DESCRIPCIÓN DE ESTRUCTURAS USADAS

Dentro de este proyecto se implementaron 4 estructuras de datos distintas:

- Listas doblemente enlazadas
- Diccionarios / tablas hash
- Matrices dispersas
- Árboles

Ahora a continuación habrá una breve descripción de cada una junto al por qué se implementaron. Ahora cabe aclarar que las listas doblemente enlazadas no se explicaran ya que estas no son de importancia para este proyecto:

1. Diccionarios / tablas hash

La plataforma en si para que tenga sentido su existencia requiere de usuarios, series y películas. Estos elementos se representan por medio de clases junto a sus atributos y a su vez la plataforma requiere guardarlos en alguna parte a la que pueda acceder a los mismos de manera eficiente tanto en términos de tiempo como de memoria.

Para poder guardar estos datos podemos usar diferentes estructuras de datos como listas, arreglos o colas, pero dado que queremos una acceso rápido debemos usar una estructura que no itere sobre todos los elementos para poder encontrar el indicado, y como la cantidad de datos puede llegar a ser muy alta esto complica la asociación entre posiciones y el elemento guardado, por lo que para evitar esto teniendo un acceso rápido usamos tablas hash que usen como clave el nombre de los atributos dado que estos se consideran únicos dentro del programa (no habrá dos películas con el mismo nombre), esto permitiéndonos realizar búsquedas por nombre manejando una complejidad temporal O(1) y sirviendo como apoyo en los algoritmos de búsqueda donde se ingresan palabras para realizar la búsqueda de algo solicitado.

2. Matrices dispersas

Dentro del programa nosotros establecemos relaciones entre las instancias de las clases, por ejemplo:

- Una película forma parte de una o varias categorías/etiquetas
- Un usuario mira y califica ninguna o muchas películas

Estas relaciones no dejan de ser información relevante que puede usar el programa para mejorar ciertos algoritmos o aplicar nuevos, esto hace que resulte muy útil guardar estas relaciones para que el programa las uses en diferentes, sin embargo esta información no se puede guardar de cualquier manera, si por ejemplo guardamos las relaciones en listas

resulta difícil operar o acceder a las mismas sabiendo a qué es a lo que se accede. Esto nos lleva a usar matrices para representar estas relaciones donde a las filas y columnas se les hace una referencia entre índice-id tal que la casilla en la posición **(a, b)** guarda un valor que establece la relación entre los elementos **a** y **b**.

Ahora, si analizamos las relaciones planteadas dentro del programa podemos darnos cuenta que muchas de ellas guardan valores 0, por ejemplo, una película a lo mucho tendrá 4 etiquetas con las que se relaciona, por lo que en una matriz etiquetas – películas de 20 etiquetas podremos llegar a tener una columna de hasta 16 ceros o más, por lo que el hecho de llegar a tener demasiados 0 en esta matriz nos lleva a usar una matriz dispersa como estructura de datos para guardar esta información tal que no gaste mucho espacio de manera innecesaria y nos permita operar sobre ella.

Dentro del programa estas matrices dispersas nos permiten:

- Buscar contenido por categorías buscando en columnas de manera eficiente (no hace falta ver todo el contenido).
- Crear un sistema de recomendaciones basándose en operaciones sobre la matriz usuarios -contenido.

3. Árboles binarios de búsqueda

Con los diccionarios / tablas hash y matrices dispersas ya podemos manejar de buena muchas de las búsquedas anteriormente mencionadas, pero si queremos implementar búsquedas basadas en rangos de un atributo estas dos se quedan muy atrás, pues usando solo las tablas hash tendría que comprobar data por dato cuales están dentro del rango establecido para retornar, mientras que la matriz dispersa no me da la información necesaria para realizar esa clase de búsquedas. Para estos casos lo que se puede hacer es implementar un árbol binario que guarda los datos de cierta manera que me permite al buscar por rangos ignorar de inmediato muchos datos que me permitirían obtener lo buscado con una complejidad temporal **O(log n)**, siendo más eficiente que usar la tabla hash. Ahora, la idea es no duplicar información por lo que para estos árboles se guardan dos cosas, donde queremos realizar búsquedas por valoración y duración de contenido lo que se hace es guardar en cado nodo el atributo de búsqueda para comparar y el nombre del contenido al que hace referencia para después ser buscado en la tabla hash, así evitando el duplicado de las instancias o un manejo complicado de punteros.

La razón por la que el árbol funciona es por su forma de guardado donde se usa comparador para guardar los de atributo más pequeño a la izquierda y los de atributos más grande a la derecha.

Para la implementación del árbol se usa **multimap** que ya actúa como un árbol en **c++** con sus métodos de inserción y con algoritmos que lo mantienen balanceado.