Ruby - Feature #19294

Enumerator.product works incorrectly with consuming enumerators

01/01/2023 05:56 PM - zverok (Victor Shepelev)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

```
s = StringIO.new('abc')
Enumerator.product([1, 2, 3], s.each_char).to_a
# Expected: => [[1, "a"], [1, "b"], [1, "c"], [2, "a"], [2, "b"], [2, "c"], [3, "a"], [3, "b"], [3
, "c"]]
# Actual: => [[1, "a"], [1, "b"], [1, "c"]]
```

The implementation consumes the non-first enumerator to produce the first combination.

Somewhat related to the dilemma of consuming and non-consuming enumerators (#19061).

PS: I noticed I don't understand why it is Enumerator.product and not Enumerable#product, but probably it is too late to raise the questions :(

History

#1 - 01/01/2023 05:57 PM - zverok (Victor Shepelev)

- Backport changed from 2.7: UNKNOWN, 3.0: UNKNOWN, 3.1: UNKNOWN, 3.2: UNKNOWN to 2.7: DONTNEED, 3.0: DONTNEED, 3.1: DONTNEED, 3.2: UNKNOWN

#2 - 05/14/2023 05:33 AM - jeremyevans0 (Jeremy Evans)

- Backport deleted (2.7: DONTNEED, 3.0: DONTNEED, 3.1: DONTNEED, 3.2: UNKNOWN)

- Tracker changed from Bug to Feature

@headius (Charles Nutter) and I discussed this. We don't think it is possible to get the output requested without keeping all enumerated elements for arguments after the first argument in memory, which is not acceptable by default. It's possible that behavior could be considered via a keyword argument, but that would be a feature request and not a bug fix.

#3 - 05/14/2023 01:25 PM - nevans (Nicholas Evans)

Perhaps #rewind could be called, but (IMO) that shouldn't be the default either. Two kwargs?

```
Enumerator.product(*enums, rewind: boolish, memoize: boolish) {|elements| ... }
```

Or one?

```
Enumerator.product(rewind: (bool | :rewind | :memoize) {|elements| ... }
```

In the meantime, it should at least be documented, and that documentation should include simple workarounds such as:

When the consumable enumerator doesn't take up too much memory:

```
Enumerator
  .product([1, 2, 3],
           s.each_char.to_a)
  .to_a
# =>
# [[1, "a"],
#  [2, "a"],
#  [3, "a"],
#  [1, "b"],
#  [2, "b"],
#  [3, "b"],
#  [1, "c"],
#  [2, "c"],
#  [3, "c"]]
```

If rewinding works (to_a is just for the example. presumably you wouldn't use to_a if memory use is a motivator):

```
rewinder = Enumerator.new do |y|
  s.rewind
  s.each_char(&y)
end
Enumerator
  .product([1, 2, 3], rewinder)
  .to_a
# =>
# [[1, "a"],
#  [2, "a"],
#  [3, "a"],
#  [1, "b"],
#  [2, "b"],
#  [3, "b"],
#  [1, "c"],
#  [2, "c"],
#  [3, "c"]]
```

If you only have a single consumable enumerator, it might not fit in memory and it can't or shouldn't rewind:

```
Enumerator
  .product(s.each_char,
           [1, 2, 3])
  .lazy
  .map(&:reverse)
  .to_a
# =>
# [[1, "a"],
#  [2, "a"],
#  [3, "a"],
#  [1, "b"],
#  [2, "b"],
#  [3, "b"],
#  [1, "c"],
#  [2, "c"],
#  [3, "c"]]
```