UNIVERSIDAD DE MÁLAGA

Análisis y Diseño de Algoritmos

Práctica 2

v

1.- Dado un vector V de n elementos (no necesariamente ordenables), se dice que un elemento x es mayoritario en V cuando el número de veces que x aparece en V es mayor que n/2.

a) Implementar un algoritmo de fuerza bruta que devuelva el elemento mayoritario.

b) Implementar un algoritmo utilizando la técnica Divide y Vencerás. La idea consiste en encontrar primero un posible candidato, es decir, el único elemento que podría ser mayoritario, y luego comprobar si realmente lo es. De no serlo, el vector no admitiría elemento mayoritario. Para encontrar el candidato, suponiendo que el número de elementos del vector es par, vamos a ir comparándolos por pares (a[1] con a[2], a[3] con a[4], etc.). Si para algún k = 1,3,5,7, ... se tiene que a[k] = a[k+1] entonces copiaremos a[k] en un segundo vector auxiliar b. Una vez recorrido todo el vector a, buscaremos recursivamente un candidato para el vector b, y así sucesivamente. Este método obedece a la técnica de Divide y Vencerás pues en cada paso el número de elementos del vector se reduce como poco a la mitad.
Comprobar que el orden de crecimiento del algoritmo es *O(n)*.

c) Implementar el algoritmo de Voto Mayoritario de Boyer–Moore, que también encuentra el elemento mayoritario en *O(n)*. En esencia usa una variable `m` para el elemento mayoritario y un contador `i`, que empieza en `0`. Según se va procesando cada elemento `x` se hace lo siguiente:

```
SI i==0, entonces m=x; i=1
SINO
    SI m== x, entonces incrementar i
    SINO decrementar i
    FIN_SI
FIN_SI
```

Al final en `m` está el mayoritario si lo hay (hay que comprobar que el candidato propuesto es mayoritario)

2.- Dado un array ordenado que contiene m elementos distintos (de 0 a m-1) que pueden estar repetidos, queremos calcular la frecuencia de cada uno de esos elementos, es decir, el número de veces que se repite cada uno de dichos elementos.

Un posible algoritmo que utiliza el enfoque por fuerza bruta sería el siguiente:

Precondición: $\{frecuencias.length = m \wedge \forall i \in \mathbb{Z}, 0 \leq i < m : frecuencias[i] = 0\}$

```
public FrecElemsFB(int [] a, int [] frecuencias) {
      for (int i = 0; i < a.length; i++) {
            int elem = a[i];
            frecuencias[elem]++;
      }
}
```

a) Determinar la función de complejidad de dicho algoritmo en base al número de accesos a componentes de un array. ¿Cuál es su orden de crecimiento?

b) Se nos ha ocurrido una variante que recorrerá el array y calculará la frecuencia de los elementos de la siguiente manera:

1. Comenzamos en la posición *i = 0*, que contiene el valor *a[i]*;

2. Buscamos la posición *p* más a la derecha en la que aparece el valor *a[i]*. Para ello **se ha de diseñar** un algoritmo Divide y Vencerás que, dado un array ordenado y un valor, encuentre la posición más a la derecha en la que está dicho valor.
3. La frecuencia del elemento *a[i]*, será *(p-i+1)*
4. Se actualiza *i* con el valor *p+1* y se vuelve al paso 2.

Estimamos que el orden de crecimiento de esta variante será $O(m \cdot T(n))$, donde m es el número de elementos distintos del array y T(n) es la complejidad del algoritmo que hay que diseñar en el paso 2.

Implementar esta forma de resolver el problema y discutir si merece la pena esta variante con respecto al algoritmo basado en fuerza bruta. ¿En qué condiciones de los datos de entrada sería interesante o no lo sería?

c) Intentar diseñar un método alternativo que se ajuste a la siguiente signatura

```
public void frecElemsDyV(int [] a, int izq, int der, int [] frecuencias)
```

y que, en vez de aplicar DyV para diseñar una parte del algoritmo solución, proporcione una solución que se basa completamente en la estrategia DyV.

El método buscado debe tener una complejidad de orden *O(n),* es decir, podemos diseñar un algoritmo DyV de complejidad lineal.