SEPTIEMBRE 2015 – ORIGINAL (MODELO A)

1. ¿Cuál de las siguientes afirmaciones es cierta con respecto al coste de algunos algoritmos y su eficiencia?

(a) El algoritmo de Kruskal tiene un coste que está en O(n log n).

Falso. El coste del algoritmo Kruskal es O(a log n).

(b) En el algoritmo de Prim cuando se combina el uso de la lista de adyacencia con el uso de un montículo es más eficiente cuando el grafo es denso porque de esta manera el coste está en $O(n^2 \log n)$.

Falso. El coste del algoritmo Dijkstra es:

- O(n) → Tareas de inicialización
- $O(n^2)$ → Tiempo de ejecución
- O(log n) → Instrucción que elimina la raíz del montículo
- O(log n) → Coste cuando se actualiza el montículo con w ubicándolo según su valor de especial[w] cuando se encuentra un nodo w tal que, a través de v, se encuentra un camino menos costoso.
- O((n+a) log n) → La raíz del montículo se elimina n-2 veces, y se realizan operaciones de flotar un máximo de a veces.
- O(a log n) → Implementación con montículo en los casos:
  - Grafo Conexo: $n-1 <= a <= n^2$
  - Grafo Disperso: El número de aristas es pequeño y cercano a n
- $O(n^2 \log n)$ → Se da cuando el grafo es denso, siendo preferible la implementación con montículo.

(c) El coste del algoritmo Quicksort en el caso peor es de orden O(n log n).

Falso. El coste de Quicksort es:

- Normal: $O(n^2)$
- Mejorado: O(n log n)

(d) La función que crea un montículo a partir de una colección de valores, si se utiliza el procedimiento Hundir puede llegar a tener un coste lineal.

Respuesta D)

2. Sea un grafo no dirigido representado con la siguiente matriz de adyacencia:

	1	2	3	4	5	6	7
1	-	10			6		
2		-	5	2			
3			-	4			
4				-		3	
5					-	9	1
6						-	8
7							-

Si se utiliza el algoritmo de Kruskal para calcular un árbol de recubrimiento mínimo, indica cuál de las siguientes secuencias de aristas (que incluye las rechazadas) representa el orden en el que las evalúa el algoritmo:

(a) {5,7}, {2,4}, {4,6}, {3,4}, {2,3}, {1,5}, {6,7}.

(b) {5,7}, {2,4}, {4,6}, {3,4}, {1,5}, {6,7}, {5,6}.

(c) {5,7}, {1,5}, {6,7}, {4,6}, {2,4}, {4,3}.

(d) {1,5}, {5,7}, {6,7}, {4,6}, {2,4}, {4,3}.

Se va tomando siempre la arista de menor peso, por lo que la primera de ella es la {5, 7}

La tabla de Kruskal resultante es la siguiente:

Aristas	Componentes Conexas
5, 7	{5, 7}, 1, 2, 3, 4, 6
2, 4	{5, 7}, {2, 4}, 1, 3, 6
4, 6	{5, 7}, {2, 4}, {4, 6}, 1, 3
3, 4	{5, 7}, {2, 4}, {4, 6}, {3, 4}, 1
2, 3	Se rechaza, ya que no es factible al provocar un bucle
5, 1	5, 7, 1 // 2, 4, 6, 3
6, 7	5, 7, 1, 2, 4, 6, 3

Respuesta A)

3. Con respecto a la resolución de colisiones en las funciones Hash, indicar cuál de las siguientes afirmaciones es cierta:

(a) El recorrido lineal permite mayor dispersión de las colisiones que el cuadrático.

Falso. El recurrido cuadrático ofrece mayor dispersión que el recorrido lineal.

(b) Si el factor de carga es 1 se puede resolver mediante hashing cerrado.

Falso. El factor de carga se usa en el Hashing Abierto.

(c) El hashing abierto contempla un método de resolución conocido como "recorrido mediante doble hashing".

Falso. El recorrido mediante el doble hashing lo contempla el hashing cerrado.

(d) Ninguna de las anteriores es correcta.

Respuesta D)

4. Durante la ejecución de un algoritmo de Ramificación y Poda se halla una solución que es mejor que la mejor solución existente en ese momento y que mejora la estimación optimista de la cima del montículo. Esto implica que:

(a) Definitivamente el algoritmo ha encontrado la solución.

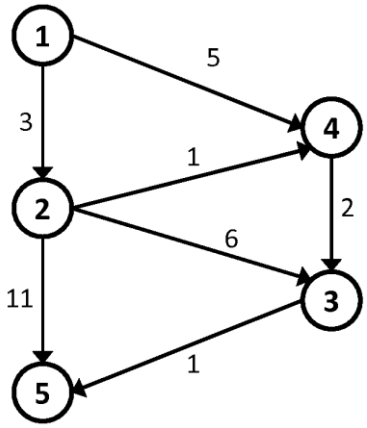
(b) Se actualiza la cota y se sigue con la exploración porque no hemos terminado.

(c) Se actualiza la cima del montículo y se sigue con la exploración porque no hemos terminado.

(d) Ninguna de las anteriores es correcta.

Respuesta A)

5. Dado el siguiente grafo, indique cuál sería el orden en que se seleccionarían los nodos (pasan a estar explorados) al aplicar el algoritmo de Dijkstra para encontrar todos los caminos de menor coste desde el nodo 1:



(a) {1, 2, 4, 3, 5}

(b) {1, 4, 3, 5, 2}

(c) {1, 2, 3, 4, 5}

(d) Ninguna de las anteriores

Nodos que han salido	Nodos que no han salido	Vector Distancia especial [ ]				Predecesores			
		2	3	4	5	2	3	4	5
1	2, 3, 4, 5	**3**	∞	5	∞	1	1	1	1
1, 2	3, 4, 5	3	9	**4**	14	1	2	2	2
1, 2, 4	3, 5	3	**9**	4	10	1	2	2	2
1, 2, 4, 3	5	3	9	4	10	1	2	2	3

Respuesta A)

6. Considérese el vector [10,6,3,5,2,3,2] que representa un montículo. ¿Cuál sería la representación resultante de insertar (función Insertar del texto base) en este montículo el valor 6 usando la función flotar?

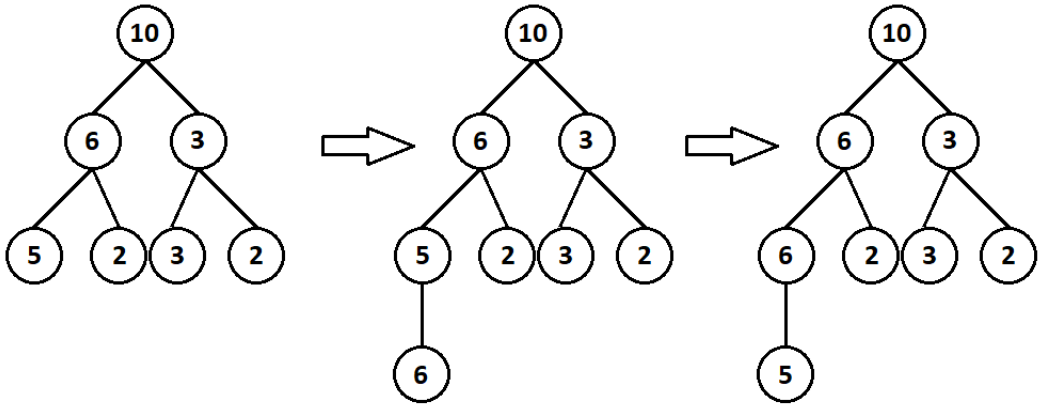
(a) [10,6,6,3,3,2,5,2]

(b) [10,6,5,3,6,2,2,3]

(c) [10,6,3,6,2,3,2,5]

(d) Ninguna de las opciones anteriores.

- Insertar: Añadir el nodo al final y aplicar la operación flotar.
- Flotar: Intercambiar el nodo a un nivel superior.



El vector resultante es: {10, 6, 3, 6, 2, 3, 2, 5}

Respuesta C)

PROBLEMA (4 Puntos)

Una pareja decide divorciarse tras 10 años de matrimonio. Deciden repartir su patrimonio a partes iguales. Cada uno de los n activos (indivisibles) que hay que repartir tiene un valor entero positivo. Los cónyuges quieren repartir dichos activos a medias y, para ello, primero quieren comprobar si el conjunto de activos se puede dividir en dos subconjuntos disjuntos, de forma que cada uno de ellos tenga el mismo valor.

La resolución del problema debe incluir, por este orden:

1. Elección del esquema más apropiado, el esquema general y explicación de su aplicación al problema (0,5 puntos).

No se puede encontrar una función de selección que garantice, sin tener que reconsiderar decisiones, una elección de los activos que cumpla con la restricción del enunciado, por ello no se puede aplicar el esquema voraz.

Tampoco se puede dividir el problema en subproblemas que al combinarlos nos lleven a una solución. Al no ser un problema de optimización, el esquema de exploración de grafos más adecuado es el esquema vuelta atrás.

Vuelta atrás es un recorrido en profundidad de un grafo dirigido implícito. En él una solución puede expresarse como una n-tupla [x1, x2, x3, … xn], donde cada xi, representa una decisión tomada en la etapa i-ésima, de entre un conjunto finito de alternativas.

El esquema general de la técnica de vuelta-atrás es el siguiente:

```
fun VueltaAtras (v: Secuencia, k: entero)
    {v es una secuencia k-prometedora}
    IniciarExploraciónNivel(k)
    mientras OpcionesPendientes(k) hacer
        extender v con siguiente opción
        si SoluciónCompleta(v) entonces
            ProcesarSolución(v)
        sino
            si Completable (v) entonces
                VueltaAtras(v, k+1)
            fsi
        fsi
    fmientras
ffun
```

En este caso, el espacio de búsqueda es un árbol de grado 2 y altura n+1. Cada nodo del i-ésimo nivel tiene dos hijos correspondientes a si el i-ésimo activo va a un cónyuge o al otro.

Para poder dividir el conjunto de activos en dos subconjuntos disjuntos, de forma que cada uno de ellos tenga el mismo valor, su valor debe ser par. Así, si el conjunto inicial de activos no tiene un valor par, el problema no tiene solución.

2. Descripción de las estructuras de datos necesarias (0,5 puntos solo si el punto 1 es correcto).

- El conjunto de activos y sus valores se representa en un array de enteros v = [v1, v2, v3, …, vn], donde cada vi representa el valor del activo i-ésimo.
- La solución se representa mediante un array de valores x = [x1, x2, x3, …, xn], donde cada xi podrá tomar el valor 1 o 2 en función de que el activo i-ésimo se asigne al subconjunto de un cónyuge o del otro.
- Un array de dos elementos, suma, que acumule la suma de los activos de cada subconjunto.

3. Algoritmo completo a partir del refinamiento del esquema general (2,5 puntos solo si el punto 1 es correcto).

Una posible condición de poda consistirá en que se dejarán de explorar aquellos nodos que verifiquen que alguno de los dos subconjuntos que se van construyendo tiene un valor mayor que la mitad del valor total.

```
función Solucion (k: entero; suma: array[1..2]) dev booleano
        si (k = n) AND (suma[1] = suma [2]) entonces
                dev Verdadero
        sino
                dev Falso
        fsi
ffuncion


función SeparacionConyugesMitad (v: array[1..n], k: entero, suma: array[1..2], sumaTotal: entero)
        dev x: array[1..n]; separacion: boolean

si Solucion (k, suma) entonces
        dev x, Verdadero
sino
        para i desde 1 hasta 2 hacer
                x[k] ← i
                suma[i] ← suma[i] + v[k]
                si suma[i] <= (sumaTotal DIV 2) entonces
                        si k < n entonces
                                x ← SeparacionConyugesMitad(v, k+1, suma, sumaTotal)
                        fsi
                sino
                        suma[i] ← suma[i] - v[k]
                fsi
        fpara
fsi
ffunción


función ProblemaSeparacionConyugesMitad (v: n-tupla) dev x: n-tupla; separacion: boolean

        para i desde 1 hasta n hacer
                sumaTotal ← sumaTotal + v[i]
        fpara

        si par(sumTotal) entonces
                dev SeparacionConyugesMitad (v, 1, suma[0,0], sumaTotal)
        sino
                dev 0
        fsi
ffuncion
```

Llamada inicial ProblemaSeparacionConyugesMitad(v);

4. Estudio del coste del algoritmo desarrollado (0,5 puntos solo si el punto 1 es correcto).

En este caso, el coste viene dado por el número máximo de nodos del espacio de búsqueda, esto es: $T(n) \in O(2^n)$.