Ruby - Feature #16757

Add intersection to Range

04/03/2020 09:29 PM - stuyam (Stuart Yamartino)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

It would be great if there was a way to do an intersection with ranges. I wrote a method recently to solve this but it gets complicated and doesn't solve for all edge cases.
The example I was using it for was getting the intersection of two date ranges. I was using it to calculate pricing discounts for overlapping dates for a reservation system.

I would propose something like:

```
(Date.new(2020, 04, 03)..Date.new(2020, 04, 20)) & (Date.new(2020, 04, 15)..Date.new(2020, 04, 30
))

=> Wed, 15 Apr 2020..Mon, 20 Apr 2020
```

There are a handful of array methods that you can just convert a range to and do on an arrays rather than the ranges themselves, like so:

```
(Date.new(2020, 04, 03)..Date.new(2020, 04, 20)).to_a & (Date.new(2020, 04, 15)..Date.new(2020, 04
, 30)).to_a

=> [Wed, 15 Apr 2020, Thu, 16 Apr 2020, Fri, 17 Apr 2020, Sat, 18 Apr 2020, Sun, 19 Apr 2020, Mon,
 20 Apr 2020]
```

There are a few issues with this however:

1. Performance: creating the ranges into arrays and doing array calculations can me slower based on the span of the range.
2. Returning a range: if you want to go back to a range, you need to convert the resulting array back to a range, which involves min and maxing the array and handling nil edge cases.
3. Infinite ranges: Now that there are endless and beginless ranges added in ruby 2.6 and 2.7 respectively, you can not convert an infinite range to an array and therefore can't do an intersection without more complex logic.

I think the added infinite range support makes it a good reason to add range intersections since that is a feature you can't accomplish with just an array.

It looks like there was an Array#intersection alias added for Array#& recently. I would propose Range#& and Range#intersection to have parity with Array and Set methods.

History

#1 - 01/22/2022 07:26 AM - baweaver (Brandon Weaver)

I'd like to see this explored, as I find myself solving a number of problems that involve Range merging, and potential intersections.

My current implementation, in Ruby, would be something to the note of:

```
class Range
  # To do this we would have to either inline this overlap checking
  # code in the methods, or introduce overlap checks for ranges
  # beyond single values as `cover?` does.

  # If other Range begins inside of the current Range
  #   (0..5).left_overlap?(3..7) # true
  def left_overlap?(other)
    cover?(other.begin)
  end

  # If self Range begins inside the other Range
  #   (3..7).left_overlap?(0..5) # true
```

```
  def right_overlap?(other)
    other.cover?(self.begin)
  end

  # If the current Range overlaps the other
  def overlap?(other)
    left_overlap?(other) || right_overlap?(other)
  end

  def join(other)
    return self.class.new(self.begin, other.end) if left_overlap?(other)
    return self.class.new(other.begin, self.end) if right_overlap?(other)

    # This is contentious if there is no overlap. Do we return `self`,
    # an empty Array, nil (breaks chaining), or exception (breaks a lot more)
    self
  end
  # Though I really hesitate to add `+` here in case it fails from
  # no overlaps
  alias_method :+, :join

  def intersection(other)
    return self.class.new(other.begin, self.end) if left_overlap?(other)
    return self.class.new(self.begin, other.end) if right_overlap?(other)

    # This one I feel better about returning an empty Array for, but
    # it does break some type integrity, so not sure what would be
    # more correct here.
    []
  end
  # I do feel better about this
  alias_method :&, :intersection
end
```

There are a few concerns I can see with these implementations which would need to be addressed for consistency and correctness, there may be some I miss.

First is what to do when the origin Range is "greater" than the other Range (10..20 against 0..5 versus the reverse), would it be ignored as with (10..1).to_a? I could see a few different cases for this.

Second is how to handle beginless and endless ranges, though cover? prevents issues with nil? as is the case with >= and other ordering comparators.

Third is how to approach exclusive versus inclusive ranges, which may be something like self.class.new(other.begin, self.end, self.exclude_end?) and the reverse in the other case.

In general I am a fan of type-specific methods rather than converting to Array, as is the case with Hash specific methods like select returning Hash rather than requiring to_h.