	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: TISTA GARCÍA EDGAR

Asignatura: ESTRUCTURAS DE DATOS Y ALGORITMOS II

Grupo: 5

No de Práctica(s): 03

Integrante(s): GÓMEZ LUNA ALEJANDRO

No. de Equipo de cómputo empleado: 51

No. de Lista o Brigada: 16

Semestre: 2020-1

Fecha de entrega: 13/SEPTIEMBRE/2019

Observaciones:

CALIFICACIÓN: _____

Objetivo de la práctica

EL ESTUDIANTE CONOCERÁ E IDENTIFICARÁ LAS CARACTERÍSTICAS NECESARIAS PARA REALIZAR BÚSQUEDAS POR TRANSFORMACIÓN DE LLAVES

Desarrollo

a) Ejercicios de la guía: En la guía se nos presenta otra manera de realizar la búsqueda de un elemento, que es a través de la transformación de llaves. Estas llaves son como un tipo de identificador para cada elemento, al proceso de transformar una llave a un cierto elemento se le conoce como «hashing». Esto nos sirve para poder encontrar un elemento con mayor facilidad.

   Existen diversas funciones que realizan el hashing, para este caso, se utilizo la función hashing por división.

   La implementación se realiza en el lenguaje de programación Python, en donde se da un diccionario, donde la llave es una cadena de caracteres, mientras que el valor es un número.

   La primera función es la de formarTabla(), que será la encargada de crear la tabla donde se almacenen los valores con su correspondiente llave ya transformada.

   La siguiente función es convertirLlave(), la cual nos devolverá una llave que representa a la cadena de caracteres mediante un número entero. A este número resultante se le aplica la función hash por división y con esta llave transformada se indicará la posición en donde insertar el elemento. El elemento siempre se insertará en la posición donde no exista ningún elemento, pues las colisiones para este caso se manejan utilizando el hashing lineal, es decir, si un elemento será insertado en una posición ya ocupada por un elemento, entonces se irá sumando de uno en uno la posición hasta encontrar una que este vacía.

   Por último, encontramos la función buscar, la cual usará el hashing por división para encontrar la posición en el diccionario donde estará el elemento que se busca. Una vez llegada a esa posición, si el elemento buscado no coincide con el que se encuentra en esa posición, se empezará a buscar en las posiciones siguientes, debido a la forma en como se manejaron las colisiones.

   Se intenta agregar un sexto elemento, sin embargo, ya no es posible porque el diccionario solo fue creado para almacenar cinco llaves con sus respectivos elementos, por lo que, al intentar insertar un sexto elemento, nos muestra en pantalla “No hay espacio”.

   Por otro lado, si se intenta buscar un elemento que no se encuentra en el diccionario, la función buscar() regresará como valor un -1, haciendo referencia a que no se encuentra el elemento, pues esta posición es inválida.

   La captura de implementación del código proporcionado nos muestra lo siguiente:

```
[None, None, None, None, None]
No hay lugar
[['Hola5', '12213295'], ['Hola4', '12213294'], ['Hola3', '12213293'], ['Hola2', '12213292'], ['Hola1', '12213291']]
0
-1
```

b) Ejercicios propuestos:
    a. Ejercicio 1: Para este ejercicio se nos podía implementar una función hash por división de manera manual, en donde el usuario ingresara un elemento y, mediante la función hash módulo 19, se le asignara una posición en una lista ligada a dicho elemento, en donde la lista es un miembro privado de la clase HashModulo, pues los miembros es recomendable que solo puedan ser manipulados por métodos de la misma clase donde se encuentra.

    Se crearon cinco métodos para la clase Hashmodulo. El primer método es el constructor y es el encargado de inicializar la lista con veinte referencias null, mediante el método add() de la colección LinkedList.

    El segundo método es el agregarElemento(), el cual se encargará de ir adicionando los elementos a la lista. Para lograrlo, se implementa la función hash y, si la posición resultante está desocupada, entonces el elemento se almacenará ahí. En caso contrario, el elemento irá recorriendo toda la lista hasta encontrar una posición desocupada. En caso de que se llegue al final de la lista, se volverá a iniciar desde la primera posición de la lista y se seguirá buscando la posición, hasta que se vuelva a llegar a la posición inicial indicada por la función hash, pues esto indica que se ha recorrido toda la lista y no hay espacio donde almacenar el elemento. Se realizó de esta manera para poder implementar el método de prueba lineal para resolver colisiones.

    El tercer método es imprimirLista(), el cual imprimirá en pantalla todos los elementos que se encuentran en la lista. Para lograrlo, se utiliza una estructura de repetición for each.

    El cuarto método es buscarElemento(), el cual se encargará de verificar si un elemento se encuentra en esta lista. En este caso, se sigue básicamente el mismo principio que agregarElemento(), solamente que, en cuanto se encuentra el elemento deseado, se deja de recorrer la lista y se muestra en pantalla que se ha encontrado el elemento buscado.

    Por último, se encuentra el método menuHashModulo(), el cual creará un menú en el cual se puede escoger entre las acciones de agregar elemento, imprimir lista, buscar elemento o salir de este menú.

    Cabe resaltar que este último método es el único que no es privado, puesto que este método es el único que queremos, pueda ser llamado por clases ajenas, porque este será el encargado de implementar las otras funciones.

    Las capturas de implementación son las siguientes:

    De igual forma, me gustaría remarcar la importancia del uso de preincrementos y postincrementos, pues para las funciones buscarElemento() y agregarElemento(), existía casos donde primero era necesario comparar y luego realizar un incremento, mientras que en otros primero se necesitaba realizar un incremento y luego comparar, entonces, en estas funciones se puede apreciar un poco la importancia y diferencia entre los preincrementos con respecto a los postincrementos.

```
run:
Ingresa la opcion a realizar:
1)Funcion hash por modulo
2)Encadenamiento
3)Manejo de Tablas Hash en Java
4)Salir
1
Selecciona una de las siguientes opciones:
1)Agregar elemento
2)Imprimir lista
3)Buscar elemento
4)Salir
3
Ingrese el elemento a buscar en la lista:
5
El elemento no se encuentra en la lista
Selecciona una de las siguientes opciones:
1)Agregar elemento
2)Imprimir lista
3)Buscar elemento
4)Salir
1
Ingrese el elemento a adicionar a la lista:
15
Elemento 15 agreagdo en la posicion 15
Selecciona una de las siguientes opciones:
1)Agregar elemento
2)Imprimir lista
3)Buscar elemento
4)Salir
1
Ingrese el elemento a adicionar a la lista:
65
Elemento 65 agreagdo en la posicion 8
```

```
Selecciona una de las siguientes opciones:
1)Agregar elemento
2)Imprimir lista
3)Buscar elemento
4)Salir
2
Los elementos en la lista son:
null
null
876
null
23
null
63
6
65
null
542
null
4325
null
null
15
null
986
null
null
```

```
Selecciona una de las siguientes opciones:
1)Agregar elemento
2)Imprimir lista
3)Buscar elemento
4)Salir
1
Ingrese el elemento a adicionar a la lista:
986
Elemento 986 agreagdo en la posicion 17
Selecciona una de las siguientes opciones:
1)Agregar elemento
2)Imprimir lista
3)Buscar elemento
4)Salir
1
Ingrese el elemento a adicionar a la lista:
63
Elemento 63 agreagdo en la posicion 6
Selecciona una de las siguientes opciones:
1)Agregar elemento
2)Imprimir lista
3)Buscar elemento
4)Salir
1
Ingrese el elemento a adicionar a la lista:
876
Elemento 876 agreagdo en la posicion 2
```

```
Selecciona una de las siguientes opciones:
1)Agregar elemento
2)Imprimir lista
3)Buscar elemento
4)Salir
3
Ingrese el elemento a buscar en la lista:
5
El elemento no se encuentra en la lista
Selecciona una de las siguientes opciones:
1)Agregar elemento
2)Imprimir lista
3)Buscar elemento
4)Salir
3
Ingrese el elemento a buscar en la lista:
8
El elemento no se encuentra en la lista
Selecciona una de las siguientes opciones:
1)Agregar elemento
2)Imprimir lista
3)Buscar elemento
4)Salir
3
Ingrese el elemento a buscar en la lista:
6
Elemento 6 encontrado en la posicion 7
```

b. Ejercicio 2: Para este ejercicio se nos pedía realizar un encadenamiento, más específicamente una «lista de listas». Para poder implementarlo, en Java se utilizó la colección de LinkedList. Se creo un miembro privado de la clase Encadenamiento, el cual es una lista ligada de listas ligadas de enteros.
   En el constructor de esta clase, en cada espacio de la lista ligada principal, se adiciona otra lista ligada secundaria, en total, dieciséis listas ligadas, una principal y otras quince secundarias. En este caso, se hace uso de las referencias anónimas, pues solamente queremos que exista la referencia en memoria de estas listas secundarias, ya que éstas nos las utilizaremos independientemente, si no, mediante la lista principal.
   Se define el método imprimirLista(), el cual hace uso de dos for each, uno anidado dentro del otro, para que, en cada posición de la lista ligada principal, se vayan recorriendo todos los elementos de la lista ligada secundaria.
   Otro método es el de agregarElemento(), en donde adicionaremos un elemento en cualquiera de las posiciones de la lista ligada principal. Para determinar la posición, se crea un objeto de la clase Random para poder usar el método nextInt() que recibe como parámetro un valor frontera y genera un número pseudo-random entre 0 y el valor que le pasamos como frontera, exclusive. En este método es en donde podemos ver el encadenamiento, ya que, siempre que existan colisiones (se genere una posición en donde ya existe un elemento), entonces este nuevo elemento se almacenará en esa posición, pero adicionándose a la lista ligada secundaria usada en dicha posición. Aunado a esto, cada que se agregue un nuevo elemento, se imprimirá en pantalla el estado actual de la lista.
   El último método es menú(), el cual creará un menú donde el usuario pueda elegir entre adicionar un nuevo elemento o salir del menú.
   Como en el ejercicio anterior, este es el único método que no es privado, por lo explicado con anterioridad.

   En la captura de implementación, solo se muestra el estado final de la lista, pues se tuvieron que agregar varios elementos para observar de mejor manera como se realiza el encadenamiento.

```
Selecciona una de las siguientes opciones:
1)Agregar elemento
2)Salir
1
Ingrese el elemento a adicionar a la lista:
99
El elemento 99 se agregara en la poisicon 0
[0] -> 158 -> 99 ->
[1] ->
[2] ->
[3] -> 666 ->
[4] -> 99 ->
[5] -> 10 -> 865 ->
[6] -> 12121 ->
[7] -> 743 ->
[8] ->
[9] ->
[10] ->
[11] -> 777 ->
[12] -> 743 ->
[13] -> 8674 ->
[14] ->
```

c. Ejercicio 3: Para esta actividad se nos pedía implementar una colección Hashtable o Hashmap, cabe resaltar que ambas realizan lo mismo, solamente que Hashmap es una colección más actual en comparación con Hashtable.
   En la clase TablasHash podemos encontrar dos miembros privados. El primero es la tabla hash que se encuentra compuesta por cadenas a manera de llave y arreglos de enteros a manera de valor. El segundo miembro es un arreglo de enteros. Este arreglo representa las calificaciones del alumno Alejandro en la Hashtable y fue creado para facilitar la ejemplificación del uso del método containsValue() de la clase Hashtable, que veremos más adelante.
   Además de estos miembros, encontramos dos métodos. Un método que es el constructor de la clase e inicializa la Hashtable con cinco valores, que en este caso son visualizados como cinco alumnos con sus respectivas calificaciones.
   El segundo método es probando() y es el encargado de visualizar algunos métodos en Hashtable¡. En este método, es importante notar que se empieza por mostrar la Hashtable y se observa que, el valor de cada alumno es una dirección de memoria, los cuales representan los lugares en memoria donde se almacenaron los arreglos de enteros.
   En cuanto a los métodos a probar, tenemos que:
   - Contains y containsKey: Son lo mismo. El porqué se tienen dos métodos que realizan lo mismo es debido a la existencia de la colección Hashtable que ya contenía el método contains(), pero cuando llego la interfaz Map, como se tienen que implementar todos los miembros de dicha, entonces llegó el otro método containsKey().
   - containsValue: Nos regresa un boolean indicando si está el valor que le pasamos como parámetro en la Hashtable. En esta función, para poder mostrarlo, se utilizo el miembro definido con anterioridad que contenía el arreglo con enteros, debido a que, si se le pasaba como parámetro un arreglo con los mismos enteros, arrojaría false, porque, aunque tengan los mismos valores, se comparan sus direcciones de memoria, y estas no son iguales.
   - equals: Verifica si dos Hashtable son iguales. Para este caso, se creó una copia de la Hashtable con el método clone() y se comparó con la original para verificar el funcionamiento de este método. El resultado fue que, efectivamente ambas Hashtable eran iguales.
   - get: Se le pasa como parámetro una llave y recupera el valor asociado a dicha llave en la Hashtable. Aquí, como el valor es un arreglo de enteros, entonces se utilizó el método toString de la clase Arrays para mostrar dichos enteros, ya que, si no se utiliza, solo se mostraría en pantalla la dirección en memoria de dicho arreglo.
   - put: Agrega una nueva llave con su respectivo valor a la Hashtable.
   - remove: Toma como parámetro una llave y, junto con el valor asociado a dicha, los elimina de la Hashtable.
   - size: Muestra el tamaño actual de la Hashtable.

En la captura de implementación se puede apreciar el resultado de estos métodos.

```
run:
Ingresa la opcion a realizar:
1)Funcion hash por modulo
2)Encadenamiento
3)Manejo de Tablas Hash en Java
4)Salir
3
La Hashtable tiene los siguientes elementos: {Samantha=[I@14ae5a5, Edgar=[I@7f31245a, Bestia=[I@6d6f6e28, Zlatan=[I@135fbaa4, Alejandro=[I@45ee12a7}
A continuacion se probaran algunas de los metodos en la coleccion Hashtable:
Contiene al alumno Raul: false
Contiene a la alumna Samantha: true
Contiene el arreglo de calificaciones [8,9,10,9,9,10]: true
Contiene el arreglo de calificaciones [6,9,8,7,0,2]: false
Comparando esta Hashtable con una copia suya: true
Obteniendo las calificaciones del alumno Zlatan en la Hashtable: [4, 2, 1, 3, 5, 5]
Agregando el alumno Nadie con las calificaciones [6,7,8,9,10,10]:
Eliminando al alumno Bestia:
El tamaño final de la Hashtable es: 5
Ingresa la opcion a realizar:
1)Funcion hash por modulo
2)Encadenamiento
3)Manejo de Tablas Hash en Java
4)Salir
4
BUILD SUCCESSFUL (total time: 6 seconds)
```

Conclusiones

Para esta práctica se lograron realizar los ejercicios propuestos relacionado con el uso de las funciones hash para poder facilitar la búsqueda de elementos. Se visualizó la implementación de la función hash módulo y el como resolver las colisiones por prueba lineal, en donde se iba buscando un espacio que estuviera desocupado para almacenar algún elemento en caso de que su posición correspondiente estuviera ocupada por algún otro elemento. Además de esta manera de resolver colisiones, se implementó el encadenamiento, mediante una lista de listas y una función random, donde, siempre que se agregaban elementos con la misma posición en la lista principal, estos se iban adicionando a la lista secundaria, pues cada posición de la lista principal se puede visualizar como el «head» de una lista secundaria. En último lugar, vimos el funcionamiento de las colecciones Hashtable, que ya se encuentran definidas en Java, y el uso de algunos de los métodos con los que cuenta.

El hashing es una herramienta muy poderosa para la búsqueda si se escoge una función hash adecuada que no genere muchas colisiones, porque permite buscar cualquier elemento en alguna colección en un tiempo constante o muy cercano. La desventaja es que las funciones hash siempre generan colisiones, por lo que, el hecho de saber manejar las colisiones también es de suma importancia. Un mal manejo de colisiones puede llevar a que la búsqueda por transformación de llaves se vuelva una búsqueda lineal.

Se logró el objetivo de la práctica, ya que se lograron identificar las características para realizar una búsqueda por transformación de llaves, además de que se logró implementar dicha en el lenguaje de programación Java, así como algunas formas de resolver colisiones.

Mediante los ejercicios de la práctica logramos un análisis de las características básicas de la transformación de llaves, y no solo eso, si no que además nos permite implementar distintas maneras de resolver las colisiones. Una práctica que cumple con el objetivo y además, permite que el alumno implemente el manejo de colisiones, agregando mayor dificultad a la práctica pero también permitiendo un mayor aprendizaje.