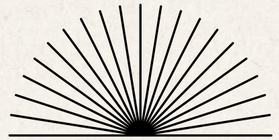


Esto será subido a Github

HASHING
B+ TREES

Joaquín - Steven - Elías - Alexander - Paula

IIC2133
MINIONS



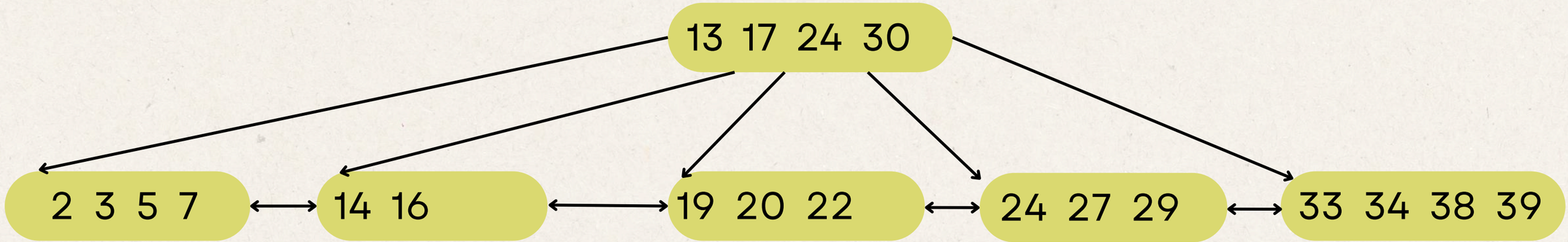
RESUMEN
B+ TREES

B+ TREES

Estructura de datos en árbol balanceado usada en sistemas de bases de datos y archivos, optimizada para lecturas y escrituras en disco.

Detalles

- **Balance**: Todas las hojas están al mismo nivel.
- **Nodos internos**: Solo almacenan claves para guiar la búsqueda.
- **Hojas**: Contienen todas las claves y apuntan a los registros (o punteros). Además, están enlazadas.
- **Uso**: Todos los nodos tienen un uso **mínimo** del 50% excepto la raíz
- **Orden:** Un B+ de orden ***d*** asegura que cada nodo interno (excepto raíz) tenga entre *d y 2d* hijos, manteniendo el balance.



APLICACIÓN B+ TREES

Contexto

Suponga que tiene que almacenar un conjunto de registros, cada uno de los cuales representa datos de una persona. En particular, por cada persona se almacena su RUT, teléfono, email y nombre. Suponga que sobre un conjunto de datos de ese tipo se tienen los dos siguientes árboles de búsqueda:

Rut-Index	Un árbol B+ como el visto en clases, cuyo directorio (nodos internos) están ordenados por el RUT de las personas y sus nodos externos (hojas) almacenan los registros de forma ordenada
Fono-Index	Un árbol cuyo directorio (nodos internos) son como el de un árbol B+ y que está ordenado por teléfono, y en cuyos nodos externos los registros se almacenan desordenados

Consejo

Comprender qué tienes disponible para usar en tu solución

a) Escribir algoritmos usando B+

RangoRUT(i, f)	Trabaja sobre Rut-Index, devolviendo un arreglo (o lista) con todos los registros tal que el RUT se encuentra en el rango [i, f].
RangoFono(i, f)	Trabaja sobre Fono-Index, devolviendo un arreglo (o lista) con todos los registros tal que el teléfono se encuentra en el rango [i, f].

Consejo

Se recomienda modificar el algoritmo correspondiente de búsqueda en rango de los árboles B+.

~~a) Escribir algoritmos~~ SOLUCIÓN

```
GetHoja(T, i):
    if T es hoja:
        return T
    s ← hijo entre llaves k_1, k_2 tal que k_1 <= i <= k_2
    return GetHoja(s, i)
```

Consejo

Se recomienda modificar el algoritmo correspondiente de búsqueda en rango de los árboles B+.

~~a) Escribir algoritmos~~ SOLUCIÓN

RangoRUT(i, f):

```
T ← GetHoja(RutIndex, i)
m ← T.start el menor de la hoja
A ← lista vacía
while m < i:
    m ← m.next
    M ← m
    while M ≤ f :
        A.add(M)
        M ← M.next
return A
```

GetHoja(T, i):

```
if T es hoja:
    return T
s ← hijo entre llaves k_1, k_2 tal que k_1 <= i <= k_2
return GetHoja(s, i)
```

Consejo

Se recomienda modificar el algoritmo correspondiente de búsqueda en rango de los árboles B+.

~~a) Escribir algoritmos~~ SOLUCIÓN

RangoFono(i, f)

```
T1 ← GetHoja(fono-index, i)
T2 ← GetHoja(fono-index, f )
A ← lista vac´ıa
foreach T in lista T1 → . . . → T2 do
   foreach m in T do
      if i ≤ m ≤ f :
         A.add(m)
return A
```

GetHoja(T, i):

```
if T es hoja:
    return T
s ← hijo entre llaves k_1, k_2 tal que k_1 <= i <= k_2
return GetHoja(s, i)
```

Consejo

Se recomienda modificar el algoritmo correspondiente de búsqueda en rango de los árboles B+.

b) Escribir Insert(r)

Insert(r)	Escriba el algoritmo Insert(r), el cual sea una modificación del algoritmo de inserción en árboles B+ para la inserción de un nuevo registro r representando los datos de una persona. Dicho algoritmo debe actualizar ambos índices mencionados.

~~b) Eseribir Insert(r)~~ SOLUCIÓN

Insert(r):

InsertClasico(Rut-index, r.rut, r)
InsertDesordenado(r.fono, r)

Detalle

InsertDesordenado opera igual que insertClasico pero ubica el dato en la última posición libre dentro de la hoja que le corresponde, los split operan de la misma manera.

RESUMEN HASHING

HASHING

Hashing convierte información en un valor entero (hash) mediante una función determinística, que sirve para ubicar los datos en una tabla de manera eficiente (O(1) promedio). No olvidar que pueden haber colisiones.

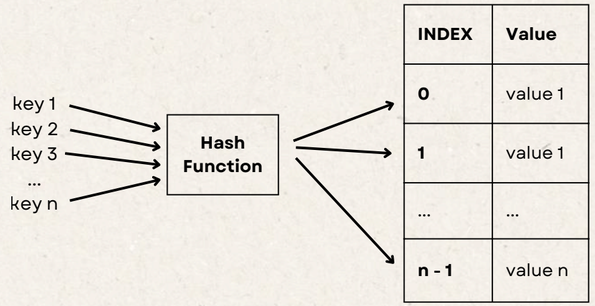


Tabla de Hash

EDD para almacenar los datos
key → hash → data

Colisiones

Encadenamiento, Sondeo lineal, Otros

Factor de Carga

$$\alpha = n/m$$

Razón entre #elementos almacenados (n) y el *table size (m)*.

APLICACIÓN HASHING

Contexto a)

Suponga que en una tabla (**inicialmente vacía**) de tamaño m = 10 **se han insertado** los elementos **a, b, c, d, e, f, g, h, i, en ese orden.** Se usó una estrategia de resolución de colisiones tal que si para una llave k se tiene h(k) = H, entonces se sondean las posiciones H + 3 · i, con i = 0, 1, . . ., aplicando la operación módulo como se revisó en clases. Luego de insertar todos los elementos en la tabla, el resultado es el siguiente:

g	**d**	**b**	**i**	**e**	**a**		**f**	**c**	
(5)	(1)	(0)	(7)	(1)	(0)		(2)	(2)	

Debajo de cada elemento en la tabla se muestra (entre paréntesis) la cantidad de colisiones que ocurrieron al momento de insertar el elemento correspondiente. Por ejemplo, al insertar el elemento **c** ocurrieron 2 colisiones. A partir de los datos indicados, indique el valor de la función h para cada elemento de la tabla, de manera que al insertar los elementos en el orden dado se obtenga la tabla indicada

Consejo

COMPRENDER cómo funciona la operación módulo ~ llevar en el resumen

~~Contexto a)~~ SOLUCIÓN

Estrategia de resolución de colisiones
La tabla es de tamaño m = 10 y usa sondeo lineal con salto 3:

Posiciones: $H, (H+3) \bmod 10, (H+6) \bmod 10, (H+9) \bmod 10, \ldots$

Es decir, si el índice original H está ocupado, el algoritmo prueba la casilla H+3, luego H+6, etc., siempre en módulo 10.

Ejemplo con h(e), nos tiene que dar $(H + 3) \bmod 10 = 4$ → H debe ser 1 para que se cumpla → **h(e) = 1**

g	**d**	**b**	**i**	**e**	**a**		**f**	**c**	
(5)	(1)	(0)	(7)	(1)	(0)		(2)	(2)	

~~Contexto a)~~ SOLUCIÓN

h(a) = 5 | h(b) = 2 | h(c) = 2 | h(d) = 8 | h(e) = 1 | h(f) = 1 | h(g) = 5 | h(i) = 2

g	**d**	**b**	**i**	**e**	**a**		**f**	**c**	
(5)	(1)	(0)	(7)	(1)	(0)		(2)	(2)	

Estrategia de **resolución de colisiones** tal que
si para una **llave k** se **tiene h(k) = H**, entonces se sondean las posiciones H + 3 · i, con i = 0, 1, . . .,
aplicando la operación módulo como se revisó en clases

Contexto b)

Queremos saber cuáles son las palabras más frecuentes que aparecen en la novela Fortunata y Jacinta, de Benito Pérez Galdós. Explica cómo abordar esta pregunta eficientemente usando tablas de hash, teniendo presente las siguientes condiciones: Fortunata y Jacinta es una de las novelas más extensas de la literatura española; contar “a mano” no sirve. No sabemos cuántas palabras tiene la novela, ni tampoco cuántas palabras distintas tiene. Dispones de una función, f, que recibe un string de caracteres y devuelve un número entero en un rango muy grande. Según el método de resolución de colisiones que elijas usar, y para que el desempeño de la tabla sea eficiente, el factor de carga no puede ser mayor que 1/3 en un caso, o mayor que 3 en el otro caso (tú tienes que saber cuál es cuál). **Específicamente, explica i, ii, iii y iv**

Contexto b)

i)	¿Cómo hay que ir ajustando el tamaño m de la tabla a medida que se van procesando más y más palabras de la novela? La idea es que se respete el limite superior para el factor de carga, pero también es importante no desperdiciar memoria como consecuencia de una tabla demasiado grande.
ii)	Exactamente, ¿Cómo se procesa una palabra? Es decir, ¿qué debe hacer tu programa cuando lee la próxima palabra? ¿Cómo se usan la función f y el tamaño m de la tabla?
iii)	Exactamente, ¿Cómo se procesa una palabra? Es decir, ¿Qué debe hacer tu programa cuando lee la próxima palabra? ¿Cómo se usan la función f y el tamaño m de la tabla?
iv)	Exactamente, ¿Cómo se procesa una palabra? Es decir, ¿Qué debe hacer tu programa cuando lee la próxima palabra? ¿Cómo se usan la función f y el tamaño m de la tabla?

~~Contexto b)~~ SOLUCIÓN i)

i)	¿Cómo hay que ir ajustando el tamaño m de la tabla a medida que se van procesando más y más palabras de la novela? La idea es que se respete el limite superior para el factor de carga, pero también es importante no desperdiciar memoria como consecuencia de una tabla demasiado grande.
RESPUESTA	La idea básica es definir un m inicial, más o menos arbitrariamente, e ir ajustándolo –aumentándolo a aproximadamente el doble– mediante rehashing cada vez que sea necesario, es decir, cuando el factor de carga λ se aproxime mucho, o sobrepase levemente, el valor 3 (si usamos resolución de colisiones mediante encadenamiento).

~~Contexto b)~~ SOLUCIÓN ii)

ii)	Exactamente, ¿Cómo se procesa una palabra? Es decir, ¿qué debe hacer tu programa cuando lee la próxima palabra? ¿Cómo se usan la función f y el tamaño m de la tabla?
RESPUESTA	Definimos una tabla de hash T de tamaño m, inicialmente vacía; un contador n de palabras distintas almacenadas en T, con valor inicial 0; y un factor de carga λ, también con valor inicial 0. Cuando se lee la próxima palabra s de la novela, se calcula su valor de hash, h, y se mira la tabla, en particular la casilla T[h]. Si T[h] está vacía, entonces se agrega s a T[h], con s.count = 0, se incrementa n, y se re-calcula λ. Si T[h] no está vacía y una de las palabras almacenadas allí es s, entonces se sólo incrementa s.count. Los m's deberían ser números primos (por eso el aumento en la letra a) no es exactamente al doble), en particular si la función de hash consiste en aplicar la operación módulo m al número entregado por la función f para la palabra –el string– que se está procesando.

~~Contexto b)~~ SOLUCIÓN iii)

iii)	Exactamente, ¿Cómo se procesa una palabra? Es decir, ¿Qué debe hacer tu programa cuando lee la próxima palabra? ¿Cómo se usan la función f y el tamaño m de la tabla?
RESPUESTA	Si cuando se mira T[h], esta celda no está vacía, pero ninguna de las palabras almacenadas en la lista ligada que empieza ahí es s, entonces hay una colisión. Entonces, hay que agregar s a la lista ligada, inicializar s.count = 0, incrementar n, y re-calcular λ.

~~Contexto b)~~ SOLUCIÓN iv)

iv)	Exactamente, ¿Cómo se procesa una palabra? Es decir, ¿Qué debe hacer tu programa cuando lee la próxima palabra? ¿Cómo se usan la función f y el tamaño m de la tabla?
RESPUESTA	Una posibilidad es declarar un array R de tamaño n (el número de palabras distintas), y recorrer la tabla T y almacenar en cada casilla de R una palabra s distinta junto a su s.count. Luego, hay que ordenar R de mayor a menor de acuerdo con los s.count; y finalmente hay que imprimir los contenidos de las primeras casillas de R. Otra posibilidad es usar un heap –un array de tamaño igual al número de palabras más frecuentes que nos interesa– que se va actualizando cada vez que se procesa una nueva palabra.

~~Contexto b)~~ SOLUCIÓN i, ii, iii, iv

i)	La idea básica es definir un m inicial, más o menos arbitrariamente, e ir ajustándolo –aumentándolo a aproximadamente el doble– mediante rehashing cada vez que sea necesario, es decir, cuando el factor de carga λ se aproxime mucho, o sobrepase levemente, el valor 3 (si usamos resolución de colisiones mediante encadenamiento).
ii)	Definimos una tabla de hash T de tamaño m, inicialmente vacía; un contador n de palabras distintas almacenadas en T, con valor inicial 0; y un factor de carga λ, también con valor inicial 0. Cuando se lee la próxima palabra s de la novela, se calcula su valor de hash, h, y se mira la tabla, en particular la casilla T[h]. Si T[h] está vacía, entonces se agrega s a T[h], con s.count = 0, se incrementa n, y se re-calcula λ. Si T[h] no está vacía y una de las palabras almacenadas allí es s, entonces se sólo incrementa s.count. Los m's deberían ser números primos (por eso el aumento en la letra a) no es exactamente al doble), en particular si la función de hash consiste en aplicar la operación módulo m al número entregado por la función f para la palabra –el string– que se está procesando.
iii)	Si cuando se mira T[h], esta celda no está vacía, pero ninguna de las palabras almacenadas en la lista ligada que empieza ahí es s, entonces hay una colisión. Entonces, hay que agregar s a la lista ligada, inicializar s.count = 0, incrementar n, y re-calcular λ.
iv)	Una posibilidad es declarar un array R de tamaño n (el número de palabras distintas), y recorrer la tabla T y almacenar en cada casilla de R una palabra s distinta junto a su s.count. Luego, hay que ordenar R de mayor a menor de acuerdo con los s.count; y finalmente hay que imprimir los contenidos de las primeras casillas de R. Otra posibilidad es usar un heap –un array de tamaño igual al número de palabras más frecuentes que nos interesa– que se va actualizando cada vez que se procesa una nueva palabra.



Esto será subido a Github

HASHING
B+ TREES

Joaquín - Steven - Elías - Alexander - Paula

IIC2133
MINIONS

