Algoritmos y Estructuras de Datos
1er Semestre 2025
Parcial 1 – Partes 2 y 3

Análisis de Solución

Escenario

Una plataforma de streaming busca mejorar la organización y la búsqueda dentro de su catálogo de películas para optimizar la experiencia de sus usuarios. Para lograrlo, ha contratado a su empresa unipersonal.

Su objetivo es analizar las distintas estructuras de datos vistas en la cursada y seleccionar la(s) más adecuada(s) para resolver el problema de forma eficiente. Es importante considerar que el catálogo se actualiza semanalmente con nuevas películas, por lo que la estructura seleccionada debe permitir inserciones frecuentes sin perder eficiencia

Escenario

Una plataforma de streaming busca mejorar la organización y la búsqueda dentro de su catálogo de películas para optimizar la experiencia de sus usuarios. Para lograrlo, ha contratado a su empresa unipersonal.

Su objetivo es analizar las distintas estructuras de datos vistas en la cursada y seleccionar la(s) más adecuada(s) para resolver el problema de forma eficiente. Es importante considerar que el catálogo se actualiza semanalmente con nuevas películas, por lo que la estructura seleccionada debe permitir inserciones frecuentes sin perder eficiencia

Escenario

Una plataforma de streaming busca mejorar la organización y la búsqueda dentro de su catálogo de películas para optimizar la experiencia de sus usuarios. Para lograrlo, ha contratado a su empresa unipersonal.

Su objetivo es analizar las distintas estructuras de datos vistas en la cursada y seleccionar la(s) más adecuada(s) para resolver el problema de forma eficiente. Es importante considerar que el catálogo se actualiza semanalmente con nuevas películas, por lo que la estructura seleccionada debe permitir inserciones frecuentes sin perder eficiencia

Descripción del Problema

Se debe desarrollar un sistema que permita:

- Almacenar las películas en una estructura de datos eficiente, manteniendo el catálogo ordenado para facilitar búsquedas tanto por puntaje como por título.
- Buscar películas según su puntaje, permitiendo al usuario definir un rango opcional (mínimo y/o máximo).
- Buscar películas por género, ordenando los resultados por año de estreno.
- Incorporar nuevas películas de manera eficiente, preservando las propiedades mencionadas.

Descripción del Problema

Se debe desarrollar un sistema que permita:

- Almacenar las películas en una estructura de datos eficiente, manteniendo el catálogo ordenado para facilitar búsquedas tanto por puntaje como por título.
- Buscar películas según su puntaje, permitiendo al usuario definir un rango opcional (mínimo y/o máximo).
- Buscar películas por género, ordenando los resultados por año de estreno.
- Incorporar nuevas películas de manera eficiente, preservando las propiedades mencionadas.

Objetivos y Requisitos del Sistema

1. **Modelo de Película:**
    a. Título: `String`
    b. Año de estreno: `Int`
    c. Género: `String`
    d. Puntaje: `Float`
2. **Estructura de Datos:**
    a. Se debe especificar y justificar la estructura de datos elegida para almacenar la información.
    b. Se debe proveer un método para insertar una película en el catálogo.

Objetivos y Requisitos del Sistema

1. **Modelo de Película:**
    a. Título: `String`
    b. Año de estreno: `Int`
    c. Género: `String`
    d. Puntaje: `Float`
2. **Estructura de Datos:**
    a. Se debe especificar y justificar la estructura de datos elegida para almacenar la información.
    b. Se debe proveer un método para insertar una película en el catálogo.

Objetivos y Requisitos del Sistema

3. **Búsqueda por Puntaje:**
    a. Implementar una función que devuelva todas las películas cuyo puntaje esté en un rango opcional definido por un mínimo y/o máximo.
        i. Si no se especifica mínimo, incluir todas las películas con puntaje menor o igual al máximo.
        ii. Si no se especifica máximo, incluir todas las películas con puntaje mayor o igual al mínimo.
        iii. Si no se especifica ni mínimo ni máximo, devolver todas las películas.
    b. Los resultados deben estar ordenados por año de estreno (ascendente).

Objetivos y Requisitos del Sistema

3. **Búsqueda por Puntaje:**
   a. Implementar una función que devuelva todas las películas cuyo puntaje esté en un rango opcional definido por un mínimo y/o máximo.
      i. Si no se especifica mínimo, incluir todas las películas con puntaje menor o igual al máximo.
      ii. Si no se especifica máximo, incluir todas las películas con puntaje mayor o igual al mínimo.
      iii. Si no se especifica ni mínimo ni máximo, devolver todas las películas.
   b. Los resultados deben estar ordenados por año de estreno (ascendente).

Objetivos y Requisitos del Sistema

3. **Búsqueda por Puntaje:**
   a. Implementar una función que devuelva todas las películas cuyo puntaje esté en un rango opcional definido por un mínimo y/o máximo.
      i. Si no se especifica mínimo, incluir todas las películas con puntaje menor o igual al máximo.
      ii. Si no se especifica máximo, incluir todas las películas con puntaje mayor o igual al mínimo.
      iii. Si no se especifica ni mínimo ni máximo, devolver todas las películas.
   b. Los resultados deben estar ordenados por año de estreno (ascendente).

Objetivos y Requisitos del Sistema

4. **Búsqueda por Género:**
    a. Implementar una función que devuelva todas las películas de un género específico, ordenadas por año de estreno (ascendente).

Objetivos y Requisitos del Sistema

4. **Búsqueda por Género:**
    a. Implementar una función que devuelva todas las películas de un género específico, ordenadas por año de estreno (ascendente).

Objetivos y Requisitos del Sistema

4. **Búsqueda por Género:**
    a. Implementar una función que devuelva todas las películas de un género específico, ordenadas por año de estreno (ascendente).

Consideraciones Importantes

- La lógica necesaria para resolver cada uno de los problemas debe desarrollarse en detalle, aunque se pueden omitir las implementaciones de métodos internos de los TDAs vistos en clase.
  - TDAs vistos: Lista enlazada simple (con inserción al final), Árbol Binario de Búsqueda, Árbol AVL.
- La eficiencia de los algoritmos será evaluada, priorizando implementaciones con mejor rendimiento.
- Se debe justificar claramente la elección de la(s) estructura(s) de datos utilizada(s).

Consideraciones Importantes

- La lógica necesaria para resolver cada uno de los problemas debe desarrollarse en detalle, aunque se pueden omitir las implementaciones de métodos internos de los TDAs vistos en clase.
  - TDAs vistos: Lista enlazada simple (con inserción al final), Árbol Binario de Búsqueda, Árbol AVL.
- La eficiencia de los algoritmos será evaluada, priorizando implementaciones con mejor rendimiento.
- Se debe justificar claramente la elección de la(s) estructura(s) de datos utilizada(s).

Consideraciones Importantes

- Se deben incluir todos los pasos estándar de desarrollo de algoritmos: Lenguaje natural, precondiciones, postcondiciones, pseudocódigo y análisis del orden de ejecución.
- **La salida de los métodos que buscan datos debe ser mediante una ILista<Pelicula>, correspondiente con el TDALista estándar visto en clase.**

Consideraciones Importantes

- Se deben incluir todos los pasos estándar de desarrollo de algoritmos: Lenguaje natural, precondiciones, postcondiciones, pseudocódigo y análisis del orden de ejecución.
- **La salida de los métodos que buscan datos debe ser mediante una ILista<Pelicula>, correspondiente con el TDALista estándar visto en clase.**

Decisiones

Estructura del catálogo de películas

Estructura del catálogo de películas

- Árbol AVL:
  - Operación optimizada: inserción
- Las operaciones de búsqueda son indiferentes con respecto a la estructura elegida.
  - En el peor caso, las búsquedas deben recorrer toda la estructura.
  - Ejemplo: todas las películas tienen el puntaje dentro del rango elegido.
- ¿Qué atributo usar como clave?

Decisiones

Clave de los nodos del árbol

Clave de los nodos del árbol

- **Requerimiento:** catálogo ordenado para facilitar búsquedas tanto por puntaje como por título.
- Se necesita una clave compuesta por ambos valores
  1. Nuevo tipo de datos con ambos valores (Ej.: “ClavePelicula”).
  2. Normalización lexicográfica.

Opción 2: Normalización lexicográfica

- **Problema:** el orden lexicográfico de la representación directa en String de los números no preserva el orden numérico.
  - Esto evita que se pueda formar una clave compuesta por la concatenación directa del puntaje y el título de la película.

Opción 2: Normalización lexicográfica

```java
// Numeric values
float a = 2.0f;
float b = 10.0f;

// Numeric comparison
System.out.println(Float.compare(a, b));
// Output: -1 → 2.0 < 10.0 (Correct numeric comparison)

// Convert to direct strings (not padded)
String aStr = Float. toString(a); // "2.0"
String bStr = Float. toString(b); // "10.0"
System.out.println(aStr.compareTo(bStr));
// Output: 1 → "2.0" < "10.0" (WRONG lexicographically - due to string comparison rules)
```

Opción 2: Normalización lexicográfica

- **Problema:** el orden lexicográfico de la representación directa en String de los números no preserva el orden numérico.
- **Solución:** Transformar el dato numérico antes de concatenarlo con el título de la película.
  - **Usar una transformación de largo fijo.**
  - Se debe asumir que los puntajes cumplen con las siguientes propiedades:
    - Rango de valores: 0.0 a 10.0
    - Cantidad fija de números decimales (ejemplo: 2 decimales)
  - El nombre formal de la técnica es **codificación de cadena canónica para ordenación**

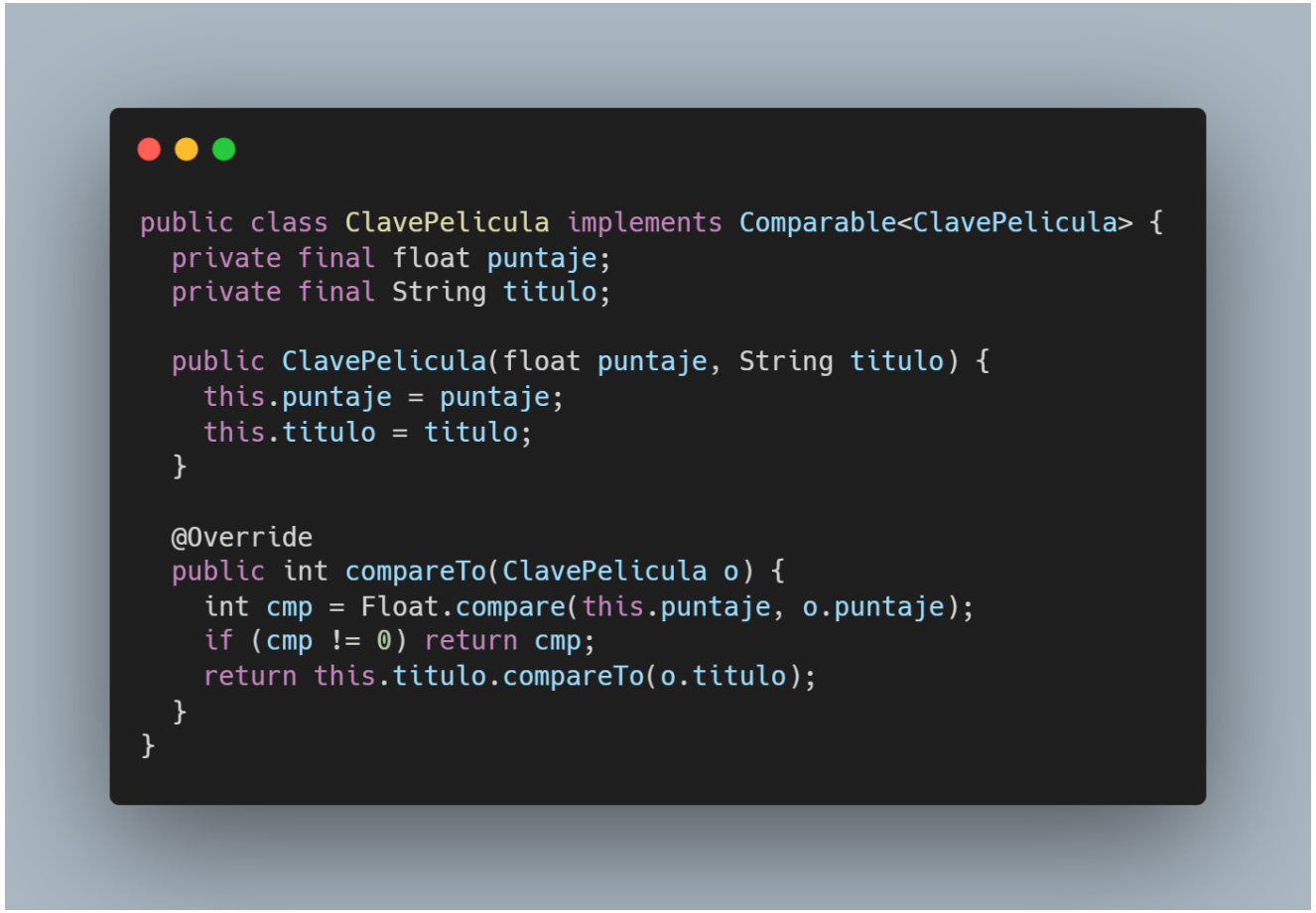
Opción 2: Normalización lexicográfica

```java
// Numeric values
float a = 2.0f;
float b = 10.0f;

// Numeric comparison
System.out.println(Float.compare(a, b));
// Output: -1 → 2.0 < 10.0 (Correct numeric comparison)

// Fixed-width formatted strings with zero-padding
String aFormatted = String.format("%05.2f", a);  // "02.00"
String bFormatted = String.format("%05.2f", b);  // "10.00"

// Lexicographic comparison of formatted strings
System.out.println(aFormatted.compareTo(bFormatted));
// Output: -1 → "02.00" < "10.00" (CORRECT lexicographically – matches numeric)
```

Opción 1: Nuevo tipo de datos con ambos valores

- **Crear una nueva clase que sea *Comparable*** que combine los valores del puntaje y el título.
- Implementar correctamente el método ***compareTo*** para respetar la precedencia de los valores.
  - Primero comparar por puntaje
  - Si el puntaje es igual, comparar por título

Opción 1: Nuevo tipo de datos con ambos valores



```java
public class ClavePelicula implements Comparable<ClavePelicula> {
  private final float puntaje;
  private final String titulo;

  public ClavePelicula(float puntaje, String titulo) {
    this.puntaje = puntaje;
    this.titulo = titulo;
  }

  @Override
  public int compareTo(ClavePelicula o) {
    int cmp = Float.compare(this.puntaje, o.puntaje);
    if (cmp != 0) return cmp;
    return this.titulo.compareTo(o.titulo);
  }
}
```

Opción 1: Nuevo tipo de datos con ambos valores

```java
/**
 * Clase que representa un catálogo de películas utilizando un árbol AVL.
 * Permite insertar películas y realizar búsquedas por puntaje y género.
 */
public class CatalogoPeliculasAVL implements CatalogoPeliculas {
  private ArbolPeliculasAVL arbolPorPuntaje;

  public CatalogoPeliculasAVL() {
    this.arbolPorPuntaje = new ArbolPeliculasAVL();
  }

  @Override
  public void insertarPelicula(Pelicula pelicula) {
    ClavePelicula clave = new ClavePelicula(pelicula.getPuntaje(), pelicula.getTitulo());
    arbolPorPuntaje.insertar(clave, pelicula);
  }
}
```

Clave de los nodos del árbol

- **Requerimiento:** catálogo ordenado para facilitar búsquedas tanto por puntaje como por título.
- Se necesita una clave compuesta por ambos valores
  1. Nuevo tipo de datos con ambos valores (Ej.: “ClavePelicula”).
  2. Normalización lexicográfica.
- **Beneficio**: no hay que modificar el árbol AVL visto en clase, ya que ambas alternativas representan la clave mediante una instancia que es *Comparable*.

Decisiones

Lista para la salida de las búsquedas

Lista para la salida de las búsquedas

- No se puede utilizar una árbol binario de búsqueda por limitación del problema.
  - Esta sería una estructura ideal porque es auto ordenable.
  - La clave debería ser el año de estreno.
- Se debe implementar una ListaOrdenada
  - Es un derivado del TDALista estándar visto en clase.
  - Se debe sobrescribir el método insertar para insertar en forma ordenada los valores.

Lista para la salida de las búsquedas

```java
public class ListaOrdenada<T> extends Lista<T> {
    @Override
    public boolean insertar(T dato, Comparable clave) {
        Nodo<T> nuevo = new Nodo<>(clave, dato);
        if (primero == null || primero.etiqueta.compareTo(clave) > 0) {
            nuevo.siguiente = primero;
            primero = nuevo;
            return true;
        }
        Nodo<T> actual = primero;
        while (actual.siguiente != null && actual.siguiente.etiqueta.compareTo(clave) <= 0) {
            actual = actual.siguiente;
        }
        nuevo.siguiente = actual.siguiente;
        actual.siguiente = nuevo;
        return true;
    }
}
```

Búsquedas

Algoritmos de búsquedas a implementar

Búsqueda por género – en árbol

```java
public class ArbolPeliculasAVL extends ArbolAVL<Pelicula> {
    /**
     * Método que busca películas por genero en el árbol.
     * @param genero Género de la película a buscar.
     * @return Lista ordenada de películas que cumplen con el género
     */
    ListaOrdenada<Pelicula> buscarPorGenero(String genero) {
        ListaOrdenada<Pelicula> resultado = new ListaOrdenada<Pelicula>();

        if (raiz != null) {
            ((NodoPelicula) raiz).buscarPorGenero(genero, resultado);
        }

        return resultado;
    }
}
```

Búsqueda por género – en nodo

```java
/**
 * Clase interna que representa un nodo de película en el árbol.
 * Extiende la clase ElementoAB para almacenar películas y permite búsquedas específicas.
 */
private class NodoPelicula extends ElementoAB<Pelicula> {
    /**
     * Método que busca películas por género en el árbol.
     * @param genero Género de la película a buscar.
     * @param resultado Lista ordenada de películas que cumplen con el género.
     */
    public void buscarPorGenero(String genero, ListaOrdenada<Pelicula> resultado) {
        if (getHijoIzq() != null)
            ((NodoPelicula) getHijoIzq()).buscarPorGenero(genero, resultado);
        if (getDatos().getGenero().equalsIgnoreCase(genero)) {
            resultado.insertar(getDatos(), getDatos().getAnio());
        }
        if (getHijoDer() != null)
            ((NodoPelicula) getHijoDer()).buscarPorGenero(genero, resultado);
    }
}
```

Búsqueda por rango de puntajes – en árbol

```java
public class ArbolPeliculasAVL extends ArbolAVL<Pelicula> {
    /**
     * Método que busca películas por puntaje en el árbol.
     * @param minimo Puntaje mínimo (incluido).
     * @param maximo Puntaje máximo (incluido).
     * @return Lista ordenada de películas que cumplen con el rango de puntaje.
     */
    ListaOrdenada<Pelicula> buscarPorPuntaje(Float minimo, Float maximo) {
        ListaOrdenada<Pelicula> resultado = new ListaOrdenada<Pelicula>();

        if (raiz != null) {
            ((NodoPelicula) raiz).buscarPorPuntaje(minimo, maximo, resultado);
        }

        return resultado;
    }
}
```

Búsqueda por rango de puntajes – en nodo

```java
private class NodoPelicula extends ElementoAB<Pelicula> {
    /**
     * Método que busca películas por puntaje en el árbol.
     * @param minimo Puntaje mínimo (incluido). Si es null, no se aplica mínimo.
     * @param maximo Puntaje máximo (incluido). Si es null, no se aplica máximo.
     * @param resultado Lista ordenada de películas que cumplen con el rango de puntaje.
     */
    public void buscarPorPuntaje(Float minimo, Float maximo, ListaOrdenada<Pelicula> resultado) {
        Pelicula pelicula = getDatos();

        float puntaje = pelicula.getPuntaje();

        // Verifica si el puntaje de la película está dentro del rango especificado
        boolean mayorMin = (minimo == null) || (puntaje >= minimo);
        boolean menorMax = (maximo == null) || (puntaje <= maximo);

        // El chequeo de puntaje se realiza por diferencia e igualdad para incluir los extremos del
        // intervalo
        if (minimo == null || puntaje >= minimo) {
            // Si el hijo izquierdo no es nulo, busca en el subárbol izquierdo
            // para encontrar películas con puntajes mayores al mínimo
            if (getHijoIzq() != null)
                ((NodoPelicula) getHijoIzq()).buscarPorPuntaje(minimo, maximo, resultado);
        }

        // Si la película cumple con el rango de puntaje, se agrega a la lista de resultados
        if (mayorMin && menorMax) {
            resultado.insertar(pelicula, pelicula.getAnio());
        }

        if (maximo == null || puntaje <= maximo) {
            // Si el hijo derecho no es nulo, busca en el subárbol derecho
            // para encontrar películas con puntajes menores al máximo
            if (getHijoDer() != null)
                ((NodoPelicula) getHijoDer()).buscarPorPuntaje(minimo, maximo, resultado);
        }
    }
}
```