Problemas 2

Enunciados

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

Camilo Palazuelos Calderón

Objetivos

1. Resolver problemas de las clases de complejidad P y NP-completo
2. Elegir y aplicar la mejor estrategia algorítmica para cada problema

Duración y fecha de entrega

- Las dos sesiones destinadas a los problemas son las de los días 30 de abril y 7/8 de mayo
- Los problemas deben entregarse, a través de la plataforma Moodle, antes del 12 de mayo a las 23:59

Qué debéis entregar

- Memoria con respuestas a las preguntas formuladas en este documento

---

Problema 1 [3 PUNTOS]. La *scriptio continua* fue predominante en los textos occidentales hasta finales del siglo X o principios del siglo XI. Se trataba de un estilo de escritura caracterizado por la ausencia de espacios para separar las palabras. Diseñad un algoritmo de vuelta atrás que, dados un entero positivo m y un array de letras $A[1..n]$, indique, identificando los subarrays correspondientes, si $A[1..n]$ es separable en m palabras; si no lo es, el algoritmo devolverá NULL. Suponed que disponéis del algoritmo ESPALABRA($A[i..j]$), que determina si $A[i..j]$ es una palabra. Por ejemplo, el array de letras[1]

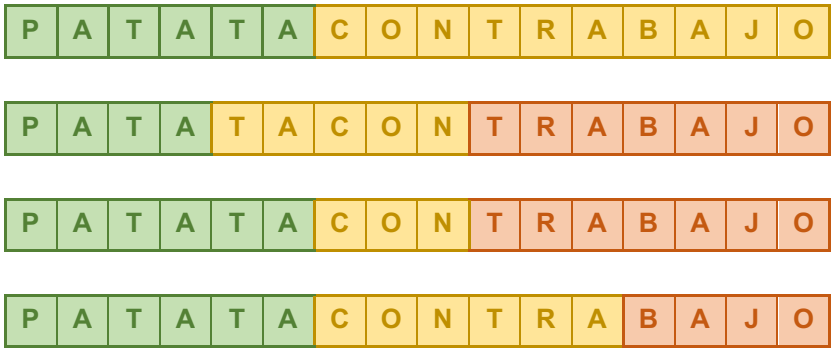
$A[1..16]$ =

P	A	T	A	T	A	C	O	N	T	R	A	B	A	J	O

,

si ABAJO, AJO, BAJO, CON, CONTRA, CONTRABAJO, PATA, PATATA, TACO, TACÓN, TRABA y TRABAJO son palabras, es separable en $m = 2$ o $m = 3$ palabras solo así:

[1] El ejemplo está basado en el propuesto por mis alumnos del curso 2022-2023 Ramón Belmonte, Sergio Lanza, Miguel Martín, Gonzalo Peña y David Ruiz: PATATACONTOMATE (😊).



Problema 2 [4 PUNTOS]. El problema 1 pertenece, en realidad, a la clase de complejidad P, y el algoritmo SEPARABLE, cuyo diseño se muestra a continuación, lo resuelve (como problema de decisión) en tiempo polinómico.

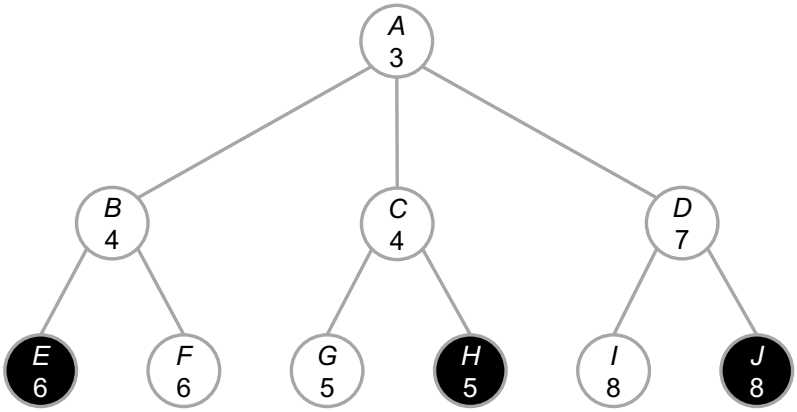
```
SEPARABLE(A[1..n], m):
    B[0..m, 1..n + 1] ← ((NULL,…,NULL),…, (NULL,…,NULL))
    return SEPARABLE(A[1..n], B[0..m, 1..n + 1], m, 1)

SEPARABLE(A[1..n], B[0..m, 1..n + 1], i, j):
    if i = 0 and j = n + 1
        return TRUE
    else if i = 0 and j < n + 1
        return FALSE
    else if i > 0 and j = n + 1
        return FALSE
    else
        if B[i, j] = NULL
            B[i, j] ← FALSE
            for k ← j to n
                if ESPALABRA(A[j..k])
                    if SEPARABLE(A[1..n], B[0..m, 1..n + 1], i – 1, k + 1)
                        B[i, j] ← TRUE
        return B[i, j]
```

a) [2 PUNTOS] Diseñad un algoritmo SEPARABLE iterativo cuyo coste temporal sea asintóticamente equivalente al de su homónimo recursivo.

b) [2 PUNTOS] Calculad el coste temporal del algoritmo SEPARABLE en el caso peor, indicando claramente cómo lo habéis calculado y por qué.

Problema 3 [3 PUNTOS]. Se tiene un algoritmo de ramificación y poda que resuelve un problema de minimización recorriendo el siguiente árbol de búsqueda:



Los nodos blancos y negros son soluciones parciales y soluciones al problema, respectivamente. Para los primeros, el número que acompaña a la letra que los identifica es una estimación no pesimista (cota) del coste de la solución; para los segundos, se trata del coste real de la solución. El algoritmo utiliza la función de cota para ordenar los nodos en la cola de prioridad.

a) [2 PUNTOS] Añadid filas a la siguiente tabla y rellenadlas de acuerdo con el algoritmo de ramificación y poda (hasta que vaciéis la cola de prioridad):

Nodo actual	Cola de prioridad	Mejor solución
A	((*B*, 4), (*C*, 4), (*D*, 7))	NULL

b) [1 PUNTO] ¿Qué nodos del árbol de búsqueda habéis podado y por qué?