	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Professor: Jesus Cruz Navarro

Asignatura: Estructura de Datos y Algoritmos 2

Grupo: 1

No de Práctica(s): 05

Integrante(s): Diego Santiago Gutiérrez

No. de Equipo de cómputo empleado:

No. de Lista o Brigada:

Semestre: Tercer Semestre

Fecha de entrega: 09/11/2020

Observaciones:

CALIFICACIÓN: _____

PRACTICA 05: ALGORITMO DE BÚSQUEDA 2: TABLA HASH

OBJETIVO:

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

INTRODUCCIÓN:

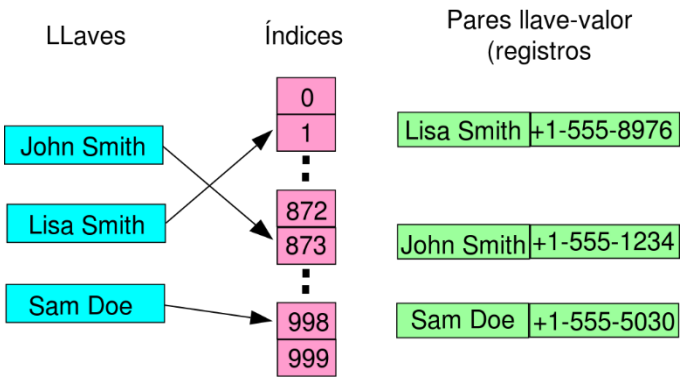
Las tablas hash se suelen implementar sobre vectores de una dimensión, aunque se pueden hacer implementaciones multi-dimensionales basadas en varias claves. Como en el caso de los arrays, las tablas hash proveen tiempo constante de búsqueda promedio O(1),2 sin importar el número de elementos en la tabla.
Comparada con otras estructuras de arrays asociadas, las tablas hash son más útiles cuando se almacenan grandes cantidades de información.

Inserción:

La forma de implementar en función esta operación es pidiendo la llave y el valor, para con estos poder hacer la inserción del dato.
Para almacenar un elemento en la tabla hash se ha de convertir su clave a un número. Esto se consigue aplicando la función resumen (hash) a la clave del elemento.
El resultado de la función resumen ha de mapearse al espacio de direcciones del vector que se emplea como soporte, lo cual se consigue con la función módulo. Tras este paso se obtiene un índice válido para la tabla.

Búsqueda:

La forma de implementar en función esta operación es pidiendo la llave y con esta devolver el valor.
Para recuperar los datos, es necesario únicamente conocer la clave del elemento, a la cual se le aplica la función resumen.
El valor obtenido se mapea al espacio de direcciones de la tabla.
Si el elemento existente en la posición indicada en el paso anterior tiene la misma clave que la empleada en la búsqueda, entonces es el deseado. Si la clave es distinta, se ha de buscar el elemento según la técnica empleada para resolver el problema de las colisiones al almacenar el elemento.



DESARROLLO:

```
def creteTableHash(m):
    HashTable = [None] * m
    return HashTable
```

A .1) Función createTableHash

Esta función creará y retornará una tabla de “m” None elementos

```
def Hashing(keyvalue):
    suma=0
    for item in keyvalue:
        suma = suma + ord(item)

    key = (( ( ( 49 * suma ) + 689 ) % 999983 ) % len((tablaCreada)))
    return key
```

Función Hashing

Esta función obtendrá el valor de la suma del carácter recibido “keyvalue”, en el cual terminará siendo una llave para ingresar los elementos en el índice key. (Key se obtiene por medio de una formula de Hashing Universal, los valores fueron elegidos por el programador)

```
def insert(tablaCreada, keyvalue, nameValue):

    Lista = []
    hash_key = Hashing(keyvalue)

    if tablaCreada[hash_key] == None:
        tablaCreada[hash_key] = Lista
            #La llave Hasheada
    tablaCreada[hash_key].append(nameValue)
```

a.2) Función insert

Insert es una función que recibe la lista para meter los elementos, el usermae <keyvalue> y el nombre de la persona a ingresar <nameValue>. Para ello se crea una lista y a hash_key será la función Hashing del username, el cual será para tener la llave en la cual será acomodado este elemento. Se realiza una comparación con la lista y None, es decir existe, se le guardará este valor a Lista.
Finalmente insertamos nameValue en tablaCreada de la posición correspondiente de su llave.

```python
def searchElement(tablaCreada, keyvalue):

    hash_key = Hashing(keyvalue)

    if(tablaCreada[hash_key] is None):
        return None
    else:
        for item in tablaCreada [hash_key]:
            if item.username == keyvalue:
                return item
```

a.3) Función searchElement

Lo que hace esta función es buscar un elemento por medio de una tabla y el username <keyvalue> recibidos como parámetros. Para ello hacemos el hashing del username para buscar en el índice correspondiente de tablaCreada. Si la tabla esta vacia se devuelve un vacio, de lo contrario se realiza un ciclo for en la tabla creada hasta hacer match con el valor del username y el username que nosotros estamos dando, es decir que existe el elemento, de ser esto regresa la posición en la cual se encontró esta comparación.

```python
def login(tablaCreada, keyvalue, valuePasword):

    print("-----------------------------------------")
    print("LOGIN:   usr: " + keyvalue + "   pass: " + valuePasword)

    hash_key = Hashing(keyvalue)

    a = searchElement(tablaCreada, keyvalue)
    print(a)
    if (a == None):
        print("Usuario no existe: ")
    elif(a.password == valuePasword):
        print("ACCESO concedido: ", a.fullname)
    else:
        print("Acceso no autorizado contraseña incorrecta")

    print("-----------------------------------------")
```

Función login

Para su funcionamiento es requerido una tabla, el usuario <keyvalue> y su contraseña <valuePasswrod>. Lo que hará será sacar el Hashing del valor del usuario para poder buscar en listaCreada si es que estos valores se encuentran dentro de ella y en qué posición con searchElement. Después de realizar esto verificamos si a se encuentra vacio, de ser así significa que no hay usuario. Si la contraseña de a es igual a la que nosotros le estamos pasando se concederá el acceso, de lo contrario significa que está el usuario, pero la contraseña dada es incorrecta.

```python
def BusqLineal(list, keyusername):
    for k in range(0, len(list)):
        if ( list[k] is not None ):
            for j in range(0 , len(list[k])):
                if list[k][j].username == keyusername:
                    return list[k][j]
        else: continue
    return None
```

b) Función BusquedaLineal

Búsqueda lineal es un método de búsqueda que ya conocemos, por lo tanto no abundaremos en su explicación. Busca el elemento en la posición que se encuentre el elemento de la lista dada, como es Hashing con Chaining es necesario tener dos índices.

```python
def Compare(tablaHash, usrList):
    sumLineal = sumHash = 0
    for i in range(10):
        usr = usrList[random.randint(0, len(usrList))].username
        t1 = time.perf_counter()
        e1 = searchElement(tablaHash, usr)
        t2 = time.perf_counter()

        t3 = time.perf_counter()
        e2 = BusqLineal(tablaHash, usr)
        t4 = time.perf_counter()
        print("\nPara encontrar al usuario: "+ usr + " tardé:")
        print("Con Busqueda Lineal:", t4 - t3, "\t Con Busqueda en Tabla Hash:", t2 - t1)
        sumLineal += t4 - t3
        sumHash += t2 - t1
    print("\nEl tiempo promedio de 10 casos fue de: ")
    print("Para la Busqueda Lineal:", sumLineal/10)
    print("Para la Busqueda en Tabla Hash:", sumHash/10)
```

c) Función Compare

Lo que hace esta función básicamente es comparar los tiempos de ejecución de la tabla Hashing y Búsqueda lineal. Como los requerimientos del maestro son que se realicen por lo esto se realizara 10 veces y se tomarán los tiempos de ejecución de ambos métodos de búsqueda. Estos valores serán almacenados en su correspondiente acumulador par adespues imprimir el valor de estas 10 pruebas, es decir el promedio de ejecución.

RESULTADOS:

```
tablaCreada= creteTableHash(999)
usrList = Database.Usuario.GetUsuariosDB(1000)


for item in usrList:
    insert(tablaCreada, item.username, item)

login(tablaCreada, "mvicker23", "TNqqef")
login(tablaCreada, "tnewborn6x", "FnZzuo")
login(tablaCreada, "tnewborn6x", "123123")
login(tablaCreada, "Jesus Cruz", "123123")

print("Parámetros utilizados: \n a = 49\n b = 689\n p = 999983\n m (tamaño tabla) =", len(tablaCreada))
f = len(usrList)/len(tablaCreada)
print("Factor de carga: F =", f,"\n")

Compare(tablaCreada, usrList)


print("-----------------------------------")
```

Estos será el Main, donde realizamos la llamada a las funciones importantes como login, compare para poder mostrar en pantalla lo siguiente.

```
-----------------------------------------
LOGIN:   usr: mvicker23   pass: TNqqef
usr=mvicker23 , name=Marabel Vicker , pass=TNqqef
ACCESO concedido:  Marabel Vicker
-----------------------------------------
-----------------------------------------
LOGIN:   usr: tnewborn6x   pass: FnZzuo
usr=tnewborn6x , name=Tim Newborn , pass=FnZzuo
ACCESO concedido:  Tim Newborn
-----------------------------------------
-----------------------------------------
LOGIN:   usr: tnewborn6x   pass: 123123
usr=tnewborn6x , name=Tim Newborn , pass=FnZzuo
Acceso no autorizado contraseña incorrecta
-----------------------------------------
-----------------------------------------
LOGIN:   usr: Jesus Cruz   pass: 123123
None
Usuario no existe:
-----------------------------------------
Parámetros utilizados:
 a = 49
 b = 689
 p = 999983
 m (tamaño tabla) = 999
Factor de carga: F = 1.001001001001001
```

Esto lo obtenemos de la función login pasándole los parámetros que el maestro mostró en la práctica.

```
Para encontrar al usuario: daustenj1 tardé:
Con Busqueda Lineal: 0.0007458999999999938      Con Busqueda en Tabla Hash: 3.790000000000737e-05

Para encontrar al usuario: sgwillyam2e tardé:
Con Busqueda Lineal: 0.0008278999999999925      Con Busqueda en Tabla Hash: 3.7300000000017874e-05

Para encontrar al usuario: gelcockk3 tardé:
Con Busqueda Lineal: 0.0002783000000000091      Con Busqueda en Tabla Hash: 2.2300000000002873e-05

Para encontrar al usuario: khalsedn tardé:
Con Busqueda Lineal: 0.0016595000000000082      Con Busqueda en Tabla Hash: 6.870000000000487e-05

Para encontrar al usuario: nmagillrf tardé:
Con Busqueda Lineal: 0.000861899999999971       Con Busqueda en Tabla Hash: 3.0599999999991745e-05

Para encontrar al usuario: rmcaviyb1 tardé:
Con Busqueda Lineal: 0.0004605999999999777      Con Busqueda en Tabla Hash: 2.140000000000475e-05

Para encontrar al usuario: bbarkessd2 tardé:
Con Busqueda Lineal: 0.0007264000000000159      Con Busqueda en Tabla Hash: 2.1299999999946362e-05

Para encontrar al usuario: bfilippiim tardé:
Con Busqueda Lineal: 0.0023833000000000326      Con Busqueda en Tabla Hash: 4.500000000001725e-05

Para encontrar al usuario: scaghanp3 tardé:
Con Busqueda Lineal: 0.0005178000000000127      Con Busqueda en Tabla Hash: 2.199999999996649e-05

Para encontrar al usuario: kmoroq4 tardé:
Con Busqueda Lineal: 0.006121700000000008       Con Busqueda en Tabla Hash: 1.8000000000018e-05

El tiempo promedio de 10 casos fue de:
Para la Busqueda Lineal: 0.001458330000000002
Para la Busqueda en Tabla Hash: 3.244999999999776e-05
-----------------------------------
```

Como es posible observar, se imprimen los tiempos de la tabla Hash y BúsquedaLineal
Al final el color morado es de búsqueda Lineal y el color azul corresponde a la tabla Hash

CONCLUSIONES:

En esta práctica hemos aprendido lo que es una **Hash Table**, una función de **Hash**, lo que son las **Hash Collisions** y una manera de resolverlas, llamada **Hashing con Chaning**. Es muy importante reconocer la mejor manera de enfrentar estos problemas y la implementación que conllevan los mismos. Una razón por la cual la tabla Hash es considerada como la mejor tabla de búsqueda es debido a su estructura, la cual permite que almacenar una gran cantidad de datos y seguir siendo de tipo O(1). Siendo así una estructura que permite tener sub-arreglos que es relativamente sencillo acceder en ellos. La forma en que una tabla Hash se acomoda, dependerá de el tamaño de la tabla, el numero de valores que se ingresen y los datos elegidos para poder implementar un **Hash_Key** correspondiente.
Como es posible observar, los tiempos de ejecución se ven con un menor tiempo de ejecución debido a la complejidad del algoritmo en frente de una búsqueda lineal, que con suerte podría tener los elementos a buscar al inicio del arreglo, pero desafortunadamente los datos que se trabajan funcionan de manera distinta y es por ello que **Hash** termina siendo un algoritmo mejor.