FEBRERO 2018 – SEMANA 2 (MODELO A)

Grado en Ingeniería Informática y Grado en Ingeniería en Tecnologías de la Información

Normas de valoración del examen:

- La nota del examen representa el 80% de la valoración final de la asignatura (el 20% restante corresponde a las prácticas).
- Cada cuestión contestada correctamente vale 1 punto.
- Cada cuestión contestada incorrectamente baja la nota en 0.3 puntos.
- Debe obtenerse un mínimo de 3 puntos en las cuestiones para que el problema sea valorado (con 3 cuestiones correctas y alguna incorrecta el examen está suspenso).
- La nota total del examen debe ser al menos de 4.5 para aprobar.
- **Las cuestiones se responden en una hoja de lectura óptica**.

Examen tipo A:

Cuestiones:

1. Sea el problema de la mochila, en el que tenemos una mochila de capacidad *M*, y *n* objetos con beneficios b_1, b_2, b_3, ... b_n y pesos p_1, p_2, p_3, ... p_n. El objetivo es maximizar el valor de los objetos transportados, respetando la limitación de la capacidad impuesta *M*. Indica, de los esquemas siguientes, cuál es el más adecuado en el caso de que cada objeto pueda meterse en la mochila entero o fraccionado.

   (a) El esquema voraz.
   (b) El esquema divide y vencerás.
   (c) El esquema de vuelta atrás.
   (d) El esquema de ramificación y poda.

a) El esquema o algoritmo Voraz admite tanto objetos enteros como fraccionables.
b) El esquema Divide y Vencerás no resuelve el problema de la mochila.
c) El esquema Vuelta Atrás admite solo elementos enteros. Resuelve el problema de la mochila con coste $O(2^n)$.
d) El esquema Ramificación y Poda admite solo elementos enteros, resolviendo el problema de la mochila con coste $O(2^n)$.

Si hablásemos del esquema de Programación Dinámica, este admite en el problema de la mochila solo elementos enteros, cuyo coste es:

- Sin Optimización de memoria: O(nm)
- Con Optimización de memoria: O(m)

Respuesta A)

2. Indica cuál de las siguientes afirmaciones es cierta con respecto al problema de la devolución de cambio de moneda utilizando un número mínimo de monedas y suponiendo que la disponibilidad de cada tipo de moneda es ilimitada:

   (a) El problema de la devolución de cambio de moneda se puede resolver para todo sistema monetario utilizando una estrategia voraz, siempre que en el sistema se disponga de un tipo de moneda de 1.
   (b) Si se dispone de n tipos de moneda $T = \{m^0, m^1, m^2, \ldots m^{n-1}\}$ siendo $m > 1$ y $n > 0$, el problema de la devolución de cambio de moneda no se puede resolver utilizando una estrategia voraz.
   (c) El problema de la devolución de cambio de moneda para cualquier sistema monetario, siempre que en el sistema se disponga de un tipo de moneda de 1, se puede resolver utilizando los esquemas de programación dinámica y de ramificación y poda.
   (d) El problema de la devolución de cambio de moneda para cualquier sistema monetario, siempre que en el sistema se disponga de un tipo de moneda de 1, se puede resolver utilizando el esquema de programación dinámica pero no el de ramificación y poda.

   a) **Falso**. Puede existir o no una moneda de 1 para poder resolver este problema.
   b) **Falso**. El esquema Voraz si es capaz de resolver el problema.
   c) **Cierto**. Tanto el esquema de Programación Dinámica (Coste: O(NC)), así como el de Ramificación y Poda, pueden resolver este problema.
   d) **Falso**. Tanto un esquema como otro pueden resolver el problema.

Respuesta C)

3. Indica cuál de las siguientes afirmaciones es cierta con respecto al algoritmo Quicksort para un vector de tamaño n.

   (a) Se compone de dos invocaciones recursivas con tamaño n/2 más un procedimiento que combina los subvectores ordenados resultantes que es de coste lineal.
   (b) Su coste en el caso peor es $O(n^2)$, pero existe una versión mejorada eligiendo como pivote la mediana del vector, lo que daría que el coste del Quicksort en el caso peor sería O(n).
   (c) Está compuesto de dos recorridos lineales del vector y posteriormente una llamada recursiva al subvector no ordenado.
   (d) Ninguna de las anteriores es cierta.

   a) **Falso**. El proceso del algoritmo Quicksort es:
      - Toma un elemento cualquiera del vector denominado *pivote*.
      - Toma los valores del vector que son menores que el pivote y forma un subvector. Se procede análogamente con los valores mayores o iguales.
      - Se invoca recursivamente al algoritmo para cada subvector.

      Necesita O(n) llamadas recursivas. Como la función *Pivotar* tiene un coste lineal, el algoritmo es de orden $O(n^2)$.
   b) **Falso**. En el caso peor el coste es O(n log n).
   c) **Falso**. Ver explicación en el apartado a).
   d) **Cierto**.

Respuesta D)

4. Una filmoteca ha organizado un maratón de cortometrajes. Durante 24 horas se proyectarán cortos de cine (todos diferentes) en las n salas disponibles. Un cinéfilo ha conseguido la programación completa donde aparecen todas las películas que se van a proyectar durante el maratón, incluyendo el título, duración del corto, sala en la que se proyecta y hora de comienzo. Si se quiere planificar el maratón del cinéfilo de forma que pueda ver el máximo número posible de cortos, ¿Cuál es el esquema más apropiado para hacer la planificación eficientemente?

   (a) Esquema voraz.
   (b) Divide y vencerás.
   (c) Esquema de vuelta atrás.
   (d) Esquema de ramificación y poda.

Respuesta A)

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

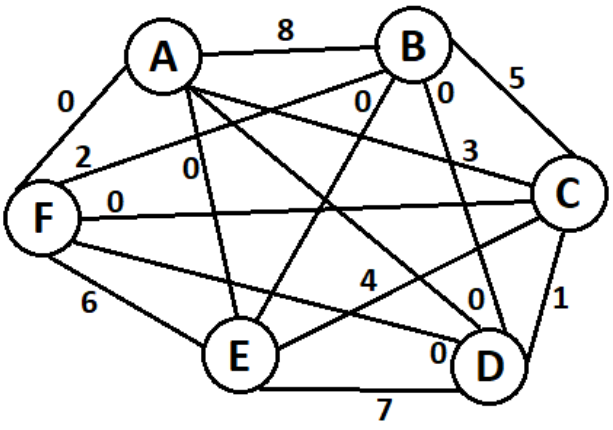
	A	B	C	D	E	F
A	-	8	3	0	0	0
B		-	5	0	0	2
C			-	1	4	0
D				-	7	0
E					-	6
F						-

   Si se utiliza el algoritmo de Prim para calcular un árbol de recubrimiento mínimo tomando como raíz del árbol el nodo A, indique 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) {A,C},{C,D},{C,E},{B,C},{B,F}.
   (b) {A,C},{C,D},{B,F},{C,E},{B,C}.
   (c) {A,C},{C,D},{C,E},{B,F},{B,C}.
   (d) Ninguna de las anteriores.

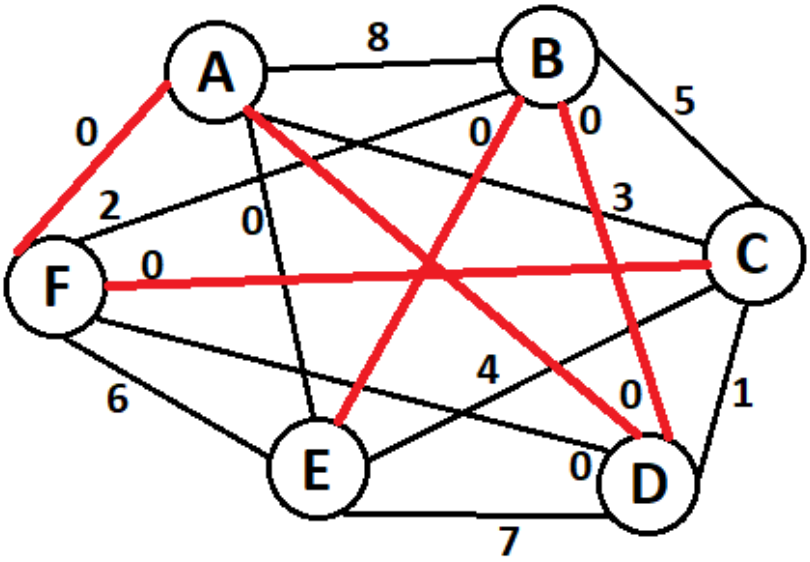
El algoritmo de PRIM (Página 66 del Libro de Texto Base de la asignatura) es utilizado siempre, únicamente, con grafos no dirigidos. Con él se pretende encontrar un árbol de recubrimiento mínimo (ARM) en un grafo conexo.

Un grafo denso es aquel en el que el número de aristas está cercano al número máximo de aristas. En contraparte, un grafo disperso es aquel que tiene solo algunas aristas. Para poder entender mejor la resolución del ejercicio, vamos a mostrar el grafo al completo en función a la matriz de adyacencia dada por el enunciado.



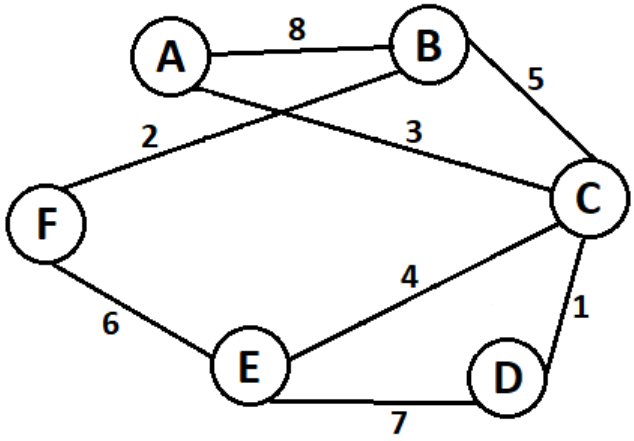
La respuesta realmente correcta es la D), ya que, como indica el propio enunciado, puede haber aristas con peso 0. En este caso, el algoritmo de PRIM escogería como primera arista {A, D}, {A, E} o {A, F} (cualquiera de ellas sería válida, en función a como se haya implementado la estructura de datos a la hora de recorrer el grafo).

Una de las posibles soluciones podría ser la de las aristas {A, D}, {D, B}, {B, E}, {A, F} y {F, C}, todas ellas con coste 0, por lo que el ARM tiene un coste total de 0. Para que se entienda mejor, es como si pudiéramos encontrar una red de carreteras que une a todas las ciudades, la cual está totalmente libre de peajes.



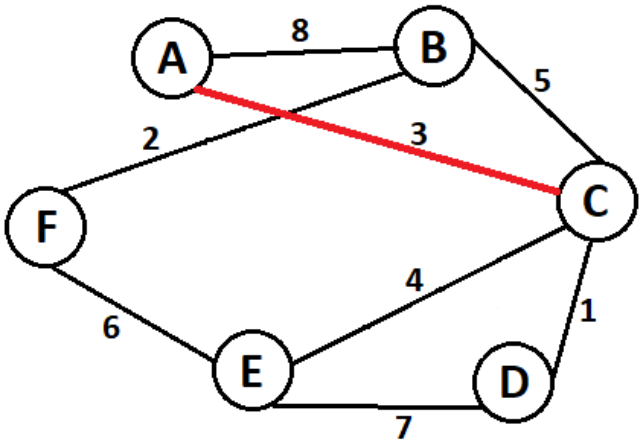
Sin embargo, la respuesta que el equipo docente proporcionó en su momento, además de admitir la D) como correcta, fue la respuesta A).

Para que la respuesta A) fuese válida, se tiene que obviar la frase que indica “en la que puede haber pesos de valor 0”. El grafo general podríamos dibujarlo como:



Por tanto, si no consideramos los pesos 0, la solución sería:

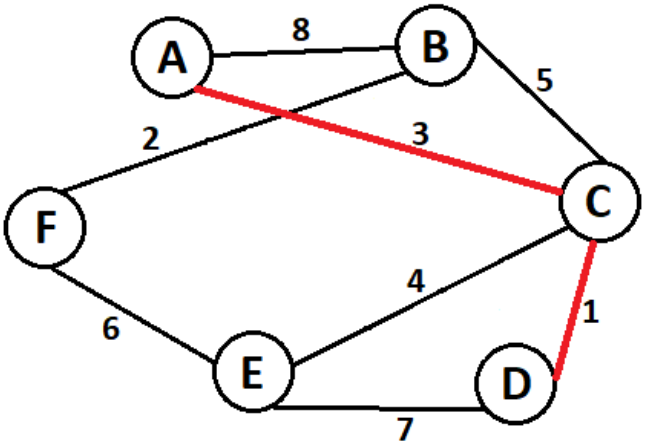
- Paso 1: Partimos del nodo A, por lo que el nodo de menor peso mas cercano a él es el C, con peso 3.



Aristas Elegidas	Nodos	Aristas (S)
A, C	A	{A, C}

- Paso 2: Podemos tomar cualquier nodo desde A o C, que tenga menor peso, mas cercano a alguno de dichos nodos (ya que, al ser un grafo no dirigido, no estamos situados en un nodo concreto).

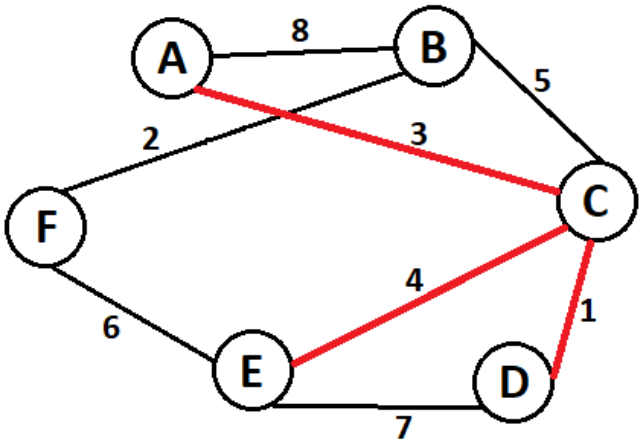
  Del nodo A al nodo B hay un peso de 8, y desde el nodo C al D hay peso 1, por lo que este es un camino de menor coste para el ARM:



Aristas Elegidas	Nodos	Aristas (S)
A, C	A	{A, C}
C, D	A, C	{A, C}, {C, D}

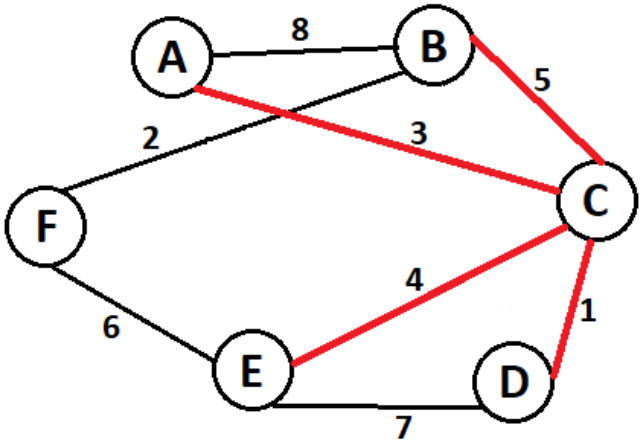
- Paso 3: Ahora elegiremos, de los nodos que no han salido, aquel que esté más cercano (con menor coste) a los nodos que ya han salido, A, C y D.

  Desde A al nodo B se tiene peso 8, de C a E peso 4, y de D a E peso 7. En este caso, el siguiente nodo que deberíamos elegir es el E:



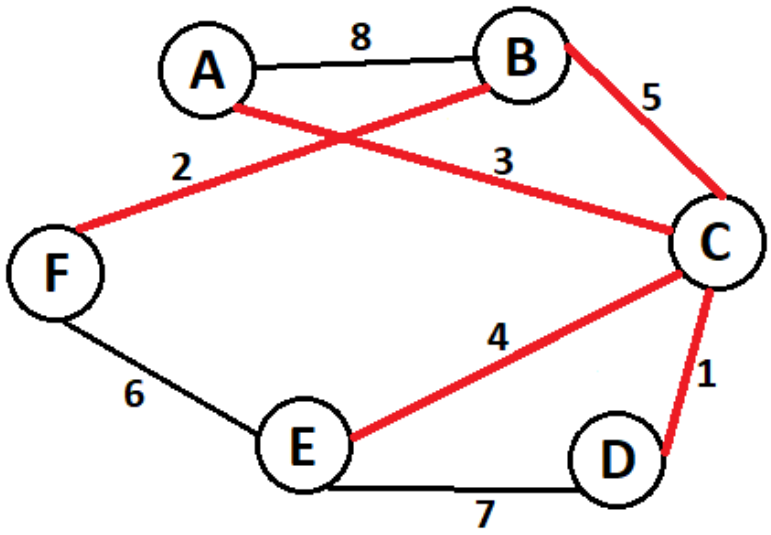
Aristas Elegidas	Nodos	Aristas (S)
A, C	A	{A, C}
C, D	A, C	{A, C}, {C, D}
C, E	A, C, D	{A, C}, {C, D}, {C, E}

- Paso 4: Se tiene ahora del nodo A al nodo B peso 8, del nodo C al nodo B peso 5, del nodo E al nodo D peso 7. En este caso el de menor peso es el camino del nodo C al B, por tanto:



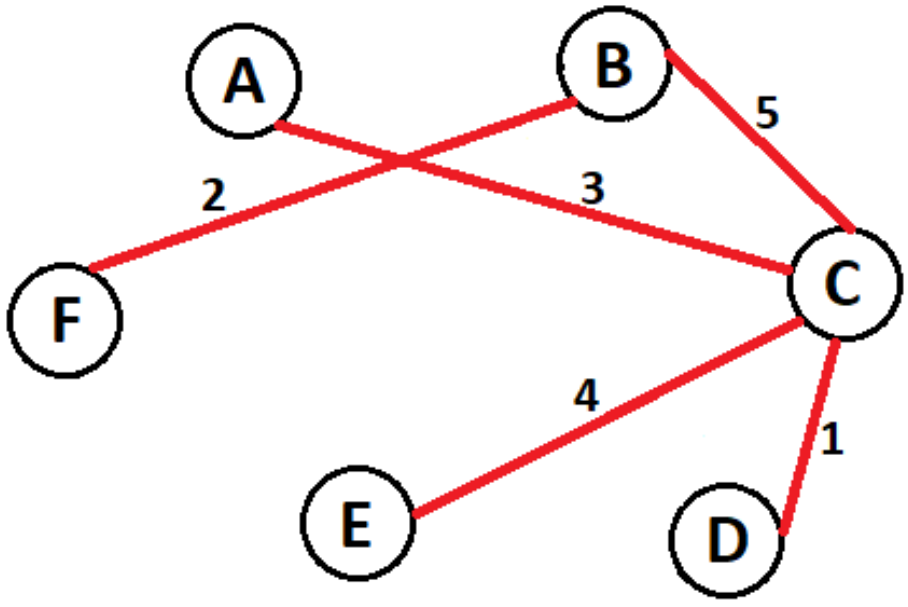
Aristas Elegidas	Nodos	Aristas (S)
A, C	A	{A, C}
C, D	A, C	{A, C}, {C, D}
C, E	A, C, D	{A, C}, {C, D}, {C, E}
B, C	A, C, D, E	{A, C}, {C, D}, {C, E}, {B, C}

- Paso 5: Ahora tendríamos del nodo A al nodo B peso 8, del nodo B al nodo F peso 2, del nodo D al nodo E peso 7, y del nodo E al nodo F peso 6. El que se debe elegir en este caso, con menor peso/coste, es el nodo F, para la arista {B, F}:



Aristas Elegidas	Nodos	Aristas (S)
A, C	A	{A, C}
C, D	A, C	{A, C}, {C, D}
C, E	A, C, D	{A, C}, {C, D}, {C, E}
B, C	A, C, D, E	{A, C}, {C, D}, {C, E}, {B, C}

Finalmente, el ARM resultante sería el siguiente:

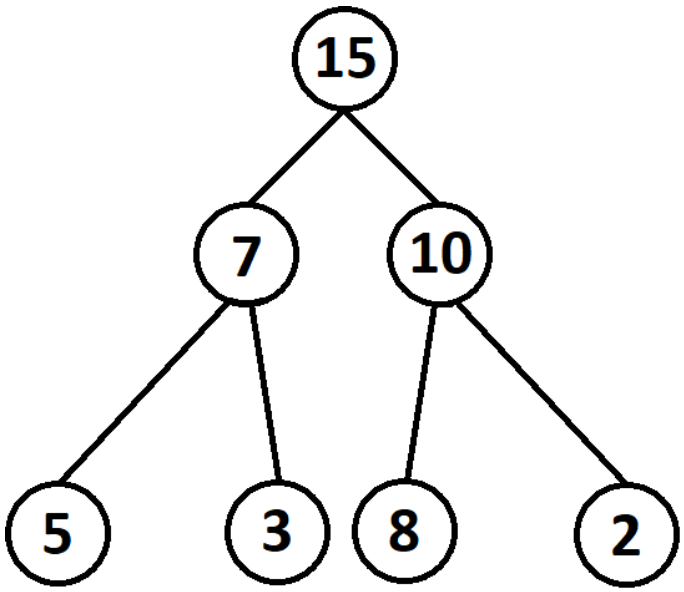


<u>Respuesta A) o D)</u>

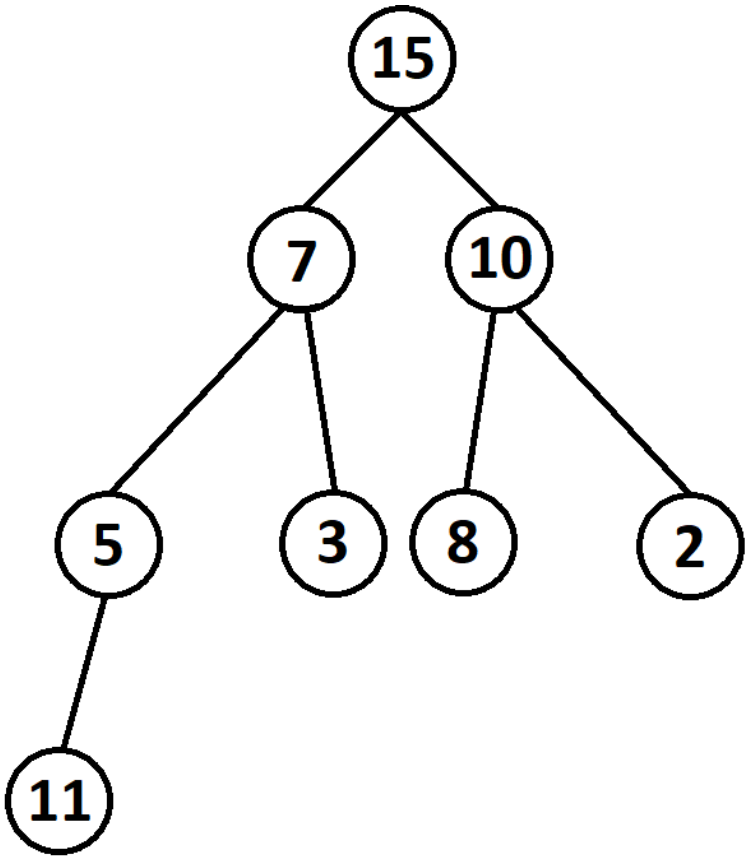
6. Considérese el vector [15,7,10,5,3,8,2] que representa un montículo de máximos. ¿Cuál sería la representación resultante de insertar en este montículo el valor 11 usando la función flotar?

   (a) [15,11,10,7,5,8,2,3]
   (b) [15,11,10,7,3,8,2,5]
   (c) [15,10,11,7,3,8,2,5]
   (d) Ninguna de las opciones anteriores.

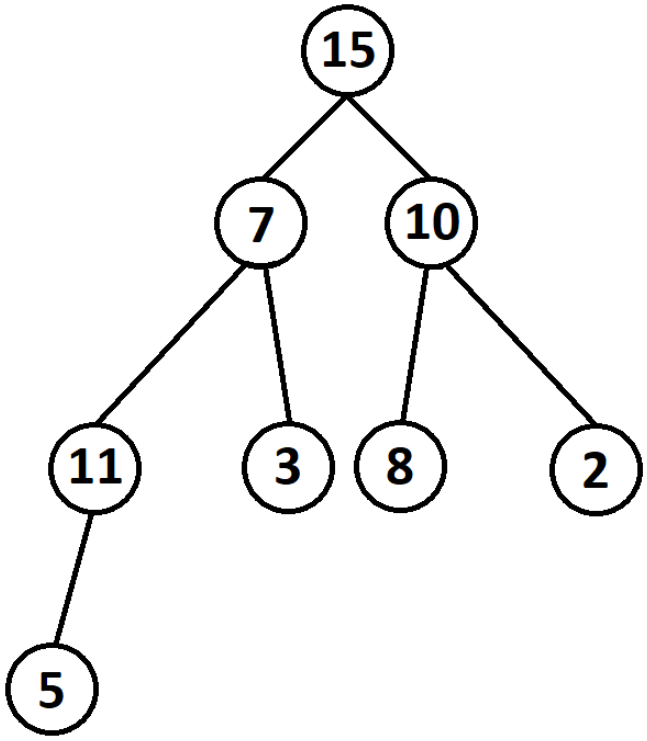
Paso 1: Se dibuja el árbol correspondiente al vector que representa el montículo de máximos.



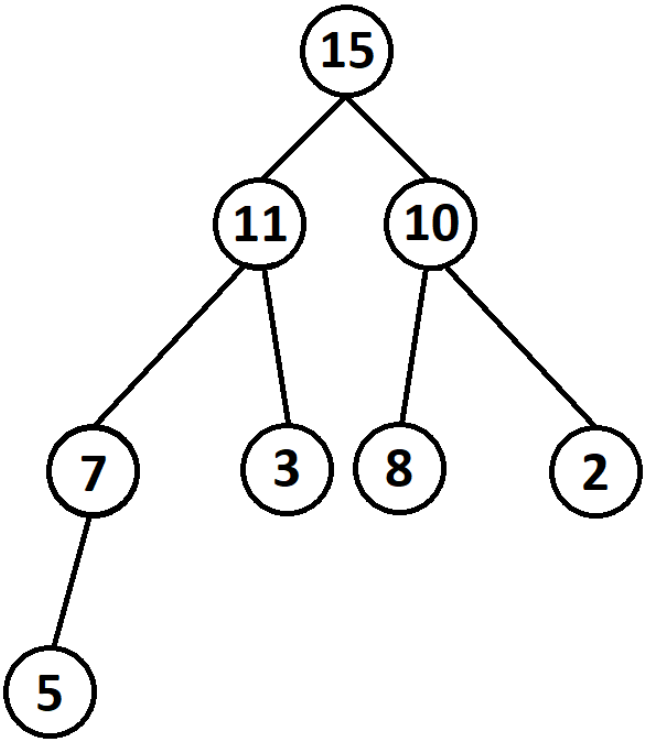
Paso 2: Como 5 es menor que 11, se intercambian dichos nodos el uno por el otro.



Paso 3: Ahora como el nodo 7 es menor que 11, estos se intercambian el uno por el otro.



Paso 4: La raíz 15 es mayor que 11, por lo que no se produce ningún otro cambio. Además de ello, ya es que hemos llegado a la raíz.



Finalmente, el vector solución sería: [15, 11, 10, 7, 3, 8, 2, 5]

Respuesta B)

Problema (4 puntos). Teseo se adentra en el laberinto en busca de un minotauro que no sabe dónde está. Se trata de implementar una función *ariadna* que le ayude a encontrar el minotauro y a salir después del laberinto. El laberinto debe representarse como una matriz de entrada a la función cuyas casillas contienen uno de los siguientes tres valores: 0 para "camino libre", 1 para "pared" (no se puede ocupar) y 2 para "minotauro". Teseo sale de la casilla (1,1) y debe encontrar la casilla ocupada por el minotauro y salir posteriormente del laberinto deshaciendo el camino recorrido. En cada punto, Teseo puede tomar la dirección Norte, Sur, Este u Oeste siempre que no haya una pared. La función *ariadna* debe devolver la secuencia de casillas que componen el camino de regreso desde la casilla ocupada por el minotauro hasta la casilla (1,1).

La resolución de este problema debe incluir, por este orden:

1. Elección del esquema más apropiado de entre los siguientes: vuelta atrás, divide y vencerás, programación dinámica y ramificación y poda. Escriba la estructura general de dicho esquema e indique cómo se aplica al problema (0,5 puntos).

Como no se indica nada al respecto de la distancia entre casillas adyacentes, y ya que se sugiere utilizar únicamente una matriz, es lícito suponer que la distancia entre casillas adyacentes es siempre la misma (1, sin pérdida de generalidad).

Por otra parte, no se exige hallar el camino más corto entre la entrada y el minotauro. Tras estas consideraciones previas, ya es posible elegir el esquema algorítmico más adecuado.

El tablero puede verse como un grafo en el que los nodos son las casillas y en el que como máximo surgen cuatro aristas (N, S, E, O). Todas las aristas tienen el mismo valor asociado (por ejemplo, 1).

Como no es necesario encontrar el camino más corto, sino encontrar uno cualquiera, una búsqueda en profundidad resultará más adecuada que una búsqueda en anchura.

Es posible que una casilla no tenga salida, por lo que es necesario habilitar un mecanismo de retroceso.

Por último, es necesario que no se exploren por segunda vez casillas ya exploradas anteriormente.

Por los motivos expuestos, se ha elegido el esquema de Vuelta Atrás, cuyo esquema general viene descrito en la página 161 del Libro de Texto base de la asignatura.

```
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).

Para almacenar las casillas bastará con un registro de dos enteros, x e y. Vamos a emplear una lista de casillas para almacenar la solución y otra para las compleciones.

Para llevar el control de los visitados bastará con una matriz de igual tamaño que el laberinto, pero de valores booleanos. Será necesario implementar las funciones de lista:

- crear_lista
- vacia
- añadir
- primero

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

```
tipoCasilla = registro
                  x, y: entero;
              fregistro
tipoLista
fun vuelta-atrás (laberinto: vector [1..LARGO, 1..ANCHO] de entero;
                           casilla: tipoCasilla
                           visitados: vector [1..LARGO, 1..ANCHO] de booleano;
                   ) dev (es_solución: booleano; solución: tipoLista)
visitados [casilla.x, casilla.y] ← verdadero;
si laberinto[casilla.x, casilla.y] == 2 entonces
        solución ← crear_lista();
        solución ← añadir (solución, casilla);
        devolver (verdero, solución);
si no
        hijos ← crear_lista();
        hijos ← compleciones (laberinto, casilla)
        es_solución ← falso;
        mientras ¬ es_solución ^ ¬ vacia (hijos)
                hijo ← primero (hijos)
                si ¬ visitados [hijo.x, hijo.y] entonces
                        (es_solución, solución) ← vuelta-atrás (laberinto,hijo,visitados);
                fsi
        fmientras
        si es_solución entonces
                solución ← añadir (solución, ensayo);
        fsi
        devolver (es_solución, solución);
fsi
ffun
```

En el caso de encontrar al minotauro, se detiene la exploración en profundidad, y al deshacer las llamadas recursivas, se van añadiendo a la solución las casillas que se han recorrido.

Como se añaden al final de la lista, la primera será la del minotauro, y la última casilla (1, 1), tal como pedía el enunciado. La función compleciones comprobará que la casilla no es una pared y que no está fuera del laberinto.

```
fun compleciones (laberinto: vector [1..LARGO, 1..ANCHO] de entero;
                  casilla: tipoCasilla) dev tipoLista
    hijos crear_lista();
    si casilla.x+1 <= LARGO entonces
        si laberinto[casilla.x+1,casilla.y] <> 1 entonces
            casilla_aux.x=casilla.x+1;
            casilla_aux.y=casilla.y;
            hijos añadir (solución, casilla_aux);
        fsi
    fsi
    si casilla.x-1 >= 1 entonces
        si laberinto[casilla.x-1,casilla.y] <> 1 entonces
            casilla_aux.x=casilla.x-1;
            casilla_aux.y=casilla.y;
            hijos añadir (solución, casilla_aux);
        fsi
    fsi
    si casilla.y+1 <= ANCHO entonces
        si laberinto[casilla.x,casilla.y+1] <> 1 entonces
            casilla_aux.x=casilla.x;
            casilla_aux.y=casilla.y+1;
            hijos añadir (solución, casilla_aux);
        fsi
    fsi
    si casilla.y-1 >= 1 entonces
        si laberinto[casilla.x,casilla.y-1] <> 1 entonces
            casilla_aux.x=casilla.x;
            casilla_aux.y=casilla.y-1;
            hijos añadir (solución, casilla_aux);
        fsi
    fsi
ffun
```

4. Estudio del coste del algoritmo desarrollado (0.5 puntos solo si el punto 1 es correcto).

El espacio de búsqueda del problema es un árbol en el que cada nodo da lugar como máximo a tres ramas correspondientes a las cuatro direcciones de búsqueda, menos la casilla de la que procede el recorrido.

El número de niveles del árbol es el número de casillas del tablero, ANCHOxLARGO. Luego, una cota superior es $3^{ANCHOxLARGO}$.