Escritura Formal Basica De Un Problema*

Andres Vergara Cano[a], Felipe Morales Espitia[1]

[a] *Pontificia Universidad Javeriana, Bogotá, Colombia*

Abstract

En este documento se presenta el problema de como escribir formalmente un algoritmo en LATEX sobre como encontrar un elemento en un arreglo utilzando una combinación entre QuickSort y Busqueda Binaria con fundamento en las bases de dividir y conquistar

Keywords: algoritmo, escritura formal, secuencia, elemento, búsqueda

Análisis del Problema

El objetivo de este ejercicio es diseñar un algoritmo para encontrar un elemento específico dentro de una secuencia NO ordenada de datos. Este problema puede ser abordado de manera eficiente utilizando técnicas similares a las empleadas en la búsqueda binaria o quicksort, que es un algoritmo clásico para la búsqueda en listas ordenadas.

Descripción del Algoritmo "QuickBinario"

El algoritmo QuickBinario utiliza la estrategia de dividir y conquistar para buscar un elemento en una secuencia ordenada. El proceso se describe a continuación:

1. **Inicialización:**
   - Se definen dos índices, `left` y `right`, que marcan los límites inferior y superior del rango de búsqueda dentro del array `A`.
2. **Búsqueda:**
   - Mientras el índice `left` sea menor o igual al índice `right`, se calcula el índice del punto medio de la subsecuencia actual utilizando la fórmula:

$$\text{pivote} = \frac{\text{left} + \text{right}}{2}$$

*Este documento presenta la escritura formal de un algoritmo.

Email addresses: andres-vergarac@javeriana.edu.com (Andres Vergara Cano), moralese.j@javeriana.edu.com (Felipe Morales Espitia)

Algoritmo 1 Quick Binario.

```
1: procedure QuickBinario(A, left, right, x)
2:     if left > right then
3:         return -1                                  ▷ Elemento no encontrado
4:     end if
5:     pivote ← (left + right) ÷ 2
6:     pivote ← A[pivote]
7:     if pivote = x then
8:         return pivote                              ▷ Elemento encontrado
9:     else if x < pivote then
10:        return QuickBinario(A, left, pivote − 1, x)
11:    else
12:        return QuickBinario(A, pivote + 1, right, x)
13:    end if
14: end procedure
```

- Se obtiene el valor en la posición del punto medio:

$$\text{pivote} = A[\text{pivote}]$$

3. **Comparación:**
   - Se compara el valor en el punto medio con el elemento que estamos buscando (`x`):
     - Si pivote $= x$, se retorna `pivote`, ya que el elemento ha sido encontrado.
     - Si $x <$ pivote, se ajusta el índice `right` para buscar en la mitad inferior de la subsecuencia. La búsqueda se realiza de manera recursiva en el rango `left` a `pivote - 1`.
     - Si $x >$ pivote, se ajusta el índice `left` para buscar en la mitad superior de la subsecuencia. La búsqueda se realiza de manera recursiva en el rango `pivote + 1` a `right`.
4. **Resultado:**
   - Si el elemento se encuentra en alguna de las llamadas recursivas, se devuelve su posición.
   - Si el proceso termina sin encontrar el elemento (es decir, `left` supera a `right`), se retorna `-1`, indicando que el elemento no está en la secuencia.

Complejidad

La complejidad temporal del algoritmo Quick Binario es $O(\log n)$, donde n es el número de elementos en la secuencia. Esta eficiencia se debe a que, en cada paso, se elimina la mitad de la subsecuencia restante.

- **Complejidad Temporal:** En cada iteración del algoritmo, la longitud del rango de búsqueda se reduce a la mitad. Por lo tanto, el número de iteraciones necesarias para reducir el rango a un solo elemento es proporcional al logaritmo en base 2 del número de elementos, lo que resulta en una complejidad temporal de $O(\log n)$.

- **Teorema Maestro:** Para el algoritmo `QuickBinario`, la relación de recurrencia es:

$$T(n) = T(n/2) + O(1)$$

Donde:

- a: Número de subproblemas. Aquí, $a = 1$, ya que se busca en una mitad del array en cada paso.
- b: Factor de reducción del tamaño del subproblema. Aquí, $b = 2$, ya que se divide el array en dos partes.
- $f(n)$: Costo de la combinación de las soluciones de los subproblemas, aquí $f(n) = O(1)$.

Cálculo de $\log_b a$:

$$\log_b a = \log_2 1 = 0$$

Evaluación de $f(n)$:

$$f(n) = O(1)$$
$$n^{\log_b a} = n^0 = 1$$

Comparando $f(n)$ con $n^{\log_b a}$:

- **Caso 1:** Si $f(n) = O(n^c)$ con $c < \log_b a$.
- **Caso 2:** Si $f(n) = \Theta(n^{\log_b a})$.
- **Caso 3:** Si $f(n) = \Omega(n^c)$ con $c > \log_b a$ y se cumplen ciertas condiciones.

En este caso:

$$f(n) = O(1) \text{ y } n^{\log_b a} = 1$$

Dado que $f(n) = \Theta(1)$ es igual a $n^{\log_b a}$, aplica el **Caso 2**.

Resultado del Teorema Maestro: Para el **Caso 2**, la fórmula del Teorema Maestro nos da:

$$T(n) = \Theta(n^{\log_b a} \log^k n)$$

Dado que $\log_b a = 0$ y $k = 0$:

$$T(n) = \Theta(1 \cdot \log^0 n) = \Theta(\log n)$$

Por lo tanto, la complejidad del algoritmo `QuickBinario` es:

$$T(n) = \Theta(\log 2n)$$

1. Diseño del Problema

El objetivo de este problema es diseñar un algoritmo para encontrar un elemento específico en una secuencia ordenada de datos utilizando la estrategia de dividir y vencer. El algoritmo a desarrollar debe ser eficiente y estar basado en la búsqueda binaria.

1. Entradas:
   a) A: Un arreglo ordenado de elementos.
   b) $left$: El índice inferior del rango de búsqueda.
   c) $right$: El índice superior del rango de búsqueda.
   d) x: El elemento que se desea encontrar en el arreglo.
2. Salidas:
   a) La posición del elemento x en el arreglo A si el elemento se encuentra.
   b) -1 si el elemento no está en el arreglo.

2. Algoritmo de Solución

El algoritmo de solución propuesto es Quick Binario, que utiliza la estrategia de dividir y vencer para encontrar un elemento en una secuencia ordenada. A continuación se describen dos enfoques para implementar este algoritmo.

2.1. Algoritmo Recurrente

La versión recursiva de Quick Binario se basa en la llamada recursiva para buscar en la mitad del arreglo.

2.1.1. Invariante

El resultado intermedio p_i refleja el cálculo de la búsqueda en el subarreglo definido por los índices $left$ y $right$.

- Se establece el caso base donde $left > right$, indicando que el elemento no está en el subarreglo.
- Si $x <$ pivote, se realiza una llamada recursiva para buscar en la mitad inferior. Si $x >$ pivote, se realiza una llamada recursiva para buscar en la mitad superior.
- Se encuentra el elemento x o se determina que no está en el arreglo.

2.1.2. Análisis de Complejidad

El algoritmo recursivo Quick Binario tiene una complejidad $O(\log n)$. Cada llamada recursiva divide el rango de búsqueda a la mitad, lo que resulta en una profundidad de recursión proporcional al logaritmo en base 2 del número de elementos.

2.2. Notas de Implementación

Al implementar el algoritmo Quick Binario, se deben considerar las siguientes notas:

- Asegurarse de que el arreglo esté ordenado antes de aplicar el algoritmo.
- Manejar correctamente los índices para evitar desbordamientos y errores en los límites del arreglo.
- La implementación recursiva puede tener problemas de stack overflow para arreglos muy grandes; en estos casos, se puede considerar una versión iterativa del algoritmo.
- El algoritmo debe manejar el caso cuando $left > right$ correctamente, retornando -1 para indicar que el elemento no está presente.