Escritura del problema de secuencia mayoritaria

Jorge Luis Esposito Albornoz[1]

[1]Departamento de Ingeniería de Sistemas, Pontificia Universidad Javeriana
Bogotá, Colombia
{jesposito}@javeriana.edu.co

11 de agosto de 2022

Resumen

En este documento se presenta la formalización del problema de indicar si una secuencia es mayoritaria, junto con la descripción de 2 algoritmos que la solucionan. **Palabras clave:** secuencia, algoritmo, mayoritaria.

Índice

1. Formalización del problema

El problema de la secuencia mayoritaria es uno que a simple vista puede resultar muy sencillo para los humanos, ya que al visualizar una secuencia nuestro cerebro ubicará rápidamente patrones en aquellos números que se repitan. No obstante, esto cambia cuando estamos ante secuencias de números muy grandes y con valores muy diversos, es acá donde se procede a evaluar paso a paso cada dígito de la secuencia para poder determinar si cumple la condición para ser mayoritario ó no.

1.1. Definición del problema del "Secuencia mayoritaria"

Así, el problema de la secuencia mayoritaria se define a partir de:

1. una secuencia S de elementos $a \in \mathbb{T}$

Imprimir una indicación acerca de sí la secuencia es mayoritaria e informar acerca del elemento mayoritario.

- Entradas:
  - $S = \langle a_i \in \mathbb{T} \rangle \mid 1 \leq i \leq n$.
- Salidas:
  - Si $maxCount > n/2$ imprimir s_i, en caso contrario indicar que no hay elemento mayoritario.

2. Algoritmos de solución

2.1. Método iterativo

La idea de este algoritmo es por medio de fuerza bruta, usar dos ciclos uno que recorra el arreglo y otro que compare cada elemento a ver si se repite. Luego se irá almacenando el elemento con el mayor número de coincidencias y finalmente se comprobará si es mayor a la mitad del tamaño de la secuencia.

Algoritmo 1 Secuencia Mayoritaria Iterativa.

Require: $S = \langle s_i \in \mathbb{T} \rangle$
Ensure: Notificar si se encuentra el elemento mayoritario

```
1:  procedure findMajority(S, |S|)
2:      maxCount ← 0
3:      index ← −1
4:      for i ← 1 to |S| do
5:          count ← 0
6:          for j ← 1 to |S| do
7:              if s_i == s_j then
8:                  count + = 1
9:              end if
10:         end for
11:         if count > maxCount then
12:             maxCount ← count
13:             index ← i
14:         end if
15:         if maxCount > n/2 then
16:             print(s_index)
17:         end if
18:         if maxCount <= n/2 then
19:             print (No majority element)
20:         end if
21:     end for
22: end procedure
```

2.1.1. Análisis de complejidad

Este algoritmo tiene una complejidad O(n^2) debido a los dos ciclos aninados que deben recorrer todo el arreglo.

2.1.2. Invariante

Después de cada iteración controlada por el contador i, se guarda el elemento de mayor coincidencia.

1. Inicio: i = 1 todos los elementos son iguales.
2. Iteración: Por cada elemento del arreglo se corre otro ciclo para encontrar sus similares.
3. Terminación: Si la cuenta máxima es mayor que la mitad del tamaño de la secuencia, se imprime el elemento. Si no, imprime que no hay ningún elemento mayoritario.

2.2. Método Divide y vencerás

La idea de este algoritmo es dividir la secuencia en dos mitades y encontrar el elemento mayoritario de ambas mitades de manera recursiva, luego se determina el elemento mayoritario global.

Algoritmo 2 Secuencia Mayoritaria Divide y Vencerás 1/3.

```
1: procedure getMajorityElement(S, |S|)
2:     return getMajority(S, 0, n − 1)
3: end procedure
```

Algoritmo 3 Secuencia Mayoritaria Divide y Vencerás 2/3.

```
1: procedure getMajority(S, l, r)
2:     if l == r then
3:         return S_l     end if
4:             mid ← (r − l)/2 + 1
6:             leftMajority ← getMajority(S, l, mid)
7:             rightMajority ← getMajority(S, mid + 1, r)
8:             if leftMajority == rightMajority then
9:                 return leftMajority
10:            end if
11:            leftCount ← countFrequency(S, l, r, leftMajority)
12:            rightCount ← countFrequency(S, l, r, rightMajority)
13:            if leftCount > rightCount then
14:                return leftMajority
15:            end if
16:            if leftCount < rightCount then
17:                return rightMajority
18:            end if
19:        end procedure
```

Algoritmo 4 Secuencia Mayoritaria Divide y Vencerás 3/3.

```
1: procedure countFrequency(S, l, r, majority)
2:     count ← 0
3:     for i ← l to r do
4:         if S_i == majority then
5:             count ← count + 1
6:         end if
7:     end for
8:     return count
9: end procedure
```

2.2.1. Análisis de complejidad

Ecuación de recurrencia: $T(n) = 2T(n/2) + O(n)$ esta ecuación es similar a la del algoritmo merge sort, por lo cual sabemos que tiene una complejidad O(logn).

2.2.2. Invariante

Después de cada recursión, se obtienen los elementos mayoritarios de los sub arreglos.

1. Inicio:Si los sub-arreglos son iguales se retorna un solo elemento. (left == right)
2. Iteración: Se obtiene el mayoritario de cada mitad, y se comparan entre ellos para determinar el elemento mayoritario global.
3. Terminación: Si la cuenta máxima es mayor que la mitad del tamaño de la secuencia, se imprime el elemento. Si no, imprime que no hay ningún elemento mayoritario.