Primer Parcial de Estructura de Datos y Algoritmos

Ejer 1.a	Ejer 1.b	Ejer 2	Ejer 3	Nota
/2	/3	/4	/1	/10

Duración: 2 horas 10 minutos

Condición Mínima de Aprobación. Deben cumplir la siguiente condición:

Sumar **no menos de cuatro** puntos entre los ejercicios 1 y 2.

Muy Importante

1. **Al terminar el examen deberían subir los siguientes archivos, según lo explicado en los ejercicios: *SortedCompactedList.java*, y Laberinto.java** según lo pedido. No colocar código auxiliar por fuera de estas clases.

2. Todos los códigos que les subimos para este examen pertenecen al **package default. Colocarlos en un package con el nombre del usuario de Uds. en campus. Ej: package lgomez**

Ejercicio 1

Se tiene el código provisto en el archivo ***SortedCompactedList.java*** de una lista **ordenada simplemente enlazada**. En esta implementación **se permiten elementos repetidos** y se guarda **un Node por cada uno de los repetidos por lo que no se encuentra compactada.**

Los métodos provistos son:

- **insert**: inserta ordenadamente el item en la lista
- **remove**: borra el primer item que contenga ese dato. Es importante aclarar que si hay items repetidos únicamente se borra una de las repeticiones y no todas.
- **dump**: imprime a consola todos los items uno por línea
- **dumpNodes**: imprime a consola todos los nodos que determinan la lista.
- **size**: devuelve la cantidad total de items en la lista.

En la implementación provista todos los métodos (salvo dump y dumpNodes) **delegan en los nodos la lógica principal de su funcionamiento**.

Se pide lo siguiente:

1.a) modificar la implementación actual para que la misma tenga el mismo comportamiento que el original salvo porque la representación interna sea **compacta**.
Se entiende por compacta que cuando haya items repetidos los mismos no deben generar nuevos Nodos sino que en los Nodos se debe agregar la cantidad de repeticiones de cada item que contiene la lista.
El único método que tendrá un resultado distinto entre la implementación sin compactar y la implementación compactada será **dumpNodes** dado que en el caso de la implementación sin compactar tiene el mismo comportamiento que **dump** pero en la implementación compactada imprimirá sólo una vez cada valor distinto independientemente del número de repeticiones.

Por ejemplo, al insertar los valores:
hola, tal, ah, veo, ah, bio, ah, veo, ah, tal

en el caso de la implementación original da lugar a la lista donde los nodos son:

ah→ah→ah→ah→bio→hola→tal→tal→veo→veo

mientras que después de modificada la implementación los nodos deben ser los siguientes (entre paréntesis se consigna la cantidad de repeticiones):

ah(4) →bio(1) →hola(1) →tal(2) →veo(2)

Se agregan los métodos Test1 y Test2 con casos de inserción y borrado.

1.b) agregar un iterador de lectura y borrado **a la implementación con Compactación** realizada en el punto **1.a**. Para esto hay, en la implementación original, una clase private ***SortedCompactedListIterator*** sin comportamiento. Se agregan los métodos Test3, Test4 y Test5 con algunos casos de iteración y borrado desde el iterator. Cuando Uds. implementen los métodos del iterador, debe estar andando esos testeos.

Aclaraciones Importantes para ambos puntos:

- **No se pueden agregar miembros de datos a las clases provistas** con la excepción de un contador de tipo int tanto en Node como en SortedCompactedListIterator.
- No se pueden usar otras clases auxiliares ni otras provistas por java con la excepción de clases para lanzar excepciones.
- No se puede modificar el código de dumpNodes

Ejercicio 2

Implementar el método ***existeCamino*** de la clase ***Laberinto.java.*** Este método permite determinar si existe un camino desde una celda de inicio hasta una celda de fin, dentro de un laberinto representado por una matriz de enteros.

```
public class Laberinto{

    public static boolean existeCamino(int[][] laberinto,
                                       int filaInicio, int columnaInicio,
                                       int filaFin, int columnaFin) {}
}
```

Descripción del Laberinto:

- El laberinto es una cuadrícula bidimensional donde cada celda está representada por un entero:
    - 0: Indica un pasillo por el que se puede transitar.
    - 1: Indica una pared, una obstrucción que no se puede atravesar.
- El método recibirá la matriz del laberinto, las coordenadas de la celda de inicio (fila y columna) y las coordenadas de la celda de fin (fila y columna). **Considerar que la fila 0, columna 0 es la celda que se encuentra arriba a la izquierda**.

Objetivo:

Determinar si existe una secuencia de movimientos válidos desde la celda de inicio hasta la celda de fin, moviéndose únicamente a través de las celdas que son pasillos y sin pasar por las paredes.

Consideraciones:

- Debe evitar ciclos infinitos en la búsqueda. Una forma de hacerlo es marcar las celdas por las que ya se ha pasado para no volver a explorarlas innecesariamente. Utilice el valor **-1** en la matriz del laberinto para marcar una celda como visitada.
- El método debe devolver **true** si se encuentra al menos un camino desde el inicio hasta el fin, y **false** en caso contrario.

Direcciones de Movimiento:

Respecto de una celda solo hay 4 movimientos posibles y como se observa, **lo único válido es moverse a una celda adyacente en alguna de estas 4 direcciones:**

- Arriba: (fila - 1, columna)
- Abajo: (fila + 1, columna)
- Izquierda: (fila, columna - 1)
- Derecha: (fila, columna + 1)

Recuerde verificar que los movimientos resultantes siempre estén dentro de los límites del laberinto.

Ejercicio 3

Usando el mismo ejemplo visto en clase con Lucene:
a) mismos 4 archivos (**nombres: a.txt. b.txt c.txt y d.txt**) asociados al campo **content**
b) tanto indexación como búsqueda usan **StandardAnalyzer**

Se pide:

Indicar qué documentos (**solo el nombre de los mismos**) devuelve Lucene al ejecutar cada una de las siguientes consultas con el **QueryParser**. No hace falta indicar el ranking (no ordenar el resultado por ranking). **Explicar claramente** por qué los documentos que indiquen en la segunda columna son los recuperados, caso contrario no se considerará respondido el ejercicio.

3.1)

queryString	**Documentos obtenidos en la respuesta**
"content:IEDO~ AND content:[PI TO SOS] "	

Explicación: …..

3.2)

queryString	**Documentos obtenidos en la respuesta**
"content:\"review game game,\" OR content:\",,store,,game\""	

Explicación: …..