



Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: TISTA GARCÍA EDGAR

Asignatura: ESTRUCTURA DE DATOS Y ALGORITMOS II

Grupo: 5

Número de Práctica(s): Guía práctica de estudio 5: Algoritmos de Búsqueda Parte II

Integrante(s): MURRIETA VILLEGAS ALFONSO

*Núm. De Equipo de
cómputo empleado :* 38

Semestre : 2019 - 1

Fecha de entrega: 18 DE SEPTIEMBRE DE 2018

Observaciones:

CALIFICACIÓN: _____

ALGORITMOS DE BÚSQUEDA. PARTE II

OBJETIVOS DE LA PRÁCTICA

- El estudiante conocerá e identificará algunas de las características necesarias para realizar búsquedas por transformación de llaves.

INTRODUCCIÓN

Recuperar información de una computadora es una de las actividades más útiles e importantes, es muy comúnmente se tiene un nombre o llave que se quiere encontrar en una estructura de datos tales como una lista u arreglo. Al proceso de encontrar un dato específico llamado “CLAVE” en alguna estructura de datos, se denomina búsqueda. Este proceso termina exitosamente cuando se localiza el elemento que contiene la llave buscada, o termina sin éxito cuando no aparece ningún elemento con esa llave.

Los algoritmos de búsqueda se pueden clasificar en:

- 1) Algoritmos de comparación
- 2) Algoritmos de transformación de llaves

Algoritmos de transformación de llaves

Un diccionario es un conjunto de pares (llave, valor), siendo una abstracción que vincula un dato con otro. En estos se pueden realizar operaciones de búsqueda, inserción y borrado de elementos a través de una llave. Es de esta forma que una Tabla Has o Hashtable es una estructura de datos para implementar un tipo de dato abstracto (TDA) diccionario.

El algoritmo o método de búsqueda por transformación de llaves los datos son organizados con ayuda de una tabla hash, la cual permite el acceso a los datos mediante una llave que indica la posición donde están guardados. Para esto es necesario de una función que transforme la llave o dato clave en una dirección dentro de la estructura tabla, esta función es conocida como “hash”

Dicho de forma matemática, para determinar un elemento con una llave x se aplica una función hash

$$h(x)$$

para de esta forma obtener la dirección en la tabla.

Sólo si h mapea o contiene a las llaves U donde U es el universo donde se contienen todas las llaves para la tabla hash.

$$h: \rightarrow [0,1,2,3 \dots n - 1]$$

Como se puede apreciar en el diagrama inferior, en ocasiones se puede generar una colisión que se define como una misma dirección para dos llaves o más llaves distintas, sin embargo, esto puede resolverse fácilmente.

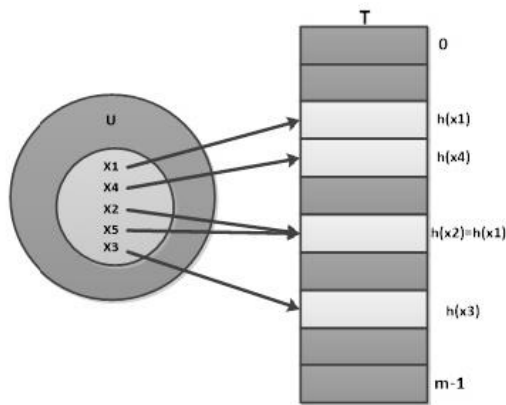


Imagen 1: Diagrama de Ven donde se muestra la relación entre el espacio U y el espacio T

La función hash es del tipo biyectiva es decir que a cada elemento le corresponde un único y solo un único elemento, dicho en el caso de esta función solamente existe una dirección para una sola llave y así recíprocamente.

Para poder emplear este algoritmo para búsqueda es necesario considerar 2 aspectos importantes:

- 1) Calcular el valor de la función de transformación, la cual transformara la llave de búsqueda en una dirección o entrada a la tabla hash.
- 2) Disponer de un método para tratar las colisiones. Dado que la función hash puede generar la misma dirección para diferentes llaves.

Universo de Llaves

Cabe destacar que si quiere utilizarse letras al momento de emplear llaves en el hash es necesario asignarles una posición numérica lo cual se puede expresar como $\text{ord}(x)$

$$K = \sum_{i=1}^n \text{Clave}[i](p[i])$$

Las llaves se pueden obtener en forma de enteros positivos a través de la sumatoria anterior, donde n es el número de caracteres de la llave, $\text{clave}(i)$ corresponde a la representación ASCII del i-ésimo carácter y $p(i)$ es un entero procedente de un conjunto de pesos generados aleatoriamente para un intervalo entre 1 y n donde i es un elemento entre ambos elementos

Funciones Hash

Una buena función hash es aquella que realiza una distribución o asignación de direcciones uniforme, es decir que para cada llave de entrada cualquiera de las posibles salidas tenga la misma probabilidad de suceder.

Para llevar a cabo esta función es necesario llevarla a través de funciones de transformación mediante distintos métodos, a continuación, se mencionarán algunos con sus respectivas características:

1) Método de división

Se mapea una llave x en una celda de la tabla tomando el residuo de x dividido entre m (Por lo regular m no debe ser potencia de 2), esto es comúnmente conocido como modulo.

$$h(x) = x \bmod m$$

Cabe destacar que comúnmente este método con lleva un pequeño problema conocido como colisión que posteriormente será abordado.

2) Método de multiplicación

En este método se opera en 2 partes, primero multiplicamos la llave x por una constante en este caso A dada en un rango de 0 a 1, posteriormente se extrae la parte fraccional que se multiplica por m (es un valor potencia de 2), por último se obtiene el mayor número entero menor o igual al resultado obtenido

$$h(x) = [m(xA \bmod 1)]$$

3) Hashing universal

Puede darse el caso donde exista una función donde un conjunto de llaves produzca la misma salida o les sea asignado la misma dirección en una tabla. Para evitar esto es que se lleva a cabo una función hash de forma aleatoria.

Sea U un universo de llaves y sea H un conjunto finito de funciones hash que mapean U a $\{0,1,2,\dots,m-1\}$ Entonces el conjunto H es llamado universal si para toda x , y que pertenecen a U donde x diferente a y .

Resoluciones de Colisiones

Las técnicas de resolución de colisiones se clasifican en 2 principalmente.

1) Direcccionamiento abierto (Open Addressing o closed hashing)

Cuando el número de n elementos de una tabla se puede estimar con anticipación, existen métodos de almacenamiento de n registros en una tabla donde m el tamaño de esta se define a través de m siendo m mayor a n .

Una llave x se redireccionar a una entrada de la tabla que ya está ocupada, se elige la secuencia de localizaciones alternativas, esto a través de lo siguiente.

$$h_j = (h(x) + j) \bmod m \quad \text{para } 1 \leq j \leq m-1$$

$m = \text{tamaño de la tabla}$

2) Enlazamiento (Chainning u open hashing)

Consiste en colocar los elementos mapeados a la misma dirección de la tabla hash en una lista ligada, la posición o dirección j contiene un apuntador al inicio de la lista ligada que contiene todos los elementos que son mapeados a la posición j . si no hay elementos entonces la localidad tendrá un NULL.

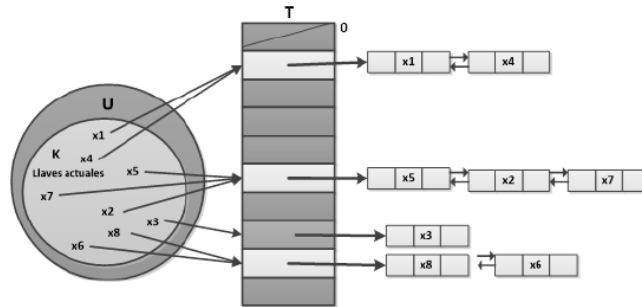


Imagen 2: Relación entre el universo de llaves, la tabla hash y las listas ligadas.

DESARROLLO

Actividad 1. Análisis y ejercicios del manual de prácticas

1.1 Tabla Hash

Para esta actividad solamente se pidió realizar el código en Python dado en el Manual y analizar paso a paso que es lo que realiza cada función de este.

- 1) La función **formaArreglo** sirve para crear una lista vacía de tamaño n .
- 2) La función **obtenerLlaveNumerica** sirve para representar la llave formada por una cadena de caracteres como un valor entero, es por ello que se suma el valor de cada carácter en ASCII y posteriormente este representará a la llave.
- 3) Como función hash esta **H** la cual es la encargada de llevar a cabo la función en su versión divisible o sea a través del módulo
- 4) Por otro lado, está la función **agregar** un elemento el cual se encarga principalmente del manejo de las colisiones originadas a lo largo de la función, los parámetros que se le pasan a esta función son la llave, el valor a insertar, la tabla hash y el tamaño de esta.
- 5) Por último, está función de **búsqueda** la cual simplemente se encarga de llevar a cabo la búsqueda de un elemento en la tabla hash, para poder llevar a cabo esta función se necesitan tanto la tabla como el valor del elemento que se quiere buscar.

Por último, se muestra la salida del programa empleando todas las funciones mencionadas anteriormente:

```
In [8]: runfile('C:/Users/AlfonsoMV/OneDrive/Documentos/
Universidad/3º Semestre/EDA II/Prácticas_LAB/5_practice/
Codes_manual/hash.py', wdir='C:/Users/AlfonsoMV/OneDrive/
Documentos/Universidad/3º Semestre/EDA II/Prácticas_LAB/
5_practice/Codes_manual')
Tabla llena 2
Tabla llena 3
Tabla llena 4
Tabla llena 1
[[['Hola4', 1221324]], [['Hola5', 1221325]], [['Hola1',
1221321]], [['Hola2', 1221322]], [['Hola3', 1221323]], None,
None, None, None]
1221322
```

Imagen 3. Salida del programa hash

Como se puede ver en la imagen anterior, se puede primero ver los resultados de la tabla llena debido a que probablemente hubo colisiones al momento de asignar direcciones, por otro lado, se encuentra los valores insertados en la tabla con sus correspondientes llaves.

Por último, se encuentra la llave de un elemento que se buscó a través de la función de búsqueda previamente mencionada.

Actividad 2. Ejercicios del laboratorio de prácticas

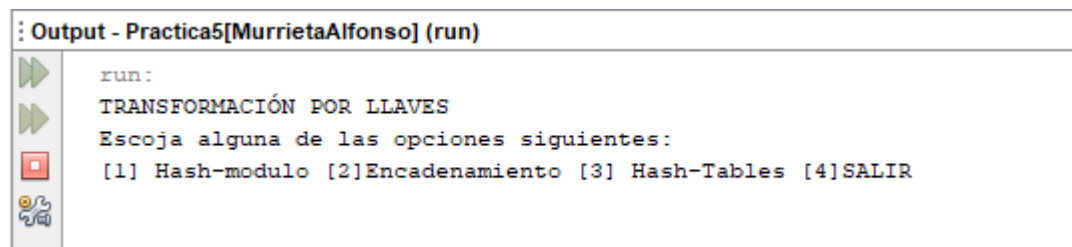
2.1 Esqueleto de la práctica

Para este apartado simplemente se creó un proyecto en NetBeans donde la clase principal “Practica5MurrietaAlfonso” fuera la encargada de llamar directamente a los 3 métodos o formas de ordenar los datos mediante la transformación de llaves o funciones hash.

Par poder llevar a cabo de manera más ordenada y opcional el apartado de las otras 3 clases simplemente se implementó un menú en consola a través de un switch con sus opciones correspondientes y además un ciclo do while para poder hacer que el menú se repitiera tantas veces el usuario quisiera.

Otro aspecto importante es que se decidió no hacerse los métodos estáticos de cada una de las clases de los métodos o funciones hash, por lo que para poder llamar o utilizar los correspondientes métodos se tuvo que instanciar cada clase dentro de los cases del switch.

A continuación, se muestra la salida del programa donde se puede apreciar el menú:



```
Output - Practica5[MurrietaAlfonso] (run)

run:
TRANSFORMACIÓN POR LLAVES
Escoja alguna de las opciones siguientes:
[1] Hash-modulo [2] Encadenamiento [3] Hash-Tables [4] SALIR
```

Imagen 4. Salida del programa, mostrando el menú principal

2.2 Función Hash por módulo

Para este apartado se pidió crear un método en el proyecto dedicado solamente a la función Hash del tipo Módulo, donde a través de la creación de una lista (ArrayList) se llevará el guardado de los elementos.

CONDICIONES

Tener un menú donde el usuario pueda escoger entre, agregar elementos, imprimir la lista o buscar elementos.

Cabe destacar que se deberá crear un método donde se utilizará la función hash por módulo donde el valor que se empleará será el de 17 (Esto debido a que es el valor primo más cercano a 20 o sea el tamaño del arraylist), también se debe considerar que el método de inserción debe calcular la posición en la que el elemento será insertado. Por último, para resolver las posibles colisiones debe hacerse un método donde se resuelva este problema a través del método de doble dirección hash, utilizando la función hash cuadrado.

NOTA: En caso de no poder insertar un elemento después de 3 intentos mandar el mensaje de que no ha sido posible la inserción

Incluir un método dedicado a la búsqueda de elementos en el arraylist, el cual informara si existe o no el elemento y además de indicar la posición donde se encuentra.

DEFINICIÓN DE MÉTODOS:

A continuación, se presentan de manera explícita y particular las definiciones de cada uno de los métodos de esta clase, esto con el fin de poder explicar con el mayor detalle posible el programa:

1] Menu

Es el método principal de la clase, en este se lleva a cabo tanto el llamado de los otros métodos, como la parte del orden y el menú para hacer de manera ordenada y modular el programa.

Es un método que no regresa nada por lo tanto se declaró del tipo void.

2] hashfunction

Como dice su nombre es el método encargado de llevar a cabo la función de asignación o inserción en la lista, es una función que no regresa nada solamente modificar la lista, por lo tanto, es del tipo static void, para poder llevar a cabo la lógica de este método es necesario pasarle como argumentos el dato que se quiere insertar como la lista principal (La declarada en el método menú).

Dentro de la lógica del algoritmo se decidió condicionarse a un ciclo do while para poder limitar la inserción de un elemento (recordemos que después de 3 veces de intentarse debe mandarse un mensaje), por otro lado, para poder llevar a cabo el módulo se decidió hacer uso del método módulo de la biblioteca math.

NOTA: Es necesario aclarar que, debido a una recomendación dada por el profesor, se empleó una variable booleana para poder llevar a cabo la comprobación de un elemento en la lista.

Dentro del ciclo `do while` se para poder llevar a cabo la lógica del algoritmo de hash -modulo se emplearon 2 veces condiciones `if` con sus respectivos `else` para de esta forma en caso de colisión poder llevar a cabo nuevamente el módulo mediante otro método “fixCuadrado” que no es más que el método del cuadrado para las colisiones en tablas hash.

NOTA 2: Por último, para poder indicar al usuario en caso de que no se haya podido insertar un elemento se colocó un último condicional `if` para poder indicar en caso de haberse tratado 3 veces la inserción, el mensaje de que no se pudo insertar el elemento.

3] fixCuadrado

Este método es el encargado de llevar las soluciones de las colisiones creadas a lo largo de la ejecución del programa a través de la inserción de elementos en la lista. Este método al regresar el valor corregido (Para eliminar la colisión) es necesario que sea un método del tipo entero.

Cabe destacar que nuevamente para llevar este algoritmo a cabo se tuvo que hacer varias condiciones, la primera es demostrar que el dato este en el rango de la lista, posteriormente aplicar el módulo tanto a la cantidad de las unidades como a las decenas esto con el fin de poder omitir las colisiones.

Cabe destacar que al principio del desarrollo de este método se decidió tomar en cuenta los 2 primeros elementos del valor a insertar, sin embargo, debido a que esto ocasionaba constantes colisiones, se decidió llevar a cabo al final de cuentas la lógica de tomar los elementos intermedios del valor a insertar.

NOTA: Para llevar a cabo lo anteriormente mencionado, se tuvo que emplear un casteo del valor a string para de esta forma poder abordar el problema mediante el método de los strings conocido como `.length`.

NOTA 2: La parte del cuadrado realmente se lleva a cabo al momento de mandarle como argumento el valor del módulo al cuadrado a este método.

4] search

Este método es el encargado de llevar a cabo la búsqueda de un elemento, debido a que este método no regresa nada simplemente se declaró del tipo estático y `void`, para llevar a cabo la lógica de búsqueda simplemente me base en la lógica que tenía el método de inserción.

Como argumentos simplemente se le pasan la lista y el correspondiente valor a buscar.

APLICACIÓN EN CÓDIGO Y SALIDA DEL PROGRAMA:

A continuación, se muestran capturas de pantalla correspondientes a las salidas de cada uno de los métodos de la clase presente:


```
: Output - Practica5[MurrietaAlfonso] (run)
run:
TRANSFORMACIÓN POR LLAVES
Escriba alguna de las opciones siguientes:
[1] Hash-modulo [2]Encadenamiento [3] Hash-Tables [4]SALIR
1
HASH-MODULO
1) Agregar elemento 2)Imprimir lista 3)Buscar elemento 4)SALIR
1
INSERTAR ELEMENTOS
Ingrese el dato: 12
El modulo es 12
12 está en posición: 12
HASH-MODULO
1) Agregar elemento 2)Imprimir lista 3)Buscar elemento 4)SALIR
1
INSERTAR ELEMENTOS
Ingrese el dato: 3215
El modulo es 2
3215 está en posición: 2
HASH-MODULO
1) Agregar elemento 2)Imprimir lista 3)Buscar elemento 4)SALIR
3451
Escriba una opción válida:
HASH-MODULO
1) Agregar elemento 2)Imprimir lista 3)Buscar elemento 4)SALIR
1
INSERTAR ELEMENTOS
Ingrese el dato: 3216
El modulo es 3
3216 está en posición: 3
HASH-MODULO
1) Agregar elemento 2)Imprimir lista 3)Buscar elemento 4)SALIR
```

Imagen 5. Salida del programa, mostrando tanto el método que muestra el menú como el método de inserción de elementos el cual emplea tanto hashfunction como fix cuadrado.

```
HASH-MODULO
1) Agregar elemento 2)Imprimir lista 3)Buscar elemento 4)SALIR
1
INSERTAR ELEMENTOS
Ingrese el dato: 3211
El modulo es 15
3211 está en posición: 15
HASH-MODULO
1) Agregar elemento 2)Imprimir lista 3)Buscar elemento 4)SALIR
2
IMPRESION DE LA LISTA
[0, 3214, 3215, 3216, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12, 0, 0, 3211, 0, 0, 0, 0]
HASH-MODULO
1) Agregar elemento 2)Imprimir lista 3)Buscar elemento 4)SALIR
```

Imagen 6. Salida del programa, mostrando el método de inserción de elementos el cual emplea tanto hashfunction como fix cuadrado, por último se muestra el apartado de impresión de la lista, donde se puede ver donde han sido insertados los elementos.

```
IMPRESION DE LA LISTA
[0, 3214, 3215, 3216, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12, 0, 0, 3211, 0, 0, 0, 0]
HASH-MODULO
1) Agregar elemento 2)Imprimir lista 3)Buscar elemento 4)SALIR
3
BUSQUEDA DE ELEMENTOS
Ingrese el valor del elemento:
3214
El dato 3214 está en la posición1
HASH-MODULO
1) Agregar elemento 2)Imprimir lista 3)Buscar elemento 4)SALIR
3
BUSQUEDA DE ELEMENTOS
Ingrese el valor del elemento:
12
El dato 12 está en la posición12
HASH-MODULO
1) Agregar elemento 2)Imprimir lista 3)Buscar elemento 4)SALIR
```

Imagen 6. Salida del programa, mostrando el método de búsqueda donde se puede ver como es que efectivamente los elementos se encuentran en la posición en la que se indica.

2.3 Encadenamiento

Para este apartado, se decidió agregar una nueva clase del tipo “Encadenamiento” la cual tendría una lista principal de 15 elementos de tamaño en donde a través del concepto de listas encadenadas, se planteó que cada uno de los elementos de la lista principal fuera otras listas donde se colocarían los elementos que serían insertados en la lista.

CONDICIONES

La sub-lista o listas encadenadas debían ser de enteros, además de que para insertar los elementos en las correspondientes listas se debía implementar una función aleatoria.

NOTA: Cada vez que el usuario ingresara un número en la lista, la función aleatoria indicará la posición donde ubicara el número y mostrará el estado global de la lista.

Por otro lado, el usuario debía tener la posibilidad de agregar elementos a la lista o simplemente salirse de la sección.

DEFINICIÓN DE MÉTODOS:

A continuación, se presentan de manera explícita y particular las definiciones de cada uno de los métodos de esta clase, esto con el fin de poder explicar con el mayor detalle posible el programa:

1] Menu

Es el método principal de la clase, en este se lleva a cabo tanto el llamado de los otros métodos, como la parte del orden y el menú para hacer de manera ordenada y modular el programa.

Es un método que no regresa nada por lo tanto se declaró del tipo void.

NOTA: Como se mencionó en el apartado de condiciones, se tuvo que crear una lista donde sus elementos fueran igualmente listas encadenadas, esto con el motivo de eliminar las colisiones de la manera más fácil tanto de programar.

2] agregarElemento

Debido a que en las condiciones se sugirió y pidió que la inserción se hiciera de manera aleatoria simplemente se empleó el método random de la biblioteca math para poder llevar a cabo de esa forma la inserción, también cabe destacar que este método se tuvo que realizar del tipo cadena o string debido a que en las condiciones del programa se pidió que se indicara donde había sido insertado el elemento dentro de las sub-listas o listas encadenadas.

Por último, cabe destacar que este método solamente se le pasa el elemento a insertar debido a que a diferencia de por ejemplo de la clase anterior se le pasaba a la vez la lista, sin embargo, en este caso se decidió hacer una variable o estructura global dentro de la clase.

3] printList

Este método es un apartado extra que decidí realizar debido a que en un principio tenía problemas con la salida del anterior método es por ello que para cumplir con la condición de mostrar donde se estaba insertando los elementos, decidí mostrarlos a través de este método además de ser un método general que ayuda a la visualización de la lista principal.

NOTA: Solamente requiere una lista para imprimir.

APLICACIÓN EN CÓDIGO Y SALIDA DEL PROGRAMA:

A continuación, se muestran capturas de pantalla correspondientes a las salidas de cada uno de los métodos de la clase presente:

```
Output - Practica5[MurrietaAlfonso] (run)
Escoja alguna de las opciones siguientes:
1) Agregar elemento 2)Imprimir Lista 3)SALIR
1
AGREGAR ELEMENTO:
Ingrese la llave:
1234
Escoja alguna de las opciones siguientes:
1) Agregar elemento 2)Imprimir Lista 3)SALIR
1
AGREGAR ELEMENTO:
Ingrese la llave:
2314
Escoja alguna de las opciones siguientes:
1) Agregar elemento 2)Imprimir Lista 3)SALIR
2
IMPRESIÓN LISTA
Lista 1: [1234]
Lista 2: []
Lista 3: []
Lista 4: []
Lista 5: []
Lista 6: []
Lista 7: []
Lista 8: []
Lista 9: []
Lista 10: []
Lista 11: [2314]
Lista 12: []
Lista 13: []
Lista 14: []
Lista 15: []
```

Imagen 7. Salida del programa, mostrando el método de inserción de elementos, donde además se muestra a través del método de impresión de lista, donde han sido colocados.

```
AGREGAR ELEMENTO:
Ingrese la llave:
31256
Escoja alguna de las opciones siguientes:
1) Agregar elemento 2)Imprimir Lista 3)SALIR
2
IMPRESIÓN LISTA
Lista 1: [1234]
Lista 2: [543]
Lista 3: []
Lista 4: []
Lista 5: []
Lista 6: [12]
Lista 7: []
Lista 8: [345]
Lista 9: []
Lista 10: [1235, 7684]
Lista 11: [2314]
Lista 12: []
Lista 13: [21, 31256]
Lista 14: []
Lista 15: []
```

Imagen 8. Salida del programa, mostrando como para la solución de colisiones simplemente se van agregando más elementos a las listas encadenas como es el caso de la lista 10 y 13.

2.4 Tablas-Hash

PREVIO:

El manejo de tablas hash o hash-tables en Java se realiza a través de la interfaz Map<E> la cual cuenta con diferentes implementaciones y métodos ya definidos. Los casos más usados son principalmente HashMap y Hashtable, la diferencia principal de estos dos radica principalmente en que la primera permite elementos nulos y además conserva un cierto orden, sin embargo, la segunda da la opción de ser sincronizada.

NOTA: Sincronización:

Cuando en un programa tenemos varios hilos corriendo simultáneamente es posible que varios hilos intenten acceder a la vez a un mismo sitio (un fichero, una conexión, un array de datos) y es posible que la operación de uno de ellos entorpezca la del otro. Para evitar estos problemas, hay que sincronizar los hilos.

DEFINICIÓN DE MÉTODOS:

Antes de implementar cada uno de los siguientes métodos en el programa, es preferible y recomendable saber qué necesitan y qué es lo que hacen, a continuación, se describen cada uno de los métodos empleados en la clase Hash-Table:

contains:

Sirve para saber si existe un valor en específico (elemento) a través del valor que se le pasa comparando a los que hay en el mapa (Hashtable), es un método del tipo booleano.

containsKey:

Sirve para saber si existe un valor en específico (elemento) a través de claves del mapa (Hashtable), es un método del tipo booleano y solamente se le pasa la clave respectiva al elemento que se quiere buscar.

containsValue:

Es un método booleano, al cual se le pasa como argumento el valor, este método a diferencia del primero indica si hay una o más llaves para el elemento que se está buscando.

equals:

Dependiendo de la definición o del tipo de declaración que se le haya hecho al hash compara un elemento en específico con respecto a un elemento del mapa:

Es un método del tipo booleano y solamente como argumento se le pasa al elemento a comparar

get:

Devuelve el valor al que se asigna la clave especificada, o nulo si en el mapa no existe ninguna asignación para la clave. Al método se le pasa como argumento el valor que se está buscando.

Remove:

Este método se encarga de eliminar o remover tanto la llave como el elemento que se ha indicado, a través de la clave que se le ha pasado.

put:

Es un método del tipo inserción donde se le pasan como argumentos una clave y un valor o elemento, el método se encarga de asignar la clave que se ha pasado al elemento especificado.

size:

Es un método del tipo entero donde regresa la cantidad o número de claves que hay en la tabla -Hash

APLICACIÓN EN CÓDIGO Y SALIDA DEL PROGRAMA:

A continuación se muestran las capturas de pantalla de los correspondientes métodos de la clase o collection del tipo Map de Java , “Hash Table”.



```
run:
TRANSFORMACIÓN POR LLAVES
Escoja alguna de las opciones siguientes:
[1] Hash-modulo [2] Encadenamiento [3] Hash-Tables [4] SALIR
3
HASH TABLE - JAVA
Escoja alguna de las opciones siguientes:
[1] Agregar elemento [2] Imprimir /Buscar Lista [3] Eliminar Elemento [4] SALIR
1
Insertar llave:
1
Valor de la llave:
12
Método get(Inserte valor):
1
12
Escoja alguna de las opciones siguientes:
[1] Agregar elemento [2] Imprimir /Buscar Lista [3] Eliminar Elemento [4] SALIR
1
Insertar llave:
2
Valor de la llave:
34
Método get(Inserte valor):
3
null
Escoja alguna de las opciones siguientes:
[1] Agregar elemento [2] Imprimir /Buscar Lista [3] Eliminar Elemento [4] SALIR
1
```

Imagen 9: Salida del programa enseñando en primer lugar el método put y además el método get, como se puede ver en la imagen en la primera parte el método get regresa el valor mientras en el segundo caso al no existir simplemente regresa un null.

```

Escoja alguna de las opciones siguientes:
1] Agregar elemento 2]Imprimir /Buscar Lista 3]Eliminar Elemento 4]SALIR
2
La cantidad de llaves es: 3
{3=34, 2=24, 1=12}
Buscar llave:
2
true
Buscar valor:
24
true
Buscar valor:
35
false

```

Imagen 10: Salida del programa Hash-Table, en primera instancia podemos ver el uso del método size para conocer el tamaño del hash, también se puede ver la impresión de todo el hash y por último el uso de los métodos contains tanto value, key y el común donde la salida de estos es simplemente un valor booleano indicando si está o no en el Hash-table.

```

Escoja alguna de las opciones siguientes:
1] Agregar elemento 2]Imprimir /Buscar Lista 3]Eliminar Elemento 4]SALIR
3
Eliminar elemento (key):
3
Escoja alguna de las opciones siguientes:
1] Agregar elemento 2]Imprimir /Buscar Lista 3]Eliminar Elemento 4]SALIR
2
La cantidad de llaves es: 2
{2=24, 1=12}
Buscar llave:
.

```

Imagen 11: Salida del programa Hash-table, en esta captura de pantalla se puede observar el método restante que en este caso es el de remove,, en la primera parte se puede ver como se elimina el elemento con la llave 3, posteriormente se imprime la tabla-hash para mostrar que efectivamente había sido borrado el elemento.

CONCLUSIONES

En la presente práctica se pidió desarrollar un proyecto, donde a través de un programa se podía buscar un elemento en una lista empleando el concepto de “Búsquedas por Transformación de Llaves”, en esta primera versión lamentablemente no se pudo concluir ningún programa.

En la siguiente versión de la práctica evidentemente se tratará de resolver los problemas presentes.

A lo largo del desarrollo de esta práctica realmente tuve que investigar muchísimo más de lo que habitualmente he hecho, las tablas hash aunque no lo parezca es una estructura de datos que realmente es muy empleada sobre todo por la forma en que se puede emplear para almacenar elementos, además y aunque no era el punto principal de la práctica el desarrollo de las funciones hash como de las funciones de solución a colisiones son realmente un tema realmente ocupado en lo que hoy en día sucede, sobre todo porque resulta que no solamente las funciones Hash sirven para colocar elementos en una tabla sino que estas también tienen en cierta forma relación en la forma en que se puede cifrar elementos dentro de alguna estructura de forma en que estos solamente puedan ser accedidos a través de una clave en específico.

Por otro lado, realmente tengo que destacar que en la primera función , el caso del hash modulo, tuve bastantes problemas debido a que en un principio planteé el no emplear los números intermedios de una cifra ingresada en la tabla hash (En el método dedicado a la solución de colisiones), es por ello que finalmente tuve que realizar aunque fuera más complicado un método dedicado a obtener los números intermedios y posteriormente emplearlos para la solución de colisiones , fue de esta forma que realmente se pudo reducir la cantidad de colisiones obtenidas al inicio.

REFERENCIAS

Bradley N. Miller y David L. Ranum, Franklin, Beedle & Associates. (2011). Problem Solving with Algorithms and Data Structures using Python. Segunda Edición.

Elba Karen Saenz García. (2017). *Manual de Prácticas del laboratorio de Estructuras de Datos y algoritmos II*. UNAM, Facultad de Ingeniería.

Mark J. Guzdial, Barbara Ericson. (2015). *Introducción a la computación y programación con Python*. 3ra Edición. Madrid España.

Consultado el 21 de septiembre de 2018, de <https://docs.oracle.com/javase/7/docs/api/>