FEBRERO 2017 – SEMANA 1 (MODELO A)

1. Un servidor tiene que atender *tres* clientes que llegan todos juntos al sistema. El tiempo que requerirá dar servicio a cada cliente es conocido, siendo t_1=5, t_2=10 y t_3=3. El objetivo es minimizar el tiempo medio de estancia de los clientes en el sistema. Se quiere implementar un algoritmo voraz que construya la secuencia ordenada óptima de servicio a los distintos clientes. Según este algoritmo, el tiempo mínimo de estancia en el sistema del conjunto de clientes es:

   (a) 26
   (b) 29
   (c) 31
   (d) 34

   $t_1 = 5$ $C = [t_1, t_2, t_3]$

   $t_2 = 10$ $S = 0$

   $t_3 = 3$ Se parte de un conjunto de soluciones S vacío

   Este tipo de problemas pueden resolverse con un Esquema Voraz, donde:

   - Solución: se da cuando el tamaño de S es igual al tamaño de C
   - Factible: la consideramos cierta (true), es decir, el que exista una solución
   - Selección: Se elige al cliente con menor tiempo de servicio
   - La estructura de datos es un array

   Orden Cliente < Tiempo Servicio

   $$t_3 < t_1 < t_2$$

   $$t_3 + (t_3 + t_1) + (t_3 + t_1 + t_2) = 3 + (3 + 5) + (3 + 5 + 10) = 3 + 8 + 18 = \mathbf{29}$$

   El tiempo mínimo de estancia de un cliente en el sistema, en el conjunto de todos los clientes C, es 29.

Respuesta B)

2. Sea el problema de la devolución de cambio con monedas de valores 1,6 y 10 solucionado con programación dinámica para pagar una cantidad de 12 unidades. Identifica cuál de las siguientes respuestas correspondería al contenido de la tabla de resultados parciales de cantidades en la fila correspondiente a la moneda de valor 6, si dichas monedas se consideran por orden creciente de valores:

   (a) 0 1 2 3 4 5 6 2 3 4 5 6 3
   (b) 0 1 2 3 4 5 6 2 3 4 5 6 7
   (c) 0 1 2 3 4 5 1 2 3 4 5 6 2
   (d) Ninguna de las anteriores.

   El modo de calcular los valores de la tabla, es el siguiente:

   - Si el valor de la moneda actual (m_i) es mayor que la cantidad, se paga con el resto de monedas, es decir, se toma el resultado de la fila y columna anteriores.
   - Si el valor de la moneda m_i es menor o igual que la cantidad, se toma el valor mínimo entre:
     a. Pagar con el resto de monedas (fila y columna anteriores)

b. Pagar con una moneda igual a m_i, y el resto con el resultado que se hubiese obtenido al pagar la cantidad actual a la que se le ha restado el valor de la moneda actual.

Mínimo valor entre…

- 1 + C[Fila, Columna] → 1+ C[Fila, (Moneda en Uso – Moneda Actual)]
- C[Fila anterior, Columna actual]

Ejemplo.- $m_2 = 6$, Columna 6

6 – 6 = **0*** → Fila Anterior, Columna **0*** → 6

0 + 1 = 1

El valor mínimo entre 6 y 1 → 1

	CANTIDAD A PAGAR												
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
$m_1 = 1$	0	1	2	3	4	5	6	7	8	9	10	11	12
$m_2 = 6$	0	1	2	3	4	5	1	2	3	4	5	6	2
$m_3 = 10$	0	1	2	3	4	5	1	2	3	4	1	2	2

Respuesta C)

3. Se tiene una tabla hash de tamaño m=11 y las funciones $h_1(k) = k \bmod m$ y $h_2(k) = (k \bmod (m - 1)) + 1$. Se pide insertar los valores [22,1,13,11,24,33,18,42,31] mediante doble hashing usando $h(k,i) = (h_1(k) + ih_2(k)) \bmod m$, donde *i* indica el incremento en caso de colisión.

(a)

0	1	2	3	4	5	6	7	8	9	10
22	1	13		11	18	31	24	33	42	

(b)

0	1	2	3	4	5	6	7	8	9	10
22	1	13	18	11		31	24	33	42	

(c)

0	1	2	3	4	5	6	7	8	9	10
22	1	13	11		18	31	24	33	42	

(d)

0	1	2	3	4	5	6	7	8	9	10
22	1	13	24	33	18	31		11	42	

$h_1(k) = k \bmod m$

$h_2(k) = (k \bmod (m-1)) + 1$

$h(k, i) = (h_1(k) + i \cdot h_2(k)) \bmod m$

La función más importante es la última de ellas. Los cálculos a realizar para poder dibujar la tabla Hash resultante, son los siguientes:

Cálculo 1

$h(22, 0) = (h_1(22) + \cancel{0 \cdot h_2(22)}) \bmod 11 = 0 \bmod 11 = 0$

→ $h_1(22) = 22 \bmod 11 = 0$

Cálculo 2

$h(1, 0) = (h_1(1) + \cancel{0 \cdot h_2(1)}) \bmod 11 = 1 \bmod 11 = 1$

→ $h_1(1) = 1 \bmod 11 = 1$

Cálculo 3

$h(13, 0) = (h_1(13) + \cancel{0 \cdot h_2(13)}) \bmod 11 = 2 \bmod 11 = 2$

→ $h_1(13) = 13 \bmod 11 = 2$

Cálculo 4

$h(11, 0) = (h_1(11) + \cancel{0 \cdot h_2(11)}) \bmod 11 = 0 \bmod 11 = 0$ → Hay colisión, i+1 = 1

→ $h_1(11) = 11 \bmod 11 = 0$

$h(11, 1) = (h_1(11) + 1 \cdot h_2(11)) \bmod 11 = (0 + 2) \bmod 11 = 2 \bmod 11 = 2$ → Colisión, i+1 = 2

→ $h_1(11) = 11 \bmod 11 = 0$

$h_2(11) = 11 \bmod (11-1)) + 1 = (11 \bmod 10) + 1 = 1 + 1 = 2$

$h(11, 2) = (h_1(11) + 2 \cdot h_2(11)) \bmod 11 = (0 + (2 \cdot 2)) \bmod 11 = (0 + 4) \bmod 11 = 4 \bmod 11 = 4$

Cálculo 5

$h(24, 0) = (h_1(24) + \cancel{0 \cdot h_2(24)}) \bmod 11 = 2 \bmod 11 = 2$ → Colisión, i+1 = 1

→ $h_1(24) = 24 \bmod 11 = 2$

$h(24, 1) = (h_1(24) + 1 \cdot h_2(24)) \bmod 11 = (2 + (1 \cdot 5)) \bmod 11 = 7 \bmod 11 = 7$

→ $h_2(24) = (24 \bmod 10) + 1 = 4 + 1 = 5$

Cálculo 6

$h(33, 0) = (h_1(33) + \cancel{0 \cdot h_2(33)}) \bmod 11 = 0 \bmod 11 = 0$ → Colisión, i+1 = 1

→ $h_1(33) = 33 \bmod 11 = 0$

$h(33, 1) = (h_1(33) + 1 \cdot h_2(33)) \bmod 11 = (0 + (1 \cdot 4)) = 4 \bmod 11 = 4$ → Colisión, i+1 = 2

→ $h_2(33) = (33 \bmod 10) + 1 = 3 + 1 = 4$

$h(33, 2) = (h_1(33) + 2 \cdot h_2(33)) \bmod 11 = (0 + 2 \cdot 4) \bmod 11 = 8 \bmod 11 = 8$

Cálculo 7

$h(18, 0) = (h_1(18) + \cancel{0 \cdot h_2(18)}) \bmod 11 = 7 \bmod 11 = 7$ → Colisión, i+1 = 1

→ $h_1(18) = 18 \bmod 11 = 7$

$h(18, 1) = (h_1(18) + 1 \cdot h_2(18)) \bmod 11 = (7 + 9) \bmod 11 = 16 \bmod 11 = 5$

→ $h_2(18) = (18 \bmod 10) + 1 = 8 + 1 = 9$

Cálculo 8

$h(42, 0) = (h_1(42) + \cancel{0 \cdot h_2(42)}) \bmod 11 = 9 \bmod 11 = 9$

→ $h_1(42) = 42 \bmod 11 = 9$

Cálculo 9

$h(31, 0) = (h_1(31) + \cancel{0 \cdot h_2(31)}) \bmod 11 = 9 \bmod 11 = 9$ → Colisión, i+1 = 1

→ $h_1(31) = 31 \bmod 11 = 9$

$h(31, 1) = (h_1(31) + 1 \cdot h_2(31)) \bmod 11 = (9 + 2) \bmod 11 = 11 \bmod 11 = 0$ → Colisión, i+1 = 2

→ $h_2(31) = (31 \bmod 10) + 1 = 1 + 1 = 2$

$h(31, 2) = (h_1(31) + 2 \cdot h_2(31)) \bmod 11 = (9 + 2 \cdot 2) \bmod 11 = (9 + 4) \bmod 11 = 13 \bmod 11 = 2$ → Colisión, i+1 = 3

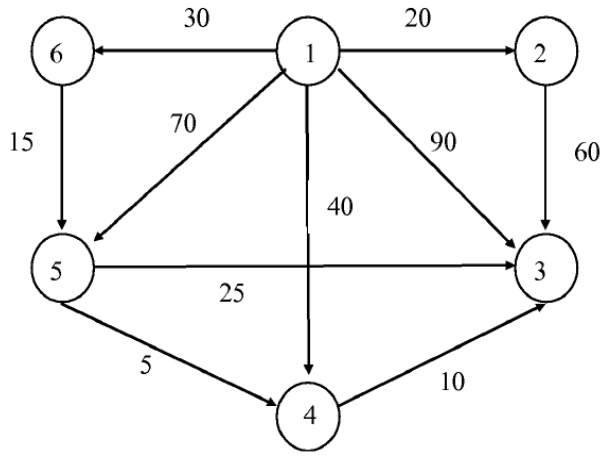
$h(31, 3) = (h_1(31) + 3 \cdot h_2(31)) \bmod 11 = (9 + 3 \cdot 2) \bmod 11 = (9 + 6) \bmod 11 = 15 \bmod 11 = 4$ → Colisión, i+1 = 4

$h(31, 4) = (h1(31) + 4 \cdot h2(31)) \bmod 11 = (9 + 4 \cdot 2) \bmod 11 = (9 + 8) \bmod 11 = 17 \bmod 11 = 6$

Índice	0	1	2	3	4	5	6	7	8	9	10
Clave	22	1	13	-	11	18	31	24	33	42	-

Respuesta A)

4. Dado el grafo de la figura:



Indique cuál sería el orden en que se seleccionan los nodos del conjunto de candidatos al aplicar el algoritmo de Dijkstra comenzando por el nodo 1:

(a) 1 2 6 5 3 4
(b) 1 2 6 5 4 3
(c) 1 2 6 4 5 3
(d) Ninguna de las anteriores

Iteración 1

Sale el nodo 1, y el resto, quedan sin salir. Se toma el nodo cuya distancia sea menor, que es la del nodo 2 (Distancia 20). El predecesor es el nodo 1 en todos los casos.

Iteración 2

Salen los nodos 1 y 2, quedándose sin salir el resto. El nodo con menor distancia es el 6 (Distancia 30). Se calcula la distancia desde el nodo 1 al resto de nodos, pasando por el nodo 2. De 1 a 3, la distancia es menor pasando por el nodo 2 que sin pasar por él (Distancia 80), por lo que su predecesor es el 2.

Iteración 3

- Salen: 1, 2, 6
- No salen: 3, 4, 5
- Predecesor nuevo: 6 del nodo 1 al 5
- Distancias nuevas: 45 del nodo 1 al 5
- Nodo que se toma: 4 (Distancia menor: 40)

Iteración 4

- Salen: 1, 2, 6, 4
- No salen: 3, 5
- Predecesor nuevo: 4 del nodo 1 al 3
- Distancias nuevas: 50 del nodo 1 al 3
- Nodo que se toma: 5 (Distancia menor: 45)

Iteración 5

- Salen: 1, 2, 6, 4, 5
- No salen: 3
- Predecesor nuevo: Ninguno
- Distancias nuevas: Ninguna
- Nodo que se toma: 3 (Distancia menor: 50)

Nodos que han salido	Nodos que no han salido	Vector Distancia especial [ ]					Predecesores				
		2	3	4	5	6	2	3	4	5	6
1	2, 3, 4, 5, 6	**20**	90	40	70	30	1	1	1	1	1
1, 2	3, 4, 5, 6	20	80	40	70	**30**	1	2	1	1	1
1, 2, 6	3, 4, 5	20	80	**40**	45	30	1	2	1	6	1
1, 2, 6, 4	3, 5	20	50	40	**45**	30	1	4	1	6	1
1, 2, 6, 4, 5	3	20	50	40	45	30	1	4	1	6	1

La solución pasa por tomar los nodos que han ido saliendo en el orden en que estos han salido, es decir: 1, 2, 6, 4, 5, 3

Respuesta C)

5. Indique cuál de las siguientes afirmaciones con respecto a los grafos es **falsa**:

(a) Un grafo es simple si entre cada par de vértices existe a lo sumo una arista.
(b) La longitud de un camino es el número de aristas que contiene.
(c) Un grafo dirigido es conexo si al reemplazar todas sus aristas dirigidas por aristas no dirigidas resulta un grafo conexo no dirigido.
(d) Se denomina componente conexa de un grafo a un subgrafo conexo maximal.

a) **Cierto**.
b) **Cierto**.
c) **Falso**. Un grafo es conexo si para cualquier par de vértices distintos existe un camino que los contiene. Un grafo dirigido es fuertemente conexo si para cada par de vértices distintos n y m hay un camino de "n" a "m" y también hay un camino de "m" a "n".
d) **Cierto**.

Respuesta C)

6. En el problema de colorear un grafo con *n* nodos utilizando *m* colores, de manera que no haya dos vértices adyacentes que tengan el mismo color, una cota superior ajustada del coste de encontrar una solución utilizando un esquema adecuado es del orden de:

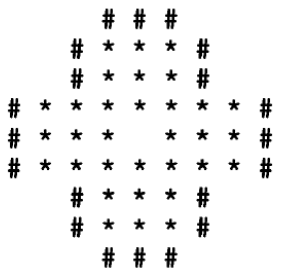
(a) $O(n^{m/2} \log n)$.
(b) $O(m \log n)$.
(c) $O(n^m)$.
(d) $O(nm^n)$.

Se utiliza el algoritmo de Vuelta Atrás. En situación normal su coste es $O(m^n)$.

Respuesta D)

Problema (4 puntos).

Se distribuyen 32 fichas (*) sobre un tablero como muestra la figura:



El juego consiste en mover las fichas hasta que solo quede una, con las siguientes condiciones:

- Las fichas * se mueven saltando una sobre otra, desapareciendo la ficha sobre la que se salta. No se salta en diagonal. Las # indican fin del tablero. No se puede saltar sobre ellas.
- Solo se salta sobre una ficha (y solo una) si tras ella hay un hueco.

Se pide programar un algoritmo que resuelva el problema en el menor número de movimientos.

La resolución de este 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 Vuelta Atrás. Hay que destacar que el algoritmo no precisa estimar el coste de la solución, sino el número de movimientos.

No tiene sentido el esquema de Ramificación y Poda porque la estimación del número de movimientos hasta la solución es conocida (31) y además no puede variar de una solución a otra.

```
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).

Se puede representar el tablero como una matriz bidimensional en la que los elementos pueden tomar los valores *novalida,* libre u ocupada, dependiendo de que esa posición no sea válida (no corresponda a una de las posiciones que forman la "cruz"), o bien siendo válida exista ficha o no. La solución vendrá dada en una matriz $X = [x_1, x_2, ..., x_m]$, en donde xi representa un movimiento (salto) del juego.

Cada movimiento se representará con la posición de la ficha que va a efectuar el movimiento, la posición hacia donde da el salto y la posición de la ficha "comida".

El valor m representa el número de movimientos (saltos) efectuados para alcanzar la solución. Puesto que en cada movimiento ha de comerse obligatoriamente una ficha, sabemos que m podrá tomar como máximo valor 31.

3. Algoritmo completo 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.

La solución vendrá dada en una tupla de valores X = [x1, x2, ..., xm] en donde xi representa un movimiento (salto) del juego.

En este valor se almacenará la posición de la ficha que va a efectuar el movimiento, la posición hacia donde da el salto y la posición de la ficha "comida". El valor m representa el número de saltos (movimientos) efectuados para alcanzar la solución.

Puesto que en cada movimiento ha de comerse obligatoriamente una ficha, sabemos que m podrá tomar a lo sumo el valor 31. Con esto, el algoritmo que resuelve el problema es:

```
CONST m = 31;  (* numero maximo de movimientos *)
      n = 7;   (* tamano del tablero *)
TYPE ESTADO   = (libre,ocupada,novalida);(* tipo de casilla *)
     TABLERO  = ARRAY[1..n],[1..n] OF ESTADO;
     PAR      = RECORD x,y:INTEGER END; (* coordenadas *)
     SALTO    = RECORD origen,destino,comido:PAR END;
     SOLUCION = ARRAY [1..m] OF SALTO;

PROCEDURE Continental(VAR k:CARDINAL;VAR t:TABLERO;
                      VAR encontrado:BOOLEAN;VAR sol:SOLUCION);
  VAR i,j:CARDINAL;
BEGIN
  IF Fin(k,t) THEN encontrado:=TRUE
  ELSE
    FOR i:=1 TO n DO
      FOR j:=1 TO n DO
        IF Valido(i,j,1,t,encontrado) THEN(* a la izquierda *)
          INC(k);
          sol[k].origen.x:=i;
          sol[k].origen.y:=j;
          sol[k].destino.x:=i;
          sol[k].destino.y:=j-2;
          sol[k].comido.x:=i;
          sol[k].comido.y:=j-1;
          NuevaTabla(t,i,j,1); (* actualiza el tablero *)
          Continental(k,t,encontrado,sol)
        END;
        IF Valido(i,j,2,t,encontrado) THEN (* hacia arriba *)
          INC(k);
          sol[k].origen.x:=i;
          sol[k].origen.y:=j;
          sol[k].destino.x:=i-2;
          sol[k].destino.y:=j;
          sol[k].comido.x:=i-1;
          sol[k].comido.y:=j;
          NuevaTabla(t,i,j,2);(* actualiza el tablero *)
          Continental(k,t,encontrado,sol)
        END;
```

```
            IF Valido(i,j,3,t,encontrado) THEN (* a la derecha *)
               INC(k);
               sol[k].origen.x:=i;
               sol[k].origen.y:=j;
               sol[k].destino.x:=i;
               sol[k].destino.y:=j+2;
               sol[k].comido.x:=i;
               sol[k].comido.y:=j+1;
               NuevaTabla(t,i,j,3);(* actualiza el tablero *)
               Continental(k,t,encontrado,sol)
            END;
            IF Valido(i,j,4,t,encontrado) THEN (* hacia abajo *)
               INC(k);
               sol[k].origen.x:=i;
               sol[k].origen.y:=j;
               sol[k].destino.x:=i+2;
               sol[k].destino.y:=j;
               sol[k].comido.x:=i+1;
               sol[k].comido.y:=j;
               NuevaTabla(t,i,j,4);(* actualiza el tablero *)
               Continental(k,t,encontrado,sol)
            END;
         END
      END;
      IF NOT encontrado THEN (* cancelar anotacion *)
         RestaurarTabla(t,k,sol);
         AnularSalida(sol,k);
         DEC(k)
      END
   END
END Continental;
```

La función Fin determina si se ha llegado al final del juego, esto es, si sólo queda una ficha y ésta se encuentra en el centro del tablero, y la función Valido comprueba si una ficha puede moverse o no:

```
PROCEDURE Valido(i,j,mov:CARDINAL;VAR t:TABLERO;e:BOOLEAN):BOOLEAN;
BEGIN
   IF mov=1 THEN (* izquierda *)
      RETURN ((j-1>0) AND (t[i,j]=ocupada) AND(t[i,j-1]=ocupada)
               AND (j-2>0) AND (t[i,j-2]=libre) AND (NOT e))
   ELSIF mov=2 THEN (* arriba *)
      RETURN ((i-1>0) AND (t[i-1,j]=ocupada) AND(t[i,j]=ocupada)
               AND (i-2>0) AND (t[i-2,j]=libre) AND (NOT e))
   ELSIF mov=3 THEN (* derecha *)
      RETURN ((j+1<8) AND (t[i,j+1]=ocupada) AND(t[i,j]=ocupada)
               AND (j+2<8) AND (t[i,j+2]=libre) AND (NOT e))
   ELSIF mov=4 THEN (* abajo *)
      RETURN ((i+1<8) AND (t[i+1,j]=ocupada) AND(t[i,j]=ocupada)
               AND (i+2<8) AND (t[i+2,j]=libre) AND (NOT e))
   END
END Valido;
```

Un aspecto interesante de este problema es que pone de manifiesto la importancia que tiene el orden de generación de los nodos del árbol de expansión. Si en vez de seguir la secuencia de búsqueda utilizada en la anterior implementación (izquierda, arriba, derecha y abajo) se intentan los movimientos en otro orden, el tiempo de ejecución del algoritmo pasa de varios segundos a más de dos horas.

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

El coste de la solución es 4^{31}.