Programming Pearls in Ruby

This is my rewrite of most of the challenges in the programming-pearls book. Ruby allows for a more readable and simple syntax, saving huge ammouts of code for any given example (sometime 1 to 10 LOC).

Some challenges and discussions are not included since they are pointless for ruby, like "how much space needs a hash" or "integers use 8 bit on an 8 bit system and...". Most discussions are very condensed, since this should be a quick read.

The original can be found here. This version is printer friendly and maybe available as printed copy if i find a cheap way of getting it printed.

If you want to contribute, feel free to fork at github.

Copyright

Chapter 01 sorting integers

Tasks and solutions

- sort a list of 1.000.000 unique integers
- do not loading them into memory

Quick inline file sort

Take a value from the middle of the file,
put anything larger in file right, anything smaller in file left.
Recurse until the file is smaller than our memory-limit, then sort it with array.sort.
At the end, put all temp-file-pieces back together.

Disadvantage:
Needs many temp-files, that stay opened until the end

```
# lib/01-sorting-integers/1.rb

  def quick_sort_with_files(input)
    num_lines = (input.stat.size / @chars_per_num)
    return sort_in_memory(input) if num_lines <=
      ⏎@max_memory

    left = new_temp_file
    right = new_temp_file

    middle = line_in_middle(input).to_i

    input.rewind
    while line = input.gets
      if line.to_i < middle
        left.puts line
      elsif line.to_i > middle
        right.puts line
      end
    end
```

```
  left.flush
  right.flush

  return [quick_sort_with_files(left), middle,
    ⏎quick_sort_with_files(right)]
end
```

Chapter 02 1 vector rotation

Task

- rotate a given vector of length n, m places
- rotate without extra memory usage (no copies of the vector)

Solutions

`[1,2,3]` rotated one place is `[2,3,1]`.
Rotation of more than one step at a time is difficult.

1 * x

rotate one place, x times

```
# lib/02-1-vector-rotation/1.rb

#helper
def swap(vector,i,j)
  j %= vector.length
  vector[i], vector[j] = vector[j], vector[i]
end

def rotate(vector,steps)
  steps.times {
    (vector.length-1).times {|i| swap(vector,i,i+1)}
  }
  vector
end
```

re-arrange in a copy

Making copies is not allowed, but if it where it would be this simple. Make a copy, and re-arrange inside of it.

```
# lib/02-1-vector-rotation/2.rb
```

```
def rotate(vector,steps)
  copy = vector.dup
  vector.each_index do |i|
    vector[i]=copy[(i+steps) % vector.length]
  end
end
```

juggeling solution

temp<-1 then 1<-4<-8<-12 then 12<-temp

```
# lib/02-1-vector-rotation/3.rb

require 'rational'
def rotate(vector,distance)
  length = vector.length
  distance.gcd(length).times do |start|
    temp = vector[start]
    offset = start
    while true do
      swap_with = (offset + distance) % length
      break if swap_with == start
      vector[offset] = vector[swap_with]
      offset = swap_with
    end
    vector[offset] = temp
  end
  vector
end
```

Chapter 02 2 anagrams

Task

- For a given dictionary, find all anagrams and list them
- output in lines: `eenprsst = presents, serpents`

Solution

An anagram for ‘no’ is ‘on’, meaning same letters - different order.

- Store each word in a Hash, where the key are the words sorted letters and the values are all words with the same letters
- Remove all keys that only have a single anagram

```
# lib/02-2-anagrams/1.rb

anagrams = Hash.new([])
input.each {|word| anagrams[word.split('').sort] += [word]}
anagrams.each do |anagram,words|
  words.uniq!
  next if words.size == 1
  puts "#{anagram} = #{words*', '}"
end
```

Chapter 04 binary search

Task

- for a sorted set `[1,2,4,5,6]` what index has 4 ?
- return nil when it is not contained

Solution

Simple

```
list.index(value)
```

Binary search

Grab the middle element, if it is to large, continue left of the middel. If it is to small, continue right of the middel.
Return nil when no elements are left in the search space.

```
# lib/04-binary-search/1.rb

def binary_find(range,searched)
  middle = range.length/2
  chosen = range[middle]

  return middle if chosen == searched
  return nil if range.length <= 1

  if chosen < searched
    #search in the upper range, add middle to resulting
      ⏎index
    found = binary_find(range[middle..-1],searched)
    if found then middle+found else nil end
  else
    #search in lower range
    binary_find(range[0...middle],searched)
  end
```

```
end
```

Test

```
# lib/04-binary-search/2.rb

require 'spec'
describe :binary_find do
  {
    []=>1,
    [1]=>1,
    [1,3]=>3,
    [1,3,4]=>3,
    [1,3,4,6,8,9,12,15,17,20]=>17
  }.each do |range,searched|
    it "find inside #{range.length} element array" do
      binary_find(range,searched).should ==
        ⏎range.index(searched)
    end
    it "returns nil when nothing could be found inside a
      ⏎#{range.length} element array" do
      binary_find(range,2).should == nil
    end
  end
end
```

Chapter 06 algorithm design

Task

Find the maximum sum inside a range.
`[1,2,-4,2,4,-3,1]` => `[2,4]` => 6

Solutions

Cubic time O(n³)

Find the maximum of every possible sub-range.

```
# lib/06-algorithm-design/1-cubic.rb

#runtime O(n³) 2 loops + the sum loop
def find_max_sum_range(range)
  max = 0
  0.upto(range.length) do |start|
    start.upto(range.length-1) do |end_at|
      max = [range[start..end_at].sum, max].max
    end
  end
  max
end
```

Quadratic time O(n²)

Find the maximum of every possible sub-range.
Do not calculate the sum for every sub-range, rather just add each new element.

`[1,2,3]`: `[1]` => 1; `[1,2]` = 1+2 = 3; `[1,2,3]` = 3+3 = 6

```
# lib/06-algorithm-design/2-quadratic.rb

def find_max_sum_range(range)
  max = 0
```

```
  0.upto(range.length) do |start|
    sum=0
    start.upto(range.length-1) do |end_at|
      sum+=range[end_at]
      max = [sum, max].max
    end
  end
  max
end
```

Less than quadratic time O(n²)

Find the maximum of every possible sub-range.
Build sums, where `sums[2] == range[0...2].sum` and therefore `range[1...3] = sums[1]-sums[3]`

```
# lib/06-algorithm-design/3-quadratic-culmulative.rb

def find_max_sum_range(range)
  sums=[]
  range.each{|x|sums << sums.last.to_i+x}

  max = 0
  0.upto(range.length) do |start|
    start.upto(range.length-1) do |end_at|
      start_sum = if start == 0 then 0 else sums[start-1]
        ⏎end# sums[-1] == sums.last
      sum_of_range = sums[end_at]-start_sum
      max = [sum_of_range, max].max
    end
  end
  max
end
```

Logarithmic time O(n log n)

Divide the problem into 2 smaller, equal problems. (compare: Binary search) Maximum of left / right and the maximum of ranges, that cross the middle are comparen.

```
# lib/06-algorithm-design/4-logarithmic.rb

def find_max_sum_range(range)
  #trivial
  return 0 if range.empty?
  return [0,range[0]].max if range.length == 1

  #divide
  middle = range.length / 2
  left = range[0...middle]
  right = range[middle..-1]

  #start from the middel, and build sums to left/right end,
    ⏎to find max, then add them
  middle_max = [left.reverse, right].sum do |sub_range|
    max = sum = 0
    sub_range.each do |value|
      sum+=value
      max = [max,sum].max
    end
    max
  end

  [
    middle_max,
    find_max_sum_range(left),
    find_max_sum_range(right)
  ].max
end
```

Linear time O(n)

Compare the sums of all sub-parts. A part ends when its sum drops below 0.

```ruby
# lib/06-algorithm-design/5-linear.rb

def find_max_sum_range(range)
  max = partial_max = 0
  range.each do |x|
    # keep adding to partial max unless it sinks below 0
    partial_max = [partial_max+x, 0].max
    max = [partial_max, max].max
  end
  max
end
```

Chapter 07 estimation

Task and solutions

How much water flows out of the Mississippi River in a day?

Guessing:

- Near its mouth the river is about 1 km wide and 5 m deep
- Rate of flow is 5km / h , equals 120 km/day

Calculation:
1 km * 0.005 km * 120 km/day = 0.60 km^3/day

How long does it take to fill your pool with the gardenhose … ?

Little's Law

"The average number of things in the system is the product of the average rate at which things leave the system and the average time each one spends in the system."

For example a restaurant has 60 seats and a normal person will eat for 1 hour.
When there are 15 people in line before you you will have to wait for ?

```
Leaving * Time spent = In System

In the system = 60 people
Time spent = 1 hour

Leaving = 60 people / 1 hour = 1 person / minute
You wait: 15 minutes!
```

What is your cities death rate … ?

Chapter 13 searching

Task

- insert randomly generated numbers into a set
- insert one-by-one
- the set must not contain duplicates after a insertion
- return a sorted set, with the requested size, after the last insertion

Solutions

1. Hash

Insert unless key exists, sort at the end

```
# lib/13-searching/1-hash.rb

class SetHash
  def initialize(maximum_value)
    @maximum = maximum_value
  end

  def generate(size)
    raise "size too big" if size >= @maximum #would runs
      ⏎endless
    @set = {}
    while @set.size < size
      random = rand(@maximum)
      @set[random]=true unless @set[random]
    end
    @set.keys.sort
  end
end
```

2. Array

Insert unless element is included, sort at the end

```
# lib/13-searching/2-naive-array.rb

class SetArray
  def initialize(maximum_value)
    @maximum = maximum_value
  end

  def generate(size)
    raise "size too big" if size >= @maximum #would runs
      ⏎endless
    @set = []
    while @set.size < size
      random = rand(@maximum)
      @set << random unless @set.include?(random)
    end
    @set.sort
  end
end
```

3. Sorted-Linked-List

A linked list of sorted elements, for easy injection. @head = } Go thorugh the elements until the insertion position is found.

```
# lib/13-searching/3-linked.rb

class SetLinked
  def initialize(maximum_value)
    @maximum = maximum_value
  end

  def generate(size)
    raise "size too big" if size >= @maximum #would runs
      ⏎endless
    reset
```

```
  insert(rand(@maximum)) while @size < size
  flatten
end

def reset
  @head = nil
  @size = 0
end

def flatten
  flat = []
  link = @head
  while true
    flat << link[:value]
    break unless link[:next]
    link = link[:next]
  end
  flat
end

#insert a number if its new, and update @size counter
def insert(number)
  return if include?(number)
  @head = insert_link(@head,number)
  @size += 1
end

def insert_link(link,number)
  return {:value=>number,:next=>link} if not link or
    ⏎number <= link[:value]
  link[:next] = insert_link(link[:next],number)
  link
end

def include?(number)
  link = @head
```

```
      while link
        return true if link[:value] == number
        return false if link[:value] > number
        link = link[:next]
      end
      return false
    end
  end
```

Using an array (`[2,[3,[5,nil]]]`) instead of the more-readable `hash[:next]`, only saves 1/100th of the time.

4. Sorted Array

Place elements in a sorted Array, find insertion position using binary-search.

```
  # lib/13-searching/4-binary-array.rb

  class BinaryArray
    def initialize(maximum_value)
      @maximum = maximum_value
    end

    def generate(size)
      raise "size too big" if size >= @maximum #would runs
        ⏎endless
      @set = []
      while @set.size < size
        random = rand(@maximum)
        found, position = binary_find(random)
        @set.insert(position,random) unless found
      end
      @set
    end

    # search inside the ordered @set
```

```
    # return [found,position] where position is where it was
      ⏎found
    # or if not found, where it should be inserted
    def binary_find(value,left=0,right=@set.length-1)
      length_to_search = right+1-left
      middle = left + length_to_search/2

      #found or not found?
      return [true,middle] if @set[middle] == value
      if length_to_search <= 1
        middle += 1 if @set[middle] and @set[middle] < value
        return [false,[middle,0].max]
      end

      #not sure yet, recurse!
      if @set[middle] > value
        binary_find(value,left,middle-1)
      else
        binary_find(value,middle+1,right)
      end
    end
  end
```

Time

1. O(n) - lookups always take the same time
2. O(n²) - the longer the array, the longer the lookup takes
3. O(n²) - the longer the array, 1/2(mean) the longer the lookup takes
4. n log n - divide and conquer

Performance

Inserting x items in 2 seconds:

1.
2. 000
3.
4. 000
5. ~1.250
6.

7. 000

Conclusion

- nothing beats linear time
- when in doubt choose native objects over self-built (2. vs 3.)
- when self-built has a lower ‘time’, investigate performance

Chapter 14 heaps

Task

- build and maintain a sorted heap
- insert from bottom or top
- extract the smallest element
- build a sort method, to sort any array, using the heap

Solutions

A heap, where each nodes children are higher then their parent. 2 3 4 5 7 10 12
Converted to an array, by going from top to bottom and left to right,
results in `[2,3,4,5,7,10,12]`.

Insertion from bottom

Append to the array and then let the new element flow up (swap with parent)
until it reaches a parent that is higher than itself.

```
# lib/14-heaps/1.rb

class Heap
  def initialize
    @values = []
  end

  def insert_bottom(value)
    @values << value
    position = @values.length - 1
    while true
      return if position == root
      return if @values[parent(position)] < value
      swap(position, parent(position))
      position = parent(position)
    end
  end
```

```
  private

    def swap(a,b)
      @values[a], @values[b] = @values[b], @values[a]
    end

    def root
      0
    end

    def parent(i)
      (i+1)/2 - 1
    end

    def left_child(i)
      (2*i)+1
    end

    def right_child(i)
      (2*i)+2
    end
  end
```

Insertion from top

Prepend to the array and then let the new element sink down (swap with smallest child) until it reaches a pair of children that are lower than itself or a position without any children.

```
  # lib/14-heaps/2.rb

  class Heap
    def insert_top(value)
      @values.unshift(value)
      position = 0
```

```
    while true
      smallest_child = smallest_child(position)
      break if not @values[smallest_child] or
        ⏎@values[smallest_child] >= value
      swap(position, smallest_child)
      position = smallest_child
    end
  end

protected

  def smallest_child(position)
    left,right = left_child(position),
      ⏎right_child(position)
    return left unless @values[right]
    return right unless @values[left]
    if @values[left] <= @values[right]
      left
    else
      right
    end
  end
end
```

Extract the smallest element

The heap is sorted, so the top element will always be the smallest.
Take it out and insert the last element from top, to restore the order (since we do not know if the second or third element is the smallest)

```
# lib/14-heaps/3.rb

class Heap
  def extract_smallest
    smallest = @values.shift
    insert_top(@values.pop)
```

```
      smallest
    end
  end
```

Sort an Array using a heap

Fill the heap with the elements of the array and then extract the smallest until the array is empty.

```
# lib/14-heaps/4.rb

class Heap
  def self.sort(array)
    h = Heap.new
    array.each {|x|h.insert_top(x)}
    array.map{h.extract_smallest}
  end
end
```

Runtime: (n * time to find the minimum) + (n/2 * time for insert), see table below.

Comparison of runtime to other implementations

	insert	**find minimum**	**n of each**
Sorted Array	O(n)	O(1)	$O(n^2)$
Heap	O(log n)	O(log n)	O(n log n)
Unsorted Array	O(1)	O(n)	$O(n^2)$

Chapter 15 1 count words

Task & Solutions

Unique words

Show all words that are inside a given text (unique)!

```
# lib/15-1-count-words/1.rb

puts input.readlines(' ').uniq * "\n"
```

Word counting

Show all words sorted by number of occurance!

Use a 0-based Hash to mark how often a word has occured.

```
# lib/15-1-count-words/2.rb

occurances = Hash.new(0)
input.each(' ').each{|word| occurances[word]+=1}
puts occurances.to_a.sort_by{|word,count|count}.map{|x|x*'
  ⏎times '} * "\n"
```

Benchmarking

Benchmark occurance counting solution!

```
# lib/15-1-count-words/3-1.rb

occurances = nil

processing = measure do
  occurances = Hash.new(0)
  input.each(' '){|word| occurances[word]+=1}
end
```

```
printing = measure do
  puts
    ⏎occurances.to_a.sort_by{|word,count|count}.map{|x|x*'
    ⏎times '} * "\n"
end
```

Optimization

Optimize for performance!

Reading the File once and splitting later.

```
# lib/15-1-count-words/3-2.rb

occurances = nil
processing = measure do
  occurances = Hash.new(0)
  input.read.split(' ').each{|word| occurances[word]+=1}
end
printing = measure do
  puts
    ⏎occurances.to_a.sort_by{|word,count|count}.map{|x|x*'
    ⏎times '} * "\n"
end
```

Benchmarking of optimized occurance search
shows that it is faster (2.5s vs 3.0s for processing)

Memroy usage: (“puts memory” inside the processing loop) normal: mapped: 17652K writeable/private: 2180K shared: 0K optimized: mapped: 109892K writeable/private: 94420K shared: 0K optimized: the text (4.x mb) is in memory twice (once for `IO.read` and
once for `text.split`, no garbage collection was run)

Chapter 15 2 duplicate phrases

Task

Find the longest duplicate substring in a text.

Solutions

Simple but slow

Go through all possible substrings, and see if they are

- longer than the current longes duplicate substring
- occure > 2 times in the text

Time: $O(n^2)$

```
# lib/15-2-duplicate-phrases/1.rb

longest = ''
time = measure do
  0.upto(text.length) do |start|
    start.upto(text.length) do |last|
      substring = text[start..last]
      longest = substring if substring.length >
        ⏎longest.length and text.scan(substring).count ==
        ⏎2
    end
  end
end
```

Suffix array

- Build a suffix-array, where each possible suffix is stored
  abc -> `[abc,bc,c]`
- Sort this array to to move substrings that start with the same letters nearby
- find the longest commong prefix 2 neighboring suffixes share
  `[a,abc,abd,ac]` -> longest duplicate substring is ‘ab’

Time: O(n)

```
# lib/15-2-duplicate-phrases/2.rb

longest = ''
time = measure do
  #store each suffix
  suffixes = []
  0.upto(text.length) do |start|
    suffixes << text[start..text.length]
  end

  #sort them so that equals are next to each other
  suffixes.sort!

  suffixes.each_with_index do |suffix,i|
    common = common_prefix(suffix,suffixes[i+1])
    longest = common if common.length > longest.length
  end
end
```

Chapter 15 3 markov

Task

Generate a markov text from given seed data.

Solution

Generating random text by truely random placement of letters would be unnatural. sovudhbyeonnapdubvne Every letter in the alphabet has a certain probability to occur after another letter. l -> l=5% e=10% o=10% i=8% …. This yields somewhat natural text lleno deister If we take more than one letter into account, readability inncreases. # markov-4 (4 letters used when calculating propability) “Hell” -> o = 80%, ’ ‘=20% The more letters we take into account, the more natural the generated words sound.
What we can do with letters, we can do with whole words!

- calculate the probability word A follows on word B, using seed data
- generate random text

```
# lib/15-3-markov/1.rb

#build possible_successors for any used word
possible_successors = Hash.new([])
0.upto(text.length-1) do |i|
  possible_successors[text[i]] << text[i+1]
end

#build text by randomly adding possible_successors
random_text = [text.first]
0.upto(50).map do |i|
  random_text << possible_successors[random_text.last].rand
end

puts random_text * ' '
```

Chapter 16 random samples

Tasks and Solution

Simple

Select n random, unique samples from a list.

```
# lib/16-random-samples/1-n-samples.rb

list = (1..200).to_a
samples = []
while samples.length != n
  samples << rand(list.length)
  samples.uniq!
end
samples.map!{|i|list[i]}
```

More complex

Select n random, unique samples from a list, with a maximum index of m.

```
# lib/16-random-samples/2-n-samples.rb

#Same as before, but the list is limited...
list = (1..200).to_a[0..m]
```

Combination

Randomly combine experiments 1,2,3 with stress factors low, medium and high.
Example result: 2 high, 1 low, 3 medium

```
# lib/16-random-samples/3-combinations.rb

experiments = [1,2,3]
stresses = ['low','medium','high']
result = []
```

```
experiments.count.times {
  experiment =
    ⏎experiments.delete_at(rand(experiments.count))
  stress = stresses.delete_at(rand(stresses.count))
  result << "#{experiment} #{stress}"
}
```