DataFusionDB: Reporte

José Joaquín Osnayo Matos

October 1, 2024

1 Introducción:

El objetivo de este proyecto es encontrar la técnica de indexación de archivos que permita la búsqueda puntual y por rango más eficiente. Para esto se proporcionarán y compararán 3 técnicas de indexación, la técnica de archivo secuencial, la técnica de árbol AVL, y la técnica de extendible hashing.

El dominio de datos a utilizar para este proyecto es la base de datos de kaggle "Anime Dataset 2023" https://www.kaggle.com/datasets/dbdmobile/myanimelist-dataset, la cual contiene información de 55734 animes. Para el proyecto se trabajará con una versión modificada de esta base de datos, con solo 8 columnas y con 100000 entradas formadas por filas de la base de datos original en orden aleatorio donde varias entradas estarán repetidas. La base de datos filtrada tendrá filas que formarán la estructura Anime con las siguientes columnas:

id int, clave identificadora del anime.
nombre char[100], nombre original del anime.
puntaje float, nota del 1 al 10 que indica su calidad.
genero char[50], tipo de trama del anime.
tipo char[10], formato del anime.
temporada float, fecha de estreno del anime codificado como año.estacionDelAño, donde verano = año.1, otoño = año.2, invierno = año.3 y primavera = año.4.
estado char[20], estado del anime (aún no emitido, en emisión, finalizado).
estudio char[30], empresa que emitió el anime.

Los resultados que se esperan obtener de la indexación por archivo secuencial es que la búsqueda puntual para la parte ordenada tenga complejidad algorítmica O(log(n)), y para la parte auxiliar tenga complejidad algorítmica O(k), donde k sea menor o igual a log(n), por lo que en general se espera una búsqueda puntual con tiempo O(log(n)). En cuanto al AVL, se espera una búsqueda puntual con complejidad algorítmica O(log(n)), y tanto en el archivo secuencial como en el AVL se espera una búsqueda por rango O(log(n)) + el número de coincidencias en el rango. Finalmente para el extendible hashing se desea que la búsqueda puntual sea constante, aunque en realidad se espera que su complejidad algorítmica sea O(1) + número de colisiones, además se espera que no soporte la búsqueda por rango.

2 Técnicas utilizadas:

2.1 Archivo secuencial:

Un archivo secuencial es un archivo ordenado mediante punteros, donde tiene un área que está ordenada físicamente, y un área auxiliar no ordenada, la cual tiene como mucho log(n) elementos, donde n es el número de registros.

Su característica principal es que siempre se mantiene ordenado mediante los punteros, debido a esto, cuando el área auxiliar supera un límite, el archivo se reconstruye, ordenando físicamente todos los todos los registros hasta ese momento.

2.1.1 Inserción:

La inserción funciona de la siguiente manera:

Existen 2 casos: Hay parte ordenada o no hay parte ordenada.

Si no hay parte ordenada se leerá el header, que contiene la posición del registro con menor llave de la lista. Luego se iterará a través de los punteros mientras el registro tenga clave menor o igual, si el registro a comparar tiene clave mayor modificar el puntero del previo por la posición en la que se insertará el registro, y hacer que el registro apunte hacia donde apuntaba el registro modificado, luego insertar al final En caso sea menor que todos hacer la modificación con el header.

En el mejor de los casos se realiza una lectura y escritura, la lectura del header y la inserción del nuevo registro. En el peor de los casos se leen k registros y se escriben 2, el registro modificado y el nuevo registro, donde k es el número de registros del área auxiliar que siempre es menor o igual a log(n), siendo n el número de registros.

Si hay parte ordenada, realizar búsqueda binaria en el área ordenada para encontrar el registro que debe modificar su puntero, si apunta a parte ordenada modificar el puntero del registro por la posición a insertar el nuevo elemento y hacer que el nuevo elemento apunte a donde apuntaba antes el registro modificado, si apunta al área auxiliar, iterar a través de la lista auxiliar hasta encontrar la posición adecuada y modificar el registro indicado, esto mientras no se vuelva a apuntar al área ordenada. De la misma forma si el elemento es menor al primer registro ordenado leer el header e iterar a través de la lista de menores hasta encontrar la posición adecuada.

En el mejor de los casos se realizan log(n) lecturas para encontrar la posición correcta, donde n es el número de registros ordenados, luego una lectura y 2 escrituras, una para encontrar el registro a comparar, y 2 para modificar el registro e insertar el nuevo.

2.1.2 Búsqueda:

La búsqueda funciona de la siguiente manera:

Inicializar un registro por defecto en caso de no encontrarse el elemento. Se tienen 2 casos: Hay parte ordenada o no hay parte ordenada.

Si no hay parte ordenada leer el header y buscar la clave en la lista auxiliar a través de los registros, si se llegó al final retornar el default.

En el peor de los casos se tienen que leer los k elementos del área auxiliar, en el mejor de los casos solo el elemento del header.

Si hay parte ordenada hacer búsqueda binaria y encontrar la posición donde debería estar, si al leer el registro de la posición encontrada la clave es igual retornar el registro, si no, si no apunta a parte auxiliar, retornar el default, si no, buscar en la lista de auxiliares hasta encontrar un puntero a la parte ordenada, donde retorna el default, en cambio si encuentra el elemento en la lista, retornarlo.

En el peor de los casos se realizarán log(parte ordenada) búsquedas para la búsqueda binaria y 1 para retornar el elemento. En el peor de los casos además de las lecturas de la búsqueda binaria también se leen todos los k registros auxiliares hasta llegar a un puntero del área ordenada, donde k es menor o igual a log(n) con n siendo el número de registros.

2.1.3 Búsqueda por rango:

Para la búsqueda por rango también hay 2 casos: Con parte ordenada o sin parte ordenada.

Si no hay parte ordenada leer el header e iterar a través de la lista de auxiliares, insertar en un vector todos los registros que coincidan con elrango hasta encontrar un elemento con clave mayor al final del rango.

En el peor de los casos se hacen k lecturas, donde k es el número de elementos auxiliares. En el mejor caso solo se lee el registro del header en caso de que este ya tenga una clave mayor al rango.

Si hay parte auxiliar aplicar búsqueda binaria a la clave del inicio de rango, insertar en un vector todos los elementos a partir de allí que pertenezcan al rango hasta que un elemento tenga clave mayor al final del rango.

En el peor de los casos el rango ocupa todo el registro, para lo cual se realizan n + log(n) lecturas. En el mejor de los casos solo se hacen log(n) + 1 lecturas, con una lectura para el elemento encontrado.

2.1.4 Eliminación:

2.2 Árbol binario balanceado AVL:

Un árbol AVL es un árbol binario balanceado, donde cada nodo, en este caso cada registro, tiene punteros derecho e izquierdo, donde todos los nodos en el camino del nodo izquierdo son menores a la clave del nodo y todos los nodos en el camino del nodo derecho son mayores o iguales a la clave del nodo, además este árbol también contiene un puntero al padre para manejar las modificaciones en los punteros a la hora del balanceo.

Para que las operaciones en el árbol mantengan una complejidad logarítmica se requiere que el árbol esté balanceado, el AVL usa un balanceo por alturas, donde la diferencia de alturas de cada subárbol no debe ser mayor a 1, sino se realizan rotaciones para equilibrar las diferencias de altura, para lo cual se modifican punteros en el nodo debalanceado y sus asociados, por eso en cada inserción y eliminación se verifica el balanceo de cada nodo en el camino.

2.2.1 Inserción:

2.2.2 Búsqueda:

2.2.3 Búsqueda por rango:

2.2.4 Eliminación:

2.3 Archivo hash extensible

Un archivo hash extensible es un archivo con un índice que apunta a páginas de registros, mediante la técnica hash los registros se guardan en páginas, donde cada página tiene un límite de registros pequeño, de esta forma la búsqueda puntual tiene complejidad algorítmica constante.

Inicialmente se tienen un número determinado de bloques, a los cuales se ingresan mediante el módulo del número de bloques con respecto a una función hash aplicada a la clave a insertar, la característica principal del extendible hashing es que este número de bloques se puede incrementar de ser necesario modificando el índice de acuerdo a los cambios, para esto se dispone de una profundidad global, que será del índice, y una profundidad local, definida en cada bloque, donde si un bloque está lleno y tiene la misma profundidad que la profundidad global, se duplica el índice mientras que se parte únicamente el índice lleno, optimizando así el espacio del archivo.

El extendible hashing también dispone de las características de un archivo hash estático, entre las que está la propiedad de encadenamiento. Para que el índice no se haga demasiado grande, se dispone de un umbral hasta donde puede crecer la profundidad global, así si un bloque está lleno y su profundidad llega al umbral, se procede con encadenamiento, que es asociar mediante un puntero el bloque con uno nuevo.

2.3.1 Inserción:

2.3.2 Búsqueda:

2.3.3 Búsqueda por rango:

2.3.4 Eliminación:

2.4 Parser

Las 3 técnicas están implementadas como subclases de una clase virtual llamada DataFusion, de esa forma se usa un solo parser para la transformación de las consultas a funciones. El parser funciona de la siguiente manera:

Al inicio se puede elegir con cuál de las subclases de DataFusion trabajar.

Recibe la clase DataFusion, tiene 4 funciones, ejecutar select, ejecutar insert, ejecutar select range, ejecutar delete, las cuales ejecutan por dentro las funciones virtuales insertar, buscar, buscar por rango y remover.

Insertar: Para insertar recibe la consulta "Insert into Anime values (atributos);", reconoce el patrón "values(" y convierte el string restante en el input de la función.

Insertar: Para insertar recibe la consulta "Insert into Anime values (atributos);", primero reconoce el patrón "insert into Anime " para entrar a la función ejecutar insert, y luego reconoce el patrón "values(" y convierte el string restante en el input de la función.

Buscar: Para buscar recibe la consulta "select * from Anime where id = x;", primero reconoce el patrón "select * from Anime" para entrar a la función ejecutar select, y luego reconoce el patrón "where id =" y convierte el string restante en el input de la función.

Buscar por rango: Para buscar por rango recibe la consulta "select * from Anime where id between key1 and key2;", primero reconoce que es búsqueda, luego reconoce el patrón "between" y convierte el string restante en los parámetros de la función.

Remover: Para remover recibe la consulta "delete from Anime where id = x", primero reconoce el patrón "delete from Anime " para entrar a la función ejecutar delete, y luego reconoce el patrón "where id =" y convierte el string restante en el input de la función.

3 Resultados experimentales

4 Pruebas de uso