UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Programación III CS2013

Proyecto Final

Teoría 5 Lab 5.01 - 2024 - 1

DOCENTE

Chavez Alvarez, Jose Armando

INTEGRANTES - Grupo 15

- Carlos David Ordinola Ortega - 202310448
- Juan Diego Luque Segura
- Paris Lenard Herrera Torres - 202310100

Julio, 2024

1. **Introducción:**

   Objetivo del proyecto:

   Desarrollar una plataforma de streaming utilizando el lenguaje de programación c + + Esta plataforma debe permitir a los usuarios buscar y gestionar una colección de películas de manera eficiente gracias a la **estructura de datos seleccionada árbol de prefijos.**

2. **Diseño del programa:**

   El programa consiste de las siguientes clases:

2.1. Gestor Películas Interfaz

Descripción: Es una clase abstracta que define la interfaz para la gestión de películas. Declara los métodos que deben ser implementados por cualquier clase que gestione películas, asegurando una interfaz consistente.

Métodos:

- cargar_peliculas(): Método virtual puro para cargar películas desde un archivo.
- buscar_general(const string& consulta, size_t start_index, size_t max_results): Método virtual puro para realizar una búsqueda general de películas.
- buscar_tag(string s): Método virtual puro para realizar una búsqueda de películas por tags.

2.2. Gestor_peliculas_singleton

Descripción: Es la clase principal que gestiona la carga, búsqueda y almacenamiento de películas.Implementa la interfaz Gestor_peliculas_interfaz. Utiliza el **patrón Singleton** para asegurar que solo exista una instancia de la clase en el sistema.

Atributos:

- static Gestor_peliculas_singleton* gestor: Instancia única de la clase.
- arbol_trie<Pelicula> arbol_busqueda: Estructura de datos trie para la búsqueda eficiente de películas.
- int contador_hilo1: Contador de líneas procesadas por el primer hilo.
- int contador_hilo2: Contador de líneas procesadas por el segundo hilo.
- mutex mtx: Mutex para asegurar operaciones seguras en multihilo.
- vector<string> ver_mas_tarde: Vector que almacena los títulos de las películas guardadas en "ver más tarde".

Métodos:

- static Gestor_peliculas_singleton* get_instance(): Devuelve la instancia única de la clase.
- void cargar_peliculas(): Carga las películas desde un archivo CSV utilizando hilos.
- void buscar_general(const string& consulta, size_t start_index = 0, size_t max_results = 5): Realiza una búsqueda general de películas.
- void buscar_tag(string s, size_t start_index = 0, size_t max_results = 5): Realiza una búsqueda de películas por tags.
- void cargar_ver_mas_tarde(): Carga las películas guardadas en "ver más tarde" desde un archivo.
- void mostrar_ver_mas_tarde(size_t start_index = 0, size_t max_results = 5): Muestra las películas guardadas en "ver más tarde".
- void agregar_a_ver_mas_tarde(const string& titulo): Agrega una película a la lista de "ver más tarde".

2.3. Cargador_peliculas_proxy_singleton

Descripción: Es una clase proxy que controla el acceso a las funciones de Gestor_peliculas_singleton. Utiliza el patrón Singleton para asegurar que solo exista una instancia de la clase, funciona como proxy al asegurarse que el método “cargar_peliculas” se ejecute solo una vez en el código, debido a que esta es una tarea costosa y repetirla es innecesaria.

Atributos:

- static Cargador_peliculas_proxy_singleton* cargador: Instancia única de la clase.
- bool carga: Indica si las películas ya han sido cargadas.
- Gestor_peliculas_singleton* gestor: Instancia de Gestor_peliculas_singleton.

Métodos:

- static Cargador_peliculas_proxy_singleton* get_instance_cargador_proxy(): Devuelve la instancia única de la clase.
- void cargar_peliculas(): Controla la carga de películas para asegurar que no se carguen más de una vez.
- void buscar_general(const string& consulta, size_t start_index = 0, size_t max_results = 5): Delegado a Gestor_peliculas_singleton.
- void buscar_tag(string s, size_t start_index = 0, size_t max_results = 5): Delegado a Gestor_peliculas_singleton.

2.4. Pelicula

Descripción: Representa una película con sus atributos como título, sinopsis, tags, etc. Contiene métodos para interactuar con las peliculas

Atributos:

- string imdb_id: ID de IMDb de la película.
- string titulo: Título de la película.
- string plot_synopsis: Sinopsis de la película.
- vector<string> tags: Tags asociados con la película.
- string split: Información adicional (si aplica).
- string synopsis_source: Fuente de la sinopsis.

Métodos:

- Pelicula(const string& imdb_, const string& t, const string syn, const vector<string>& ta, const string& spl, const string& syn_source): Constructor de la clase.
- string get_titulo() const: Devuelve el título de la película.
- string get_plot_synopsis() const: Devuelve la sinopsis de la película.
- vector<string> get_tags() const: Devuelve los tags de la película.
- string get_imdb_id() const: Devuelve el ID de IMDb de la película.
- void mostrar_informacion() const: Muestra la información completa de la película.

2.5. Trie_nodo

Descripción: Representa un nodo en la estructura de datos trie. Cada nodo puede tener múltiples hijos, representados por un mapa de caracteres a nodos únicos.

Atributos:

- unordered_map<char, unique_ptr<Trie_nodo<T>>> hijos: Mapa de caracteres a nodos hijos.
- vector<T> datos: Vector de datos almacenados en el nodo.

2.6. arbol_trie

Descripción:

Implementa un trie para realizar búsquedas eficientes de películas por título y tags.

Atributos:

- unique_ptr<Trie_nodo<T>> raiz: Raíz del trie.

Métodos:

- arbol_trie(): Constructor de la clase.
- void insertar(const string& clave, const T& dato): Inserta una película en el trie.
- vector<T> buscar_general(const string& consulta, size_t start_index = 0, size_t max_results = 5) const: Realiza una búsqueda general en el trie.
- const Trie_nodo<T>* buscar_nodo(const string& prefijo) const: Busca un nodo en el trie basado en un prefijo.
- void buscar_aux(const Trie_nodo<T>* nodo, string prefijo, vector<T>& resultados) const: Función auxiliar para realizar búsquedas en el trie.
- void buscar_subcadena_aux(const Trie_nodo<T>* nodo, const string& subcadena, const string& titulo_actual, vector<T>& resultados) const: Función auxiliar para buscar subcadenas en el trie.

3. Funcionalidades del sistema:

3.1 Funcionamiento del Código en Conjunto

La idea principal del proyecto fue desarrollar una plataforma de streaming en C++ que permita gestionar y buscar películas de manera eficiente. Para ello, se utilizó un archivo CSV de películas proporcionado, el cual fue limpiado y procesado para evitar problemas con el uso de comas como separadores. Aquí se describe el funcionamiento completo del código.

3. 2 Preparación del Archivo CSV

Inicialmente, el archivo CSV de las películas contenía comas dentro de las sinopsis, lo que dificultaba la extracción de información correcta. Para resolver este problema, se reemplazaron las comas por comillas dobles y se utilizaron comillas dobles espacio comillas dobles (" ") para separar los campos de información. Además, se eliminó cualquier salto de línea que no correspondiera a una nueva película, asegurando que cada película estuviera en una sola línea.

3.3 Carga y Procesamiento de Películas

El archivo CSV ya limpio se lee utilizando programación concurrente con dos hilos. Cada hilo se encarga de procesar una parte del archivo, creando objetos de la clase Película y almacenándolos en un árbol de prefijos (trie). Este enfoque en paralelo mejora la eficiencia y reduce el tiempo de carga, además se uso mutex para asegurar que la inserción por tags y por título se ejecute correctamente La clase Gestor_peliculas_singleton maneja este proceso de carga.Una vez cargadas las películas, el usuario puede interactuar con la plataforma a través de tres opciones principales:

3.3.1 Búsqueda General

Descripción:
La búsqueda general permite a los usuarios buscar películas utilizando cualquier término relacionado con el título de la película.

Implementación:

Cuando el usuario ingresa un término de búsqueda, el sistema busca en el árbol de prefijos (trie) títulos que coincidan con dicho término. Los resultados encontrados se muestran al usuario, quien puede optar por agregar alguna de las películas a la lista de "ver más tarde". Si hay más resultados disponibles, se le pregunta al usuario si desea ver más resultados, y en caso afirmativo, se continúa mostrando los siguientes resultados.

3.3.2 Búsqueda por Tags

Descripción:
La búsqueda general permite a los usuarios buscar películas utilizando cualquier término relacionado con el título de la película.

Implementación:
El usuario ingresa un tag de búsqueda y el sistema busca en el trie tags que coincidan con el término ingresado. Los resultados se muestran al usuario, quien puede agregar las películas encontradas a la lista de "ver más tarde". Si hay más resultados disponibles, se le pregunta al usuario si desea ver más, y en caso afirmativo, se continúa mostrando los siguientes resultados.

3.3.3 Lista de "Ver Más Tarde"

Descripción:

La funcionalidad "ver más tarde" permite a los usuarios guardar películas para ver en el futuro.

Implementación:
Cuando el usuario elige agregar una película a "ver más tarde", el título de la película se guarda en un archivo. Al iniciar la plataforma, se cargan los títulos de las películas guardadas desde el archivo. El usuario puede optar por ver las películas guardadas en "ver más tarde", y los títulos se buscan en el trie y se muestran al usuario. Si hay más películas en la lista, se le pregunta al usuario si desea ver más, y en caso afirmativo, se continúa mostrando las siguientes películas guardadas.

4. Conclusiones

- La utilización de un árbol de prefijos (trie) ha sido fundamental para garantizar una búsqueda rápida y eficiente de títulos y tags de películas. Esta estructura de datos optimiza las operaciones de búsqueda.
- Al ser más de catorce mil películas que procesar, se esperaba que esta fuera una tarea costosa, sin embargo con el paradigma de programación concurrente y la implementación de hilos para la carga y procesamiento de datos ha permitido una inserción y lectura de datos más eficaz.
- El uso de clases bien definidas y la aplicación de patrones de diseño como Singleton y Proxy han hecho el código más entendible, mantenible y escalable. Nos permitieron tener mejor gestión y entendimiento sobre el código La separación de responsabilidades en diferentes clases ha facilitado la extensión y modificación del sistema.