20. TABLAS HASH

- insertar - lleva un índice
- find - revisa que el índice no sea 0
- remove - índice positivo

) tiempo constante ¿O(1)?

El Futuro se moldea

#ASCII con letras → matriz muy grande → solución → correspondencia entre un elemento y un índice de nombre función hash.

clave % TamañoTabla = índice donde va

↳ En ASCII q: j u n k

128^3 128^2 128^1 128^0

Colisión: cuando 2 elementos o más les corresponde la misma posición.

↳ Soluciones: sondeo → lineal, cuadrático
encadenamiento separado

Función Hash

$$A_3X^3 + A_2X^2 + A_1X^1 + A_0X^0$$

pag 3 apdo 2 exp

$$(((((A_3 \cdot x) + A_2) \cdot x) + A_1)x + A_0$$

- Sum y multi menos costosas que potencia → Ventaja
- mod después de c/ multi y Suma → resultados pequeño
  ↳ Si genera neg lo hacemos positivo

```
// Función hash aceptable
public static int hash(String key, int tableSize) {
    int hashVal = 0
    for (int i = 0; i < key.lenght(); i++)
        hashVal = (hashVal * 128 + key.charAt(i)) % tableSize;
return hashVal
}
```

```
// Hash para String
public static int hash (String key, int tableSize) {
   int valHash =0
   for (int i =0 ; i< key.lenght () ; i ++)
       hashVal = 37 * hashVal + key.charAt(i)
   hashVal % = tableSize
   if (hashVal <0)
       hashVal += tableSize
   return hashVal
}
```

tablas hash se comienzan indexando en 0.

La parte más costosa de las Op con tablas hash es el cálculo del código hashCode,

(c/String almacena su hashCode → se evitan los cálculos costosos
↳ almacenamiento en caché del código hash
↳ Fijarse pq String es inmutable

20.3.3 Análisis de la operación find

La op find sin éxito la secuencia de examinación es la misma que la de inserción. → respuesta inmediata

El coste medio para encontrar el primer elemento insertado en la tabla siempre será 1,0.

Sin éxito: $\left(1 + \frac{1}{(1-\lambda)^2}\right) \cdot \frac{1}{2}$

con éxito: $\left(1 + \frac{1}{(1-\lambda)}\right) \cdot \frac{1}{2}$

} Para una búsqueda con éxito calculamos el coste medio de inserción

PPal problema con el agrupamiento primario → el rendimiento se degrada enormemente cuando se realizan inserciones con factores de carga altos.

Para reducir el número de sondeos → esq. de resolución de colisiones

↓

evita agrupamiento primario

Sondeo Lineal

$H+1, H+2, H+3, H+4 \ldots Hi$

implementación Fácil

20.3 Sondeo Lineal

¿Qué hacer en una colisión?
Busca a la matriz hasta encontrar una celda vacía
Es tiempo muy largo

El Futuro se moldea

```
public final class String {
  public int hashCode() {
    if (hash != 0)
      return hash;
    for (int i = 0; i < length(); i++)
      hash = hash * 31 + (int) charAt(i);
      return hash
    }
  private int hash = 0;
}
```

find: si llega a celda vacía no se encontró el elemento,
si no está vacío pero el elemento no es el buscado => avanzamos de celda.
borrar: se los marca como borrados, no se los quita de la tabla (sol puede fallar si se los quita) → borrado perezoso
activo o borrado → estado

#Rendimiento de tabla hash depende de que tan llena esté

agrupamiento primario o clustering primario
Se forman bloques de celdas ocupadas

número medio de celdas examinadas en la inserción $(1+1/(1-\lambda)^2)/2$

$$\left(\frac{1}{1} + \frac{1}{(1-\lambda)^2}\right) \cdot \frac{1}{2}$$

tabla semillena → num medio 2,5

20.4 Sondeo Cuadrático

Resuelve colisiones y elimina el problema del agrupamiento primario

Fórmula para resolver las colisiones.

$F(i) = i^2$

$H+1^2, H+2^2, H+3^2 \dots H+i^2$.

- Ninguna celda será sondeada 2 veces si la carga nunca es mayor a 0,5 → Tamaño de tabla debe ser primo

Si se utiliza un sondeo cuadrático y el tamaño de la tabla es un número primo, entonces siempre se podrá insertar un nuevo elemento si la tabla está como mínimo mediovacía. Además en el curso de la inserción ninguna celda será sondeada 2 veces.

pág 14 Método para encontrar un número primo $\geq n$

Si el factor de carga es mayor que 0,5, se duplica el tamaño de la tabla hash.

↓

rehashing

las posiciones de los valores se reasignan en la función hash

↳ Dificultad para encontrar número primo $O(N^{1/2} \log N)$

↳ Menos costoso que crear una nueva tabla y pasar todos los datos $(O(N))$

20.4.1 Implementación en Java

HashSet, HashMap } requieren un método → HashCode

- No tiene parámetro TableSize
- Luego de usar la función hash del usuario, se realiza una op módulo al final.

equals = hashCode

tabla hash compuesta por una matriz de referencias
HashEntry → obj o null
↳ activa o borrada

HashEntry → boolean isActive } HashSet tiene un Array de HashEntry
↳ Object

findPos → implementa sondeo cuadrático

Si hay muchos elementos borrados se modifica el tamaño de la tabla hash

clear() → elimina todos los elementos del HashSet

null es un elemento válido en el HashSet

Pair es una clase anidada en HashMap.

3.3 Diccionarios

Permiten acceder a los elementos de datos por su contenido

El Futuro se moldea

Operaciones: K-clave D-diccionario

1) Buscar (D,K) - dada una clave devuelve el valor
2) Insertar (D,x) - elemento x se agrega a d.
3) Eliminar (D,x) - dado x lo elimina de d

4) Máximo(D) & Mínimo(D) - Permite que el diccionario funcione como una cola de prioridad
5) Predecesor (D,K) & Sucesor (D,K) - clave anterior o posterior

*El peor caso de tiempo de ejecución de c/operación ocurre cuando la estructura de datos es:
- Un arreglo no ordenado
- Un arreglo ordenado

#Solución: depediendo de cómo se implemente el diccionario, algunas operaciones serán rápidas y otras lentas

Operación	Array Desordenado	Array Ordenado
Search	$O(n)$	$O(\log n)$
Insert	$O(1)$	$O(n)$
Delete	$O(1)^*$	$O(n)$
Sucessor	$O(n)$	$O(1)$
Minimum	$O(n)$	$O(1)$
Maximum	$O(n)$	$O(1)$

Unsorted
- Search - Peor caso K no ser encontrada
- Insert - constante
- Delete - sobreescribir el x elegido con el último elemento
- Sucessor - Elemento más grande pero más pequeño que x } Hay que buscarlos
- Predecesor - Elemento más pequeño más grande que x

CAP 6

La API de Colecciones

El Futuro se moldea

6.1. Introducción

Una vez que cada estructura de datos ha sido implementada, se la puede emplear todas las veces que se quiera en diversas aplicaciones.

est. de datos: representación de los datos y de las operaciones permitidas con esos datos.

↳ Buscar, insertar, eliminar, isEmpty(), makeEmpty() → Op de la interfaz

6.5 La interfaz List

lista: colección de elementos en la que los elementos tienen una posición. Ej: matriz

↳ Amplia la interfaz Collection – get, set, ListIterator

6.5.1 La interfaz ListIterator

recorrer lista en sentido inverso → hasPrevious y previous / hasNe

remove - elimina next o previous

6.5.2 La clase LinkedList

ArrayList - solo inserción al final

Ventaja de LinkedList con respecto a ArrayList -
inserción y elimina en medio

Op de LinkedList : - addLast
- addFirst
- getFirst
- getLast
- element = primer elemento que encuentra
- removeFirst
- removeLast
- remove

6.5.3. Tiempo de Ejecución para las distintas listas

Op	ArrayList	LinkedList
add Final	O(1)	O(1)
remove Final	O(1)	O(1)
removeFirst	O(n)	O(1)
add First	O(n)	O(1)
contains	O(n)	O(n)
get/set	O(1)	O(n) → doble enlace

Si se usa un iterador para remover un elemento de una LinkedList, va a ser de tiempo lineal (sino es cuadrático)

6.7. Conjuntos

Un conjunto set es un contenedor que no contiene duplicados

El Futuro se moldea

remove == de difícil que contains (recorre todos los elementos)

SortedSet → Soporta todos los métodos de Set
↘ Permite encontrar Min y Max

6.7.1 La clase TreeSet

SortedSet implementado mediante TreeSet

printCollection - imprime en orden decreciente

No permite duplicados

Peor caso O(n) caso medio O(log n)

Encontrar Menor y Mayor O(log n)
K-ésimo elemento más pequeño no soportado x la API de Colecciones

6.7.2 La clase HashSet

difiere con TreeSet en que no se enumeran los elementos por orden. No se puede Buscar Min ni Max.

GERDAU
El futuro se moldea

HashSet - poder ter que el TreeSet

Implementación de equals y HashCode

a.equals(b) → a ≤ b

castea para tipos idénticos

instanceOf → son iguales si el dato que se les da es igual

6.8 Mapas

Un map se utiliza para almacenar una colección de entradas formadas por sus claves y sus valores.

El Map asigna claves a valores.

- claves deben ser diferentes. clave1 → valor ; clave2 → valor ; ≠ claves = valor ✓

SortedMap - mantiene el mapa ordenado según las claves

2 Implementaciones - HashMap - claves desordenadas
- TreeMap - claves ordenadas

Interfaz Map NO amplia Collection, es independiente

Op - size()
- isEmpty()
- containsKey()
- get() - key / value
- put()
- remove()
- clear()

* Se permiten valores null

* No tiene iterador, tiene Collection