Databases
Project 1

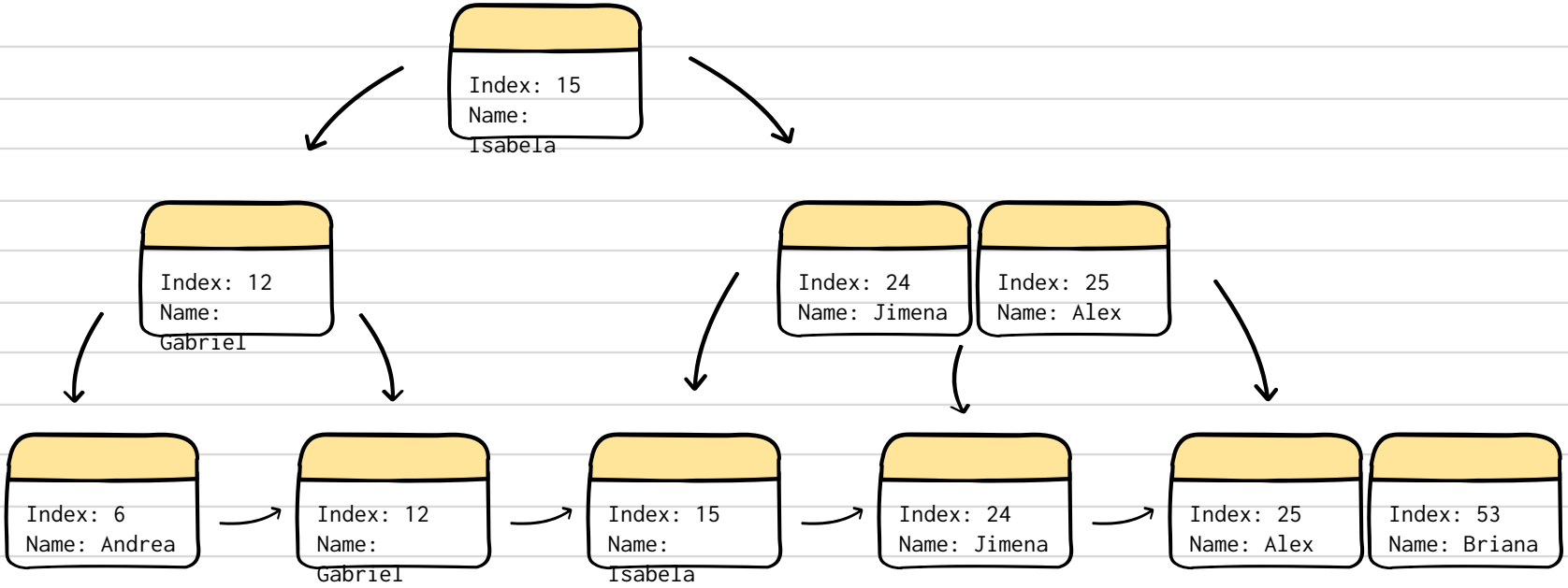
by Andrea Diaz

B+ Tree

La gran diferencia entre archivos indexados y secuenciales se encuentra en la visualización de este.

En el primero, vemos al archivo como una set de keys con las cuales podemos acceder a este mientras que en el secuencial, uno solo puede acceder al archivo secuencialmente, ir realizando operaciones en el actual o siguiente archivo que se encuentre en memoria. Los dos métodos tienen sus beneficios y desventajas, es por esto que el B+ Tree es una gran alternativa para obtener lo mejor de estos; indexando los archivos como un B Tree y accediendo a estos de manera secuencial a conveniencia.

B+ Tree structure



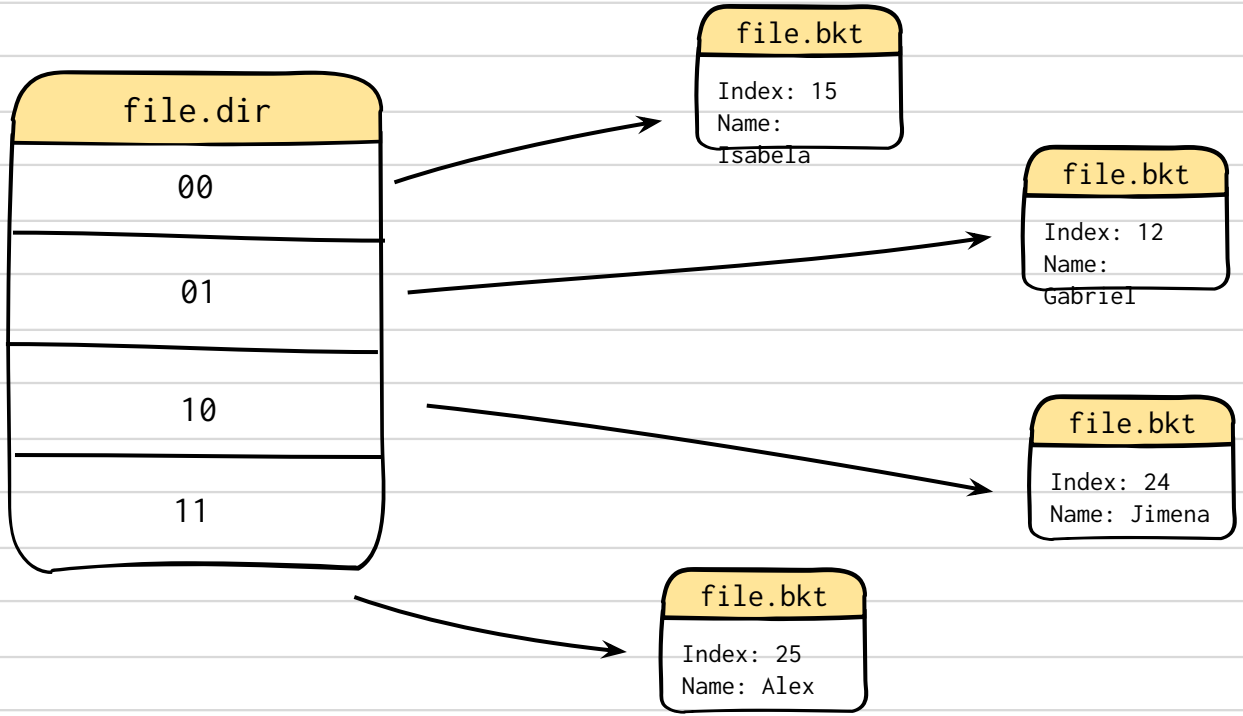
Extendible Hashing

Con la introducción del B+ Tree el tiempo de acceso a archivos mejoró a O(lgk n) y si bien ya es un buen performance, este puede mejorar a O(1) gracias al Hashing.

El core del tiempo constante por operación del Hash se encuentra en la creación de las adress cada vez que una nueva key es introducida, con esta se facilita el storing y recovery.

Si bien el addressing nos facilita muchas cosas uno de los grandes problemas de este método de indexación es el collision, dos keys pueden generar iguales hash addresses.

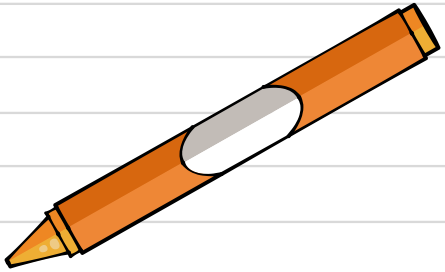
Extendible Hashing structure



01

B+ Tree

Explicando la implementación



Clases utilizadas

Btree

Estructura principal

Nodo

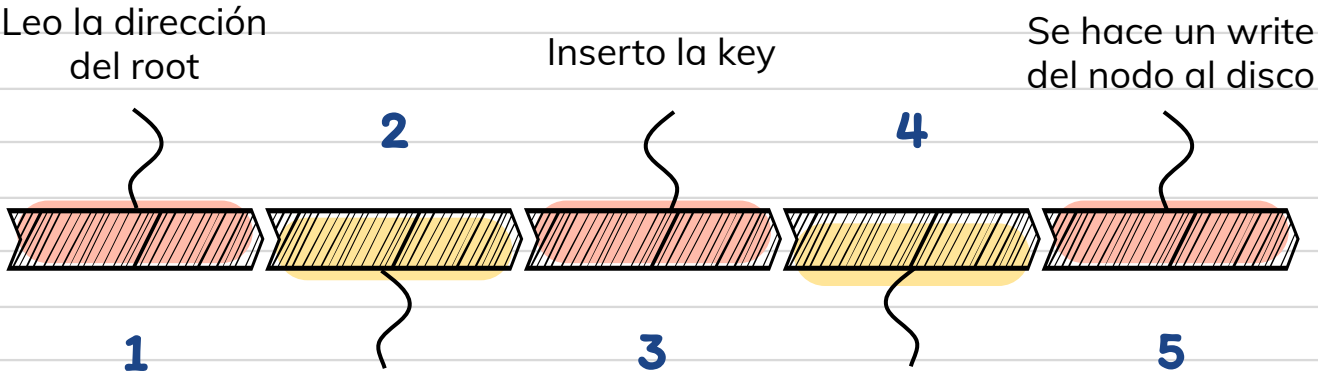
Encargado de mantener las keys y la capa anterior a el file

$$a^2+b^2=c^2$$

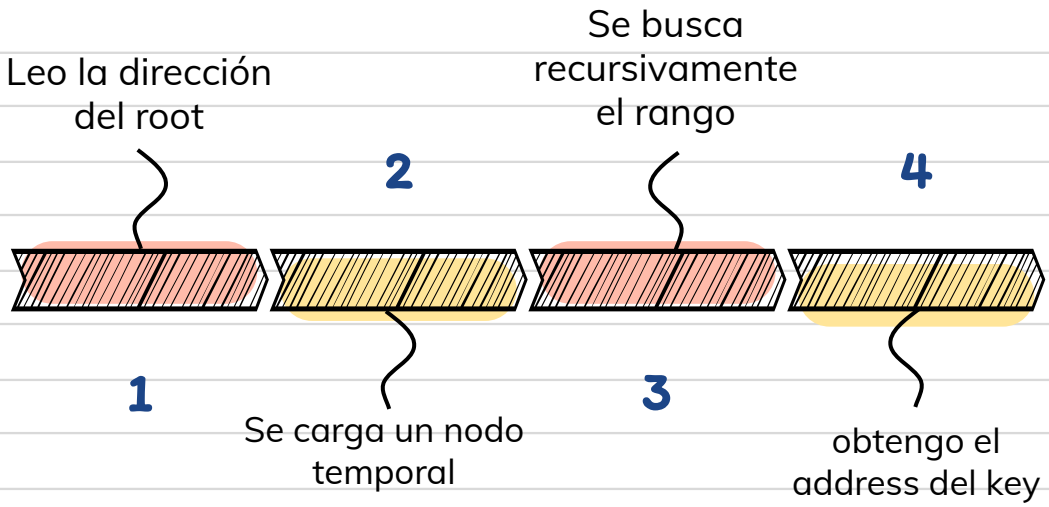
Page Manager

Todas las operaciones de manejo del file

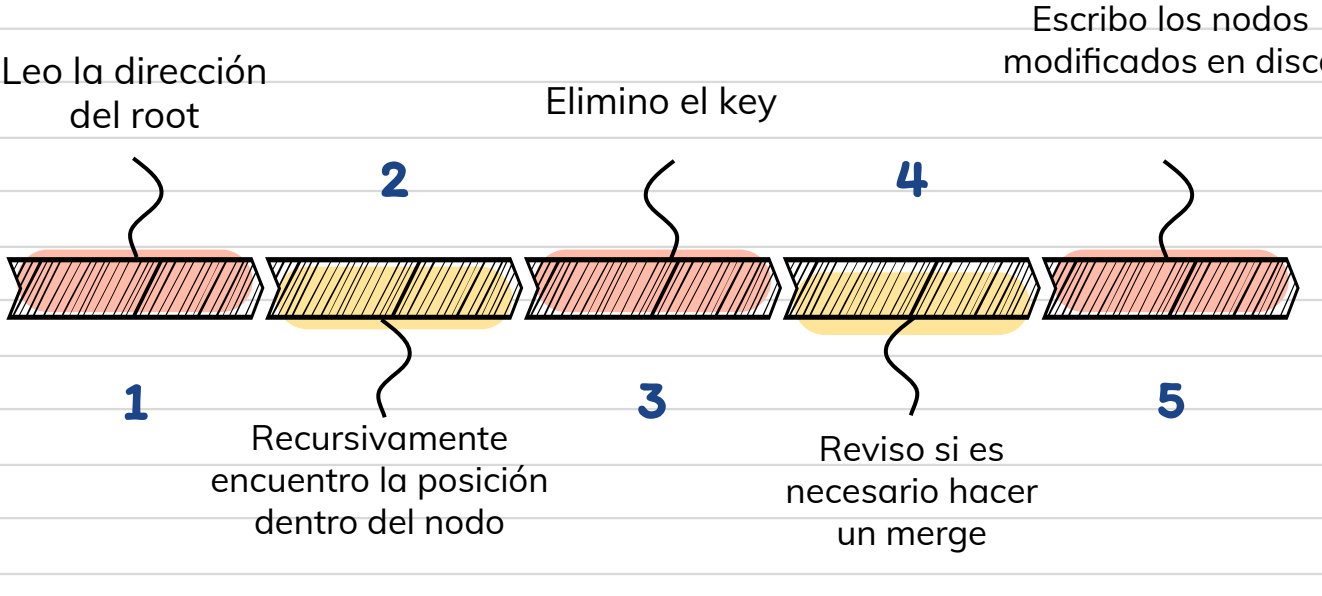
Lógica de la inserción



Lógica de la búsqueda



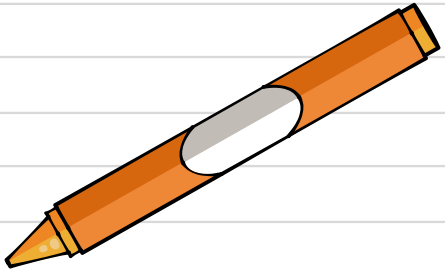
Lógica del delete



02

Extendible Hashing

Explicando la implementación



Clases utilizadas

Hashing

Estructura principal

Buckets

Encargado de mantener las keys y la estructura bucket

$$a^2+b^2=c^2$$

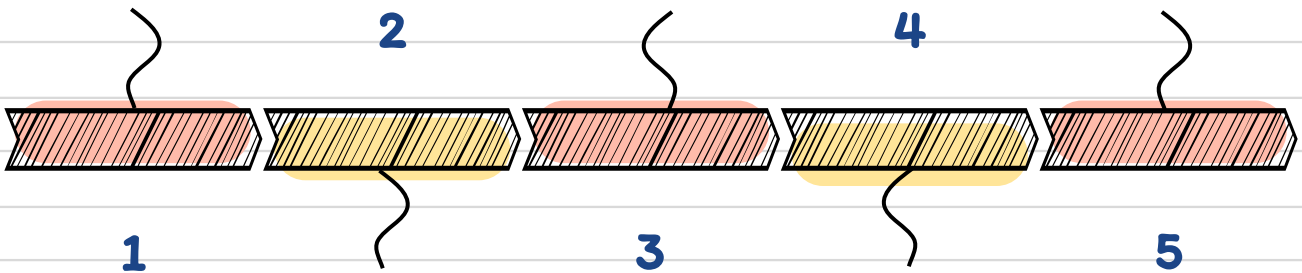
Buffer Files

Clase de control que comunica las operaciones al file

Buck Buffers

Encargado de realizar las operaciones sobre el bucket file

Lógica de la inserción



1. género la dirección del key sobre la cantidad actual de niveles
2. uso ese valor como índice en el array de bucket_addr para obtener el bucket al que pertenece.
3. traigo ese bucket de memoria
4. reviso si es necesario hacer un split del bucket. Si es así, se encuentra un bucket buddy con espacio disponible y se reparten los keys
5. inserto la key y su address en el bucket

Lógica de la búsqueda

género la dirección del key sobre la cantidad actual de niveles

1

uso ese valor como índice en el array de bucket_addr para obtener el bucket al que pertenece.

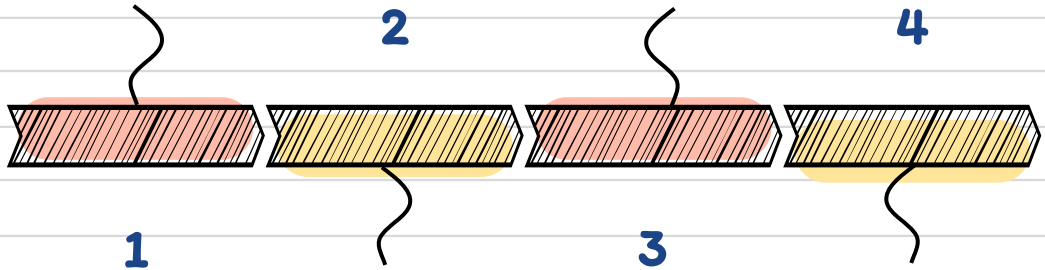
2

traigo ese bucket de memoria

3

obtengo el address del key de ese bucket

4



Lógica del delete

género la dirección del key sobre la cantidad actual de niveles

1

uso ese valor como índice en el array de bucket_addr para obtener el bucket al que pertenece.

2

traigo ese bucket de memoria

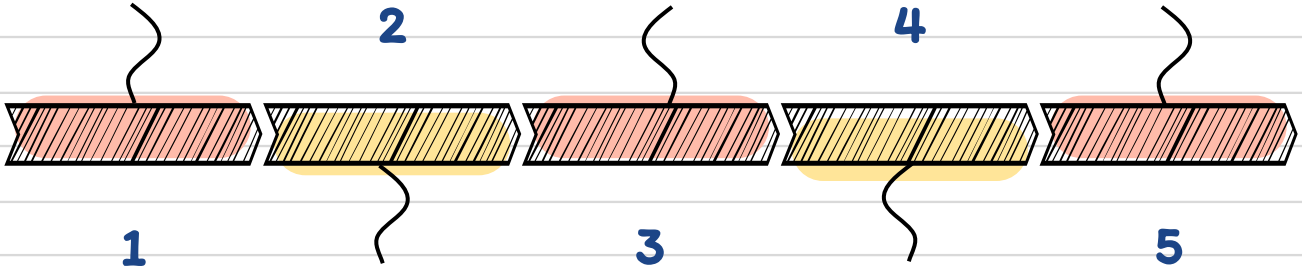
3

elimino la key y su address en el bucket

4

reviso si es necesario combinar los buckets. Si es así se combinan un bucket buddy con el current

5



03

Medidas por operaciones

Experimentación de la implementación

Inserción: B+ Tree

	1k datos	10k datos	20k datos
1	104	2437	4106
2	550	2204	2647
3	571	816	4085
4	106	2126	3202
5	510	814	2630
Promedio	368.2	1679.4	3334

Inserción: Hash

	1k datos	10k datos	20k datos
1	15	113	287
2	17	102	267
3	20	105	268
4	15	104	409
5	18	102	268
Promedio	17	103.2	299.8

Búsqueda: B+ Tree (accesos al disco)

	1k datos	10k datos	20k datos
1	11	13	14
2	11	13	14
3	11	13	14
4	11	13	14
5	11	13	14
Promedio	11	13	14

Búsqueda: Hash (accesos al disco)

	1k datos	10k datos	20k datos
1	2	2	2
2	2	2	2
3	2	2	2
4	2	2	2
5	2	2	2
Promedio	2	2	2

Delete: B+ Tree (accesos al disco)

	1k datos	10k datos	20k datos
1	13	14	16
2	12	14	15
3	12	16	14
4	12	14	14
5	12	14	15
Promedio	12.2	14.4	14.8

Delete: Hash (accesos al disco)

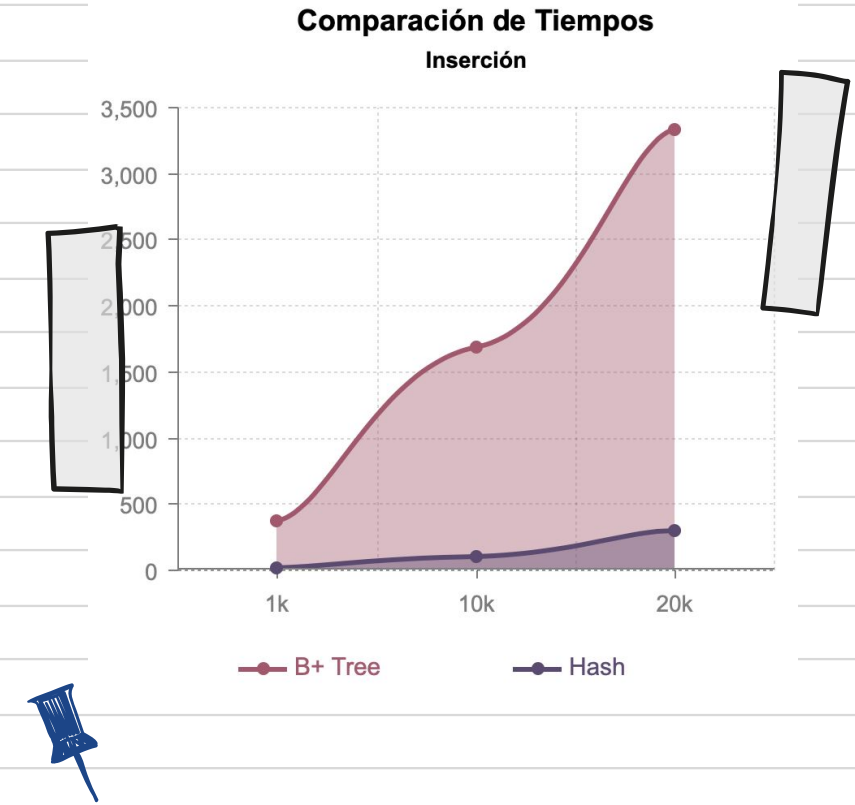
	1k datos	10k datos	20k datos
1	3	3	4
2	3	4	4
3	3	3	5
4	3	3	4
5	3	3	4
Promedio	3	3.2	4.2

04

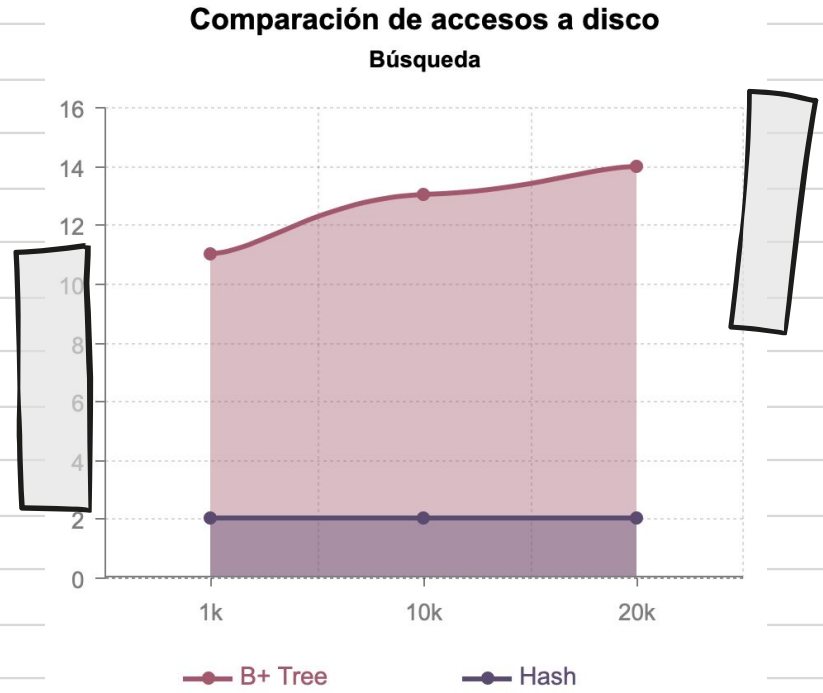
Análisis de los Resultados

Explicando los resultados

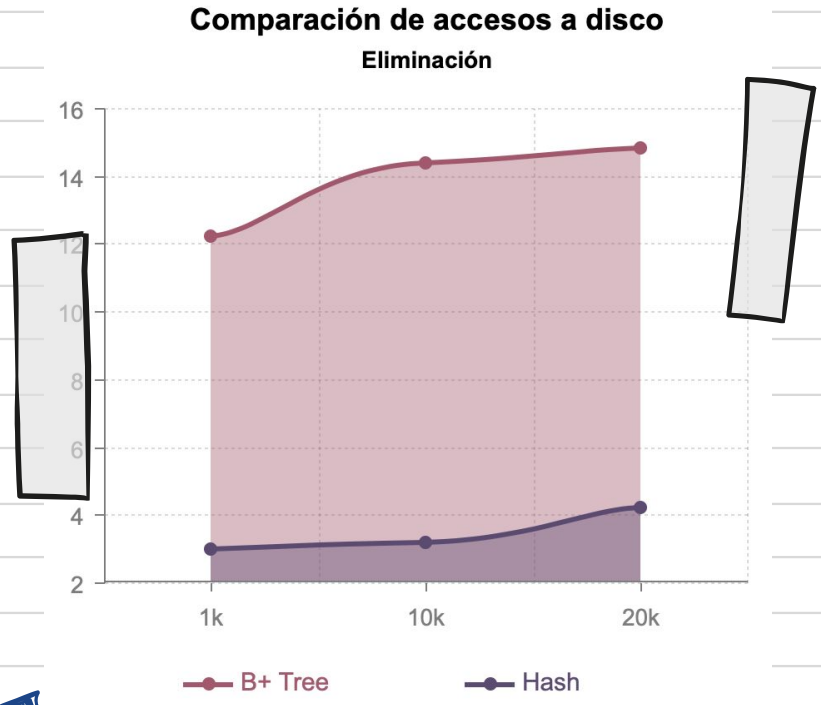
Comparación de tiempos en la inserción



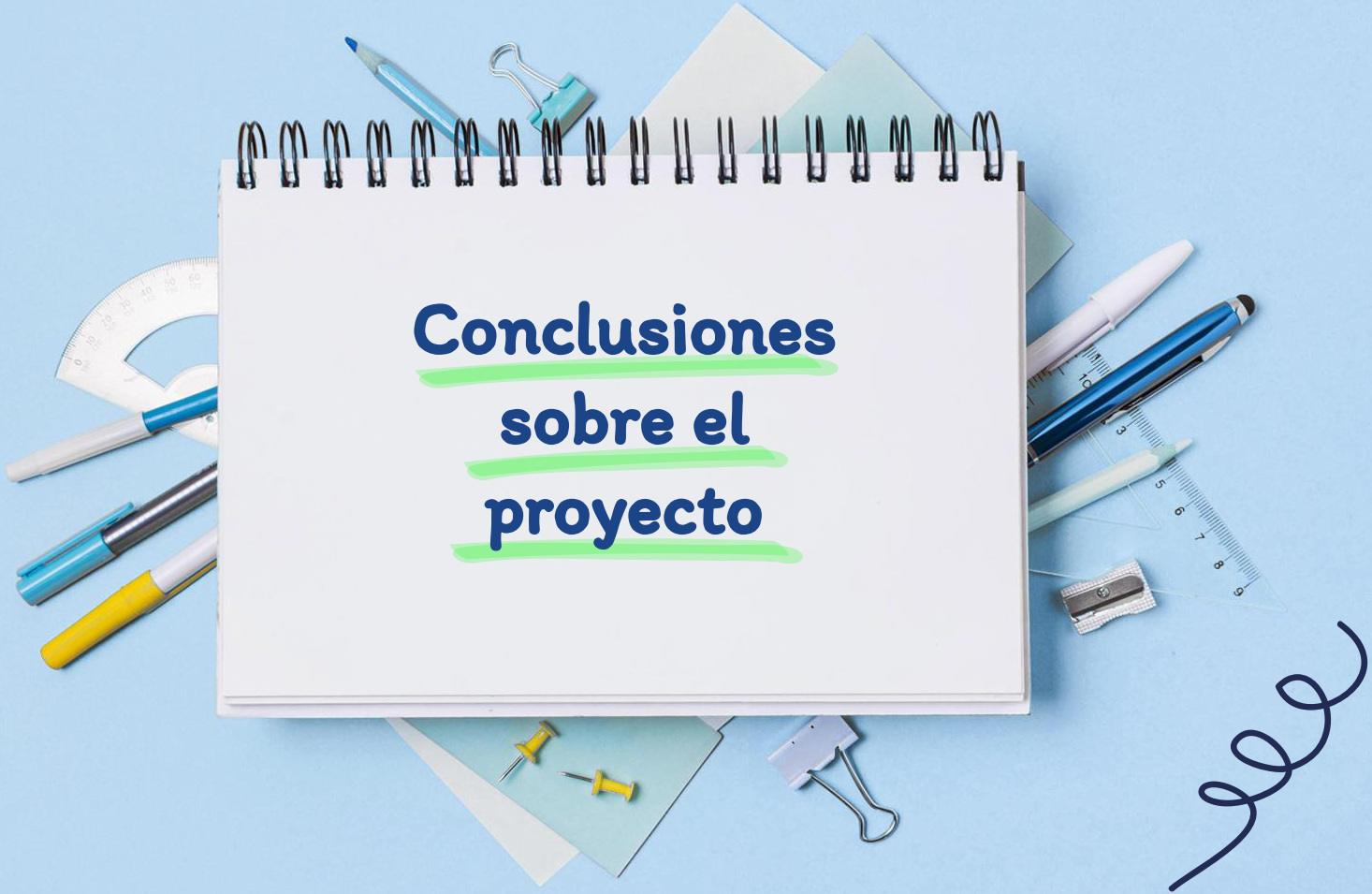
Comparación de accesos a disco en la búsqueda



Comparación de accesos a disco en el delete



Conclusiones sobre el proyecto



Conclusiones

Se pudo comprobar que el costo de recuperación de la data es, efectivamente, mejor en el Extendible Hash que en el B+ Tree, pero el problema de colisiones en la inserción causa que el Hash rinda casi igual que el B+ Tree en algunos casos.

Cabe recalcar que la elección del tipo de indexación depende de los resultados que se quieren obtener para el tipo de operaciones necesarias en su uso práctico, mientras que el Hash tiene un tiempo constante de recuperación de llaves específicas, este no es la primera opción si se necesitaran búsquedas de rango repetidamente.

Gracias!

Quieres ver el código?

Repo en github.com
Nuestra wiki