Práctica 2

Divide y vencerás

ALGORÍTMICA Y COMPLEJIDAD
Grados en Ing. Informática y Matemáticas
Universidad de Cantabria

Camilo Palazuelos Calderón

Objetivos de la práctica

1. Resolver un problema mediante «divide y vencerás»
2. Introducirse en el análisis de algoritmos recursivos
3. Comparar el coste temporal empírico de dos algoritmos

Duración de la práctica y fecha de entrega

- Las dos sesiones destinadas a la práctica son las de los días 27/28 de febrero y 5/6 de marzo
- La práctica debe entregarse, a través de la plataforma Moodle, antes del 10 de marzo a las 23:59

Qué debéis entregar

- Memoria con respuestas a las preguntas formuladas en este documento
- Código desarrollado y material suplementario que se considere oportuno

---

Elemento mayoritario de un array

Decimos que m es el elemento mayoritario de un array $A[1..n]$ cuando el número de veces que m aparece en $A[1..n]$ es estrictamente mayor que $n/2$. El algoritmo MAYORITARIO1 devuelve el elemento mayoritario del array $A[1..n]$ recibido como parámetro en caso de que exista; si no, devuelve NULL:

```
MAYORITARIO1(A[1..n]):
    if n = 1
        return A[n]
    else
        k ← ⌊n/2⌋
        m1 ← MAYORITARIO1(A[1..k])
        m2 ← MAYORITARIO1(A[k + 1..n])
```

```
    m ← NULL
    if m1 ≠ NULL
        m ← COMPROBAR(A[1..n], m1)
    if m2 ≠ NULL and m = NULL
        m ← COMPROBAR(A[1..n], m2)
    return m
```

Preguntas en grupos de 2 o 3 alumnos

Pregunta 1 [1 PUNTO]. Diseñad el algoritmo COMPROBAR en pseudocódigo para que el algoritmo MAYORITARIO1 se comporte como indica el enunciado.

Pregunta 2 [1 PUNTO]. Calculad el coste temporal del algoritmo MAYORITARIO1 de acuerdo con vuestro diseño del algoritmo COMPROBAR. Calculad también el de la función MAYORITARIO2 del fichero `practica_2.py` adjunto. ¿Cuál es más eficiente temporalmente en términos asintóticos?

El algoritmo MAYORITARIO3 también devuelve el elemento mayoritario del array $A[1..n]$ en caso de que exista; si no, devuelve NULL:

```
MAYORITARIO3(A[1..n]):
    B[1..n] ← A[1..n]
    m ← CANDIDATO(B[1..n])
    if m ≠ NULL
        m ← COMPROBAR(A[1..n], m)
    return m
```

En este caso, es el algoritmo CANDIDATO, siguiendo la estrategia «divide y vencerás», el que busca el candidato a elemento mayoritario, para lo cual compara pares de elementos: por ejemplo, $B[1]$ con $B[2]$, $B[3]$ con $B[4]$, etc. Si, para algún k perteneciente a {1, 3, 5, …}, se cumple que $B[k] = B[k + 1]$, se copia $B[k]$ en un array $C[1..\ell]$ que servirá como parámetro de la llamada recursiva. ¿Qué relación hay entre ℓ y n?

Pregunta 3 [2 PUNTOS]. Diseñad el algoritmo CANDIDATO en pseudocódigo para que el algoritmo MAYORITARIO3 se comporte como indica el enunciado. ¿Sois capaces de hacerlo sin crear el array $C[1..\ell]$? Calculad el coste temporal del algoritmo MAYORITARIO3 en el mejor caso y en el peor caso de acuerdo con vuestro diseño del algoritmo CANDIDATO.

Preguntas en grupos de 4 o 5 alumnos

Pregunta 4 [2 PUNTOS]. Implementad vuestro diseño del algoritmo COMPROBAR en Python. Mostrad que la implementación es correcta proporcionando 4 o más ejemplos de entrada junto con sus salidas correspondientes.

Pregunta 5 [2 PUNTOS]. Dibujad una gráfica con los tiempos de ejecución de las dos primeras implementaciones (MAYORITARIO1 y MAYORITARIO2) en el peor caso en función de la longitud *n* del array de entrada (para, por ejemplo, *n* entre 10^4 y 10^6 de 10 000 en 10 000). ¿Son coherentes las curvas obtenidas con los costes temporales calculados en la pregunta 2?

Pregunta 6 [2 PUNTOS]. Implementad vuestro diseño del algoritmo CANDIDATO en Python. Añadid a la gráfica de la pregunta 5 el tiempo de ejecución de MAYORITARIO3 en el peor caso en función de la longitud *n* del array de entrada (para, por ejemplo, *n* entre 10^4 y 10^6 de 10 000 en 10 000). ¿Qué conclusiones obtenéis?