Docs

Introducción

El objetivo del proyecto es organizar dos data sets usando las técnicas de Extendible Hash y Sequential File, así como entender y aplicar los algoritmos de almacenamiento de archivos físicos en estas. Para ello, vamos a hacer uso de dos datasets para cada técnica de organización.

Datasets usados

Los datasets usados fueron recuperados de la página Kaggle. El único criterio tomado en cuenta para la selección de estos datasets fue que los registros pudieran ser representados como registros de longitud fija ya que al implementar la técnica del Sequential File se deben hacer búsquedas binarias, las cuales no son posibles en registros de longitud variable.

Netflix Prize Data

El primer dataset con el que se trabajó fue el de Netflix Prize Data. Este dataset fue usado en una concurso organizado por Netflix para encontrar al mejor algoritmo para predecir el puntaje de los usuarios hacia las películas. Este dataset contiene información sobre el año de lanzamiento, nombre y ID de ciertas películas. Cuenta con 17770 registros, cada uno con 3 campos (columnas) que representan lo siguiente:

- **ID:** Identificador de cada película. Es un valor entero. Este campo fue tomado como key en la construcción del registro.
- **Year:** Año de lanzamiento de cada película. Es un valor entero.
- **MovieTitle:** Nombre de cada película. Es un string.

World Population by Year

El segundo dataset con el que se trabajó fue el de World Population by Year. Contiene información sobre la población mundial, y algunas otras métricas, en un año específico recuperadas de la página Worldmeter y registradas en un archivo csv. Trabaja con los años desde 1951 hasta 2020 y cuenta con 70 registros, cada uno con 7 campos (columnas) que representan lo siguiente:

- **Year:** Año para el que se realiza el análisis. Es un valor entero. Este campo fue tomado como key en la construcción del registro.
- **Population:** Población mundial. Es un valor entero.

- **ChangePerc:** Cambio anual en porcentaje. Es un valor decimal.
- **NetChange:** Cambio total anual. Es un valor entero.
- **Density:** Densidad en P/Km². Es un valor entero.
- **Urban:** Población urbana. Es un valor entero.
- **UrbanPerc:** Población urbana en porcentaje. Es un valor entero.

Resultados esperados

Al finalizar el análisis, se espera obtener un programa que permita al usuario acceder a estos dos datasets con cualquiera de las dos técnicas de organización de archivos y realizar sobre estos cualquiera de las operaciones básicas implementadas, las cuales son inserción, eliminación, búsqueda y búsqueda por rango.

Fundamentación y técnicas

Las técnicas de organización que implementaremos son el Extendible Hash y el Sequential File. La estructura básica de ambas técnicas así como las operaciones implementadas se describen a continuación.

Sequential file

Nuestra implementación del sequential file maneja dos archivos, uno principal de data y otro binario donde se ejecutan las operaciones. No usamos un archivo auxiliar para manejar nuevos registros, pues los insertamos al final del archivo donde van los registros válidos. Para hacer esto y seguir cumpliendo las propiedades del sequential file, se guarda el numero de registros validos y auxiliares para ejecutar las operaciones del sequential como el binary search solo con los registros válidos. Los registros validos están ordenados físicamente mientras los auxiliares no necesariamente. Nuestro sequential file maneja solo archivos de longitud fija debido a que usa binary search para la mayoría de sus métodos. Adicionalmente, puede trabajar con cualquier clase de record debido al uso de los templates T, siendo el tipo de dato de la llave del registro, y Record, siendo la clase del registro. Los registros deberían tener al menos 2 atributos llamados **llave** (identificador del registro) y **next** (posición del siguiente registro), siendo este ultimo de tipo long. Las operaciones implementadas en el Sequential hash son las siguientes:

- **Inserción:** Los registros se insertan al final del archivo como registros auxiliares. Estos registros no estarán necesariamente ordenados. Se ordenarán cuando lleguen a un limite que nosotros pondremos (auxFactor). Cuando la cantidad de registros auxiliares llega a este límite el archivo es reconstruido poniendo los registros auxiliares en orden y contándolos ahora como registros válidos. Entonces, antes de insertar un registro se verifica si los registros

auxiliares han pasado su límite o no. Luego, se lee el archivo binario y se busca la posición previa a donde se deba insertar el nuevo registro, para esto se usa el binary search. Si el registro debe ser insertado al inicio del archivo, se lee el primer registro y se escribe al final del archivo, en la parte auxiliar, y se escribe el nuevo registro al principio del archivo de modo que el registro con la menor llave siempre se mantenga en la sección de registros válidos y las demás operaciones se ejecuten correctamente. En cualquier caso, la inserción consiste en leer el registro en la posición previa a la que se debería insertar el nuevo, copiar su valor de next (posición del siguiente registro) en el next del nuevo registro, actualizar el valor de next en el registro previo para que apunte a la posición donde se inserta el nuevo (al final del archivo) y, finalmente, escribir ambos registros con las modificaciones respectivas.

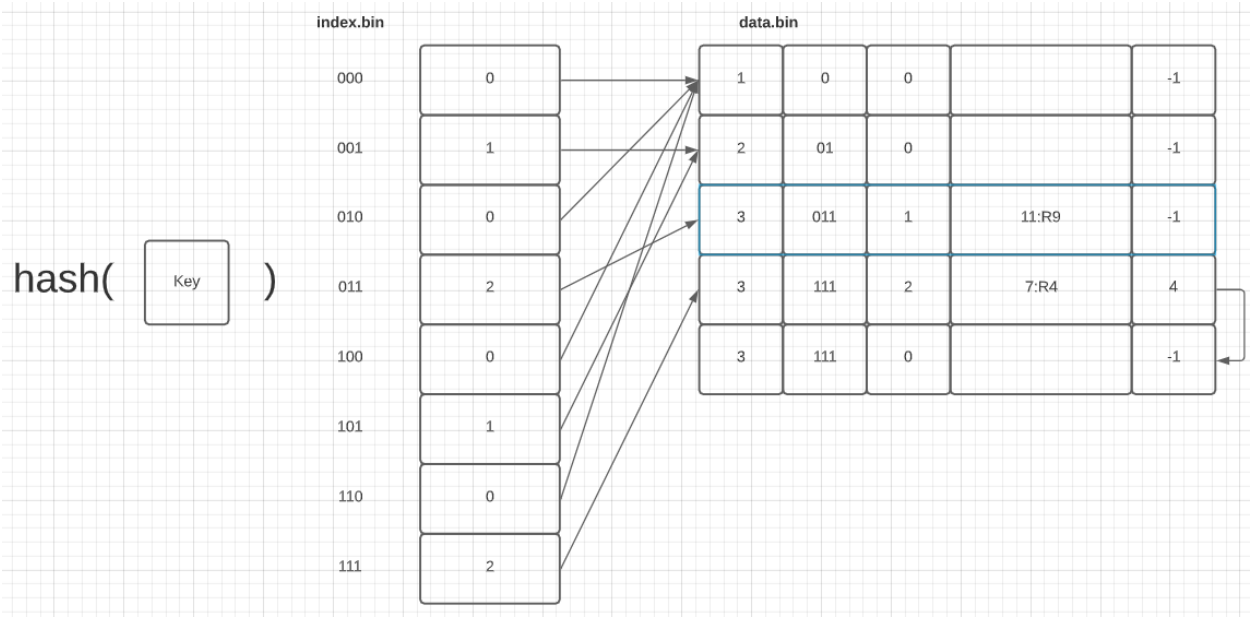
- **Eliminación:** Los registros son eliminados lógicamente, más no físicamente. Asimismo, se eliminan los registros por su llave. Entonces, primero se abre el archivo binario y se busca la posición del registro a eliminar. Luego se comprueba si se encontró la posición del registro. Si no se encontró y la posición retornada es menor a cero, significa que el registro no esta en el dataset. Si no se encontró y la posición es mayor a cero se lee el registro en esa posición. Si ese registro es el ultimo entonces el registro no esta en el dataset. Si no es ultimo se lee el siguiente registro a este. Ahora, mientras el siguiente registro este entre los registros auxiliares, seguimos buscando el registro a eliminar. Si el registro es encontrado, se elimina colocándole -2 a su atributo next y el next de su registro previo tendra el a anterior next del registro a eliminar. Después, simplemente se escriben estos registros. Sin embargo, si el registro no esta entre los registros auxiliares, simplemente no esta en el dataset. Ahora bien, si la posición del registro que se quiere eliminar sí es encontrada leemos su registro físicamente anterior y el registro siguiente a este. Seguidamente, se leen los registros desde el ultimo leído buscando el que se quiere eliminar. Finalmente, cuando se encuentre el registro a eliminar, se cambia su next a -2 y el next de su registro previo se cambia al next anterior del registro a eliminar y se escriben estos cambios.
- **Búsqueda puntual:** Para buscar un registro por su llave (key), leemos el archivo binario y aplicamos binary search para hallar la posición del registro. Si el registro no es encontrado, lo leemos y mientras el siguiente registro este entre los auxiliares, se sigue buscando la llave. Por otro lado, si sí hallamos la posición del registro, leemos esa posición y retornamos el registro.
- **Búsqueda por rango:** Se usa un binary search, aprovechando que el archivo esta ordenado físicamente. Primero, verificaremos que el ultimo valor del rango sea mayor al valor inicial de este. Si no es así, retornaremos error, si sí es así continuamos. Leemos el archivo binario y hacemos binary search para encontrar la posición del primer valor del rango. Si la posición no es encontrada se verifica si es menor a cero o mayor. Si es menor, vamos a la posición del primer registro, de lo contrario, vamos a la posición del registro siguiente. Si la posición sí fue encontrada simplemente vamos hacia esa posición. Después, se lee la posición en la que se este en el archivo y mientras no se este al final del file y la llave del registro que se lee sea

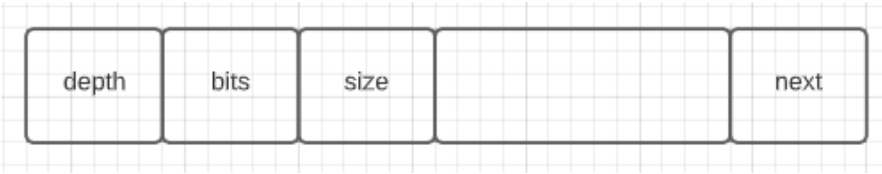
menor o igual que el valor final del rango se inserta este registro a un vector. Cuando se termine de leer todos los registros encontrados que estén dentro del rango, se retorna el vector con todos estos registros.

Extendible Hash

El Extendible Hash es una técnica de indexación de archivos que funciona apartir de una función hash, un directorio y una secuencia de buckets donde se almacenan los registros. La función hash recibe la llave del registro y devuelve una dirección en el directorio. Esta dirección nos muestra la posición de un bucket en otro archivo destinado a ellos. El elemento es insertado en el bucket. La inserción tiene casos particulares de veremos más adelante.

Búsqueda puntual: Utilizamos la función hash para encontrar la dirección a consultar en el directorio (index.bin). Dentro del directorio en la dirección previamente calculado encontraremos la ubicación de nuestro bucket en el archivo "data.bin". El archivo "data.bin" esta repleto de buckets, estos buckets contiene valores que representan la profundidad, los bits correspondientes, tamaño y la dirección al siguiente bucket. Una vez encontrado preguntamos si dentro del bucket se encuentra un registro con la llave deseada, si no lo encuentra buscamos en el siguiente bucket que pueda tener anexado.





- **Búsqueda por rango:**

Hacemos una serie de busquedas puntuales en el rango dado.

```
std::vector<R> rangeSearch(K keyA, K keyB) {
  if (keyA > keyB) throw std::invalid_argument("Second key must be greater or equal to the first one.");
  std::vector<R> result{};
  for (K k = keyA; k <= keyB; ++k) {
      auto e = search(k);
      if (e.first == true) result.push_back(e.second);
  }
  return result;
}
```

- **Inserción:**

En el caso de la inserción necesitamos conseguir el bucket donde debería estar.

- **Eliminación:**

Experimentaciones

Para este proyecto, medimos los resultados experimentales bajo el análisis de dos métricas:

Análisis de accesos a memoria secundaria

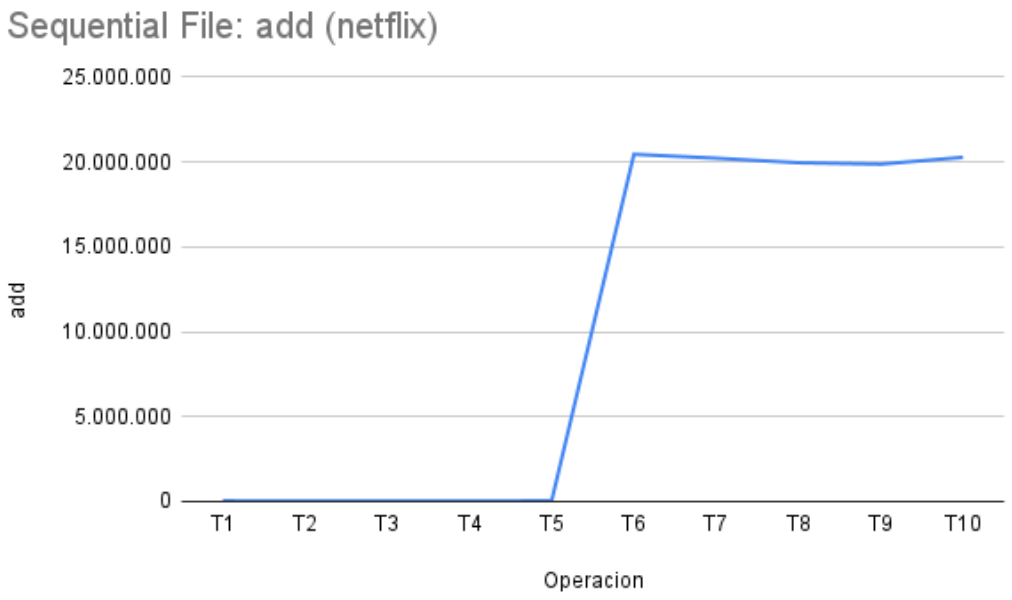
En el Sequential File los accesos a memoria secundaria de todas las operaciones poseen un log(n), siendo n la cantidad de registros válidos, pues en cada operación se requiere hacer una búsqueda binaria. En una inserción regular se hace una búsqueda binaria para encontrar la posición donde insertar el nuevo registro y se maneja el caso en el que se deba insertar un registro al inicio, aunque en este último el número de accesos es constante, dando como complejidad un O(log(n)). Sin embargo, el peor caso en la inserción ocurre cuando se debe hacer una reconstrucción. Debido a que la reconstrucción debe leer todos los registros para escribirlos ordenadamente en otro archivo, el número de accesos a memoria secundaria va a ser O(n + k), siendo k el número de registros auxiliares. Asimismo, no se considera el ordenamiento que se hace en los registros auxiliares durante la reconstrucción ya que este se hace en memoria principal. En la eliminación y la búsqueda puntual el número de accesos es O(log(n) + k) pues primero se debe encontrar la posición del registro a buscar o eliminar usando una búsqueda binaria en los registros válidos (log(n)) y, en caso de que no se encuentre, se debe verificar que el siguiente registro no pertenezca a la región auxiliar, ya que de ser así se debe hacer una búsqueda secuencial en los registros auxiliares vinculados al encontrado en los registros válidos a través de los punteros al siguiente registro, búsqueda que en el peor de los casos se va a hacer en todos los registros auxiliares (k). Finalmente, para la búsqueda secuencial se hace primero una búsqueda binaria en los registros válidos (log(n)) para encontrar la posición inicial de búsqueda.

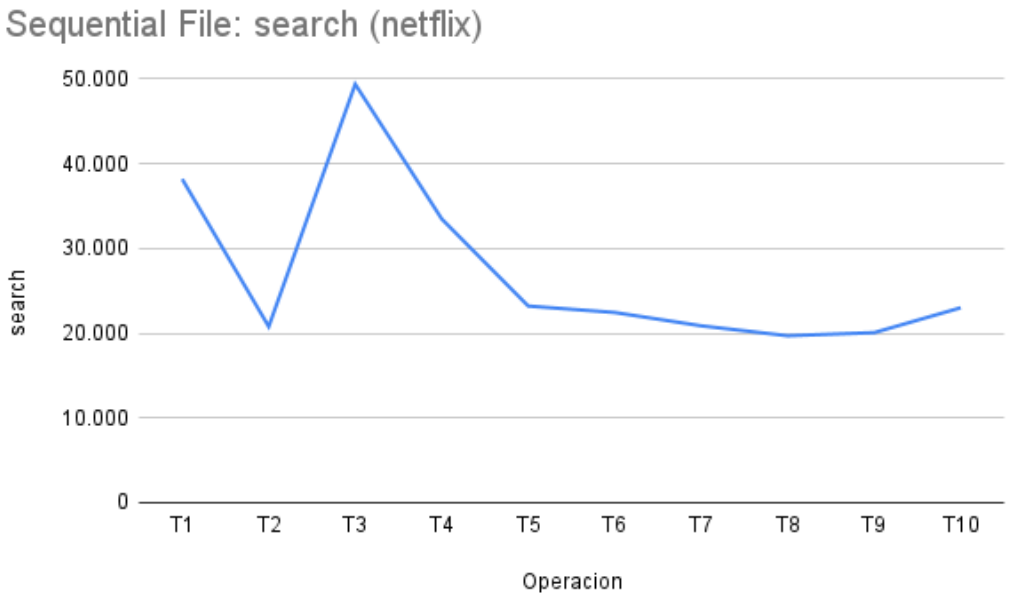
Luego, se hace una búsqueda secuencial por todo el archivo siguiendo los punteros al siguiente registro hasta llegar al límite para el rango de búsqueda. Esta búsqueda secuencial, en el peor de los casos se hace en todo el archivo (n + k) por lo que el número de accesos va a ser igual a log(n) + (n + k), es decir, O(n + k).

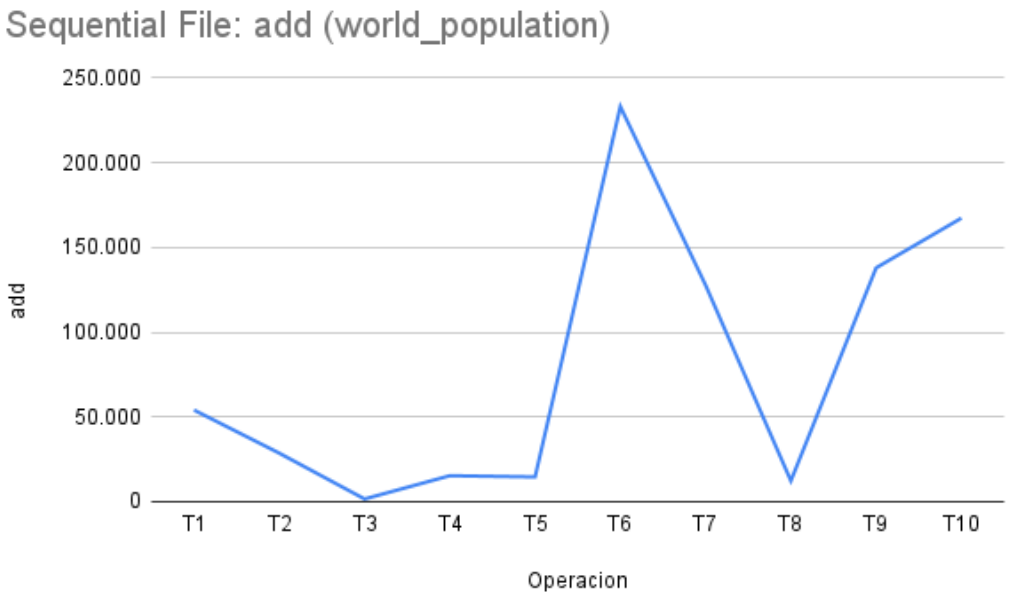
Análisis de tiempos de ejecución

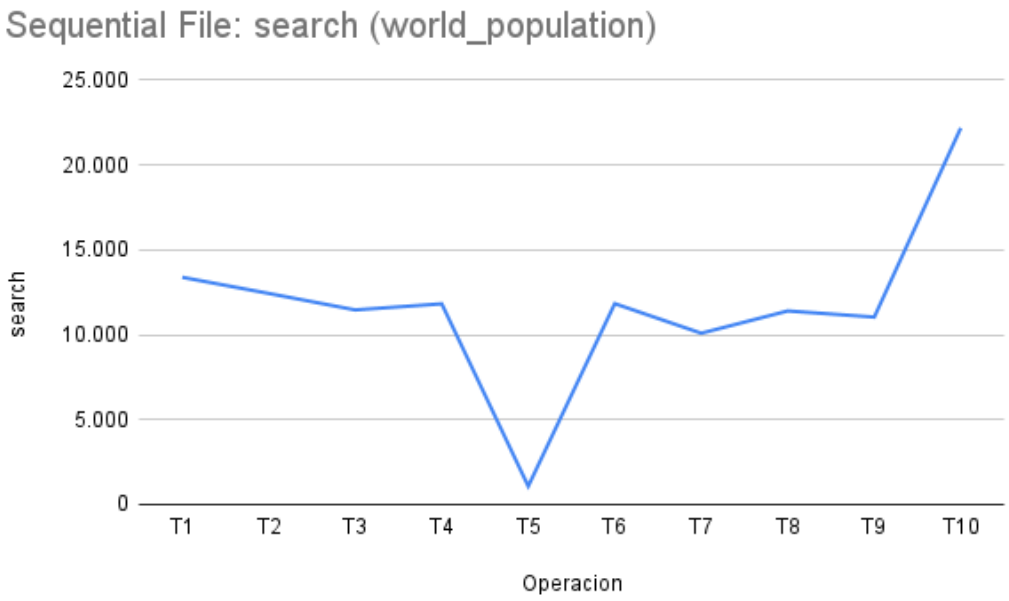
Para realizar el análisis de los tiempos de ejecución de las operaciones de inserción y búsqueda en ambas técnicas implementadas hicimos uso de las funciones testTimeNetflix, para el dataset de Netflix Prize Data, y testTimeWorld, para el dataset de World Population by Year. Estas funciones ejecutan ambas operaciones con ambas técnicas por dataset con 10 valores generados aleatoriamente, registran el tiempo de ejecución de cada operación en milisegundos, y almacenan los resultados en archivos csv. Lamentablemente, solo hemos podido hacer este testeo en el sequential file, pues tuvimos unos problemas al momento de hacerlo en Extendible hash.

Los siguientes gráficos son para los tiempos del sequential file con los datasets de Netflix y World Population. Se hicieron dos graficas por dataset, una para la función add y otra para la función search.




Sequential File: search (netflix)
50.000
40.000
30.000
20.000
10.000
0
search
T1
T2
T3
T4
T5
T6
T7
T8
T9
T10
Operacion


Sequential File: add (world_population)
250.000
200.000
150.000
100.000
50.000
0
add
T1
T2
T3
T4
T5
T6
T7
T8
T9
T10
Operacion



Pruebas de uso

Las pruebas se realizaron y documentaron en el siguiente video:

<enlace>