FEBRERO 2016 – SEMANA 2 (MODELO A)

1. Dado un grafo con n vértices y a aristas sobre el que se pueden realizar las siguientes operaciones:

- ***esAdyacente: comprueba si dos vértices son adyacentes.***
- ***borrarVertice: borra el vértice especificado y todas sus aristas.***
- ***añadirVertice: añade un nuevo vértice al grafo.***

¿Cuál es la complejidad de cada una de las operaciones anteriores según se utilice una matriz de adyacencia (MA) o una lista de adyacencia (LA)?

(a)

	MA	LA
esAdyacente	O(1)	O(1)
borrarVértice	O(n)	O(n)
añadirVértice	O(n)	O(1)

(b)

	MA	LA
esAdyacente	O(1)	O(n)
borrarVértice	O(1)	O(n+a)
añadirVértice	O(1)	O(1)

(c)

	MA	LA
esAdyacente	O(1)	O(n)
borrarVértice	O(n)	O(n+a)
añadirVértice	O(n)	O(1)

(d)

	MA	LA
esAdyacente	O(1)	O(1)
borrarVértice	O(n)	O(n)
añadirVértice	O(n)	O(n)

2. Indique cuál de las siguientes afirmaciones es <u>falsa</u>:

(a) La programación dinámica es aplicable a muchos problemas de optimización cuando se cumple el Principio de Optimalidad de Bellman.

(b) En un algoritmo de vuelta atrás es posible retroceder para deshacer una decisión ya tomada.

(c) El problema de calcular el camino de coste mínimo entre cada par de nodos de un grafo (es decir, desde todos los nodos hasta todos los restantes nodos) se resuelve utilizando el algoritmo de Dijkstra con una complejidad de $O(N^3)$.

(d) La programación dinámica no es apropiada para resolver problemas que pueden descomponerse en subproblemas más sencillos en los que haya llamadas repetidas en la secuencia de llamadas recursivas.

3. Respecto a las tablas Hash, podemos afirmar que:

(a) En el método de hashing cerrado con recorrido mediante doble hashing, las funciones h y h' que se aplican pueden ser iguales cuando el número de elementos de la tabla, m, cumple determinadas condiciones.

Falso. Ambas son o deben ser distintas.

(b) En la resolución de colisiones, el método de hashing abierto es siempre más eficiente que el hashing cerrado.

Falso. Es más eficiente siempre el Hashing cerrado.

(c) En el recorrido mediante doble hashing, la función h'(x) debe cumplir necesariamente que $h'(x) \neq 0$.

(d) En la resolución de colisiones, si el factor de carga es 1 se puede resolver mediante hashing cerrado.

4. Se quiere realizar una recopilación de poesías preferidas en un rollo de papel que se puede escribir por ambas caras. Se dispone de una lista de n poesías favoritas, junto con la longitud individual de cada una. Lamentablemente, el rollo de longitud L no tiene capacidad para contener todas las poesías, por lo que se les ha asignado una puntuación (cuanto más favorita es, mayor es la puntuación). Se pretende obtener el mejor rollo posible según la puntuación, teniendo en cuenta que las poesías deben caber enteras y no es admisible que una poesía se corte al final de una de las caras. ¿Cuál de los siguientes esquemas es más eficiente de los que puedan resolver el problema correctamente?

(a) Esquema voraz.

(b) Esquema divide y vencerás.

(c) Esquema de vuelta atrás.

(d) Esquema de ramificación y poda.

5. Una empresa de montajes tiene n montadores con distintos rendimientos según el tipo de trabajo. Se trata de asignar los próximos n encargos, uno a cada montador, minimizando el coste total de todos los montajes. Para ello se conoce de antemano la tabla de costes C[1..n, 1..n] en la que el valor c_{ij} corresponde al coste de que el montador i realice el montaje j. ¿Cuál de los siguientes esquemas es más eficiente de los que puedan resolver el problema correctamente?

(a) Esquema voraz.

(b) Esquema divide y vencerás

(c) Esquema de ramificación y poda.

(d) Esquema de vuelta atrás.

6. En relación al esquema algorítmico de vuelta atrás, ¿cuál de las siguientes afirmaciones es <u>cierta</u>?:

(a) Implementa un recorrido en amplitud de forma recursiva sobre el grafo implícito del problema.

Falso. Realiza un recorrido en profundidad del grafo implícito del problema.

(b) Aun existiendo una solución al problema, puede darse el caso de que el esquema de vuelta atrás no la encuentre.

Falso. Siempre encuentra una solución.

(c) Siempre es más eficiente que el esquema voraz, de forma que, si ambos son aplicables, nos decantaremos por utilizar el esquema de vuelta atrás.

Falso. Siempre es más eficaz el algoritmo Voraz.

(d) Todas las afirmaciones anteriores son falsas.

PROBLEMA (4 Puntos)

Se dispone de n cubos identificados con un número de 1 al n. Cada cubo tiene impresa en una de sus caras una letra distinta. Se indica además una palabra de n letras. Se trata de colocar los n cubos uno a continuación Otro, de forma que con esa disposición se pueda formar la palabra dada. Como en diferentes cubos puede haber letras repetidas, la solución puede no ser única 0 no existir.

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).

El esquema más apropiado es el de Vuelta Atrás.

```
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).

Los cubos vienen dados por un vector C[1..n], donde el cubo C[i] se representa como un vector C[i][1..6] que denota las letras en las 6 caras del cubo. La palabra viene dada por un vector P[1..n] de letras.

Podemos representar la solución como una tupla (x_1, …, x_n) de pares (i,j) indicando que se considera la letra de la cara j del cubo C[i]. Se tiene que cumplir que en cada posición se utiliza una cara válida de alguno de los cubos, que los cubos no se repitan y que se haya formado la palabra correcta. Utilizamos un vector usado [1…n] para saber que cubos ya han sido utilizados.

3. Algoritmo <u>completo</u> a partir del refinamiento del esquema general (2,5 puntos solo si el punto 1 es correcto). Si se trata del esquema voraz, debe realizarse la demostración de optimalidad. Si se trata del esquema de programación dinámica, deben proporcionarse las ecuaciones de recurrencia.

Como la solución debe ser una permutación de los n cubos, llevamos como marcador un vector usado [1..n] para saber los cubos que ya han sido colocados. Escribimos una versión que encuentra la primera solución, si es que existe.

```
tipos
    cubo  =  vector [1..6] de car
    par   =  reg
                 cubo : 1..n
                 cara : 1..6
             freg
ftipos

proc cubos-va(e P[1..n] de car, e C[1..n] de cubo, sol[1..n] de par, e k : 1..n,
              usado[1..n] de bool, éxito : bool)
    cubo := 1    { recorre los cubos }
    mientras cubo ≤ n ∧ ¬éxito hacer
        si ¬usado[cubo] entonces
            usado[cubo] := cierto    { marcar }
            cara := 1    { recorre las caras del cubo }
            mientras cara ≤ 6 ∧ ¬éxito hacer
                si P[k] = C[cubo][cara] entonces
                    sol[k].cubo := cubo ; sol[k].cara := cara
                    si k = n entonces éxito := cierto
                    si no cubos-va(P, C, sol, k + 1, usado, éxito)
                    fsi
                fsi
                cara := cara + 1
            fmientras
            usado[cubo] := falso    { desmarcar }
        fsi
        cubo := cubo + 1
    fmientras
fproc
```

La función principal es la siguiente:

```
fun cubos(P[1..n] de car, C[1..n] de cubo) dev ⟨éxito : bool, sol[1..n] de par⟩
var usado[1..n]
    éxito := falso
    usado[1..n] := [falso]
    cubos-va(P, C, sol, 1, usado, éxito)
ffun
```

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

El árbol de exploración tiene n niveles y el grado del árbol es 6n, por lo que el tamaño del espacio de búsqueda tiene como cota superior $O(6n)^n$. Una cota más ajustada sería $O(6^n(n!))$.