Ruby master - Bug #16996

Hash should avoid doing unnecessary rehash

06/27/2020 08:20 AM - marcandre (Marc-Andre Lafortune)

Status:	Open		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:		**Backport:**	2.5: UNKNOWN, 2.6: UNKNOWN, 2.7: UNKNOWN

Description

Pop quiz: Which is the fastest way to get a copy of a Hash h?

If, like me, you thought h.dup (of course, right?), you are actually wrong.

The fastest way is to call h.merge. Try it:

```
require 'benchmark/ips'

lengths = 1..50

h = lengths.to_h { |i| ['x' * i, nil] }

Benchmark.ips do |x|
  x.report("dup")        { h.dup }
  x.report("merge")      { h.merge }
end
```

I get

```
Calculating -------------------------------------
                dup    259.233k (± 9.2%) i/s -      1.285M in   5.013445s
              merge    944.095k (± 8.2%) i/s -      4.693M in   5.005315s
```

Yup, it's *3.5x faster* with this example!!

Why? Because Hash#dup does a rehash, and merge does not.

Pop quiz 2: which methods of Hash that produce a new hash do a rehash?

Answer: it depends on the method and on the Ruby version

Does this rehash?	head	2.7	2.6	2.5	2.4	2.3	2.2	2.1	2.0
h.dup / clone	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
h.select{true} / reject{false}	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
h.select!{true} / reject!{false}	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
sub_h.to_h	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
h.merge({})	Ø	Ø	Ø	Ø	Yes	Yes	Yes	Yes	Yes
h.merge	Ø	Ø	Ø	n/a					
h.transform_values(&:itself)	Ø	Ø	Yes	Yes	Yes	n/a			

(where `sub_h = Class.new(Hash).replace(h)`, Ø = no rehash)

So in Ruby head, doing h.merge({}) or even h.transform_values(&:itself) will be much faster than h.dup (but slower in Ruby 2.4) (*)

Notice that select rehashes, but select! doesn't, so the fastest way to do a select in Ruby is... not to call select and instead to actually do a merge.select!! (*)

*: on hashes with non-negligible hash functions

```
class Hash
  def fast_select(&block)
    merge.select!(&block) # don't call dup because it's slow
  end
end

Benchmark.ips do |x|
  x.report("select")           { h.select{true} }
  x.report("fast_select")      { h.fast_select{true} }
end
```

On my test case above, fast_select is *2.5x faster* than select. fast_select will always return exactly the same result (unless the receiver needed a rehash).

Pop quiz 3: Is this a bug or a feature?

It should be clear that no feature of Ruby should be re-implementable in Ruby with a 3.5x / 2.5x speed gain, so many would think "of course it's a bug".

Well, https://bugs.ruby-lang.org/issues/16121 seems to think that Hash#dup's rehash is a feature...
Why?
Because there is actually a test that dup does a rehash
Why?
Because a test of Set was failing otherwise!
Commit: https://github.com/ruby/ruby/commit/a34a3c2caae4c1fbd
Short discussion: http://blade.nagaokaut.ac.jp/cgi-bin/vframe.rb/ruby/ruby-core/48040?47945-48527
Actual test: https://github.com/ruby/ruby/blob/master/test/test_set.rb#L621-L625
Why?
This test construct a Set that needs to be rehashed (by mutating an element of the set after it is added), and then checks that rehash_me == rehash_me.clone.
That test is bogus. It passes for obscure and undocumented reasons, and rehash_me.clone == rehash_me doesn't pass.
Today, it is official that sets with elements that are later mutated must be Set#reset, so it is official that this should not be relied upon.

Probably more clear is the case of select/reject (but I didn't check for failing test), and even more clear that merge changed in Ruby 2.5 and transform_values in 2.7, but not a single NEWS file mentions the word "rehash".

My conclusion is that Hash should avoid doing an unnecessary rehash: dup/clone/select/reject. We probably should add a reminder in the NEWS that if anyone mutates a key of a Hash, or an element of a Set and does not call rehash/reset, improper behavior should be expected.

Let's make Hash#dup/clone/select/reject fast please.

Any objection?

History

#1 - 06/27/2020 11:48 AM - Eregon (Benoit Daloze)

Completely agreed, Hash#dup should not rehash (and it already doesn't on TruffleRuby).

#2 - 06/27/2020 04:00 PM - Dan0042 (Daniel DeLorme)

Very surprising results for Hash#dup. I even tried h.rehash before the benchmark, with no effect.

Shouldn't the rehash be tied to a change in the number of buckets rather than which method is used?
select{true} doesn't change the number of items so a rehash is unnecessary
select{rand<0.5} discards half the items so a rehash is likely desired
and for a select that removes one item only the number of buckets likely doesn't need to change

The same goes for merge; merging with a zero-item hash requires no additional bucket but merging with a 100-item hash should trigger a rehash, no?

#3 - 06/27/2020 04:37 PM - marcandre (Marc-Andre Lafortune)

Dan0042 (Daniel DeLorme) wrote in #note-2:

> Shouldn't the rehash be tied to a change in the number of buckets rather than which method is used?
> The same goes for merge; merging with a zero-item hash requires no additional bucket but merging with a 100-item hash should trigger a rehash, no?

I think you might be confusing the re-organizing of a Hash's internal structure into buckets with the calculation of the #hash of the keys by calling String#hash or whatever class the key is. It's the responsibility of the user to call rehash if ever that key has changed (and thus is likely to have a different #hash value).

If we already have computed the #hash value for a key, we should always consider it fixed and never recalculate it. So while big_hash.merge(another_big_hash) might require a lot of internal reorganization bucket-wise, there is no reason that I'm aware of to recalculate #hash on any key (of big_hash or of another_big_hash)