

Segunda Práctica Calificada

Alejandro Escobar Mejia 20210496g

Parte 1 Algoritmos, Programación Orientada a Objetos

1. Escribe una función que acepte una cadena que contenga todas las letras del alfabeto excepto una y devuelva la letra que falta. Por ejemplo, la cadena the quick brown box jumps over a lazy dog contiene todas las letras del alfabeto excepto la letra f. La función debe tener una complejidad temporal de $O(n)$. (1 punto)

```
def encontrar_letra_faltante(cadena)
  #Inicializamos el alfabeto
  alfabeto = 'abcdefghijklmnopqrstuvwxyz'
  # Inicializamos un contador para cada letra del alfabeto
  contador = Array.new(26, 0)

  cadena.each_char do |letra|
    if letra.match?(/[a-zA-Z]/)
      # Convierte la letra a minúscula
      letra = letra.downcase
      # Calculamos la posición en el alfabeto (0-25) usando el
      # valor ASCII
      indice = letra.ord - 'a'.ord
      contador[indice] += 1
    end
  end

  contador.each_with_index do |count, i|
    if count.zero?
      # La letra que falta es la que tiene un contador de 0
      return ('a'.ord + i).chr
    end
  end

  # Si no falta ninguna letra, devuelve nil
  nil
end

cadena = "the quick brown box jumps over a lazy dog"
letra_faltante = encontrar_letra_faltante(cadena)
puts "La letra que falta es: #{letra_faltante}"
```

```

1  # Parte 1.rb
2
3  alfabeto = 'abcdefghijklmnopqrstuvwxyz'
4  # Inicializamos un contador para cada letra del alfabeto
5  contador = Array.new(26, 0)
6
7  cadena.each_char do |letra|
8    if letra.match?(/[a-zA-Z]/)
9      # Convierte la letra a minúscula
10     letra = letra.downcase
11     # Calculamos la posición en el alfabeto (0-25) usando el valor ASCII
12     indice = letra.ord - 'a'.ord
13     contador[indice] += 1
14   end
15 end
16
17 contador.each_with_index do |integer count, integer i|
18   if count.zero?
19     # La letra que falta es la que tiene un contador de 0
20     return ('a'.ord + i).chr
21   end
22 end
23 # Si no falta ninguna letra, devuelve nil
24 nil
25 end
26
27 cadena = "the quick brown box jumps over a lazy dog"
28 letra_faltante = encontrar_letra_faltante(cadena)
29 puts "La letra que falta es: #{letra_faltante}"
30
31 encontrar_letra_faltante

```

- Un árbol binario ordenado es aquel en el que cada nodo tiene un valor y hasta 2 hijos, cada uno de los cuales es también un árbol binario ordenado, y el valor de cualquier elemento del subárbol izquierdo de un nodo es menor que el valor de cualquier elemento en el subárbol derecho del nodo. Defina una clase colección llamada `BinaryTree` que ofrezca los métodos de instancia `<<` (insertar elemento), `empty?` (devuelve cierto si el árbol no tiene elementos) y `each` (el iterador estándar que devuelve un elemento cada vez, en el orden que tú quieras). (2 puntos)
- Extiende la clase de tu árbol binario ordenado para que ofrezca los siguientes métodos, cada uno de los cuales toma un bloque: `include?(elt)` (devuelve cierto si el árbol incluye a `elt`), `all?` (cierto si un bloque dado es cierto para todos los elementos), `any?` (cierto si un bloque dado es cierto para alguno de sus elementos), `sort` (ordena los elementos) (1 puntos)

```

class BinaryTree
  include Enumerable

  attr_accessor :value, :left, :right

  def initialize(value)
    @value = value
    @left = nil
    @right = nil
  end

  def <<(element)
    if element <= @value
      @left.nil? ? @left = BinaryTree.new(element) : @left
    end
  end

```

```

<<(element)
  else
    @right.nil? ? @right = BinaryTree.new(element) : @right
  <<(element)
  end
end

def empty?
  @left.nil? && @right.nil?
end

def each(&block)
  @left.each(&block) if @left
  block.call(@value)
  @right.each(&block) if @right
end

def include?(element)
  return true if element == @value
  if element < @value
    return @left.include?(element) if @left
  else
    return @right.include?(element) if @right
  end
  false
end

def all?(&block)
  return false unless block.call(@value)
  left_result = @left.nil? ? true : @left.all?(&block)
  right_result = @right.nil? ? true : @right.all?(&block)
  left_result && right_result
end

def any?(&block)
  return true if block.call(@value)
  left_result = @left.nil? ? false : @left.any?(&block)
  right_result = @right.nil? ? false : @right.any?(&block)
  left_result || right_result
end

def sort
  sorted_elements = []
  each { |element| sorted_elements << element }
  sorted_elements.sort
end

tree = BinaryTree.new(5)
tree << 3
tree << 7
tree << 2
tree << 4
tree << 6
tree << 8

# Comprobar si el árbol contiene un elemento
puts tree.include?(3)
puts tree.include?(9)

# Comprobar si todos los elementos cumplen una condición

```

```
puts tree.all? { |element| element < 10 }
puts tree.all? { |element| element < 5 }

# Comprobar si algún elemento cumple una condición
puts tree.any? { |element| element > 8 }
puts tree.any? { |element| element > 10 }

# Ordenar los elementos del árbol
sorted_elements = tree.sort
puts sorted_elements.inspect
```

The screenshot shows a Ruby IDE with a project named 'PC2_ESCOBAR MEJIA ALEJANDRO'. The file explorer on the left shows 'Parte 1.rb' and 'Parte2-3.rb'. The main editor displays the code from the previous block, with line numbers 66 to 84. The code defines a binary tree structure and performs several operations: checking if the tree is empty, checking if all elements are less than 10 and 5, checking if any element is greater than 8 and 10, and sorting the elements. The bottom panel shows the execution output for 'Parte2-3' using 'C:\Ruby32-x64\bin\ruby.exe'. The output is as follows:

```
C:\Ruby32-x64\bin\ruby.exe "C:/Users/Alumno/Desktop/PC2_ESCOBAR MEJIA ALEJANDRO/Parte2-3.rb"
true
false
true
false
false
false
false
[2, 3, 4, 5, 6, 7, 8]
```

The status bar at the bottom indicates the file is 'Parte2-3.rb', the current line is 6, and the column is 15. The encoding is 'CRLF', the font is 'UTF-8', and there are 2 spaces.