Estructuras de Datos Avanzadas
Examen convocatoria de septiembre
8 de enero de 2024

- La duración del examen es de 120 minutos.
- Desde el campus virtual hay que descargar un archivo ZIP que contiene el esqueleto de los ejercicios del examen así como los test unitarios correspondientes. Se recomienda crear un proyecto de Netbeans vacío; copiar dentro de la carpeta del proyecto la carpeta src y la carpeta test proporcionadas; añadir las librerías JUnit 4 y Hamcrest; y finalmente cerrar y volver a abrir Netbeans.
- El alumno debe generar un fichero ZIP con el directorio en el que esté el código del alumno y subirá el fichero ZIP a la actividad examen práctico del aula virtual asegurándose de que entrega el código desarrollado durante la prueba.

Nombre y Apellidos: _____

DNI: _____ Hora de entrega: _____ Firma:

Ejercicio 1 [2 punto]

En este ejercicio se pide implementar el método `merge` que encontrará en la clase `AdditionalFeatures` del paquete `material.tree.BinarySearchTree`. Este método recibe dos árboles binarios de búsqueda y devuelve una ED `Iterable` que contiene los elementos, en orden creciente, de los dos árboles pasados como parámetro. El método no podrá tener una complejidad superior a O(n) siendo n el número de elementos de la ED `Iterable`.

Ejercicio 2 [1,5 puntos]

En este ejercicio se pide implementar método `removeHalfNodes`, que encontrará en la clase `FunHandling` del paquete `material.tree.BinaryTree`. Este método recibe un árbol binario y lo modifica eliminando todos los nodos que tenían un único hijo. En la Figura 1 se representa un árbol original que se le pasaría al método y en la Figura 2 podemos observar la transformación que sufriría.

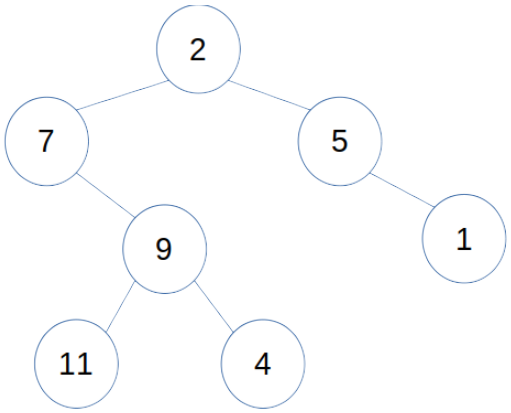

Figura 1: Árbol original

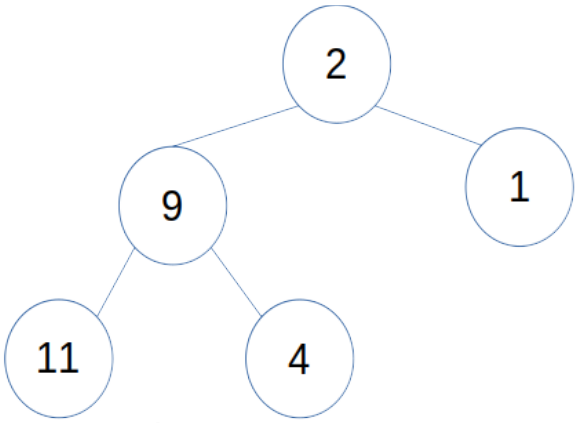

Figura 2: Árbol modificado

Ejercicio 3. Red P2P [6,5 puntos]

En un sistema P2P-TCP cada nodo actúa como servidor y cómo cliente. El modo cliente es sencillo, puesto que toda la responsabilidad recae en el servidor, el cliente solo se preocupa de solicitar la conexión al servidor. El servidor es un poco más complejo, puesto que debe recordar la información de cada cliente, es decir, su dirección IP y el puerto desde el que se ha establecido la conexión.

Se desea representar un servicio P2P-TCP empleando las ED necesarias. Cada par ejecutará un servidor que solo podrá aceptar N conexiones como máximo, además solo aceptará una conexión por cada dirección IP. Para ayudarse en esta tarea se deberá implementar la clase `ConnectionManager` que permite al servidor gestionar el número de conexiones que mantiene establecidas en cada momento. Esta clase, está incluida en la clase `Node` y será configurada para

aceptar un máximo N de conexiones con clientes, impidiendo a los clientes establecer más de una conexión desde la misma dirección IP.

En la primera parte de ejercicio se debe implementar la clase `ConnectionManager`:

a)**[0,25 puntos]** Seleccionar la estructura de datos que minimice la complejidad de las operaciones de las que consta la clase `ConnectionManager`. Implementar el método constructor que recibe como parámetro el número máximo de conexiones que podrá aceptar el servidor, la dirección IP del servidor y el puerto del servidor.

b)**[0,5 puntos]** Implementar el método `newConnection`, que recibe la IP y el puerto del host cliente como parámetros de entrada. La conexión puede aceptarse, si el servidor no ha alcanzado el número máximo de conexiones permitidas y no existe ninguna conexión activa desde la dirección IP del cliente. Si la conexión se acepta se devolverá un valor booleano verdadero. En caso contrario se devolverá falso, indicando así que el servidor ha rechazado la solicitud de conexión.

c)**[0,25 puntos]** Implementar el método `closeConnection`, que recibe la IP y el puerto del host cliente como parámetros de entrada. Si la conexión existe se cierra y se devuelve un valor booleano verdadero. Si la conexión no existía se devuelve el valor booleano falso.

d)**[0,25 puntos]** Implementar el método `closeAllConnection`, que no recibe ningún parámetro y cierra todas las conexiones activas que tenga el servidor en ese momento. Devuelve un valor entero indicando cuantas conexiones se han cerrado.

e)**[0,25 puntos]** Implementar el método `numberOfConnection`, que no recibe ningún parámetro de entrada. Devuelve un valor entero que indica el número de conexiones activas que tiene el servidor en ese momento.

Por otro lado, para poder establecer una Red P2P se debe implementar la clase `netP2P`. El constructor de la clase recibe un superservidor. Este servidor tiene de límite 100 conexiones y todos los nodos de la red se conectan a él para obtener la información necesaria para conectarse a otros servidores de la red (nunca habrá más de 100 pares en la red).

Para añadir un nuevo nodo a la red, este se conectará como cliente al superservidor. El superservidor lo añadirá a la base de datos de la red. La conexión entre el superservidor y el nodo se mantendrá activa mientras el nodo esté activo en la red. Además, el superservidor informa al nodo de qué otros nodos forman parte de la red. Con esta información, el nuevo nodo intentará iniciar conexión como cliente con dichos nodos (aunque es probable que solo unos pocos le permitan conectarse, recuerde que los servidores solo pueden aceptar un número máximo de conexiones). El superservidor también facilitará la información de este nuevo nodo al resto de nodos de la red, que intentarán conectarse a él como clientes.

Se pide:

f)**[0,25 puntos]** Seleccionar la estructura de datos que minimice la complejidad de las operaciones de las que consta la clase `netP2P`. Implementar el método constructor.

g)**[1,5 punto]** Implementar el método `newNode`, que añade un nuevo nodo a la red P2P. Este nuevo nodo tendrá un servidor de la clase `ConnectionManager`. El nodo, que para formar parte de la red debe conectarse como cliente al superservidor, recibirá información de los otros nodos que ya forman parte de la red e intentará conectarse como cliente con todos ellos. Los otros nodos también tratarán de conectarse a él como clientes. La complejidad de esta operación será como máximo de O(n).

h)**[1 punto]** Implementar el método `closeClient`**.** Este método recibe cómo parámetros el nodo cliente y el nodo servidor y si existe la conexión entre el cliente y el servidor se cierra.

i)**[0,25 puntos]** Implementar el método `shutDownServer`. Este método recibe cómo parámetro un nodo de la red y detiene su servidor (eliminando todas las conexiones), aunque el nodo siga activo como cliente en la red.

j)**[1 punto]** Implementar el método `completeShutDown`**.** Este método recibe un nodo de la red y lo elimina completamente, deja de formar parte de la red P2P.

k) **[0,25 puntos]** Implementar el método `clientServer` que devuelve una ED `Collection` de nodos de la red que actúan como cliente-servidor

l)**[0,25 puntos]** Implementar el método `servers` que devuelve una ED `Collection` con todos los nodos de la red que actúan solo como servidores

m)**[0,25 puntos]** Implementar el método `clients` que devuelve una ED `Collection` con todos los nodos de la red que actúan solo como clientes.

n)**[0,25 puntos]** Implementar el método `everyPair` que devuelve el número de nodos de la red.

NOTA: Se facilita al estudiante la clase `Node`, que implementa los nodos de la red. En la clase `ConnectionManager` se podrán agregar métodos adicionales si se considera oportuno. Sin embargo, se prohíbe modificar de cualquier forma las cabeceras de los métodos suministrados en el paquete `examen` incluido en el proyecto.