Implementación de un Sistema de Gestión de Playlist con Árboles B

facundo Sanz Palomino
Johan Marquez Zuñiga
Luis Gonzalo Basurco Monrroy

En este trabajo se presenta un sistema para la gestión de canciones y listas de reproducción utilizando árboles B. Este sistema permite una organización eficiente de los datos mediante la estructura de árbol, lo que facilita las operaciones de inserción, eliminación y búsqueda de canciones. La implementación se realizó en lenguaje C++.

Palabras clave: algunas

I. Introducción

Los sistemas de gestión de listas de reproducción son fundamentales en aplicaciones musicales y de entretenimiento. Un aspecto clave de estos sistemas es la eficiencia en la gestión de grandes cantidades de datos. En este contexto, los árboles B ofrecen una estructura adecuada para almacenar y acceder a las canciones de manera eficiente. El objetivo de este trabajo es implementar un sistema de gestión de playlists basado en árboles B, permitiendo la inserción, eliminación y búsqueda rápida de canciones.

II. Desarrollo

Estructura del Árbol B

Un árbol B es una estructura de datos balanceada, diseñada para mantener datos ordenados y permitir búsquedas, inserciones y eliminaciones eficientes. En este sistema, cada nodo del árbol B contiene información de una canción, incluyendo su nombre, artista y otros metadatos.

El árbol B es particularmente adecuado para esta tarea porque permite operaciones de búsqueda y actualización en un tiempo logarítmico, lo que es esencial para manejar grandes volúmenes de datos.

Diseño del Sistema

El sistema fue diseñado con un enfoque modular, con clases dedicadas a representar canciones, listas de reproducción y el árbol B que las organiza. Las principales operaciones soportadas por el sistema incluyen:

- Inserción de canciones en la playlist.
- Búsqueda de canciones por título o artista.
- inserción a playlist

Interfaz de usuario

La información extraída del código main proporciona una mejor búsqueda, clasificación y modificación del archivo csv como se muestra a continuación

```
PROCEDIMIENTO Mostrar Menu()

IMPRIMIR "\n--- Menú ---"

IMPRIMIR "1. Buscar canción"

IMPRIMIR "2. Agregar canción"

IMPRIMIR "3. Reproducción Aleatoria"

IMPRIMIR "4. Gestión de Playlists"

IMPRIMIR "5. Mostrar Top 100 canciones de un
año"

IMPRIMIR "6. Salir"

FIN PROCEDIMIENTO
```

la interfaz mejora la facilidad para la eleccions de opciones y dar la mejor experiencia al usuario

Función de búsqueda , agregación , playlist , top canciones

Búsqueda

```
PROCEDIMIENTO Buscar(term)

PARA i DESDE 0 HASTA n - 1 HACER

SI term EN keys[i].track_name O term EN
keys[i].artist_name ENTONCES

IMPRIMIR "Artista: " + keys[i].artist_name + " -
Canción: " + keys[i].track_name

FIN SI

FIN PARA

SI NOT leaf ENTONCES

PARA i DESDE 0 HASTA n HACER

LLAMAR children[i].Buscar(term)

FIN PARA

FIN SI
```

```
FIN PROCEDIMIENTO
```

El procedimiento de búsqueda se realiza de manera directa ingresando al csv para poder extraer la canción deseada.

Agregación de canción

```
PROCEDIMIENTO InsertarCancion(k)

SI NOT root ENTONCES

root <- NUEVO BTreeNode(t, TRUE)

root.keys[0] <- k

root.n <- 1

SINO

SI root.n == 2 * t - 1 ENTONCES

s <- NUEVO BTreeNode(t, FALSE)

s.children[0] <- root

LLAMAR splitChild(0, root)

i <- 0

SI s.keys[0].track_name < k.track_name ENTONCES

i <- i + 1

FIN SI

s.children[i].insertNonFull(k)

root <- s

SINO

root.insertNonFull(k)

FIN SI

FIN SI

FIN PROCEDIMIENTO
```

el proceso que sigue es el de ingresar una nueva canción el cual pedirá todas las características del csv

Playlist

```
PROCEDIMIENTO CrearPlaylist(nombre, grado)

SI nombre ES VACÍO ENTONCES

IMPRIMIR "El nombre de la playlist no puede
estar vacío."

RETORNAR

FIN SI

SI nombre EXISTE EN playlists ENTONCES

IMPRIMIR "La playlist \"" + nombre + "\" ya
existe."

RETORNAR

FIN SI

playlists[nombre] <- NUEVO BTree(grado)

IMPRIMIR "Playlist \"" + nombre + "\" creada
exitosamente."

FIN PROCEDIMIENTO
```

la función que cumple es la de crear una lista de reproducción que permite al usuario seleccionar y mover una canción a una lista personalizada

Top canciones

```
 PROCEDIMIENTO ObtenerCancionesPorAnio(year,
songs)

SI root EXISTE ENTONCES

allSongs <- LLAMAR collectSongs(root)

PARA CADA cancion EN allSongs HACER

SI cancion.year == year ENTONCES

LLAMAR Agregar(songs, cancion)

FIN SI

FIN PARA

FIN SI

FIN PROCEDIMIENTO
```

el top canciones muestra por año las mejores 100 canciones que se escucho, esto mejora la búsqueda por año y popularidad

III. RESULTADOS

1. BTree como estructura de datos eficiente

Un árbol B es una estructura de datos balanceada y autoajustable que permite realizar búsquedas, inserciones y eliminaciones de manera eficiente en grandes cantidades de datos. En el contexto de tu sistema, el árbol B se utiliza para gestionar las canciones dentro de las playlists, lo que permite un acceso rápido a las canciones, incluso cuando el número de ellas es grande.

- Búsqueda eficiente: La función de búsqueda en el árbol B (como se muestra en el código BTree::search) permite realizar búsquedas de canciones por nombre de canción o artista. Debido a la propiedad balanceada del árbol B, la complejidad de la búsqueda es O(log n), lo que significa que incluso con un gran número de canciones, el tiempo de búsqueda no se incrementa

significativamente.
- Inserciones rápidas: Las inserciones de canciones dentro del árbol B también son eficientes. El algoritmo de inserción en el árbol B divide los nodos cuando se alcanzan su capacidad máxima, asegurando que el árbol permanezca balanceado y las operaciones de inserción tengan una complejidad de O(log n).

2. Manejo de playlists con árboles B

El sistema permite la creación, eliminación, renombrado y transferencia de canciones entre playlists, lo cual está implementado mediante árboles B. Las playlists son gestionadas en una estructura que permite acceder a las canciones rápidamente y realizar modificaciones sin tener que recorrer toda la lista de manera secuencial.

- Eficiencia en la gestión de playlists: Las operaciones para agregar canciones a las playlists (por ejemplo, PlaylistManager::AddSongToPlaylist) o mover canciones entre playlists (como en PlaylistManager::transferirCancion) se realizan de manera eficiente gracias al uso de la estructura de árbol B para almacenar las canciones. Las búsquedas y manipulaciones dentro de cada playlist son rápidas, incluso si el número de canciones es grande.

3. Reproducción aleatoria y ordenación eficiente

El árbol B también es útil para las funciones que requieren manipulación de las canciones, como la reproducción aleatoria (BTree::shuffle) y la ordenación de canciones según diferentes criterios (por ejemplo, por nombre, popularidad, año, etc.).

- Reproducción aleatoria: El algoritmo de reproducción aleatoria utiliza un recorrido completo de las canciones en el árbol B, lo que implica que se realiza una recopilación de todas las canciones antes de mezclar y reproducirlas aleatoriamente. Gracias al método collectSongs, todas las canciones son obtenidas en un solo recorrido eficiente. Posteriormente, el uso de std::shuffle permite mezclar las canciones rápidamente, garantizando que la reproducción no siga un patrón predecible.
- Ordenación de canciones: El uso de árboles B para ordenar canciones según diferentes criterios (como popularidad, año o nombre) también es eficiente, ya que la recopilación de las canciones en un vector permite aplicar algoritmos de ordenación como std::sort, lo cual se realiza en O(n log n) tiempo, donde n es el número de canciones.

4. Análisis de rendimiento

Se pueden analizar los tiempos de ejecución de operaciones clave como la inserción de canciones, la búsqueda y la ordenación. Si bien los árboles B están diseñados para manejar grandes volúmenes de datos, el rendimiento real dependerá del número de canciones y la estructura del árbol (es decir, el grado t). Al realizar pruebas con diferentes tamaños de playlists y tipos de búsquedas, se puede observar cómo la estructura de datos escala y cómo afecta el rendimiento en función del número de canciones y el uso del sistema.

Pruebas posibles:

- Insertar un gran número de canciones en una playlist y medir el tiempo de ejecución de la operación.
- Realizar búsquedas de canciones por nombre y artista en playlists con diferentes tamaños.
- Probar la eficiencia de la ordenación y la reproducción aleatoria en playlists grandes.

IV. CONCLUSIONES

Este trabajo demuestra la viabilidad del uso de árboles B para la gestión de datos en sistemas de playlist. La implementación propuesta permite una organización eficiente de las canciones y facilita el acceso rápido a ellas, incluso cuando el número de elementos en la base de datos crece significativamente. A través de la utilización de árboles B, se ha logrado optimizar las operaciones de búsqueda, inserción y eliminación de canciones dentro de las playlists, lo que mejora el rendimiento del sistema y asegura una experiencia de usuario fluida.

Algunas conclusiones clave incluyen:

1. Eficiencia en la gestión de datos: La utilización de árboles B permite manejar grandes volúmenes de datos de manera eficiente, gracias a su capacidad para realizar operaciones de búsqueda, inserción y eliminación en O(log n), lo cual es crucial cuando se trabaja con un gran número de canciones en las playlists.
2. Escalabilidad: El sistema es escalable, lo que significa que se puede manejar un aumento significativo en la cantidad de canciones y playlists sin una pérdida considerable de rendimiento. Esto es esencial para aplicaciones a gran escala, como plataformas de música en línea.
3. Optimización de la experiencia de usuario: Las operaciones de búsqueda de canciones por nombre o artista, la ordenación por criterios como popularidad o año, y la reproducción aleatoria son altamente eficientes. El uso de un árbol B permite que estas funcionalidades se ejecuten rápidamente, incluso en sistemas con miles de canciones.
4. Simplicidad en la implementación: A pesar de ser una estructura de datos compleja, la implementación de árboles B en este sistema resulta relativamente sencilla y proporciona beneficios sustanciales en cuanto a rendimiento y eficiencia. Los algoritmos para la inserción, búsqueda y reordenación están bien definidos y se implementan sin complicaciones adicionales.
5. Futuro trabajo y mejoras: Aunque el sistema muestra un rendimiento robusto, se pueden explorar mejoras adicionales, como la optimización de las funciones de búsqueda utilizando técnicas de caching o la integración de métodos de aprendizaje automático para personalizar las recomendaciones de canciones. También se podría considerar la implementación de otros algoritmos de árboles balanceados, como los árboles AVL, y compararlos en términos de rendimiento en escenarios específicos.

En resumen, la implementación de árboles B en la

gestión de playlists y canciones es altamente eficaz y proporciona una base sólida para el desarrollo de sistemas musicales o de gestión de medios en general. Los resultados muestran que el uso de esta estructura de datos no solo mejora el rendimiento, sino que también facilita la expansión y evolución del sistema a medida que crecen los datos.

V. CONCLUSIONS

This work demonstrates the viability of using B-trees for data management in playlist systems. The proposed implementation allows for efficient organization of songs and enables fast access to them, even as the number of elements in the database grows significantly. Through the use of B-trees, we have optimized the operations of searching, inserting, and deleting songs within playlists, which enhances system performance and ensures a smooth user experience.

Key conclusions include:

1. Data Management Efficiency: The use of B-trees allows handling large volumes of data efficiently, thanks to its ability to perform search, insert, and delete operations in O(log n), which is crucial when working with a large number of songs in playlists.
2. Scalability: The system is scalable, meaning it can handle a significant increase in the number of songs and playlists without a considerable loss in performance. This is essential for large-scale applications, such as online music platforms.
3. User Experience Optimization: Operations like searching for songs by name or artist, sorting by criteria such as popularity or year, and random playback are highly efficient. The use of a B-tree ensures these functionalities execute quickly, even in systems with thousands of songs.
4. Simplicity in Implementation: Despite being a complex data structure, the implementation of B-trees in this system is relatively straightforward and provides substantial benefits in terms of performance and efficiency. The algorithms for insertion, search, and reordering are well-defined and implemented without significant complications.
5. Future Work and Improvements: Although the system shows robust performance, further improvements can be explored, such as optimizing search functions using caching techniques or integrating machine learning methods to personalize song recommendations. The implementation of other balanced tree algorithms, such as AVL trees, could also be considered and compared in terms of performance in specific scenarios.

In summary, the implementation of B-trees in playlist and song management is highly effective and provides a solid foundation for the development of music or media management systems in general. The results show that the use of this data structure not only improves performance but also facilitates the expansion and evolution of the system as the data grows.