I propose Enumerable#min_with_value, max_with_value and minmax_with_value. These methods work like this:

```ruby
%w(abcde fg hijk).min_with_value { |e| e.size } # => ['fg', 2]
%w(abcde fg hijk).max_with_value { |e| e.size } # => ['abcde', 5]
%w(abcde fg hijk).minmax_with_value { |e| e.size } # => [['fg', 2], ['abcde', 5]]
```

Corresponding to #min(n), an integer argument can be passed to #min_with_value or #max_with_value.

```ruby
%w(abcde fg hijk).min_with_value(2) { |e| e.size } # => [['fg', 2], ['hijk', 4]]
%w(abcde fg hijk).max_with_value(2) { |e| e.size } # => [['abcde', 5], ['hijk', 4]]
```

**Motivation**

When I use Enumerable#min_by, I sometimes want to get not only the minimum element but also the value from the given block. (e.g.: There are many points. Find the nearest point and get distance to it.)

```ruby
elem = enum.min_by { |e| foo(e) }
value = foo(elem)
```

This works, but I'd like to avoid writing foo() twice. (Consider a more complex case.)

This can be written without repeated foo() like belows, but it is slightly complicated and needs extra arrays.

```ruby
value, elem = enum.map { |e| [foo(e), e] }.min_by(&:first)
```

If the size of enum is enormous, it is hard to use intermediate arrays.

Enumerable#min_with_value solves this problem.

I think min_with_value is the best name I could think of, but any suggestions for better names would be appreciated.

**Benchmark**

[https://bugs.ruby-lang.org/issues/18181#note-2](https://bugs.ruby-lang.org/issues/18181#note-2)

**Example**

Solving a traveling salesman problem in nearest neighbor algorithm.

```ruby
require 'set'
Point = Struct.new(:x, :y)
points = Set.new([Point.new(1, 1), Point.new(2, 4), Point.new(3, 3), Point.new(2, 2), Point.new(0, 1)])

total = 0
current = points.first
points.delete(current)
path = [current]
```
until points.empty?
  current, distance = points.min_with_value do |point|
    Math.hypot(current.x - point.x, current.y - point.y)
  end
  total += distance
  points.delete(current)
  path << current
end

p path
p total

History

#1 - 09/20/2021 04:55 PM - sawa (Tsuyoshi Sawada)
This is a frequently seen use case. I use code like this:

%w(abcde fg hi jkl mn).group_by(&:size).min #=> [2, ["fg", "hi", "mn"]]

and I don't see a strong need to shorten it into a single method although I am not strongly against doing so. However, I have some concerns with your proposal:

1. Its use will be limited if it only picks a single element when the min/max element is not unique. I propose that it should rather hold an array of the corresponding elements.
2. I don't think the word "value" is appropriate. By this word, what comes to mind most naturally is the min/max value rather than the enumerated element. Perhaps a word like "element(s)" should be used.
3. I don't think the order of the elements you proposed is appropriate. With each_with_object, the enumerated entity is followed by the accumulating object, i.e., the word order matches the order of the elements within the subarray. I think you should follow this practice.
4. The methods min/max are used without a block, min_by / max_by are used with a block. And in this case, you are using a block.

Thus, I think your proposal should be modified into something like the following (which looks so similar to what can be done already as I have shown above):

%w(abcde fg hi jkl mn).min_by_with_elements(&:size) #=> [2, ["fg", "hi", "mn"]]

#2 - 09/21/2021 08:52 AM - kyanagi (Kouhei Yanagita)

enum.group_by { ... } holds all of the elements of enum, so it consumes a lot of memory and is very slow if enum is enormous.

require 'benchmark'
Benchmark.bm(16) do |x|
  enum = -1000000..1000000
  x.report("group_by") { enum.group_by { |i| i.abs }.min }
  x.report("map.min_by") { enum.map { |i| [i.abs, i] }.min_by{|i|first} }
  x.report("min_with_value") { enum.min_with_value { |i| i.abs } }
  x.report("each") { enum.each { |i| i.abs } }
end

<table>
<thead>
<tr>
<th>user</th>
<th>system</th>
<th>total</th>
<th>real</th>
</tr>
</thead>
<tbody>
<tr>
<td>group_by</td>
<td>1.238664</td>
<td>0.243779</td>
<td>1.482443</td>
</tr>
<tr>
<td>map.min_by</td>
<td>0.721992</td>
<td>0.047738</td>
<td>0.769730</td>
</tr>
<tr>
<td>min_with_value</td>
<td>0.236433</td>
<td>0.004105</td>
<td>0.240538</td>
</tr>
<tr>
<td>each</td>
<td>0.150815</td>
<td>0.000000</td>
<td>0.150815</td>
</tr>
</tbody>
</table>

#min and #min_by return single element even through there are multiple smallest elements. There may be cases where it is useful to return multiple elements, but single element will be enough for most cases.

Writing elements.first everytime is a little bit of a pain. Besides, size of the elements may become very large. (if almost all of the elements are mapped into the same value)

I think #min_with_value should return the element as primary (i.e. the element is first, the block's value is second) because #min and #min_by return the element. Additionally, this is similar to the way #each_with_index and #each_with_object pass their block arguments.

enum.each_with_index do |elem, i| # <- the element is first, the other is second. end
enum.each_with_object([]) do |elem, obj| # <- the element is first, the other is second.
end

Naming issue:
The behavior of this method is described as "The minimum element by the block value and its value". Based on this, the name will be something like min_element_by_value_and_value or min_element_by_value_with_value if verbosely named.

Considering the method #min which returns the element, the word "element" is thought to be able to omitted.
So I suggest the short name #min_with_value. (I think #min_by_with_value is also possible.)
The word "with" implies that "min(_element)" and "value" are different objects.

#3 - 09/21/2021 09:00 AM - kyanagi (Kouhei Yanagita)
- Description updated

#4 - 09/21/2021 09:01 AM - kyanagi (Kouhei Yanagita)
I added an example code using #min_with_value.

#5 - 09/21/2021 02:43 PM - Dan0042 (Daniel DeLorme)
I like this proposal, but I think sawa (Tsuyoshi Sawada)'s idea has value also. It would be simple to accommodate both uses with a single syntax:

```
%w(abcde fg xx hijk).min_with_value(&:size) # => [2, 'fg', 'xx']
```

in the normal case where you only want the 'first' minimum

```
v,elem = %w(abcde fg xx hijk).min_with_value(&:size)
```

#6 - 09/22/2021 12:48 AM - kyanagi (Kouhei Yanagita)
- Description updated

#7 - 09/22/2021 12:51 AM - kyanagi (Kouhei Yanagita)
Description about #min_with_value(n) added. It corresponds to #min(n).

#8 - 09/24/2021 06:46 PM - Eregon (Benoit Daloze)
:+1: I've needed this several times.
I think the semantics in the description are the best and clearest.

#9 - 12/03/2021 05:04 AM - matz (Yukihiro Matsumoto)
Just for confirmation, is it OK that min_with_value returns only a single value?
I don't like the name _with_value. It explains that the method returns an additional value with the minimum/maximum value, but no additional information on the value.

Matz.

#10 - 12/03/2021 03:00 PM - Dan0042 (Daniel DeLorme)
What about min_with ?

```
%w(abcede fg hijk).min_with(&:size) # => ['fg', 2]
```

#11 - 12/03/2021 03:40 PM - cvss (Kirill Vechera)
There's also a frequent similar problem with #find when you need to find the first matched value instead of entry. But since it involves two semantically different code parts, it a bit more complex and cannot be implemented nicely with the same approach:

```
%w(abcede fg hijk).find_with_value { |e| (size = e.size).even? && size } # => ['fg', 2]
```

We can create some lazy mapping enumerator providing pairs [entry, value], which we unroll in the second chain just as much as we need and without repeating the computation of value:

```ruby
module Enumerable
  def with_map &block
    map{|e| [e, block.call(e)]}
  end
end

%w(abcede fg hijk).lazy.with_map(&:size).find{|_2.size?} # => ['fg', 2]
```
Back to #min_with_value, it could be written as:

```
%w(abcde fg hijk).lazy.with_map(&:size).min.last # => ['fg', 2]
```

We can add some shorthand methods specific to this enumerator like these:

```
%w(abcde fg hijk).lazy.with_map(&:size).min_by_value # => ['fg', 2]
%w(abcde fg hijk).lazy.with_map(&:size).min # => ['fg', 2] - #min is overwritten as an alias to min_by_value
%w(abcde fg hijk).lazy.with_map(&:size).find_by_value(&:even?) # => ['fg', 2]
%w(abcde fg hijk).lazy.with_map(&:size).find(&:even?) # => ['fg', 2] - #find is overwritten as an alias to find_by_value
```

#12 - 12/06/2021 04:44 AM - kyanagi (Kouhei Yanagita)

matz (Yukihiro Matsumoto) wrote in #note-9:

Just for confirmation, is it OK that min_with_value returns only a single value?

I think that returning a single value will suit most cases.
It matches that min also returns a single value.
But if it is considered that returning multiple values is better, I will not disagree with it.

#13 - 12/06/2021 04:54 AM - ko1 (Koichi Sasada)

Another idea (evaluate a given block with a result):

```ruby
class Enumerator
  def with_value(&b)
    v = each(&b)
    [v, b.call(v)]
  end
end

pp %w(abcde fg hijk).min_by.with_value{|e| e.size}
#=> ['fg', 2]
```

It is general for ..._by methods. If given block has side-effect or consumes huge time, it is not acceptable, though.

#14 - 01/04/2022 10:53 PM - shan (Shannon Skipper)

For a single value, a "then" suffix might be nice.

```
%w(abcde fg hijk).min_by(&:size)
#=> "fg"

%w(abcde fg hijk).min_by.then(&:size)
#=> 2
```

It mirrors #min_by and #then with the same block.

```
%w(abcde fg hijk).min_by(&:size).then(&:size)
#=> 2
```

#15 - 01/05/2022 01:05 AM - baweaver (Brandon Weaver)

I like the more generic suggestion from ko1 (Koichi Sasada) on with_value, I had a similar idea before reading it when shan (Shannon Skipper) pinged me about this ticket.

The other potential is using the _map suffix as is the case with filter_map:

```
%w(abcde fg hijk).min_map { |e| e.size }
# => ['fg', 2]

%w(abcde fg hijk).max_map { |e| e.size }
# => ['abcde', 5]

%w(abcde fg hijk).minmax_map { |e| e.size }
# => [["fg", 2], ['abcde', 5]]

%w(abcde fg hijk).min_map(2) { |e| e.size }
# => [["fg", 2], ['hijk', 4]]

%w(abcde fg hijk).max_map(2) { |e| e.size }
# => [['abcde', 5], ['hijk', 4]]
```
This has precedence, though I do wonder how many functions we accumulate in Enumerable that are effectively the composition of two or more Enumerable methods.

More musing, and in a general sense, I'm reminded of Elixir extracting Enumerable-like methods into Enum and allowing potential composition and the rejected prior syntax of Enumerable::method which ties into a fairly old post on my hopes for the also rejected pipeline operator that was proposed.

I may write a bit on that general idea, but I do see great value in the general concept of min_with_value as I myself have needed it a few times in the past.

#16 - 01/25/2022 07:33 AM - kyanagi (Kouhei Yanagita)

Another name proposal:

When I write array.min_with_value { foo }, I mean "minimize foo with an element in array".

So array.minimizing { foo } (and maximizing) may be the candidate.