Algoritmos y Estructura de Datos: Examen 2

Grado en Ingeniería Informática y Grado en Matemáticas e Informática
Departamento de Lenguajes, Sistemas Informáticos e Ingeniería de Software

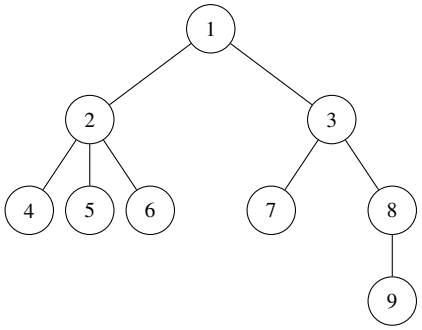
- **Se debe contestar en el espacio disponible después de cada pregunta**
- Este examen dura **100 minutos** y consta de **?? preguntas** que puntúan hasta **?? puntos**.
- Las calificaciones provisionales de este examen se publicarán el **22 de Enero de 2019** en el Moodle de la asignatura junto con la fecha y lugar de la revisión.

(3 puntos) 1. **Se pide:** Implementar en Java el método

```
static <E> void imprimirCaminosHojas(Tree<E> tree)
```

que recibe como parámetro un árbol `tree` e imprime todos los caminos del árbol que llevan desde la raiz hasta los nodos hoja (externos) del árbol, imprimiendo tanto la raiz como el nodo hoja. Los caminos que llevan desde la raiz a nodos internos no deben ser impresos. El árbol `tree` no será `null` y no contendrá elementos `null`. Para imprimir podéis usar `System.out.println(...)`.

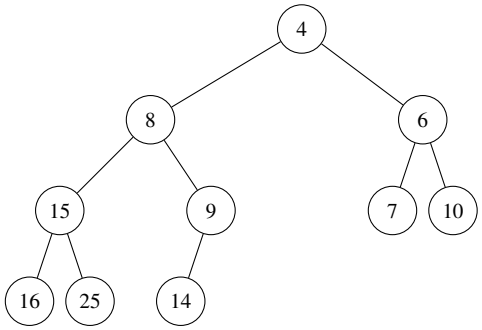
Por ejemplo, dado el siguiente árbol `tree`, el método `imprimirCaminosHojas(tree)` debe imprimir:




```
> imprimirCaminosHojas(tree);

124
125
126
137
1389
```

(2 puntos) 2. Dado el siguiente montículo (*heap*)



Se pide: Dibujar el estado final del montículo después de ejecutar las siguientes operaciones sobre dicho montículo.

(a) `enqueue(5,_)`

(b) `deqeue()`

NOTA: **NO** apliquéis `dequeue` sobre el montículo resultante de ejecutar `enqueue(5,_)`. Tanto `dequeue(5,_)` como `dequeue` se aplican sobre el montículo tal como está en el dibujo.

(2½ puntos) 3. **Se pide:** Implementar en Java el método

```
public static Map<Character,Integer> contarApariciones(String texto)
```

que devuelve un `Map<Character,Integer>` cuyas claves serán cada uno de los caracteres que aparecen en el parámetro `texto` y el valor asociado a cada clave será el número de veces que aparece el carácter en `texto`. El parámetro `texto` no será `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.

NOTA: La clase `String` dispone del método `length()` para obtener el número de caracteres del `String` y el método `char charAt(int i)` que devuelve el carácter que ocupa la posición `i` del `String` siendo 0 la primera posición y `length()-1` la última posición válida.

Por ejemplo, dado el `texto = "En un lugar de La Mancha"` la llamada al método `contarApariciones(texto)` debe devolver un `Map` que contenga los siguiente pares (clave,valor): ('h',1), (' ',5), ('M',1), ('u',2), ('E',1), ('e',1), ('r',1), ('g',1), ('l',1), ('d',1), ('L',1), ('a',4), ('n',3), ('c',1), que indica que, por ejemplo, el carácter 'n' aparece 3 veces en `texto` o que el carácter 'e' aparece 1 vez.

(2½ puntos) 4. 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 }
```

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