add this method?

Finally, regarding naming, I think I now prefer overriding cover? to accept a Range - when I described this feature to my colleague I used the phrase "does one range cover the other?" as an intuition, and indeed, that is how it's implemented.

Please let me know your thoughts.

#18 - 05/17/2018 04:46 PM - shevegen (Robert A. Heiler)

I have no particular opinion (neither pro nor con) on the issue itself.

On the comment:

> [..] this adds another useful Range method, Range#offset (which probably only makes sense for numeric ranges), would you accept a new Feature to add this method?

I think it would be best to add a new issue for Range#offset; you can then link into the issue here too, from that other issue. It makes it easier for other people to comment on it and, I think, also easier to approve it, if it is useful.

#19 - 07/12/2018 06:55 AM - tarui (Masaya Tarui)

As real-world use-case,
I want it(include,cover version) for guarding NArray's Index arguments.
NArray#[] can accept Integer and Range.
Currently, to limit the access range, We have to write complex code.

```
def check(idx,range)
  case idx
  when Integer
    range.include?(idx)
  when Range
    range.begin < idx.begin && ( (!range.exclude_end? && !idx.exclude__end? || range.exclude_end?) ? range.end
 >=idx.end : range.end > idx.end)
  end
end
```

I want to write simply

```
range.include?(idx)
```

#20 - 07/12/2018 09:39 PM - marcandre (Marc-Andre Lafortune)

Not directly related, but if your ranges are such that begin <= end, then I think you can use range.max > idx.max for the last part.

#21 - 07/14/2018 01:38 PM - znz (Kazuhiro NISHIYAMA)

How about Range#{<,<=,>,>=} like Hash#{<,<=,>,>=} for method names?

#22 - 07/18/2018 06:03 AM - matz (Yukihiro Matsumoto)

- Status changed from Feedback to Open

The method name subrange? may cause confusion that which includes which.
I propose cover? to accept ranges.

Matz.

#23 - 07/18/2018 06:04 AM - tarui (Masaya Tarui)

- Assignee set to tarui (Masaya Tarui)

- Status changed from Open to Assigned

#24 - 07/19/2018 12:03 AM - owst (Owen Stephens)

- File v4-0001-range.c-allow-cover-to-accept-Range-argument.patch added

Thank you for your proposal Matz, having thought about it over the last few months, I agree.

I have updated my patch accordingly (which has greatly simplified/improved it), I welcome any comments.

#25 - 07/30/2018 07:31 AM - tarui (Masaya Tarui)

Thank you for your patch owst.

I reviewed it and feel the behavior below is strange.

```
$ ruby -e 'p (1..3.1).cover?(1...3)'
true
$ ruby -e 'p (1..3.1).cover?(1...4)'
Traceback (most recent call last):
        1: from -e:1:in `<main>'
-e:1:in `cover?': can't iterate from Float (TypeError)
```

At (a..b).cover?(c...d) with b < d,
is it reasonable to use (c...d).max(but ignore exception) for comparison with b?

#26 - 07/31/2018 12:29 AM - owst (Owen Stephens)

- File v5-0001-range.c-allow-cover-to-accept-Range-argument.patch added

Hi tarui, thank you for reviewing and your suggestion/question.

In fact, with my v4 patch in the case you describe there is a bug: (1..3).cover?(1.0...4.0) is true not false. This is prevented using b >= (c...d).max as you suggest, but I see two issues:

First, the performance of Range#max is very bad in certain cases (as al2o3c pointed out above) e.g. on my machine (Macbook Pro, 2.8 GHz i5) I now have:

```
$ time ruby -e "p ('aaaaa'..'zzzzz').cover?('aaaaa'...'zzzzz')"
true
ruby -e "p ('aaaaa'..'zzzzz').cover?('aaaaa'...'zzzzz')"  0.19s user 0.05s system 92% cpu 0.256 total

$ time ruby -e "p ('aaaaa'..'zzzzy').cover?('aaaaa'...'zzzzz')"
true
ruby -e "p ('aaaaa'..'zzzzy').cover?('aaaaa'...'zzzzz')"  8.12s user 0.07s system 98% cpu 8.329 total
```

Second, we have the following strange behaviour that is similar to that in your comment:

```
$ ruby -e "p (1..4).cover?(1.0...4.0)"
true
$ ruby -e "p (1..3).cover?(1.0...4.0)"
Traceback (most recent call last):
    2: from -e:1:in `<main>'
    1: from -e:1:in `cover?'
-e:1:in `max': cannot exclude non Integer end value (TypeError)
```

I have attached an updated patch, which uses max, but does not address either of the above issues. Can you see an alternative that addresses these issues in a straightforward way?

#27 - 07/31/2018 01:52 AM - Anonymous

Thank you for the new patch.

At the first issue, I understood max method has performance issue at that case.
But how often will we encounter it? I think it is a very rare case.
Or please show the application.

At the second issue, as I already said 'ignore exception',
I think that it should return false rather than raising an exception.

#28 - 08/01/2018 11:41 PM - owst (Owen Stephens)

- File v6-0001-range.c-allow-cover-to-accept-Range-argument.patch added

I agree that the max performance issue is likely to be a rare case; I am happy to leave it as-is.

Apologies on the second issue - I originally misunderstood "ignore" to mean "don't rescue".

I've attached another updated patch - it rescues the TypeError raised by max, and also fixes the handling of endless/empty ranges.

#29 - 08/03/2018 03:04 AM - tarui (Masaya Tarui)

I am happy that match opinion with you.

I will commit based on your patch. It'll take a little while.

#30 - 08/03/2018 10:52 AM - owst (Owen Stephens)

Great, thank you tarui

#31 - 09/05/2018 07:06 PM - tarui (Masaya Tarui)

- Status changed from Assigned to Closed

Applied in changeset trunk|r64640.

---

range.c: Range#cover? accepts Range object. [Feature #14473]

```
* range.c (range_cover): add code for range argument.
      If the argument is a Range, check it is or is not
  covered by the reciver. If it can be treated as a
  sequence, this method treats it that way.
* test/ruby/test_range.rb (class TestRange): add tests
  for this feature.

  This patch is written by Owen Stephens. thank you!
```

Files

0001-range.c-add-subset-superset-methods.patch	8.84 KB	02/22/2018	owst (Owen Stephens)
v2-0001-range.c-add-subset-superset-methods.patch	8.85 KB	02/23/2018	owst (Owen Stephens)
v3-0001-range.c-add-subrange-superrange-methods.patch	9.24 KB	03/25/2018	owst (Owen Stephens)
v4-0001-range.c-allow-cover-to-accept-Range-argument.patch	4.45 KB	07/19/2018	owst (Owen Stephens)
v5-0001-range.c-allow-cover-to-accept-Range-argument.patch	4.82 KB	07/31/2018	owst (Owen Stephens)
v6-0001-range.c-allow-cover-to-accept-Range-argument.patch	5.51 KB	08/01/2018	owst (Owen Stephens)