Documentación del Proyecto: Sistema de Gestión de Cine

---

1. Descripción del Proyecto

El proyecto **Sistema de Gestión de Cine** es una aplicación de consola en Python diseñada para simular la operación de un cine universitario, enfocándose en la gestión de películas, funciones, usuarios y el proceso de reserva de asientos.

La solución implementa diversas **Estructuras de Datos Abstractas (TDA)** fundamentales (Lista Enlazada, Árbol Binario de Búsqueda y Tablas Hash/Diccionarios) desde cero (o como *wrappers* simples sobre tipos nativos de Python) para cumplir con objetivos pedagógicos y de rendimiento. Permite a los usuarios registrarse, ver la cartelera ordenada por hora, reservar asientos en tiempo real (utilizando una matriz 2D para la sala) y cancelar reservas.

Problemática abordada: Gestionar eficientemente la disponibilidad de funciones, el *mapping* de asientos en salas y el registro de transacciones de reserva, priorizando la **rapidez de acceso** a las entidades clave.

---

2. Diagrama de Clases UML

El sistema está compuesto por clases que representan las estructuras de datos y las entidades de negocio.

Relaciones Clave:

- **Composición (Fuerte):**
    - MiLista ← NodoLista: La lista no existe sin sus nodos.
    - ArbolFunciones ← NodoArbol: El árbol no existe sin sus nodos.
    - Usuario ← MiLista (de Reservas): El listado interno de reservas pertenece exclusivamente al usuario.
- **Agregación (Débil):**
    - Funcion ← Pelicula, Sala: Una Función utiliza una Película y una Sala, pero estas existen independientemente.
    - Reserva ← Usuario, Funcion: Una Reserva se asocia a un Usuario y una Función, pero estas entidades existen por separado.
- **Asociación/Uso:**
    - Cine utiliza las estructuras DiccionarioSimple y ArbolFunciones para gestionar sus colecciones de entidades.

---

3. Justificación de Diseño

A. Jerarquía de Herencia

En este proyecto, se optó por una estructura de **composición** e **implementación modular** en lugar de una jerarquía de **herencia** profunda.

- **Justificación:** Se decidió que la herencia no aportaba un beneficio significativo ya que las entidades de negocio (Pelicula, Sala, Funcion, etc.) son conceptualmente distintas y no comparten un comportamiento base complejo que requiera ser generalizado.
    - En su lugar, se utilizó la **composición** (ej: Funcion *tiene* una Pelicula y una Sala, Usuario *tiene* una MiLista de Reservas) para modelar las relaciones del mundo real de manera más clara y flexible.
- **Nota:** Las estructuras de datos (MiLista, ArbolFunciones) son clases independientes diseñadas para ser reutilizadas por las clases del negocio (Usuario, Cine).

B. Selección de Estructuras de Datos (TDA)

Se seleccionó cada TDA con el objetivo principal de optimizar la **velocidad de acceso** y el **ordenamiento automático** de los datos según los requerimientos de la aplicación.

Entidad Gestionada	TDA Seleccionado	Justificación de Rendimiento
Películas, Usuarios, Reservas	**Diccionario Simple** (Tabla Hash)	Permite un tiempo de acceso, inserción y verificación de existencia (**Búsqueda O(1)**) para estas entidades. Se utiliza la clave (ID para Películas/Reservas, Email para Usuarios) para una recuperación instantánea.
Funciones	**Árbol Binario de Búsqueda (BST)** (ArbolFunciones)	Las funciones deben mostrarse al usuario **ordenadas por hora**. El BST mantiene automáticamente el orden a medida que se insertan. El recorrido **In-Order** garantiza que se obtenga una lista de funciones cronológicamente ordenada con un coste de O(n).
Asientos de Sala	**Matriz 2D** (Sala.asientos)	La representación física de la sala es una cuadrícula de filas y columnas. La matriz 2D proporciona un **acceso directo**

Entidad Gestionada	TDA Seleccionado	Justificación de Rendimiento
		(**O(1)**) a cualquier asiento por sus coordenadas (fila, columna), esencial para la reserva y liberación rápida.
Reservas por Usuario	**Lista Enlazada** (MiLista)	Se eligió una lista enlazada simple para almacenar las reservas de un usuario por su simplicidad en la implementación y porque las operaciones principales son solo **agregar al final** ($O(n)$ pero rápido para listas pequeñas) y **mostrar** (recorrido $O(n)$), sin requerir una búsqueda eficiente dentro de la colección personal del usuario.

---

4. Manual de Usuario

Requisitos

- **Python 3.x** instalado.

Ejecución del Programa

1. **Clonar/Descargar el Repositorio:**

git clone https://github.com/luvitovi/upse-pa-1

cd upse-pa-1

2. Ejecutar el Script Principal:

El sistema se ejecuta directamente desde la terminal.

python sistema_cine.py

Interfaz y Operación

Al ejecutar el programa, se mostrará el menú principal:

=== CINE UNIVERSITARIO ===

1. Ver películas

2. Ver funciones

3. Registrarse

4. Hacer reserva

5. Cancelar reserva

6. Ver asientos

7. Salir

Opción:

Pasos Típicos:

1. **Registrarse (Opción 3):** Necesario antes de reservar. El sistema pedirá Nombre y Email. El email se usa como clave única.
2. **Ver Funciones (Opción 2):** Muestra la cartelera. Las funciones están **ordenadas cronológicamente** (gracias al BST) y se muestra su ID (ej: F1, F2).
3. **Hacer Reserva (Opción 4):**
    - Se solicita el Email del usuario.
    - Se pide el **ID de la función** (ej: F1).
    - Se muestra el mapa de asientos (O=Libre, X=Reservado).
    - Se ingresan los asientos deseados separados por coma (ej: A3, B4). **Importante:** Las filas se nombran con letras (A, B, C...) y las columnas con números (1, 2, 3...).
4. **Ver Asientos (Opción 6):** Muestra el estado de la **Sala 1** y la **Sala 2** para una revisión general de la ocupación.