FEBRERO 2014 – SEMANA 2 (MODELO A)

1. Indique cuál de las siguientes afirmaciones es cierta:

(a) El esquema de divide y vencerás obtiene siempre soluciones eficientes de coste O(log n) o O(n · log n)

(b) La búsqueda en profundidad recursiva obtiene siempre una eficiencia logarítmica

(c) La ordenación por Quicksort tiene coste $O(n^2)$ en el caso peor

(d) Ninguna de las anteriores es cierta.

a) Falso. No siempre, ya que el coste puede ser a veces $O(2^n)$, O(n), $O(n^2)$, etc.

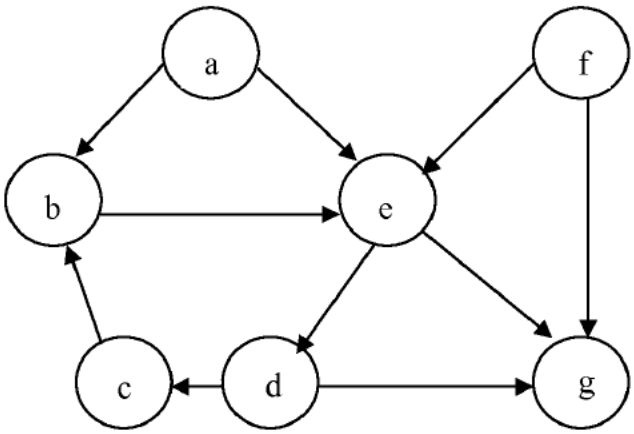
b) Falso. La búsqueda en profundidad recursiva obtiene siempre una eficiencia logarítmica de forma ordenada, pero no uniforme. El coste temporal es $O(b^m)$ y un coste espacial $O(b^d)$.

c) Cierto. El coste en el caso peor para el algoritmo Quicksort puede ser $O(n^2)$ o incluso O(n log n).

d) Falso. La respuesta anterior es cierta.

Respuesta C)

2. ¿Cuántos componentes fuertemente conexos existen en el grafo de la siguiente figura?



(a) Uno solo, formado por todo el grafo.

(b) Dos.

(c) Tres.

(d) Más de tres.

Un grafo es fuertemente conexo si para cada par de vértices distintos 'n' y 'm' existe un camino de 'n' a 'm', y también, hay un camino de 'm' a 'n'.

Existen más de 3 componentes fuertemente conexos, en concreto son:

- (b-e-d-c)
- (a)
- (f)
- (g)

Respuesta D)

3. El enemigo ha desembarcado en nuestras costas invadiendo n ciudades. Los servicios de inteligencia han detectado del número de efectivos enemigos en cada ciudad, ej. Para contraatacar, se dispone de n equipos listos para intervenir y hay que distribuirlos entre las n ciudades. Cada uno de estos equipos consta de dj efectivos entrenados y equipados. Para garantizar el éxito de la intervención en una ciudad es necesario que contemos al menos con tantos efectivos de defensa como el enemigo.

Se busca un algoritmo que indique qué equipo debe intervenir en cada ciudad, de forma que se maximice el número de éxitos garantizados. Indica cuál de las siguientes afirmaciones es cierta:

(a) El esquema de ramificación y poda es el único que puede resolver de forma óptima el problema.

(b) Se puede encontrar una estrategia voraz que permita resolver el problema.

> Cierto. No obstante, que se pueda encontrar no quiere decir que siempre se encuentre, y, realmente, el más apropiado para ello es el algoritmo Divide y Vencerás.

(c) El esquema de programación dinámica es el único que puede resolver de forma óptima el problema.

(d) El esquema de vuelta atrás es el más apropiado para resolver este problema.

Respuesta B)

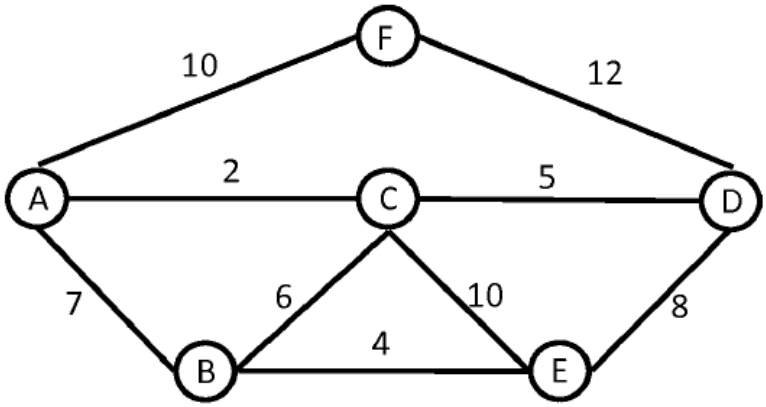
4. Indica de entre los siguientes, cuál sería el coste mínimo de un algoritmo que dado un vector C[1 .. n] de números enteros distintos no ordenado, y un entero S, determine si existen o no dos elementos de C tales que su suma sea exactamente S.

(a) $\Theta(n)$.
(b) $\Theta(n^2)$.
(c) $\Theta(n \log n)$.
(d) $\Theta(n^2 \log n)$.

Lo que podría hacerse es ordenar el vector, y luego despreciar todos los casos de números donde n > S. Tras ello separamos a la mitad el vector que nos queda, y entonces, en otro vector, almacenar los valores de S menos cada elemento de la primera mitad, y comparar con los elementos de la segunda mitad mediante divide y vencerás, como, por ejemplo, para ver si tengo coincidencia (El resultado aproximado es N log N).

Respuesta C)

5. Dado el siguiente grafo no dirigido:



Indique cuál sería el orden en que se seleccionarían las aristas al aplicar el algoritmo de Kruskal:

(a) {{A,C},{C,D},{C,B},{B,E},{D,F}}
(b) {{A,C},{B,E},{C,D},{C,B},{A,F}}
(c) {{A,C},{C,D},{A,B},{B,E},{A,F}}
(d) Ninguna de las anteriores

Se elige siempre la de menor peso mientras que no se formen ciclos repetidos.

Aristas	Componentes Conexas
A, C	{A, C}, B, D, E, F
B, E	{A, C}, {B, E}, D, F
C, D	{A, C}, {B, E}, {C, D}, F
C, B	{A, C}, {B, E}, {C, D}, {C, B}, F
A, F	{A, C}, {B, E}, {C, D}, {C, B}, {A, F}

<u>Respuesta B)</u>

6. Indique cuál de las siguientes afirmaciones es <u>cierta</u>:

(a) El enfoque de vuelta atrás realiza un recorrido en anchura del árbol implícito que representa el espacio de posibles soluciones del problema.

(b) Si se utiliza el enfoque de programación dinámica para calcular el camino de coste mínimo entre cada par de nodos de un grafo, la complejidad temporal es de $O(n^2)$.

(c) Cuando ambos son aplicables, el enfoque de ramificación y poda se comporta de manera más eficiente que el enfoque voraz en la resolución de problemas de optimización.

(d) El hecho de utilizar un algoritmo voraz para obtener la solución de un problema no garantiza que la solución obtenida sea la óptima.

a) Falso. Vuelta Atrás realiza un recorrido en profundidad. Solo encuentra una solución. Si encuentra alguna no óptima, y esta se descarta, no sigue explorándola.

b) Falso. El coste temporal es $O(n^3)$ y el coste espacial de $O(n^2)$.

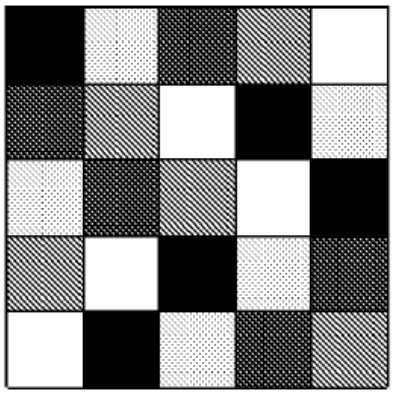
c) Falso. El esquema de Ramificación y Poda, cuando este puede emplearse junto al esquema Voraz para realizar la optimización, es menos eficiente que el segundo, por lo que la aplicación de Ramificación y Poda está indicada sólo cuando no se conoce un algoritmo voraz válido para el problema que se plantee.

d) Cierto. Explicado en la respuesta A)

Respuesta D)

PROBLEMA (4 Puntos)

Se considera un tablero n x n y un conjunto de n colores. Cuando a cada casilla se le asigna un color de los n disponibles de tal manera que no se repiten colores en ninguna fila ni en ninguna columna, el tablero se conoce como cuadrado latino. La siguiente figura muestra un ejemplo para n = 5:



Se pide diseñar un algoritmo que dados n colores presente todos los cuadrados latinos para un tablero n x n. Se pide:

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

El esquema correcto es el de Vuelta Atrás. No hay heurísticas ni criterios de búsqueda y el problema no puede descomponerse en subproblemas de su misma naturaleza, por lo que se utiliza el backtraking para la exploración del árbol de búsqueda.

```
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
```

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

Las estructuras de datos son el tablero de NxN que contiene en cada casilla valores que indiquen que el caballo no ha pasado (cero) o que ha pasado (m) donde m es un natural que indica que en el movimiento m-ésimo, el caballo llega a dicha casilla.

El nodo contaría con dicho tablero, con el número de caballos puestos en el tablero hasta el momento, y con la posición actual del caballo (el último movimiento efectuado).

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

El refinamiento del esquema precisa detallar cómo generar las compleciones y cómo realizar la recursión por el árbol implícito.

Las compleciones son las siguientes:

```
Fun compleciones(ensayo)lista ← lista_ vacía

(i,j) → nodo.ultima_posición
n ← nodo.numero_movimientos
ultimo_tablero ← ensayo tablero

para i ← -2 hasta 2 hacer
     para j ← -2 hasta 2 hacer

          si |i|+|j|==3 AND nodo.tablero[i,j] == 0 entonces hacer

               nuevoTablero ← ultimoTablero
               nuevoTablero[i,j] ← n + 1
               nuevaPosicion ← (i,j)
               nuevoNodo ←  < nuevoTablero, nuevaPosición, n+1>
               lista.add(nuevoNodo)

          fsi
     fpara
fpara
dev lista
ffun
```

El algoritmo principal queda:

```
Fun saltoCaballo (nodo)

si solucion (nodo)       /* el numero de casillas libre es cero */
entonces escribe(nodo)    /* hemos terminado */
sino hacer

     lista ← compleciones (nodo) /* lista de los posibles movimiento
                                    válidos */
     mientras  ¬ lista.vacia() hacer
          w ← lista.primerElemento()
          saltoCaballo(w)
          lista ← lista.resto()
     fmientras
fsi
ffun
```

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

Sin tener en cuenta las reglas de colocación del caballo, las n^2 casillas pueden numerarse de $(n^2)!$ maneras posibles.

Sin embargo, sabemos que el número máximo de ramificaciones del árbol es 8 por ser éstas las alternativas de movimiento de un caballo de ajedrez.

Considerando esto, el tamaño del árbol puede acotarse en $8^{n\hat{}2}$, pero se puede precisar que no hay 8 ramas en todos los casos.

De cada n^2 casillas, solo desde n^2-8(n-2) es posible realizar 8 movimientos, y, además, los últimos movimientos no tendrán prácticamente ninguna alternativa.