SEPTIEMBRE 2017 – ORIGINAL (MODELO A)

1. Sea un grafo denso no dirigido representado con la siguiente matriz de adyacencia, en la que puede haber pesos de valor 0:

	1	2	3	4	5	6	7	8
1	-	2	3	4	5	0	1	2
2		-	0	2	4	0	2	4
3			-	0	3	0	3	0
4				-	2	0	4	2
5					-	0	5	4
6						-	0	0
7							-	2
8								-

Si se utiliza el algoritmo de Prim para calcular un árbol de recubrimiento mínimo tomando como raíz del árbol el nodo 1, indica cuál de las siguientes secuencias de aristas representa el orden en el que las selecciona el algoritmo de Prim como integrantes del árbol de recubrimiento mínimo:

a. {1,6},{1,7},{6,2},{6,3},{4,6},{6,5},{6,8}.
b. {1,7},{1,6},{6,3},{6,2},{4,6},{6,5},{6,8}.
c. {1,6},{6,2},{2,3},{3,8},{4,6},{6,5},{6,7}.
d. {1,6},{1,2},{2,3},{4,6},{6,5},{6,8},{6,7}.

Solo como anotación, los costes de los recorridos para la matriz o lista de adyacencia son:

- Matriz de Adyacencia: $O(n^2)$
- Listas de Adyacencia: $O(n \cdot a)$

Para resolver este tipo de problemas, se dibuja una tabla con 3 columnas, las cuales serán:

- A: Arista a seleccionar en cada caso.
- nodos: son los nodos que van a estar conectados mediante las aristas que vamos a ir seleccionando.
- aristas (S): aristas que van a formar parte del conjunto de soluciones S.

Se comienza en el nodo 1, en base a lo indicado por el enunciado, tomando la arista de menor peso (incluyendo el 0), anotándola al conjunto de soluciones.

Ahora podemos partir del nodo 1 o el 6, según quien tenga la arista de menor peso. Este será el caso del nodo 6, pudiendo tomar las aristas 7 u 8 (peso 0 en ambos casos).

No obstante, vamos a guiarnos por la columna 6, y a seguir un orden numérico, por lo que la arista {2, 6} tiene un peso 0, seleccionándola y anotándola al conjunto de soluciones.

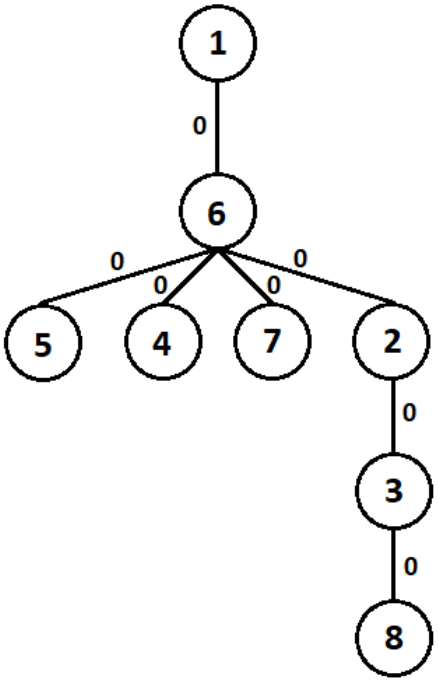
Se toma ahora la fila o columna del nodo 2, teniendo como menor peso la arista {2, 3}.

El proceso sigue yendo seleccionando la arista de menor peso desde cualquiera de las que ya se han seleccionado.

La tabla que vamos dibujando, queda finalmente como:

A	nodos	aristas (S)
1, 6	1	{1, 6}
6, 2	1, 6	{1, 6}, {6, 2}
2, 3	1, 6, 2	{1, 6}, {6, 2}, {2, 3}
3, 8	1, 6, 2, 3	{1, 6}, {6, 2}, {2, 3}, {3, 8}
4, 6	1, 6, 2, 3, 8	{1, 6}, {6, 2}, {2, 3}, {3, 8}, {4, 6}
6, 5	1, 6, 2, 3, 8, 4	{1, 6}, {6, 2}, {2, 3}, {3, 8}, {4, 6}, {6, 5}
6, 7	1, 6, 2, 3, 8, 4, 5	{1, 6}, {6, 2}, {2, 3}, {3, 8}, {4, 6}, {6, 5}, {6, 7}

El grafo resultante sería:



Respuesta C)

2. Indica cuál de las siguientes afirmaciones es **falsa**:

   a. En un algoritmo de vuelta atrás es posible retroceder para deshacer una decisión ya tomada.
   b. La programación dinámica es aplicable a muchos problemas de optimización cuando se cumple el Principio de Optimalidad de Bellman.
   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 como base el algoritmo de Dijkstra. La resolución de este problema completo tiene una complejidad de $O(n^2)$.
   d. La programación dinámica es apropiada para resolver problemas que pueden descomponerse en subproblemas más sencillos en los que haya llamadas repetidas en la secuencia de llamadas recursivas.

a) **Cierto**. Las soluciones se construyen de forma incremental, de forma que a un nodo del nivel k del árbol le corresponderá una parte de la solución construida en los k pasos dados en el grafo para llegar a dicho nodo, y que llamamos solución o secuencia k-prometedora. Si la solución parcial construida en un nodo no se puede extender o completar, decimos que se trata de un nodo de fallo. Cuando se da esta

situación, el algoritmo retrocede hasta un nodo del que quedan ramas pendientes de ser exploradas.

b) **Cierto**. Para muchos problemas de optimización es posible aplicar el esquema/algoritmo de la Programación Dinámica cuando se cumple el principio de Optimalidad de Bellman, el cual dice que, dada una secuencia óptima de decisiones, toda subsecuencia de ella es, a su vez, óptima.

c) **Falso**. El camino más corto entre un nodo y los restantes se selecciona con un algoritmo Voraz, tomando como base el algoritmo de Dijkstra, el cual tiene un coste de $O(n^2)$. Cuando se da cada par de nodos del grafo dicho coste es $O(n^3)$.

d) **Cierto**. En la Programación Dinámica se va dividiendo en subproblemas de tamaño menor que el original. Estas divisiones sucesivas del problema se plantean generalmente en forma de algoritmos recursivos. Una condición para que estos algoritmos sean eficientes es que no haya llamadas repetidas en la secuencia de llamadas recursivas. Cuando ocurren estas repeticiones conviene recurrir al esquema de programación dinámica, almacenando los resultados ya calculados para reutilizarlos. Algunos casos en los que es aplicable el algoritmo de Programación Dinámica, son problemas que también se pueden abordar con el esquema "Divide y Vencerás".

Respuesta C)

3. Tenemos una tabla hash de tamaño 7 para almacenar mediante recorrido lineal valores naturales con h(x) = x mod 7. Indicar la respuesta correcta sobre la tabla resultante de insertar en este orden los valores: 0,11,3,7,1,9.

Índice	0	1	2	3	4	5	6
Clave							

a. El índice 3 tiene clave 3
b. El índice 1 tiene clave 1
c. El índice 2 no está ocupado por ningún valor
d. El índice 6 tiene valor 9

Elementos a insertar: {0, 11, 3, 7, 1, 9}

h(x) = x % 7

Factor de Carga (F_C) = Nº elementos ocupados / nº índices

$F_C = 6/7 = 0'85 \rightarrow F_C = 85\%$

Si el F_C (Factor de Carga o Grado de Ocupación) de una Tabla Hash es alto (> 75%), el número de colisiones aumenta significativamente.

Para solucionar las colisiones, se aplica el método de Rehashing, el cual consiste en aumentar el tamaño de la tabla para reducir el FC.

Con ello, tenemos la ventaja de mantener un reducido F_C para que, las principales operaciones (insertar, buscar y eliminar), se realicen en un tiempo constante.

El único problema es que se requiere la construcción de un nuevo array e insertar nuevamente todos los datos.

Cálculos

0 mod 7 = 0

11 mod 7 = 4

3 mod 7 = 3

7 mod 7 = 0

1 mod 7 = 1

9 mod 7 = 2

(El resultado de estos cálculos corresponden al índice donde se inserta el elemento "X").

La tabla resultante es la siguiente:

Índice	0	1	2	3	4	5	6
Clave	0	1	9	3	11	-	-

Respuesta A)

4. Considérese el vector [7,10,5,2,20,15,1,5]. Los vectores argumento de la primera invocación recursiva del algoritmo de quicksort, cuando se toma el elemento 7 de la primera posición como pivote, son:

   a. [5,2,5,1] y [15,20,10]
   b. [1,5,5,2] y [15,20,10]
   c. [5,2,5,1] y [10,20,15]
   d. Ninguna de las opciones anteriores.

v = [7, 10, 5, 2, 20, 15, 1, 5] → Se toma el elemento 7 de la primera posición como pivote.

El método de ordenación por partición (Quicksort) se define como un procedimiento recursivo.

Trabaja mejor con elementos desordenados en la entrada, que con elementos semiordenados.

Quicksort se basa en el algoritmo Divide y Vencerás, siendo más rápido y fácil de ordenar dos vectores o listas de datos pequeños, que un vector o una lista grande.

Al inicio de la ordenación se toma un elemento aproximadamente en la mitad del vector, por lo que, cuando se empieza a ordenar, hay que llegar hasta dicho vector ordenado respecto al punto de división o a la mitad del vector.

Se garantiza que los elementos a la izquierda de la mitad son los menores, y los situados a la derecha son los mayores.

Paso 1

P							
7	10	5	2	20	15	1	5
	i > p						j < p

10 > 7 → Sí 5 < 7 → Sí

Se intercambian y el índice "i" y "j" se mueven una posición a la derecha e izquierda, respectivamente.

Paso 2

P							
7	5	5	2	20	15	1	10
		i > p				j < p	

Paso 3

P							
7	5	5	2	20	15	1	10
			i > p			j < p	

Paso 4

P							
7	5	5	2	20	15	1	10
				i > p		j < p	

Paso 5

P					i > p		
7	5	5	2	1	15	20	10
					j < p		

Como 15 > 7, así como 7 < 15, se intercambia 7 por 1.

Paso 6

				P			
5	5	2	1	**7**	15	20	10

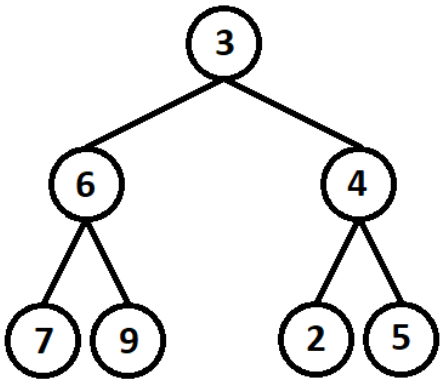
El vector resultante queda como: {1, 5, 5, 2} 7 {15, 20, 10}

Respuesta B)

5. Considérese el vector *v=[3,6,4,7,9,2,5]*. ¿Cuál de las siguientes opciones es cierta?

   a. El vector *v* es un montículo de mínimos.
   b. *v* sería un montículo de mínimos si se flotara el elemento de valor 2.
   c. *v* sería un montículo de mínimos si se hundiera el elemento de valor 6.
   d. Ninguna de las opciones anteriores.

La mejor forma de poder dar respuesta a este tipo de ejercicios, es dibujar lo primero el grafo que representa al vector, siendo éste el siguiente:



Para que el vector fuese un montículo de mínimos, cada nodo del grafo debe ser menor o igual que cualquiera de sus nodos hijos.

Las funciones de las respuestas b) y c) siguen la siguiente definición:

- Hundir: Se realiza un intercambio por el nodo hijo de menor valor.
- Flotar: Se realiza un intercambio por el nodo inmediatamente a un nivel superior.

Respuesta B)

6. Una filmoteca ha organizado un maratón de cine de terror. Durante 24 horas se proyectarán películas (todas diferentes) en las n salas disponibles. Un aficionado a este género de películas ha conseguido la programación completa donde aparecen todas las películas que se van a proyectar durante el maratón. Junto con el título, nombre del director, duración de la película y otros datos de interés, se indica la sala de proyección y la hora de comienzo. Indique qué tipo de algoritmo de entre los siguientes sería el más eficiente para el aficionado planifique su maratón de cine, teniendo en cuenta que su único objetivo es ver el máximo número de posible de películas:

   a. Esquema voraz.
   b. Esquema divide y vencerás
   c. Esquema de vuelta atrás.
   d. Esquema de ramificación y poda.

Respuesta A)

Problema

En la compleja red de metro de Tokyo, la cantidad que se paga por un billete es proporcional a la distancia que se recorre. Por tanto es necesario instalar en cada estación un panel informativo que indique el precio del billete a cualquier otra estación de la red. Describir el algoritmo más eficiente que calcule la información de todos esos paneles, de forma que el viajero se trasladará de una estación a otra por el camino más corto.

Se pide:

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

Existen dos posibles esquemas apropiados para resolver este problema:

- *Voraz*: Aplicar Dijkstra N veces, una por cada nodo de la red. Se pide el algoritmo detallado y la demostración de optimalidad (como aparece en el texto del libro base de la asignatura).

```
fun Voraz(c: conjuntoCandidatos): conjuntoCandidatos
    sol ← ∅
    mientras c ≠ ∅ ∧ ¬ solucion(sol) hacer
        x ← seleccionar(c)
        c ← c \{x}
        si factible(sol ∪ {x}) entonces
            sol ← sol ∪ {x}
        fsi
    fmientras
    si solucion(sol) entonces devolver sol
        sino imprimir('no hay solución')
    fsi
ffun
```

- *Programación Dinámica*: Algoritmo de Floyd. Se piden las ecuaciones de recurrencia y el algoritmo detallado.

```
fun ObjetosMochila(vol: vector, M:Tabla, n:entero, V:entero, objetos: Vector)
    var
        i,W: entero
    fvar
    W ← V
    para i ← n hasta 1 incremento -1 hacer
        si M[i,W] = M[i-1,W] entonces
            objetos[i] ← 0
        sino
            objetos[i] ← 1
            W ← W - vol[i]
        fsi
    fpara
ffun
```

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

c) Algoritmo completo a partir del refinamiento del esquema general (2,5 puntos solo si el punto 1 es correcto). Indicar demostración de optimalidad si es voraz. Indicar las ecuaciones de recurrencia en caso de programación dinámica.

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

Para cualquiera que haya sido el algoritmo elegido, el coste es $O(n^3)$.