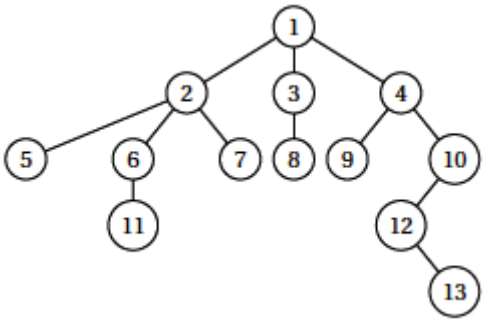
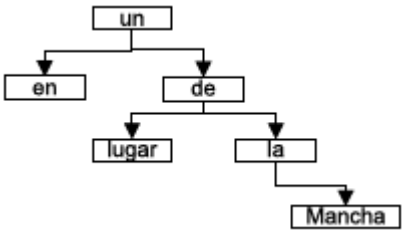
EXÁMENES SEGUNDO PARCIAL

1. Dado el siguiente árbol general con números almacenados en sus nodos, escribir la secuencia de elementos que corresponde a un recorrido en preorden y a un recorrido en postorden del árbol:



2. Dado el árbol binario de la figura siguiente, indique una secuencia de operaciones en Java que permita crearlo y, además:



(a) el resultado de recorrer el árbol en preorden.

(b) el resultado de recorrer el árbol en inorden.

(c) el resultado de recorrer el árbol en postorden.

3. Sea q una cola con prioridad implementada mediante un montículo. Las claves son números enteros y los valores son caracteres. La implementación de la cola utiliza un vector (array) de tamaño 5 para almacenar el montículo. Inicialmente q está vacía y a continuación se realiza la siguiente secuencia de operaciones:

```
q.enqueue(4,'B');
q.enqueue(7,'I');
q.enqueue(3,'A');
q.enqueue(1,'D');
q.dequeue();
```

(a) Dibujar el árbol (casi)completo del montículo resultante.
(b) Dibujar el contenido del vector resultante.

4. Implementar en Java el método `public static void aumentarPrecio (Map<String,Double> precios, String prod, Double coste)` que, dada una tabla de precios `precios`, un producto `prod` y un coste `coste`, debe aumentar en dicha tabla el precio del producto teniendo en cuenta lo siguiente: (1) si el producto ya está en la tabla aumentar su precio con el valor de coste; (2) si el producto no está en la tabla, se establece como precio del producto el valor de `coste`. La clave del map es el identificador del producto y su valor asociado es el precio. Por ejemplo, el map precios = <"1",10>,<"3",13> indica que tenemos dos productos, uno con identificador "1" y precio 10 y otro producto con identificador "3" y precio 13.
   Por ejemplo, dado precios=[<"1",10>,<"3",13>], la llamada a aumentarPrecio(precios,"1",2) debe dejar en el map precios=[<"1",12>,<"3",13>].
   O dado, precios = [<"2",14>, <"8",17>] la llamada aumentarPrecio(precios, "5", 12) deber dejar precios=[<"2",14>, <"8",17>, <"5",12>].

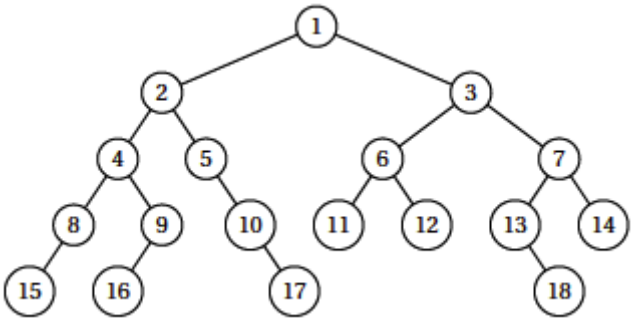
5. Se pretende implementar en Java el método `getVerticesAlcanzables` que, dado un grafo g y un vértice n, devuelve el conjunto de vértices alcanzables desde n, es decir, que se puede encontrar un camino en el grafo desde n hasta dichos vértices. Se proporciona la siguiente parte del código:

```
1 static <V,E> Set<Vertex<V>> getVerticesAlcanzables (UndirectedGraph<V, E> g,
2                                                      Vertex<V> n) {
3     Set<Vertex<V>> visited = new HashTableMapSet<V>();
4     visited.add(n);
5     getVerticesAlcanzablesRec(g,n,visited);
6     return visited;
7 }
8 static <V,E> void getVerticesAlcanzablesRec  (UndirectedGraph<V, E> g,
9                                                Vertex<V> n,
10                                               Set<Vertex<V>> visited ) {
11
12    // COMPLETAR ESTE METODO
13 }
```

   Completar el código del método `getVerticesAlcanzablesRec` para que implemente la funcionalidad indicada. **NOTA**: Para añadir elementos al conjunto `visited` podéis usar el método `visited.add(n)`, que añade el vértice n al conjunto `visited`.

6. Implementar en Java un método con la cabecera
   `public static boolean hasHeapPropertyGen(Tree<Integer> tree)`
   Queremos implementar un método en Java que compruebe si un árbol general cumple la propiedad que caracteriza a los montículos (*heaps*): *El elemento contenido en cada nodo es mayor o igual que el elemento contenido en su padre, si tiene padre*.

7. Dado el siguiente árbol binario en el que sólo se muestran los elementos en los nodos:



Indicar el tipo de recorrido (preorden, inorden, postorden, arbitrario) de los siguientes recorridos que muestran los elementos en los nodos:

(a) 15 8 16 9 4 17 10 5 2 11 12 6 18 13 14 7 3 1
(b) 15 8 4 16 9 5 10 17 2 1 11 6 12 3 18 13 7 14

8. La siguiente figura muestra el contenido del vector de una implementaci n mediante mont culo de una cola con prioridad.

null	**(2,X)**	**(4,B)**	**(3,Y)**	**(7,A)**	**null**	**null**	**null**

(a) Dibujar el árbol (casi) completo representado por el vector.
(b) Dar la secuencia de entradas que se obtendrán al invocar el método dequeue() consecutivamente hasta vaciar la cola con prioridad.

9. Implementar en Java el método `public Map<Integer,Integer> contarPracticas (PositionList<Alumno> entregas)` que recibe una lista con las entregas de los alumnos y devuelve un objeto de tipo `Map<Integer,Integer>` cuya clave será el DNI del alumno y cuyo valor será el número de entregas realizadas por dicho alumno. La lista entregas no contendrá elementos **null** ni los atributos de los alumnos contenidos en la lista serán **null**. Se dispone de la clase `HashTableMap<K,V>` que implementa el interfaz `Map<K,V>` y que cuenta con el constructor `HashTableMap<K,V>()` para crear un `Map` vacío.
Por ejemplo, dada lista=[Alumno(4,"a"),Alumno(2,"b"),Alumno(0,"c"),Alumno(4,"a")], el método contarPracticas(lista) devolverá un Map con los siguientes pares <clave,valor>: <2,1>, <0,1>, <4,2>, que indica que el alumno con DNI 2 ha entregado 1 práctica, que el alumno con DNI 0 ha entregado 1 práctica y que el alumno con DNI 4 ha entregado 2 prácticas.

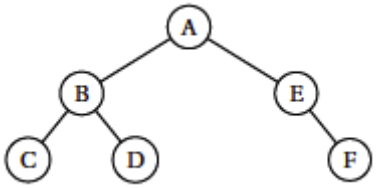
10. En un árbol binario cuyos nodos contienen enteros no positivos, definimos el valor de un camino como la suma de los valores contenidos en los nodos que constituyen ese camino. Implementar en Java el método: `static Integer maximoCamino (BinaryTree<Integer> tree)` que recibe como parámetro el árbol binario `tree`, cuyos nodos contienen enteros positivos, y devuelve el valor del camino que empieza en la raíz y cuyo valor es máximo. Si `tree` es **null**, el método debe lanzar la excepción `IllegalArgumentException`. Los nodos de `tree` no contienen elementos **null**. **Nota**: aunque pueden generarse caminos y elegir aquel con valor máximo, no es necesario hacerlo para resolver el problema.

11. Se pretende implementar en Java el método `public static <V,E> boolean isReachable (UndirectedGraph<V, E> g, Vertex<V> from, Vertex<V> to)` que devuelve `true` si desde el vértice `from` se puede alcanzar el vértice `to`, y `false` en caso contrario. Nos proporcionan el siguiente código erróneo para resolver este problema:

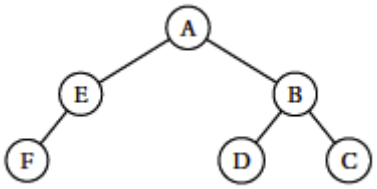
```
1 public static <V,E> boolean isReachable (UndirectedGraph<V, E> g,
2                                          Vertex<V> from,
3                                          Vertex<V> to) {
4     Set<Vertex<V>> visited = new HashTableMapSet<Vertex<V>>();
5     return isReachable(g, from, to, visited);
6 }
7
8 public static <V,E> boolean isReachable (UndirectedGraph<V, E> g,
9                                          Vertex<V> from,
10                                         Vertex<V> to,
11                                         Set<Vertex<V>> visited ) {
12
13    if (from == to) {
14          return false;
15    }
16
17    visited.add(from);
18    boolean reachable = false;
19    Iterator<Edge<E>> it = g.edges(from).iterator();
20    while (it.hasNext()) {
21          Vertex<V> other = g.opposite(from, it.next());
22          if (!visited.contains(other)) {
23                isReachable(g, other, to, visited);
24          }
25    }
26    return reachable;
27 }
```

Determinar qué cambios son necesarios para que el código devuelva el resultado correcto, no se lance ninguna excepción y sea más eficiente, evitando realizar operaciones innecesarias. Para contestar debéis indicar el número de línea en el que está el problema y cómo quedaría la línea para resolverlo.

12. Dado el siguiente árbol binario:

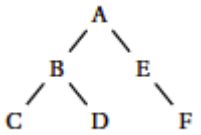


el árbol espejo de dicho árbol es el árbol obtiene visualmente dando la vuelta a esta hoja del examen, viéndose como a través de un espejo:



Implementar en Java el método `public <E> BinaryTree<E> espejo(BinaryTree<E> t)` que toma un árbol binario t y devuelve su árbol espejo como resultado.

13. Implementar en Java el método `public NodePositionList<E> breadth(BinaryTree<E> tree)` que devuelve una lista de posiciones con los elementos en los nodos del rbol seg n un recorrido en anchura (por niveles) del mismo. El recorrido en anchura de un rbol visita los nodos por niveles, primero el nivel 0 (la raíz), luego el nivel 1 (los hijos de la raíz de izquierda a derecha), luego el nivel 2, etc. Por ejemplo, la invocación de breadth sobre el árbol de la figura debe devolver la lista con los elementos A,B,E,C,D,F en ese orden de izquierda a derecha.



14. Decimos que un map es la inversa de otro cuando todos los elementos de los pares <k,v> del original aparecen en el inverso pero con v como nueva clave y k como nuevo valor – es decir, como pares <v,k>. Para que un map sea invertible, todos sus valores deben ser distintos y no nulos. Implementar en Java el método: static `Map<String,String> invertir (Map<String,String> map)` que recibe como parámetro un `Map<String,String>` invertible y devuelve el map inverso.
Por ejemplo, dado map = [<"k1","v1">,<"k2","v2">,<"k3","v3">], el método debe devolver un nuevo map con los pares [<"v1","k1">,<"v2","k2">,<"v3","k3">].

15. La siguiente figura muestra el contenido del vector de un mont culo. Se muestran las claves almacenadas, obviando los valores:

5	**6**	**10**	**7**	**11**	**11**		

(a) Dibujar el árbol (casi) completo que forma el montículo:
(b) Dibujar el árbol resultante de invocar dequeue dos veces consecutivas sobre el árbol anterior.
(c) Dibujar el vector correspondiente al árbol de la respuesta anterior.

16. Dado el siguiente código:

```
public static void desconocido (String s) {
  Map<Character,Integer> map = new HashTableMap<Character,Integer>();
  for(int i = 0; i < s.length(); i ++) {
    Integer n = map.get(s.charAt(i));
    if (n == null) {
      map.put(s.charAt(i),1);
    }
    else {
      map.put(s.charAt(i),n+1);
    }
  }
  Iterator<Entry<Character,Integer>> it = map.entries();
  while (it.hasNext()) {
    Entry<Character,Integer> entry = it.next();
    System.out.println("(" + entry.getKey() + " - " + entry.getValue() + "), ");
  }
}
```

Mostrar una posible salida por consola de la ejecuci n del siguiente m todo si se ejecuta la siguiente llamada: `desconocido("imprimo?");` **NOTA**: Recordad que el método s.charAt(i) devuelve el carácter que ocupa la posición i en s (siendo s un String).