Estructura de datos y de la información
Boletín de problemas - Tema 10

1. En el caso de que sea posible, dar un ejemplo de los siguientes puntos. Si no, explicar por qué no lo es. Considerar un valor genérico de n > 1.

   a) Un árbol binario con n nodos cuya secuencia en Preorden y en Inorden coincida.

   b) Un árbol binario con n nodos cuya secuencia en Postorden y en Inorden coincida.

   c) Un árbol binario con n nodos cuya secuencia en Preorden y en Postorden coincida.

2. Cada una de las siguientes sentencias es verdadera o falsa. Si es verdadera explica por qué lo es, y si es falsa construye un ejemplo que lo demuestre.

   a) X es descendiente de Y si, y solo si, Y precede a X en Preorden y X precede a Y en Postorden.

   b) X es descendiente de Y si, y solo si, Y precede a X en Preorden y X precede a Y en Inorden.

   c) Si X e Y son dos nodos hoja, su orden relativo es el mismo en:

      1) Preorden e Inorden
      2) Inorden y Postorden
      3) Preorden y Postorden

3. Considerar las siete operaciones siguientes:

   - Procesar la raíz.
   - Procesar el hijo izquierdo.
   - Procesar el hijo derecho.
   - Recorrer el subárbol izquierdo del hijo izquierdo.
   - Recorrer el subárbol derecho del hijo izquierdo.
   - Recorrer el subárbol izquierdo del hijo derecho.
   - Recorrer el subárbol derecho del hijo derecho.

   a) Mostrar tres formas de ordenar estas siete operaciones de manera que se correspondan con las tres formas naturales de recorrer un árbol binario: preorden, inorden y postorden.

   b) Otra manera de recorrer un árbol binario puede ser en orden de niveles, de la siguiente manera:

```
si el árbol no es vacío entonces
  nivel = 1
  mientras nivel sea menor o igual a la profundidad del árbol
    procesar todos los nodos del nivel actual
    nivel = nivel + 1
```

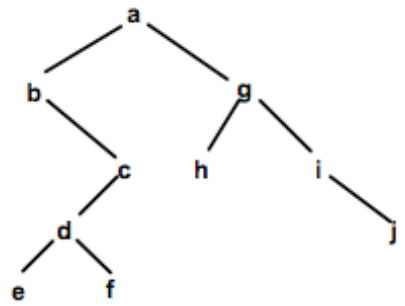
Si es posible, representar este algoritmo con las siete operaciones anteriores, y si no lo es explicar por qué.

4. Considerar el siguiente algoritmo para recorrer un árbol binario en preorden:

```
void preorden(const arbin &ab) {
  if (!ab.EsVacio()) {
    cout << ab.DatoRaiz() << endl;
    preorden(ab.Izquierdo());
    preorden(ab.Derecho());
  }
}
```

a) Modificar el procedimiento anterior para que en la secuencia de nodos en preorden que produce, cada nodo aparezca junto con información que permita al usuario reconstruir el árbol binario inicial. Esta información consistirá en unas etiquetas que indiquen si cada nodo tiene hijo izquierdo, derecho, ambos o ninguno.

b) Ejecutar el algoritmo modificado sobre el árbol siguiente, indicando la secuencia de nodos resultante junto con sus etiquetas.



c) Escribir un algoritmo que tomando como entrada una secuencia en preorden de nodos etiquetados, genere y devuelva el correspondiente árbol binario. Utilizar la estructura de datos vista en clase para representar árboles binarios. Suponer que la secuencia de nodos etiquetados que se toma como entrada consiste en un vector de registros, donde cada registro almacena el dato de un nodo más dos campos booleanos para indicar la existencia de sus hijos.

5. Utilizando una estructura de punteros como la vista en clase para representar un árbol binario, y añadiendo un campo booleano a los nodos del árbol para marcarlos, implementar los siguientes algoritmos:

   a) Diseñar un algoritmo que devuelva la altura de un árbol binario que toma como entrada.

   b) Diseñar un algoritmo que tome como entrada un árbol binario y su altura y que marque aquel de sus nodos que se encuentre a profundidad máxima y lo más a la izquierda posible en el árbol. Llamáremos a este nodo `np`.

   c) Diseñar un algoritmo que tome como entrada un árbol binario con su nodo `np` marcado y que marque todos los nodos del camino desde la raíz hasta el nodo `np`.

6. Suponer que se dispone de una función que devuelve la altura de un árbol binario:

   ```
   int altura(const arbin &ab);
   ```

   Utilizando esta función y la estructura de punteros vista en clase para representar un árbol binario, implementar los algoritmos siguientes:

   a) `bool equilibrado(const arbin &ab);` Dice si el árbol binario es equilibrado.

   b) `bool completo(const arbin &ab);` Dice si el árbol binario es completo.

   c) `bool extendido(const arbin &ab);` Dice si el árbol binario es extendido.

7. Implementar un algoritmo que busque un elemento en un árbol binario de búsqueda.

8. La siguiente estructura de datos se puede utilizar para representar los árboles binarios que cumplen la propiedad de que los nodos que tienen hijo derecho también tienen hijo izquierdo.

```
template <class T>
class arbin2 {
  public:
    // Operaciones
  private:
    class nodo {
      T info;
      arbin<T> hijo_izq; // Apunta al hijo izquierdo del nodo
      arbin<T> her_der; // Apunta al hermano derecho del nodo
      // Constructor del nodo
```

```
    };
   nodo *raiz;
};
```

a) Dibujar un árbol binario cualquiera utilizando dicha estructura de datos.

b) Implementar los algoritmos necesarios para que tomando como entrada un árbol binario construido con dicha estructura de datos, indiquen si es equilibrado o no.

9. Una posible representación para un árbol binario completo podría ser por medio de un puntero a hijo izquierdo y otro puntero a hermano derecho para cada nodo.

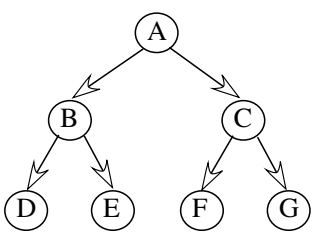
   a) Definir una estructura de datos para representar un árbol binario completo de la forma descrita.

   b) Implementar un algoritmo que tomando como entrada un árbol binario construido con dicha estructura de datos indique si es completo.

   c) Implementar un algoritmo que utilizando dicha estructura de datos, tome como entrada un árbol binario completo o casi completo y un dato, e inserte en el árbol un nuevo nodo con el dato. La inserción debe ser hecha de tal forma que el árbol construido sea siempre completo o casi completo.

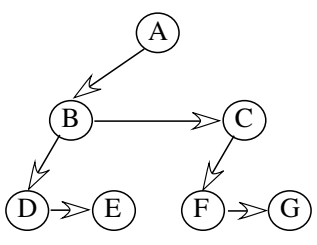
   d) Implementar los algoritmos que utilizando dicha estructura de datos, visualicen los nodos de un árbol binario completo en preorden, inorden y postorden.

10. Supongamos que tenemos las dos estructuras siguientes para almacenar un árbol binario completo. En la estructura i) se introducen los nodos dentro de un vector en secuencia de preorden, T significa true y F significa false indicando si el nodo tiene hijos. La estructura ii) utiliza punteros a hijo izquierdo y hermano derecho para representar el mismo árbol.



	nodo	hijos
0	A	T
1	B	T
2	D	F
3	E	F
4	C	T
5	F	F
6	G	F

i)



ii)

a) Diseñar dos estructuras de datos para almacenar árboles binarios según las dos figuras anteriores i) e ii).

b) Utilizando estas estructuras de datos, implementar un algoritmo que tome como entrada un árbol representado con la estructura estática i) y lo almacene en la estructura dinámica ii).

11. Se define un árbol de Fibonacci de orden n como un árbol binario que cumple una de las siguientes propiedades:

    - El árbol vacío es un árbol de Fibonacci de orden 0.
    - Un árbol con un único nodo es un árbol de Fibonacci de orden 1
    - Un árbol de Fibonacci de orden n > 1 consta de una raíz, un subárbol izquierdo que es un árbol de Fibonacci de orden n - 1 y un subárbol derecho que es un árbol de Fibonacci de orden n - 2.

    a) Escribir un algoritmo recursivo, tal que, dado un árbol binario, compruebe si es o no un árbol de Fibonacci y que en caso afirmativo, devuelva su orden.

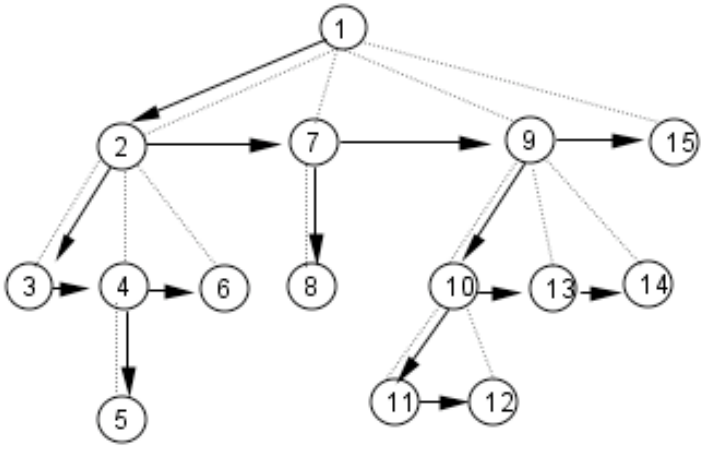
    b) ¿Cuál será la altura de un árbol de Fibonacci de orden n? Definirlo para todos los casos posibles.

12. Sea un árbol binario que almacena en sus nodos pesos, siendo un peso un número entero entre 1 y 10. El peso total de un árbol se define como la suma de los pesos de todos sus nodos. Se dice que un árbol binario es ponderado si para todo nodo se cumple que el peso total de sus dos subárboles no difiere en más de 10 unidades. Implementar un algoritmo que indique si un árbol binario es ponderado o no.

13. Teniendo en cuenta las siguientes definiciones y suponiendo que se dispone del procedimiento que calcula la altura de un árbol binario, implementar en todos los procedimientos necesarios para que dado un árbol binario construido con la estructura de datos dinámica vista en clase, calcular si es inclinado hacia la izquierda, inclinado hacia la derecha o ninguna de las dos cosas.

    - Definición 1: Un árbol binario es inclinado hacia la izquierda si para cada nodo que no es una hoja se cumple que la altura su subárbol izquierdo es mayor que la altura de su subárbol derecho. Considerar que los nodos hoja sí son inclinados hacia la izquierda.
    - Definición 2: Un árbol binario es inclinado hacia la derecha si para cada nodo que no es una hoja se cumple que la altura su subárbol derecho es mayor que la altura de su subárbol izquierdo. Considerar que los nodos hoja sí son inclinados hacia la derecha.

14. La siguiente figura representa un árbol no binario con un número indeterminado de nodos y donde cada nodo puede tener un número indeterminado de hijos. Las líneas discontinuas representan las relaciones padre-hijo, mientras que las flechas representan los punteros necesarios para su representación con una estructura de datos



a) Definir una estructura de datos con punteros para almacenar un árbol no binario de una forma similar a la indicada en la figura.

b) Implementar un algoritmo en pseudocódigo que tome como entrada un árbol implementado con la estructura de datos anterior y recorra sus nodos en preorden, de la forma indicada por la numeración de los nodos de la figura.

15. *a*) Dibujar un árbol binario de búsqueda con esta secuencia ordenada de letras y considerando el orden de las 28 letras del alfabeto:

`N, S, F, Q, D, J, X, C, U, E, L, Z, O, K`

b) Escribir la secuencia de letras que resulta de recorrer el árbol anterior en preorden, inorden y postorden.

c) Representar el árbol construido con una estructura de datos secuencial y utilizando el recorrido en inorden del árbol para insertar los nodos. Define la estructura de datos utilizada.

d) Diseñar dos algoritmos de alto nivel para borrar el nodo raíz de un árbol binario de búsqueda y ejecutarlos sobre el árbol anterior.

e) ¿Cuál de los dos algoritmos produce un árbol AVL?

16. *a*) Suponer que se tiene que implementar el funcionamiento de un conjunto de claves únicas que tienen que ser individualmente localizadas y accedidas por una aplicación en un orden aleatorio. El conjunto de claves va a sufrir muchas modificaciones (altas y bajas). Comentar las ventajas y desventajas de usar un árbol binario de búsqueda y de usar un AVL para implementar dicho conjunto.

b) Dibujar los árboles AVL que se van generando al insertar la siguiente secuencia de nodos en el orden indicado y señalar las rotaciones simples y dobles (RS, RD).

`22, 15, 20, 11, 30, 33, 27, 7, 12, 3, 40, 37`

17. Dado un árbol de búsqueda AVL,

a) Escribir un algoritmo que inserte un elemento en dicho árbol y que devuelva el puntero al nodo pivote. Si no existe el nodo pivote, devolverá NULL.

b) Suponiendo que se ha insertado un dato determinado, escribir un algoritmo que actualice los balances del camino de búsqueda desde el nodo que recibe como parámetro hasta el nodo insertado. Utilizar el siguiente prototipo de función:
`void actualizar_balances(nodoptr p, const T & dato);`

c) Escribir una función que dado el nodo pivote y un dato insertado, indique qué tipo de rotación (simple o doble) hay que realizar para reequilibrar el árbol, o si no es necesaria ninguna rotación.

18. Los siguientes procedimientos toman como entrada un vector de enteros y lo transforman en un montículo almacenado en el mismo vector.

```
const int N=100;
typedef int mont[N];
// Reconstruye el montículo a si su i-ésimo elemento está mal colocado
void reconstruir (mont a, int i, tam) {
  int mayor, izq, der;
  izq= 2 * i + 1;  // Hijo izquierdo de i
  der= 2 * i + 2;  // Hijo derecho de i
  if ((izq <= tam) && (a[izq] > a[i]))
      mayor= izq
  else mayor= i;
  if ((der <= tam) && (a[der] > a[mayor]))
      mayor = der;
  if (mayor != i) {
    intercambia(a, i, mayor);
    reconstruir(a, mayor, tam); // Llamada recursiva
    }
}
```

```
// Construye un montículo con los elementos del vector a
void crear-monticulo(mont a) {
  int  j, tam;
  tam= longitud_vector(a); //numero de elementos del vector
  for (j = tam/2; j>=0; --j)
       reconstruir(a, j, tam);
}
```

a) ¿Cuál es el efecto de ejecutar `reconstruir(a, i, tam)` cuando el elemento de `i` es mayor que sus hijos?

b) ¿Por qué se realiza la llamada recursiva `reconstruir(a, mayor, tam)`?

c) ¿Por qué se comprueba que el valor que toman las variables `izq` y `der` es menor o igual al tamaño del montículo? ¿Puede darse el caso de que sean mayores?

d) ¿Cuál sería el efecto de ejecutar `reconstruir(a, i, tam)` cuando `i` fuera mayor que el valor de `tam/2`?

e) ¿Qué sucedería si el bucle `for` fuera ascendente en lugar de descendente?

f) Modificar el algoritmo `reconstruir` para que no sea recursivo.

19. Un montículo *min-max* es una estructura de datos idéntica a la del montículo, pero con la propiedad de montículo que dice que para todo nodo X se cumple:

- a profundidad par, el valor de X es menor que el de su padre pero mayor que el de su abuelo (si existe),
- a profundidad impar, el valor de X es mayor que el de su padre pero menor que el de su abuelo (si existe).

Suponer que la raíz de un árbol está en el nivel 0, y por lo tanto a profundidad par.

a) ¿Cómo se encuentran los elementos mínimo y máximo en un montículo *min-max*?

b) Insertar los siguientes elementos en un montículo *min-max* en el mismo orden en que están a continuación. Dibujar los árboles resultantes de insertar cada uno de los elementos.

`3, 7, 5, 4, 2, 6, 1, 0, 10, 11, 9, 13, 8, 15, 12, 14`

c) Eliminar del árbol resultante del ejercicio anterior los elementos 0 y 15 en este orden. Mostrar los dos árboles resultantes.

20. A partir de los siguientes datos de entrada, aplicar el algoritmo de Huffman para asignar un código a cada uno de los caracteres de entrada. Dibujar el árbol resultante e indicar cuál sería el código para cada uno de los caracteres.

carácter	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
frecuencia	10	5	3	2	8	8	10	13	7	8	6	2	5	3	6	4

21. ¿Es posible obtener dos árboles de Huffman diferentes para un conjunto de elementos de entrada con sus pesos asignados (y sin modificarlos)?. Probar que sólo existe un árbol o dar un ejemplo de dos árboles diferentes.

22. Tomando como entrada un carácter y un árbol de Huffman previamente construido y utilizando la estructura de datos dinámica vista en clase para los árboles binarios, escribir una función que devuelva una cadena con el código binario asociado a dicho carácter de entrada según el árbol de Huffman.

23. Se dispone de un árbol de Huffman ya construido y de una cadena de unos y ceros donde se han codificado un conjunto de caracteres. Escribir una función que devuelva la secuencia de caracteres que han sido codificados.

24. Dibujar el árbol binario asociado con el siguiente esquema de codificación:

```
a = 00, b = 111, c = 0011, d = 01, e = 0, f = 1
```

    Razonar por qué el código no es unívocamente decodificable.

25. Construir un B-árbol de orden 2 siguiendo las siguientes pautas:

    a) Dibujar el B-árbol resultante de insertar la secuencia:

    ```
    30, 16, 55, 25, 2
    ```

    b) Dibujar el B-árbol resultante de insertar en el árbol del apartado a) la secuencia:

    ```
    44, 15, 22, 29, 33, 31
    ```

    c) Dibujar el B-árbol resultante de insertar en el árbol del apartado b) la secuencia:

    ```
    42, 18, 52, 35
    ```

    d) Dibujar el B-árbol resultante de insertar en el árbol del apartado c) la secuencia:

    ```
    10, 34, 38, 43
    ```

e) Dibujar el B-árbol resultante de borrar del árbol del apartado d) el número `29`.

f) Dibujar el B-árbol resultante de borrar del árbol del apartado e) el número `42`.

g) Dibujar el B-árbol resultante de borrar del árbol del apartado f) el número `30`.

26. Construir un B-árbol de orden 2 siguiendo las siguientes pautas:

a) Dibujar el B-árbol resultante de insertar la secuencia:

```
9, 26, 12, 2
```

b) Dibujar el B-árbol resultante de insertar en el árbol del apartado a) la secuencia:

```
22, 7, 11, 13, 18, 15, 20
```

c) Dibujar el B-árbol resultante de insertar en el árbol del apartado b) la secuencia:

```
8, 25, 17
```

d) Dibujar el B-árbol resultante de insertar en el árbol del apartado c) la secuencia:

```
5, 16, 19, 14, 23
```

e) Dibujar el B-árbol resultante de borrar del árbol del apartado d) el número `20`.

f) Dibujar el B-árbol resultante de borrar del árbol del apartado e) el número `13`.

g) Dibujar el B-árbol resultante de borrar del árbol del apartado f) el número `22`.