Estructuras de Datos Avanzadas
Examen Extraordinario

27 de junio de 2023

Normas generales:

- La duración del examen es de 2 horas.
- Desde el campus virtual hay que descargar un archivo ZIP que contiene el **esqueleto** de los tres ejercicios del examen así como los test unitarios correspondientes. Se recomienda crear un proyecto de ApacheNetbeans vacío; copiar dentro de la carpeta del proyecto la carpeta src y la carpeta test proporcionadas; añadir las librerías JUnit y Hamcrest; y finalmente cerrar y volver a abrir ApacheNetbeans.

Ejercicio 1 [2 puntos]

Se desea implementar un iterador que, dado un árbol, itere exclusivamente por aquellos nodos que sean “hijos únicos”. Es decir, que permita recorrer únicamente los nodos sin hermanos del árbol dado. Se recuerda a los estudiantes que la raíz de un árbol no tiene hermanos.

NOTA: La clase `WithoutSiblingIterator.java`, donde se debe implementar el iterador descrito, se encuentra en el paquete `src.material.tree`. Las pruebas para comprobar el correcto funcionamiento del iterador se encuentran en el fichero `WithoutSiblingIteratorTest.java` del paquete `test.material.tree`. Está prohibido modificar de cualquier forma las cabeceras de los métodos suministrados en el proyecto. Se valorará tanto el correcto funcionamiento de la solución como su simplicidad y eficiencia.

Ejercicio 2 [2 puntos]

Se desea implementar un árbol ternario. El árbol ternario es similar al árbol binario, pero el número máximo de hijos que puede tener cada nodo en un árbol ternario es tres (en un árbol binario, el número máximo de hijos por nodo es dos). Al igual que sucede en el árbol binario, en el árbol ternario cada nodo distingue a sus hijos por un nombre. En particular, cada nodo puede tener un hijo izquierdo, un hijo medio y un hijo derecho. En el código proporcionado, nos referiremos a estos hijos con los términos en inglés: left, middle y right.

En el paquete `tree` se ha creado un nuevo paquete `ternaryTree` que contiene la interfaz `TernaryTree` y la clase `LinkedTernaryTree`. Se pide al estudiante finalizar la clase `LinkedTernaryTree`. Los métodos que debe programar son los siguientes:

- `public Position<E> middle (Position<E> v)`
- `public boolean hasMiddle (Position<E> v)`
- `public boolean isInternal (Position<E> v)`

- `public boolean isLeaf (Position<E> v)`
- `public Iterable<? extends Position<E>> sibling (Position<E> p)`
- `public Position<E> insertMiddle (Position<E> p, E e)t`
- `public E remove (Position<E> p)`
- `public Iterable<? extends Position<E>> children (Position<E> v)`

Comentarios adicionales:

- En relación al método `public Iterable<? extends Position<E>> sibling (Position<E> p)`: Se espera que los hermanos de un nodo se devuelvan en orden de izquierda a derecha. Es decir, al recorrer los hermanos (siblings) de un nodo, se espera obtener primero el nodo izquierdo, en segundo lugar, el nodo medio y finalmente el nodo derecho. Por supuesto, ese es el orden ideal, pero no quiere decir que se tengan que obtener los tres nodos, puesto que un nodo solo puede tener dos hermanos como máximo.
- En relación al método `public E remove (Position<E> p)`: Solo se puede eliminar un nodo si podemos recolocar a todos sus hijos como hijos del padre del nodo que vamos a eliminar. Es decir, supongamos que queremos eliminar al nodo Homer, cuyo padre es Abraham y cuyos hijos son Bart, Lisa y Maggie. Entonces, solo podríamos eliminar a Homer si hay espacio para hacer que Bart, Lisa y Maggie pasen a ser hijos de Abraham.

Ejercicio 3 [6 puntos]

La empresa SocialURJC desea crear una red social sencilla. En ella se podrán dar dealta personas usando un nombre de usuario único (no puede existir otro usuario con el mismo nombre). Una vez registrado el usuario, podrá seguir a otros usuarios sin necesidad de que estos aprueben su solicitud de seguimiento. Consultar los seguidores de un usuario. Consultar a quién está siguiendo un usuario. Recibir recomendaciones de a quién seguir y saber si alguien es un bot.

Para facilitar la tarea al estudiante se proporcionan las clases `RedSocial` y la clase `Persona` que se encuentran en el paquete examen. Se pide:

a) **[0,5 puntos]** Escoger las estructuras de datos más adecuadas para la implementación de la clase `RedSocial`.

b) **[0,25 puntos]** Implementar el método `newProfile` que recibe como parámetro un nombre de usuario. Si el nombre de usuario ya está registrado el método devuelve `null` (indicando que no ha sido posible crear el nuevo perfil). En caso de no existir se crea el nuevo perfil, se registra en la red social y se devuelve el perfil.

c) **[0,25 puntos]** Implementar el método `wantToBeFollower` que recibe como parámetros dos perfiles de usuario. El primero es el usuario que desea seguir al segundo usuario. Si alguno de los dos perfiles no está registrado en la red social el método devuelve `false`.

d) **[0,25 puntos]** Implementar el método `followers` que recibe como parámetro el perfil de un usuario y devuelve una `Collection` de perfiles en las que están los seguidores de este usuario.

e) **[0,25 puntos]** Implementar el método `following` que recibe como parámetro el perfil de un usuario y devuelve una `Collection` de perfiles a los que sigue este usuario.

f) **[2 puntos]** Implementar el método `suggestions` que dado el perfil de un usuario devuelve una `Collection` de usuarios que siguen perfiles que dicho usuario sigue. Pero en esa lista no pueden existir usuarios que ya sean seguidos por el usuario.

g) **[0,5 puntos]** Implementar el método `suspiciousBot`. Este método devuelve una `Collection` con todos los perfiles que se sospecha pueden ser *bots*. Un perfil será sospechoso de ser *bot* si no tiene seguidores.

h) **[2 puntos]** Implementar el método `cleanBots`, este método elimina todos los perfiles que considere que son *bots* y los devuelve en una `Collection` de perfiles. Este método debe aplicarse a la red hasta que no queden perfiles que puedan considerarse *bots*. Tenga en cuenta que puede que un perfil tenga muchos seguidores, pero todos sean *bots*, al eliminar todos sus seguidores-bots se quedaría sin seguidores y pasaría a ser considerado *bot*.

No se permite modificar las cabeceras de las clases ni de los métodos previamente definidos. Sí se permite implementar métodos privados y clases adicionales.