Escritura del problema de la secuencia mayoritaria

Jessica Tatiana Naizaque Guevara[1]

[1]Departamento de Ingeniería de Sistemas, Pontificia Universidad Javeriana
Bogotá, Colombia
j.naizaque@javeriana.edu.co

11 de agosto de 2022

Resumen

En este documento se presenta la formalización del problema de la secuencia mayoritaria, junto con la descripción de los algoritmos de solución planteados por medio de los métodos de iteración y "dividir y vencer".

Palabras clave: secuencia mayoritaria, algoritmo, formalización, complejidad.

Índice

1. Formalización del problema

Dada una secuencia S de elementos $a_i \in \mathbb{T}$ tal que $|S| \geq 2$, determinar si es posible encontrar un número que se repita x veces cumpliendo la condición $|S| \leq x$. En caso afirmativo, es permitido llamar a la secuencia "mayoritaria", en caso contrario, la secuencia no sería mayoritaria.

1.1. Definición del problema de la "secuencia mayoritaria"

Así, el problema de la secuencia mayoritaria se define a partir de una secuencia S de elementos $a_i \in \mathbb{T}$, encontrar solo un elemento $b \in S$ donde $b_j, b_{j+1}, b_{j+2}, ..., b_x$ cumple con la relación $|S| \leq x$.

- Entrada: $S = \langle a_i \in \mathbb{T} \rangle \mid |S| \geq 2$.
- Salida: Cadena de caracteres que indica si la secuencia es mayoritaria o no, en caso afirmativo contiene el elemento.

2. Algoritmos de solución

2.1. Iterativo

La idea de este algoritmo es: crear dos arreglos que sean paralelos y tengan la función de almacenar el elemento y guardar la cantidad de veces que se encuentra repetido dicho valor, respectivamente. Lo anterior se hará recorriendo los elementos de la secuencia S hasta que alguno de los elementos tenga el valor de sus repeticiones igual o superior a $\lceil |S| \div 2 \rceil$. Al finalizar, se retornará una cadena que indique si la secuencia es mayoritaria o no, en el primer caso señalará el número o elemento mayoritario.

2.1.1. Pseudocódigo

Algoritmo 1 Algoritmo iterativo: Secuencia mayoritaria.

```
1:  procedure MajoritySequence(S)
2:      n ← |S|
3:      elements ← declare an empty array
4:      values ← declare an empty array
5:      elements.append(0)
6:      values.append(0)
7:      mayoritaria ← false
8:      i ← 1
9:      while i ≤ n ∧ mayoritaria = false do
10:         inArray ← false
11:         indexArray ← −1
12:         j ← 1
13:         while j ≤ |elements| ∧ inArray = false do
14:             if elements[j] = S[i] then
15:                 indexArray ← j
16:                 inArray ← true
17:             end if
18:             j ← j + 1
19:         end while
20:         if inArray = true then
21:             values[indexArray] ← values[indexArray] + 1
22:             if values[indexArray] ≥ ⌈n ÷ 2⌉ then
23:                 element ← elements[indexArray]
24:                 mayoritaria ← true
25:             end if
26:         else
27:             elements.append(S[i])
28:             values.append(1)
29:         end if
30:         i ← i + 1
31:     end while
32:     if mayoritaria = true then
33:         return "La secuencia es mayoritaria, el número es "+element.
34:     else
35:         return "La secuencia no es mayoritaria".
36:     end if
37: end procedure
```

2.1.2. Análisis de complejidad

Analizando el pseudocódigo anterior, se tienen tres posibles casos:

- Peor caso: $O(n^2)$, debido a que podría haber un caso en el que no haya ningún elemento repetido y deba recorrer la secuencia inicial y, luego, el arreglo de elementos buscando cada elemento sin éxito alguno.
- Caso promedio: $\theta(nlogn)$, los elementos están desorganizados, pueden existir unos al inicio, otros en la mitad u otros al final... Lo cual lo hace recorrer sin saber con anterioridad su determinado fin.
- Mejor caso: $\Omega(logn)$, debido a que si se tiene el arreglo ordenado y el elemento mayoritario se encuentra en las primeras posiciones, solo deberá recorrer la mitad de la secuencia inicial.

2.1.3. Invariante

En este caso, existe una variable "mayoritaria" que actúa como un *flag* en la ejecución, este indica si la secuencia tiene un elemento que la haga mayoritaria o no. Al inicio, se encuentra en estado $false$ porque no ha encontrado un elemento que cumpla la condición. En las iteraciones, se van encontrando diferentes repeticiones para cada uno de los elementos y en caso de que se cumpla la condición para que sea el elemento mayoritario, el valor de esta variable cambiaría a $true$.

2.2. Basado en la estrategia "dividir_y_vencer"

La idea de este algoritmo es: Dividir la secuencia por mitades e ir almacenando en dos arreglos paralelos las repeticiones de cada uno de los elementos. Al final de los recorridos, se realiza una mezcla de los resultados en los arreglos. Dentro de estos es posible buscar si existe un elemento que permita hacer la secuencia mayoritaria, en caso que sí, se retornará una cadena de caracteres determinando el caso afirmativo y señalando el número mayoritario, en caso negativo, se retornará una cadena que indique que la secuencia no es mayoritaria.

2.2.1. Pseudocódigo

A continuación, es posible encontrar el pseudocódigo de dos métodos que permiten encontrar si una secuencia es mayoritaría o no. lo anterior lo hace por la estrategia de dividir y vencer, estos son los algoritmos 2 y 3 del presente documento.
Adicionalmente, es importante aclarar que el método que hace uso de esta estrategia es $DivideConquer$, el cual es llamado dentro del método $MajoritySequence$.

2.2.2. Análisis de complejidad

Analizando el pseudocódigo anterior, se tienen tres posibles casos:

- Peor caso: $O(n^2logn)$, en cuanto al $logn$ es por las divisiones que se realizan en cada una de las entradas al método, el n^2 hace referencia a los ciclos anidados que se presentan.
- Caso promedio: $\theta(nlogn)$, el $logn$ se debe a las divisiones que se realizan y el n ocurriría cuando no se deba recorrer completamente el segundo arreglo, por lo que el segundo ciclo no debe recorrerse completamente.
- Mejor caso: $\Omega(nlogn)$, el $logn$ se debe a las divisiones que se realizan y el n, en el mejor de los casos, sería para que el elemento a buscar se encuentre en la primera posición, por lo que no se debe recorrer completamente este arreglo.

Algoritmo 2 Secuencia mayoritaria por estrategia "dividir y vencer".

```
1: procedure DivideConquer(S, elements, values, b, e)
2:     if e − b ≤ 3 then
3:         for i ← b to e do
4:             inArray ← false
5:             j ← 1
6:             while j ≤ |elements| ∧ inArray = false do
7:                 if S[i] = elements[j] then
8:                     values[j] ← values[j] + 1
9:                     inArray ← true
10:                end if
11:                j ← j + 1
12:            end while
13:            if inArray = false then
14:                elements.append(S[i])
15:                values.append(1)
16:            end if
17:        end for
18:        return [elements, values]
19:    else
20:        q ← ⌊(b + e) ÷ 2⌋
21:        elementsL, valuesL = DivideConquer(S, elements, values, b, q)
22:        elementsR, valuesR = DivideConquer(S, elements, values, q + 1, e)
23:        for i ← 1 to |elementsL| do
24:            j ← 1
25:            found ← false
26:            while j ≤ |elementsR| ∧ found = false do
27:                if elementsL[i] = elementsR[j] then
28:                    valuesL[i] ← valuesL[i] + valuesR[j]
29:                    found ← true
30:                end if
31:                if found = false then
32:                    elementsL.append(elementsR[j])
33:                    valuesL.append(valuesR[j])
34:                end if
35:                j ← j + 1
36:            end while
37:        end for
38:        return [elementsL, valuesL]
39:    end if
40:    minrun ← CalculateMinRun(|S|)
41:    size ← minrun
42: end procedure
```

2.2.3. Invariante

En este caso, se tiene un arreglo "values" que contiene las repeticiones de cada uno de los elementos pertenecientes a la secuencia. Este arreglo al inicio se encuentra vacío, lo que indica que hasta el momento no se ha encontrado ningún elemento mayoritario. En las iteraciones se almacenan las repeticiones de cada uno de los elementos, si existe un número mayoritario en la secuencia será el de mayor valor en este arreglo "values". Si no existe un número mayoritario, es que todos los elementos tienen proporciones parecidas en cuanto a sus repeticiones en la secuencia y ninguno cumple la condición para ser mayor o igual a la mitad del total de elementos incial de la secuencia dada.

Algoritmo 3 Algoritmo dividir y vencer: Secuencia mayoritaria.

```
1: procedure MajoritySequence(S)
2:     elements ← declare an empty array
3:     values ← declare an empty array
4:     elements, values = DivideConquer(S, elements, values, 1, |S|)
5:     mayoritaria ← false
6:     i ← 1
7:     while j ≤ |values| ∧ mayoritaria = false do
8:         if values[i] ≤ ⌊|S| ÷ 2⌋ then
9:             element ← elements[i]
10:            mayoritaria = true
11:        end if
12:        i ← i + 1
13:    end while
14:    if mayoritaria = true then
15:        return "La secuencia es mayoritaria, el número es "+element.
16:    else
17:        return "La secuencia no es mayoritaria".
18:    end if
19: end procedure
```