I did a quick comparison:

In Ruby

def main
  n = 10000000  # ten million
  a = randPerm(100)

  t0 = Time.now

  n.times do |i|
    a.index(i)
  end

  puts "%.5f" % [Time.now - t0]
end

def randPerm(n)
  (0...n).sort_by{rand}
end
main()

In Go

package main

import (  "fmt"  "time"  "math/rand"
)

func main() {  n := 10000000  // ten million  a := rand.Perm(100)

  t0 := time.Now()

  for i := 0; i < n; i++ {
    index(a, i)
  }

  fmt.Printf("%.5f\n", time.Now().Sub(t0).Seconds())
}

func index(slice []int, value int) int {  for i, v := range slice {    if (v == value) {      return i    }
  }
  return -1
}
The result

Ruby: 71.08961 secs
Go: 2.61975 secs

That's pretty huge difference (and worse I was told my Go index function was "crazily inefficient" too, though personally I don't see how it can be any better). So, I thought I'd mention it. Maybe it would be possible to speed up.

Associated revisions

Revision f25daa25 - 08/27/2013 07:46 AM - nobu (Nobuyoshi Nakada)
array.c: optimized equality
  - array.c (rb_ary_index, rb_ary_rindex): use optimized equality to improve performance. [Feature #8820]
  - vm_insnhelper.c (rb_equal_opt): optimized equality function.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@42704 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 42704 - 08/27/2013 07:46 AM - nobu (Nobuyoshi Nakada)
array.c: optimized equality
  - array.c (rb_ary_index, rb_ary_rindex): use optimized equality to improve performance. [Feature #8820]
  - vm_insnhelper.c (rb_equal_opt): optimized equality function.

Revision 42704 - 08/27/2013 07:46 AM - nobu (Nobuyoshi Nakada)
array.c: optimized equality
  - array.c (rb_ary_index, rb_ary_rindex): use optimized equality to improve performance. [Feature #8820]
  - vm_insnhelper.c (rb_equal_opt): optimized equality function.

Revision 42704 - 08/27/2013 07:46 AM - nobu (Nobuyoshi Nakada)
array.c: optimized equality
  - array.c (rb_ary_index, rb_ary_rindex): use optimized equality to improve performance. [Feature #8820]
  - vm_insnhelper.c (rb_equal_opt): optimized equality function.

Revision 42704 - 08/27/2013 07:46 AM - nobu (Nobuyoshi Nakada)
array.c: optimized equality
  - array.c (rb_ary_index, rb_ary_rindex): use optimized equality to improve performance. [Feature #8820]
  - vm_insnhelper.c (rb_equal_opt): optimized equality function.

Revision 42704 - 08/27/2013 07:46 AM - nobu (Nobuyoshi Nakada)
array.c: optimized equality
  - array.c (rb_ary_index, rb_ary_rindex): use optimized equality to improve performance. [Feature #8820]
  - vm_insnhelper.c (rb_equal_opt): optimized equality function.

Revision 42704 - 08/27/2013 07:46 AM - nobu (Nobuyoshi Nakada)
array.c: optimized equality
  - array.c (rb_ary_index, rb_ary_rindex): use optimized equality to improve performance. [Feature #8820]
  - vm_insnhelper.c (rb_equal_opt): optimized equality function.

Revision 42704 - 08/27/2013 07:46 AM - nobu (Nobuyoshi Nakada)
array.c: optimized equality
  - array.c (rb_ary_index, rb_ary_rindex): use optimized equality to improve performance. [Feature #8820]
  - vm_insnhelper.c (rb_equal_opt): optimized equality function.

Revision 42704 - 08/27/2013 07:46 AM - nobu (Nobuyoshi Nakada)
array.c: optimized equality
  - array.c (rb_ary_index, rb_ary_rindex): use optimized equality to improve performance. [Feature #8820]
  - vm_insnhelper.c (rb_equal_opt): optimized equality function.

Revision 42704 - 08/27/2013 07:46 AM - nobu (Nobuyoshi Nakada)
array.c: optimized equality
  - array.c (rb_ary_index, rb_ary_rindex): use optimized equality to improve performance. [Feature #8820]
  - vm_insnhelper.c (rb_equal_opt): optimized equality function.

Revision 42704 - 08/27/2013 07:46 AM - nobu (Nobuyoshi Nakada)
array.c: optimized equality
  - array.c (rb_ary_index, rb_ary_rindex): use optimized equality to improve performance. [Feature #8820]
  - vm_insnhelper.c (rb_equal_opt): optimized equality function.

History

#1 - 08/27/2013 04:59 AM - normalperson (Eric Wong)

"trans (Thomas Sawyer)" redmine@ruby-lang.org wrote:

```ruby
def main
  n = 10000000  # ten million
  a = randPerm(100)

  t0 = Time.now

  n.times do |i|
    a.index(i)
end
```

03/13/2022
puts "%.5f" % [Time.now - t0]
end

def randPerm(n)
  (0...n).sort_by{rand}
end

The performance of your code varies between runs because the ordering is always different and index is O(n) worst case. call srand(0) before any rand calls to get a consistent seed.

I suspect your Go code has the same flaw (but I don’t know Go)

The result

Ruby: 71.08961 secs
Go: 2.61975 secs

That’s pretty huge difference (and worse I was told my Go index function was “crazily inefficient” too, though personally I don’t see how it can be any better). So, I thought I’d mention it. Maybe it would be possible to speed up

From what I can tell, rb_ary_index in array.c doesn’t use the optimized $= definition in insns.def (which inlines some common comparisons) to avoid rb_funcall overhead.

Maybe that can help this case...

Otoh, I would never use anything like Array#index on large arrays and/or hot code in the first place. After all these years of Ruby, I’ve hardly ever used Array#index. The only time in recent memory I used it was on a 5-element array of short strings.

#2 - 08/27/2013 05:23 AM - Anonymous

On 08/26/2013 12:57 PM, Eric Wong wrote:

"trans (Thomas Sawyer)" redmine@ruby-lang.org wrote:

```
def main
  n = 10000000  # ten million
  a = randPerm(100)
  t0 = Time.now
  n.times do |i|
    a.index(i)
  end
  puts "%.5f" % [Time.now - t0]
end

def randPerm(n)
  (0...n).sort_by{rand}
end
```

The performance of your code varies between runs because the ordering is always different and index is O(n) worst case. call srand(0) before any rand calls to get a consistent seed.

The running time of this code won’t vary much at all. The n=10000000 setting is much higher than a.size, so most #index calls will return nil. The entire array is searched for almost all iterations.

Maybe the intent was for each iteration step to do this

a.index(i%n)
The performance of your code varies between runs because the ordering is always different and index is $O(n)$ worst case.

Call srand(0) before any rand calls to get a consistent seed.

The running time of this code won't vary much at all. The n=10000000 setting is much higher than a.size, so most index calls will return nil. The entire array is searched for almost all iterations.

I think you're right. I was on a new machine (Haswell) and enabled a bunch kernel options I normally don't use (Hyper-threading, full tickless, full preempt, automatic process group scheduling).

Lots of variables even when the system isn't loaded :o

Yes, sorry. I meant to use a random index with each iteration, not i. But per the suggestion, I think i \% 100 would be better.

I changed and reran the benchmarks. But even so the comparison still comes out about the same ratio:

Ruby: 35.66597
Go: 1.39305

This issue was solved with changeset r42704.

Thomas, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.

array.c: optimized equality
- array.c (rb_ary_index, rb_ary_rindex): use optimized equality to improve performance. [Feature #8820]
- vm_insnhelper.c (rb_equal_opt): optimized equality function.