Eduardo Espinosa Avila

Ciclo de vida del desarrollo de software

"Software eventually gained the same respect as any other discipline" (Margaret Hamilton)

"After three days without programming, life becomes meaningless" (Tao)

"On the whole it is still 'Code first, debug later' instead of 'Think first, code later', which is a pity" (Dijkstra)

a) Captura de requerimientos

> "The difference between the almost right word and the right word is really a large matter" (Twain)

"Realizar un programa que encuentre un valor dentro de una lista ordenada"

Requerimientos funcionales

Se debe realizar una función que recibirá como parámetros: una lista ordenada y el valor a buscar; y regresará el índice de la posición que ocupa el valor dentro de la lista

b) Diseño

> "When a program is being tested, it is too late to make design changes" (Tao)

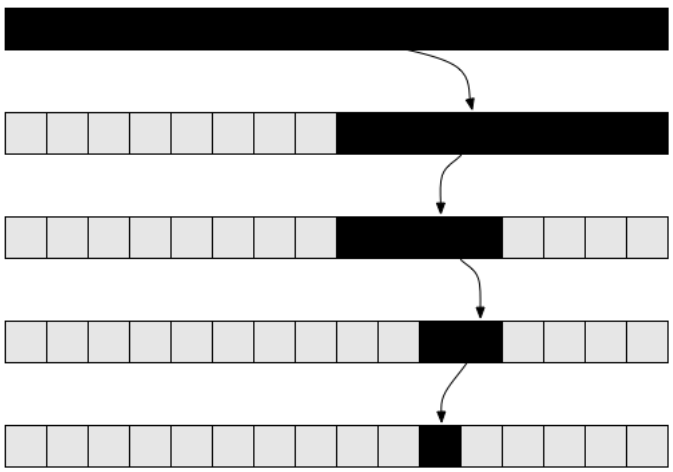
Con base en la experiencia y tomando en cuenta la complejidad algorítmica, como la lista se recibe ya ordenada, se recomienda utilizar búsqueda binaria, dado que se ejecuta en tiempo $O(log_2(n))$, mientras que la búsqueda secuencial requiere $O(n)$

> Binary Searching for Cinderella & Hunting Dragons with Binary Search

El algoritmo es sencillo:

- Comparar el valor buscado con el elemento central:
  - Si es mayor, buscar en la mitad superior con el mismo algoritmo
  - Si es menor, buscar en la mitad inferior con el mismo algoritmo
  - Si es igual, devolver el índice de dicho elemento central

Como lo ilustra la imagen:



función busqueda_binaria:

recibe:

- lista, una lista ordenada
- valor, el elemento a buscar dentro de la lista

devuelve:

- el índice que ocupa el valor dentro de la lista,
- -1 si el valor no se encuentra en la lista

```
Pseudocódigo:
busqueda_binaria(lista, valor)
  ini <- 0
  fin <- tamaño(lista)-1
  mientras ini<=fin
    med <- truncar((ini+fin)/2)
    val_med <- lista[med]

    si val_med<valor
      ini <- med+1
    otro si val_med>valor
      fin <- med-1
    si no
      regresar med
  regresar -1
```

> "We should reshape our field of programming in such a way that the mathematician's methods become equally applicable to our programming problems, for there are no other means" (Dijkstra)

¿Es posible demostrar que el algoritmo funciona de manera correcta?

¡Claro!

```
En el paso i-ésimo, la lista accesible y las variables de acceso tienen la forma:
            ... |          m          | ...
                ^          ^          ^
               ini        med        fin

Se revisan los tres posibles casos:

1. val_med < valor, indica que hay que buscar en la parte superior de la lista;
   la nueva configuración de la lista y las variables es:
            ... |          m          | ...
                           ^          ^
                          ini        fin
  Garantizando que ya no hay acceso a los índices menores que med y conservando
  verdadera la condición del ciclo: ini<=fin

2. val_med > valor, es decir, hay que buscar en la parte inferior de la lista;
   la nueva configuración de la lista y las variables es:
            ...  |          m          | ...
                 ^          ^
                ini        fin
  Garantizando que ya no hay acceso a los índices mayores que med y conservando
  verdadera la condición del ciclo: ini<=fin

3. val_med = valor, en este caso la configuración se mantiene sin cambios y la
   función devuelve el valor actual de med, que es el índice buscado

4. finalmente, en el caso extremo en que tanto ini como fin tengan el mismo
   valor, las tres variables señalan el mismo elemento de la lista y se tiene
   la configuración siguiente:
            ...  m  ...
                 ^
           [ini,med,fin]

  Se deben revisar los tres casos posibles:
  a) val_med < valor, la nueva configuración es:
            ...  m  ...
```

```
                ^^
            [fin][ini]
      Haciendo falsa la condición del ciclo: ini<=fin que termina y la función
      devuelve el valor -1 tal como se especificó en el diseño

  b) val_med > valor, la nueva configuración es:
           ...  m  ...
               ^^
           [fin][ini]
      Haciendo falsa la condición del ciclo: ini<=fin que termina y la función
      devuelve el valor -1 tal como se especificó en el diseño

  c) val_med = valor, en este caso la configuración se mantiene sin cambios y la
     función devuelve el valor actual de med, que es el índice buscado

QED

"Beware of bugs in the above code; I have only proved it correct, not tried it"
D. Knuth
```

c) Construcción

> "A well-written program is its own Heaven; a poorly-written program is its own Hell" (Tao)

```
def busqueda_binaria(lista, valor):
  """
  recibe:
  lista, una lista ordenada
  valor, el elemento a buscar dentro de la lista

  devuelve:
  el índice que ocupa el valor dentro de la lista,
  -1 si el valor no se encuentra en la lista

  Pruebas con doctest:

  >>> busqueda_binaria([10, 33, 35, 49, 55, 59, 67, 68, 77, 96], 68)
  7

  >>> busqueda_binaria([10, 33, 35, 49, 55, 59, 67, 68, 77, 96], 33)
  1

  >>> busqueda_binaria([10, 33, 35, 49, 55, 59, 67, 68, 77, 96], 12)
  -1

  #error intencional
  >>> busqueda_binaria([10, 33, 35, 49, 55, 59, 67, 68, 77, 96], 15)
  5
  """
  ini = 0
  fin = len(lista)-1
  while ini<=fin:
    med = (ini+fin)//2 # En Python 3 '//' indica división entera
    val_med = lista[med]
    if val_med < valor:
      ini = med+1
    elif val_med > valor:
      fin = med-1
    else:
      return med
  return -1

```

d) Pruebas

> "Beware of bugs in the above code; I have only proved it correct, not tried it" (Knuth)

```
# Ejecución de las pruebas con doctest:
import doctest
doctest.run_docstring_examples(busqueda_binaria, globals(), verbose=False)
```

```
PYDEV DEBUGGER WARNING:
sys.settrace() should not be used when the debugger is being used.
This may cause the debugger to stop working correctly.
If this is needed, please check:
http://pydev.blogspot.com/2007/06/why-cant-pydev-debugger-work-with.html
to see how to restore the debug tracing back correctly.
Call Location:
  File "/usr/lib/python3.8/doctest.py", line 1487, in run
    sys.settrace(save_trace)

**********************************************************************
File "__main__", line 23, in NoName
Failed example:
    busqueda_binaria([10, 33, 35, 49, 55, 59, 67, 68, 77, 96], 15)
Expected:
    5
Got:
    -1
```

También puede usarse *pytest*, más cercano a la idea de *unittest*

```
import pytest

def test_busqueda_binaria():
  lista = [10, 33, 35, 49, 55, 59, 67, 68, 77, 96]
  valores_resultados = {68:7, 33:1, 12:-1, 15:5}
  for valor,resultado in valores_resultados.items():
    assert busqueda_binaria(lista, valor) == resultado

test_busqueda_binaria()
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-3-8b5270792cdb> in <module>
      7     assert busqueda_binaria(lista, valor) == resultado
      8
----> 9 test_busqueda_binaria()

<ipython-input-3-8b5270792cdb> in test_busqueda_binaria()
      5   valores_resultados = {68:7, 33:1, 12:-1, 15:5}
      6   for valor,resultado in valores_resultados.items():
----> 7     assert busqueda_binaria(lista, valor) == resultado
      8
      9 test_busqueda_binaria()

AssertionError:
```

SEARCH STACK OVERFLOW

Con *unittest*

```
import unittest

class TestNotebook(unittest.TestCase):

  def test_busqueda_binaria(self):
    lista = [10, 33, 35, 49, 55, 59, 67, 68, 77, 96]
    valores_resultados = {68:7, 33:1, 12:-1, 15:5}
    for valor,resultado in valores_resultados.items():
      self.assertEqual(busqueda_binaria(lista, valor), resultado)

unittest.main(argv=[''], verbosity=2, exit=False)
```

```
test_busqueda_binaria (__main__.TestNotebook) ... FAIL

======================================================================
FAIL: test_busqueda_binaria (__main__.TestNotebook)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-4-6ec7e2a72b29>", line 9, in test_busqueda_binaria
    self.assertEqual(busqueda_binaria(lista, valor), resultado)
AssertionError: -1 != 5
```

```
----------------------------------------------------------------------
Ran 1 test in 0.003s

FAILED (failures=1)
<unittest.main.TestProgram at 0x7f9d4e3b3a60>
```

```
# ************************************************************************
```

e) Implementación

Cierre y gestión de cambios

> "Though a program be but three lines long, someday it will have to be maintained" (Tao)

El código realizado cumple con las especificaciones, pero ...

> "The only phrase I've ever disliked is, 'Why, we've always done it that way.' I always tell young people, 'Go ahead and do it. You can always apologize later.'" (Grace Hopper)

Si posteriormente se solicita desarrollar un programa que realice ordenamiento por *mezcla* (*mergesort*), notaremos que también debe calcularse el elemento central; puede existir la tentación de "reutilizar" el cálculo copiando y pegando esa línea.

Sin embargo hay que tener presente el principio DRY (*Don't Repeat Yourself*): ante la tentación de copiar código es mucho mejor abstraer las operaciones a funciones que realicen esa operación.

Es importante notar que no se modifican los requerimientos, el algoritmo de alto nivel ni la prueba de correctitud del pseudocódigo.

Con estas nuevas consideraciones, el nuevo ciclo de desarrollo es:

Diseño

```
Pseudocódigo:
busqueda_binaria(lista, valor)
  ini <- 0
  fin <- tamaño(lista)-1
  mientras ini<=fin
    med <- calcula_medio(ini, fin)
    val_med <- lista[med]
    si val_med<valor
      ini <- med+1
    otro si val_med>valor
      fin <- med-1
    si no
      regresar med
  regresar -1

calcula_medio(ini, fin)
  regresar truncar((ini+fin)/2)
```

```
# Construcción
def busqueda_binaria(lista, valor):
  """
  recibe:
    lista, una lista ordenada
    valor, el elemento a buscar dentro de la lista

  devuelve:
    el índice que ocupa el valor dentro de la lista,
    -1 si el valor no se encuentra en la lista

  Pruebas con doctest:

  >>> busqueda_binaria([10, 33, 35, 49, 55, 59, 67, 68, 77, 96], 68)
  7

  >>> busqueda_binaria([10, 33, 35, 49, 55, 59, 67, 68, 77, 96], 33)
  1

```

```
    >>> busqueda_binaria([10, 33, 35, 49, 55, 59, 67, 68, 77, 96], 12)
    -1
    """
    ini = 0
    fin = len(lista)-1
    while ini<=fin:
      med = calcula_medio(ini, fin)
      val_med = lista[med]
      if val_med < valor:
        ini = med+1
      elif val_med > valor:
        fin = med-1
      else:
        return med
    return -1

  def calcula_medio(ini, fin):
    """
    recibe:
      ini, fin valores de los que se obtendré el valor medio

    devuelve:
      el valor medio de los parámetros
    """
    return (ini+fin)//2 # En Python 3 '//' indica división entera

```

```
# Pruebas
import doctest
doctest.run_docstring_examples(busqueda_binaria, globals(), verbose=False)
```

```
# ********************************************************************
```

Implementación

Cierre y gestión de cambios

> "Though a program be but three lines long, someday it will have to be maintained" (Tao)

"Beware of bugs in the above code; I have only proved it correct, not tried it" (Knuth)

El nuevo código realizado además de cumplir con las especificaciones, cumple con abstracción funcional de operaciones comunes con otros programas pero ...

"Without the wind, the grass does not move. Without software hardware is useless" (Tao)

Si después de un tiempo, alguien con conocimientos de diseño digital y arquitectura de computadoras revisa el código, podría notar un posible problema (*bug*) en el programa; el análisis es el siguiente, el programa sin *docstring*:

```
def busqueda_binaria(lista, valor):
  ini = 0
  fin = len(lista)-1
  while ini<=fin:
    med = calcula_medio(ini, fin)
    val_med = lista[med]
    if val_med < valor:
      ini = med+1
    elif val_med > valor:
      fin = med-1
    else:
      return med
  return -1

def calcula_medio(ini, fin):
  return (ini+fin)//2
```

No resulta evidente, pero el problema está justamente en el cálculo del valor medio:

```
(ini+fin)//2
```

Para notarlo hay que recordar que los números enteros en las computadoras se representan en el formato llamado complemento a 2 (C2) y con cierto límite de bits; para ilustrarlo, supongamos una representación hipotética de cuatro bits.

Con 4 bits, hay 16 combinaciones posibles de ceros y unos y los números que se pueden representar en C2 están en el rango [-8,7], recordar que los negativos comienzan con 0 y los positivos con 1. Suponiendo que tenemos una lista ordenada con siete elementos y que el número es mayor que el elemento central, a nivel de bits, el cálculo implica la suma:

```
 0000   ini
+0111   fin
 ----
 0111   suma


Dividiendo entre 2:
 0011   med
```

Que es el número 3 y el nuevo valor para ini será 4, hasta aquí todo funciona de la forma esperada, pero si el valor buscado es mayor al almacenado en la posición 5, se debe volver a realizar el cálculo:

```
 0100   ini
+0111   fin
 ----
 1011   suma
```

¡Pero el número 1011 es negativo! (por la representación en C2)

```
1 # ************************************************************************
```

Una posible solución es pensar en utilizar enteros sin signo para el índice, sin embargo, el problema sigue existiendo, con los mismos 4 bits se pueden tener enteros sin signo en el intervalo [0,15] y si tenemos un arreglo con todos los elementos, podemos llegar a la situación siguiente:

```
  1000   ini
 +1111   fin
  ----
1|0111   suma => ¡Sobreflujo! (overflow)
```

¿Entonces no hay forma de evitar este problema?

¡Afortunadamente sí! => hay que cambiar la forma de obtener el valor medio evitando que la suma sobrepase el valor máximo que se puede almacenar

Diseño

¡Hagamos un poco de aritmética!

$$med = \frac{ini + fin}{2}$$

sumando un cero (*ini-ini*):

$$med = \frac{ini + fin + (ini - ini)}{2}$$

reagrupando:

$$med = \frac{2 \times ini + fin - ini}{2}$$

finalmente, separando:

$$med = ini + \frac{fin - ini}{2}$$

El nuevo pseudocódigo es:

```
busqueda_binaria(lista, valor)
  ...
```

```
calcula_medio(ini, fin)
  regresar ini+truncar((fin-ini)/2)
```

Con este cambio, se garantiza que no se presentará sobreflujo en el cálculo del valor medio.

Es importante notar nuevamente que no se modifican los requerimientos, el algoritmo de alto nivel ni la prueba de correctitud del pseudocódigo.

Finalmente, si tomamos en cuenta el rendimiento para cantidades grandes de datos, la división entre 2 puede realizarse con un *corrimiento a la derecha*, es más eficiente debido a que la operación de corrimiento necesita un sólo ciclo del reloj de la computadora para ejecutarse mientras que una división puede realizar varias restas.

Con estas consideraciones, obtenemos la siguiente versión:

```
def busqueda_binaria(lista, valor):
  """
  recibe:
    lista, una lista ordenada
    valor, el elemento a buscar dentro de la lista

  devuelve:
    el índice que ocupa el valor dentro de la lista,
    -1 si el valor no se encuentra en la lista

  Pruebas con doctest:

  >>> busqueda_binaria([10, 33, 35, 49, 55, 59, 67, 68, 77, 96], 68)
  7

  >>> busqueda_binaria([10, 33, 35, 49, 55, 59, 67, 68, 77, 96], 33)
  1

  >>> busqueda_binaria([10, 33, 35, 49, 55, 59, 67, 68, 77, 96], 12)
  -1
  """
  ini = 0
  fin = len(lista)-1
  while ini<=fin:
    med = calcula_medio(ini, fin)
    val_med = lista[med]
    if val_med < valor:
      ini = med+1
    elif val_med > valor:
      fin = med-1
    else:
      return med
  return -1

def calcula_medio(ini, fin):
  return ini+((fin-ini)>>1) # >>1 indica corrimiento a la derecha una posición
                            #     para obtener la división entre 2

```

```
# Pruebas
import doctest
doctest.run_docstring_examples(busqueda_binaria, globals(), verbose=False)
```

Implementación

Esta última versión resuelve los problemas mencionados no solamente para la búsqueda binaria sino para todos los códigos que dependen de la obtención del valor medio (como *mergesort*) y otros que pudieran requerirlo, gracias a la abstracción funcional, aún más, lo realiza de forma eficiente debido a la sustitución de la división por el corrimiento.

Además la experiencia puede usarse en otros escenarios, el método de bisección para obtener las raíces de una expresión algebráica también funciona sobre el cálculo del valor medio, en este caso de punto flotante, pero debe implementarse de forma que se prevenga el sobreflujo: el algoritmo que obtuvimos aquí.

Referencia: Extra, Extra - Read All About It: Nearly All Binary Searches and Mergesorts are Broken (Bloch)

"A pesar de que la idea básica de búsqueda binaria es realtivamente sencilla, los detalles pueden ser sorprendentemente complicados... — Donald Knuth"

"En 1946, John Mauchly mencionó por primera vez la búsqueda binaria como parte de Moore School Lectures, el primer conjunto de conferencias relacionado con las computadoras"

Wikipedia

"Never let fear get in the way! Don't be afraid to continue when things appear to be impossible, even when the so-called 'experts' say it is impossible. Don't be afraid to stand alone, to be different, to be wrong, to make and admit mistakes, for only those who dare to fail greatly can ever achieve greatly" Margaret Hamilton

```
1 #:)
```