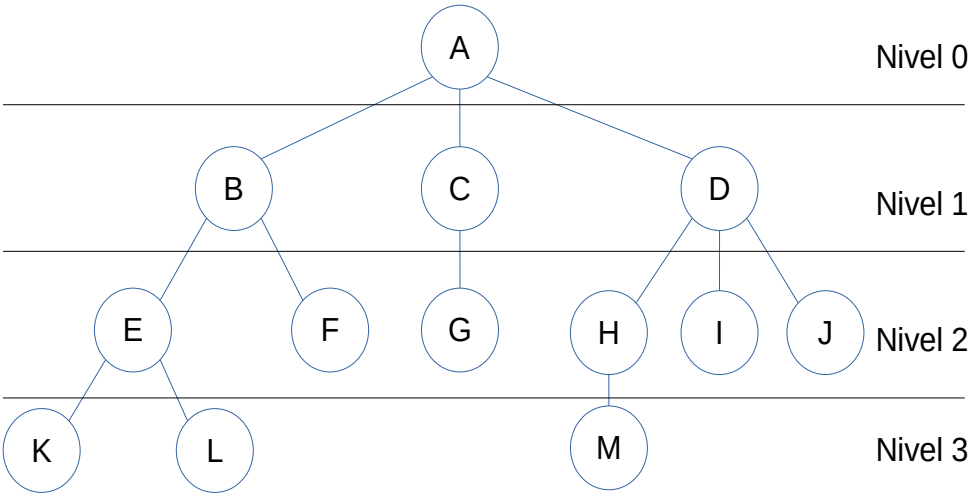
Estructuras de Datos Avanzadas
Examen Extraordinario
27 de junio de 2022

- 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 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.

Ejercicio 1 [1 punto]

Implemente un iterador por niveles, `LevelIterator`, que recibe como parámetros el árbol y el nivel del árbol que se desea recorrer. En caso de instanciar el iterador con un árbol nulo, un árbol vacío o con un nivel negativo se lanzará una excepción. Si el árbol no tiene suficiente profundidad el iterador estará vacío.

En la figura puede verse un árbol y marcado con líneas los diferentes niveles, de tal forma que si se instancia el `LevelIterator` sobre dicho árbol y le pedimos el nivel 2, recorreremos los nodos E, F, G, H, I y J.



NOTA: Se valorará un buen diseño (de clases y métodos) para obtener la máxima calificación. Las clase `LevelIterator` se encuentra en el paquete `material.tree`

Ejercicio 2 [2 puntos]

Se desea implementar la clase `ConnectionManager` que permita gestionar a un servidor web el número de conexiones que mantiene establecidas en cada momento. Esta clase se iniciará al arrancarse el servicio y será configurada para aceptar un máximo N de conexiones en paralelo desde un mismo host. Se recuerda a los estudiantes que una conexión TCP está caracterizada por la dirección IP del host cliente, la dirección IP del host servidor, el puerto origen y el puerto del servidor.

Es importante remarcar, de cara al diseño de la clase, que tanto la dirección IP del servidor como su puerto son fijos.

Se pide:

a)**[0,5 puntos]** Seleccionar la estructura de datos que minimice la complejidad de las operaciones de las que consta la clase. Implementar el método constructor.

b)**[0,5 puntos]** Implementar el método `newConnection`, que recibe la IP y el puerto del host cliente como parámetros de entrada. Si la conexión puede aceptarse, el número de conexiones con el host < N y no existe previamente, se devolverá un valor booleano verdadero. En caso contrario se devolverá falso, indicando a sí al servidor que rechace dicha solicitud de conexión.

b)**[0,25 puntos]** Implementar el método `closeConnection`, que recibe la IP y el puerto del host cliente como parámetros de entrada. Se cierra la conexión parametrizada por la IP y el puerto.

c)**[0,25 puntos]** Implementar el método `closeAllConnection`, que recibe la IP del host cliente como parámetro de entrada. Devuelve un valor entero indicando cuantas conexiones conexiones asociadas a esa IP se han cerrado.

d)**[0,25 puntos]** Implementar el método `numberOfConnection`, que recibe la IP del host cliente como parámetro de entrada. Devuelve un valor entero que indica el número de conexiones activas que hay asociadas con esa IP.

e) **[0,25 puntos]** Implementar el método `connectionInformation`, que recibe la IP del host cliente como parámetro de entrada. Devuelve una estructura de datos Iterable con todos los puertos activos de dicho cliente. En caso de que el cliente no tenga ninguna conexión activa la ED se devuelve vacía.

NOTA: 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 `examenJunio` incluido en el proyecto.

Ejercicio 3. El juego de las luces [4 puntos]

Se desea implementar una clase que sirva de base para un juego de dos jugadores, concretamente el juego de las luces. Este juego consiste en un plano de luces conectadas unas a otras. Cada vez que un jugador pulsa sobre una de ellas esta cambia de estado, si estaba apagada se enciende y si estaba encendida se apaga. Pero no lo hace sola, también modifica el estado de las luces a las que está conectada haciendo que se enciendan o apaguen dependiendo de su estado inicial. Además, si una luz está encendida y también lo están todas las luces con las que está conectada, se elimina del tablero. Lo mismo sucede si la luz está apagada y todas las que están conectadas a ella también lo están.

El jugador 1 gana si consigue que todas las luces estén apagadas y el jugador 2 gana si consigue que todas las luces estén encendidas.

a) **[0,5 puntos]** Dada la clase `playOfLight`, escoja las estructuras de datos más adecuadas para su implementación, e inicie el juego cargando la situación inicial del tablero, para ello se le pasarán dos listas, la primera con las luces y la segunda con pares de luces que están conectadas.

b) **[1 punto]** Implemente el método `changeTheState` que recibe una luz y si está encendida la apaga y si está apagada la enciende. También modifica el estado de las luces con las que esté conectada directamente. Tiene un ejemplo de funcionamiento en las figuras 1, 2, 3 y 4; siendo la situación de entrada del método la figura 1 y la de salida la de la figura 4.

c) **[1 puntos]** Implemente el método `endOfGame` que devuelve un booleano `true` cuando el juego termina. El juego puede terminar si todas las luces están encendidas o apagadas.

d) **[1,25 puntos]** Implemente el método `updateGame` que será invocado después de `chageTheState` para eliminar todas las luces que se pueda. Recuerde que una luz puede desaparecer del tablero cuando tanto ella como todas sus vecinas se encuentran en el mismo estado, es decir, apagadas o encendidas. Vea figuras 5 y 6.

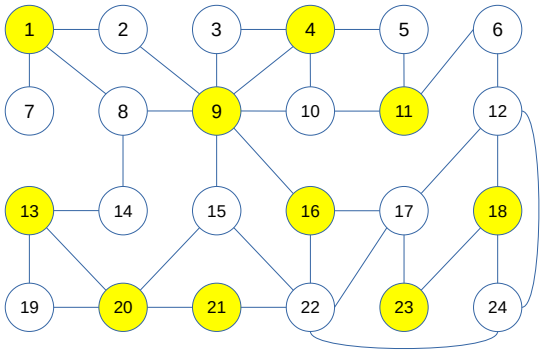


Figura 1: Situación inicial del tablero

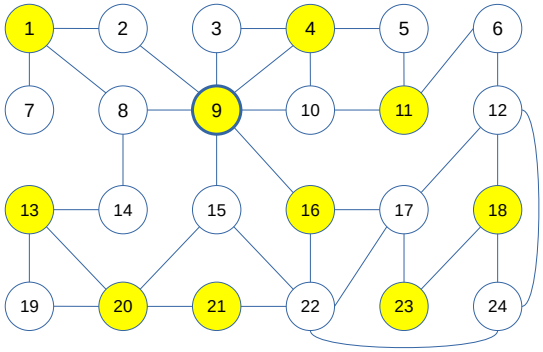


Figura 2: Seleccionamos el nodo 9

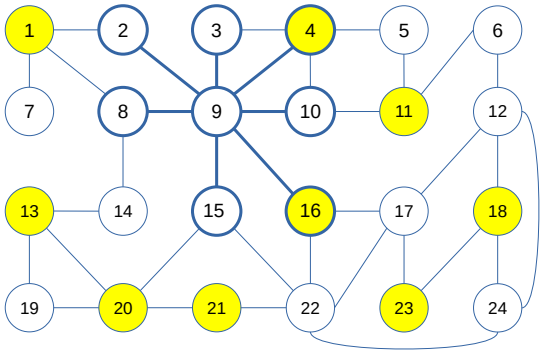


Figura 3: Las luces conectadas con 9 también cambian de estado

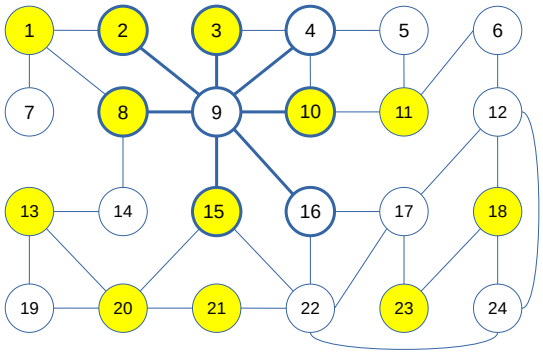


Figura 4: Como queda el tablero después de cambiar la luz 9 y sus vecinas

e)**[0,25 puntos]** Implementar los métodos `lightsOn` y `lightsOff` que devuelven una estructura de datos iterable con las luces que en ese momento están encendidas o apagadas.

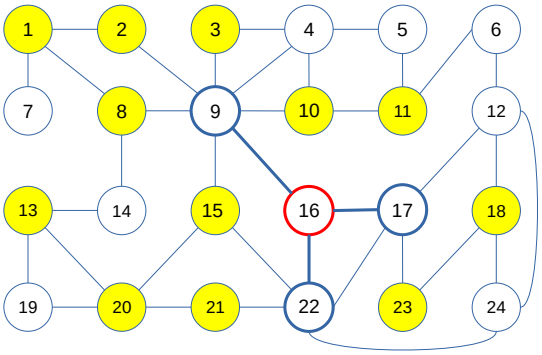


Figura 5: La luz 16 está apagada igual que sus vecinas

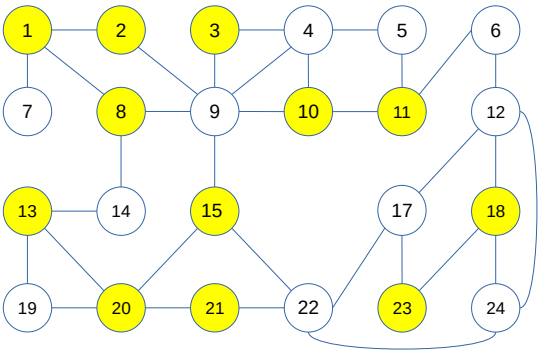


Figura 6: Tablero después de eliminar la luz 16

NOTA: En la clase `PlayOfLight` 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 `examenJunio` incluido en el proyecto. Para facilitar el trabajo del estudiante se proporciona la clase `Light` que no podrá ser modificada.