	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	49/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Guía Práctica de Estudio 5


Algoritmos de búsqueda parte 2

Elaborado por:

M.I. Elba Karen Sáenz García

Revisión:

Ing. Laura Sandoval Montaña

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	50/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Guía Práctica 5

Estructura de datos y Algoritmos II

Algoritmos de Búsqueda. Parte 2.

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

Actividades

Implementar la búsqueda por transformación de llaves utilizando alguna técnica de resolución de colisiones en algún lenguaje de programación.

Antecedentes


- Análisis previo de los algoritmos en clase teórica.
- Manejo de arreglos o listas, estructuras de control y funciones en Python 3.

Introducción

Un diccionario es un conjunto de pares (llave, valor), siendo una abstracción que vincula un dato con otro. En un diccionario se pueden realizar operaciones de búsqueda, inserción y borrado de elementos dada una llave. Una tabla hash es una estructura de datos para implementar un tipo de dato abstracto (TDA) diccionario.

En el método de búsqueda por transformación de llaves, los datos son organizados con ayuda de una tabla hash, la cual permite que el acceso a los datos sea por una llave que indica la posición donde están guardados los datos que se buscan. Se utiliza una función que transforma la llave o dato clave en una dirección dentro de la estructura (tabla), dicha función de transformación se conoce como **función hash**.

Así, para determinar si un elemento con llave x está en la tabla, se aplica la función hash h a x ($h(x)$) y se obtiene la dirección del elemento en la tabla. Esto es si la función hash se expresa como $h: U \rightarrow \{0, 1, 2, \dots, m-1\}$ se dice que h mapea el universo de llaves U en la posición de la tabla hash $T[0, 1, 2, \dots, m-1]$ donde m es el tamaño de la tabla y es típicamente menor que U . Figura 5.1.

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	51/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

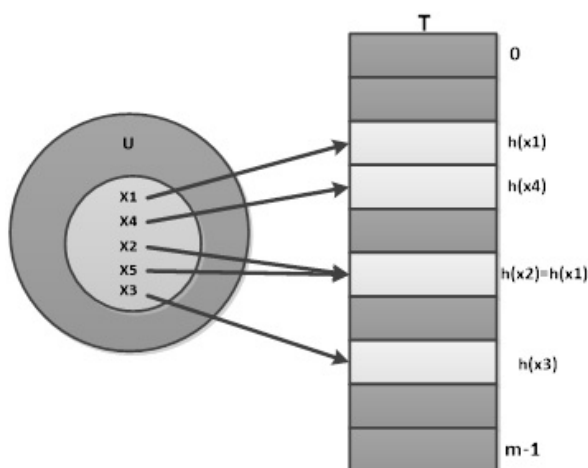


Figura 5.1 [1]

Si el universo de llaves es pequeño y todos los elementos tienen llave distinta, se realiza un direccionamiento o asignación directa, donde a cada posición en la tabla le corresponde una llave del universo U . Figura 5.2. La búsqueda y las otras funciones de diccionario se pueden escribir de forma simple como:

Búsqueda_DDirecto($T, llave$)

Inicio

Retorna $T[llave]$

Fin

Agregar_DDirecto($T, valor$)

Inicio

$T[llave.valor] = valor$

Fin

Borra_DDirecto($T, valor$)

Inicio

$T[llave.valor] = NULL$

Fin

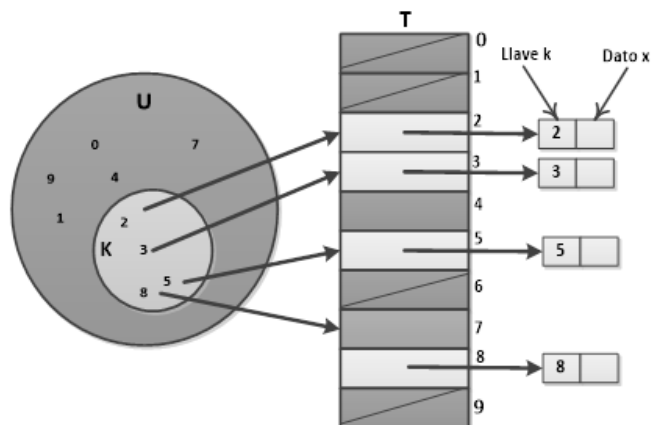



Figura 5.2

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	52/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

En ocasiones se puede generar una **colisión**, que se define como una misma dirección para dos o más llaves distintas. Figura 5.1. Afortunadamente hay algunas técnicas para resolver el conflicto de las colisiones, aunque lo ideal sería no tener este problema.

Una función hash ideal debería ser biyectiva, es decir, que a cada elemento le corresponda un índice o dirección, y que a cada dirección le corresponda un elemento, pero no siempre es fácil encontrar esa función.

Entonces para trabajar con este método de búsqueda se necesitan principalmente:

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

En lo siguiente, se mencionan algunas formas de cómo plantear una función hash y de cómo tratar las colisiones.

Universo de llaves

Para el diseño de una función hash se asume que el universo de llaves es el conjunto de números naturales o enteros positivos y si las llaves a usar no lo son, primero se busca la manera de expresarlos como tal.


Por ejemplo, si las llaves son letras de algún alfabeto, se puede asignar a cada letra el valor entero de su posición en el alfabeto o su valor correspondiente en algún código (lo que se puede expresar como $ord(x)$).

Si las llaves son cadenas de caracteres $c_0c_1c_2 \dots c_{n-1}$

El natural se puede obtener como $\sum_{i=0}^{n-1} ord(c_i)$ que es la suma de los valores numéricos asignados a cada carácter.

También se puede obtener la llave x en forma de entero positivo como:

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

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	53/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Donde n es el número de caracteres de la llave, $Clave[i]$ corresponde con la representación ASCII del i -ésimo carácter, y $p[i]$ es un entero procedente de un conjunto de pesos generados aleatoriamente para $1 \leq i \leq n$. La ventaja de utilizar pesos es que dos conjuntos de pesos diferentes $p_1[i]$ y $p_2[i]$ llevan a dos funciones de transformación diferente $h_1(x)$ y $h_2(x)$ [3].

Para explicar algunas de las funciones hash se asume que el universo de llaves es el conjunto de los naturales.

Funciones Hash

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

En la literatura se han estudiado diferentes funciones de transformación, en esta guía se explican solo los tres esquemas planteados en [1] para el diseño de una buena función hash. En dos de los esquemas se utilizan las operaciones de multiplicación y división que son heurísticas mientras que en el tercer esquema se utiliza el llamado *hashing* universal que es un procedimiento aleatorio.

Método de división

En este método se mapea una llave x en una de las celdas de la tabla tomando el residuo de x dividido entre m .

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

Se recomienda que m no sea potencia de dos. A esta operación se le llama módulo.

Ejemplo [2]. Suponer que se tienen ocho estudiantes en una escuela y sus números de cuenta son


197354864, 933185952, 132489973, 134152056, 216500306, 106500306, 216510306 y 197354865.

Se quiere almacenar la información de cada estudiante en una tabla hash en orden.

Entonces sean las llaves $x_1 = 197354864$, $x_2 = 933185952$, $x_3 = 132489973$, $x_4 = 134152056$, $x_5 = 216500306$, $x_6 = 106500306$, $x_7 = 216510306$, y $x_8 = 197354865$.

Si la tabla hash es de tamaño 13 y esta indexada del 0,1,2,3, ...,12 se define la función hash

$h: \{x_1, x_2, x_3, \dots, x_8\} \rightarrow \{0,1,2, \dots, 13\}$ como $h(x_i) = x_i \% 13$ (% representa el operador módulo) y aplicando la función a las llaves se obtiene lo siguiente:

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	54/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

$h(x1) = h(197354864) = 197354864 \% 13 = 5$	$h(x5) = h(216500306) = 216500306 \% 13 = 9$
$h(x2) = h(933185952) = 933185952 \% 13 = 10$	$h(x6) = h(106500306) = 106500306 \% 13 = 3$
$h(x3) = h(132489973) = 132489973 \% 13 = 5$	$h(x7) = h(216510306) = 216510306 \% 13 = 12$
$h(x4) = h(134152056) = 134152056 \% 13 = 12$	$h(x8) = h(197354865) = 197354865 \% 13 = 6$

Donde se puede observar que el estudiante con número de cuenta 132489973 se va a almacenar en la posición 5 de la tabla y ésta ya está ocupada por el estudiante con número de cuenta 197354864, lo mismo sucede con la posición 12. Lo anterior ejemplifica lo que es una colisión.

Método de multiplicación

Este método opera en dos partes, primero multiplicamos la llave x por una constante A en un rango $0 < A < 1$ (xA) y se extrae la parte fraccional que se multiplica por m (m es una potencia de 2) es decir se calcula $m(xA \bmod 1)$. Por último, se obtiene el mayor número entero menor o igual al resultado obtenido.

$$h(x) = \lfloor m(xA \bmod 1) \rfloor$$

Hashing Universal

A veces para una sola función hash puede haber un conjunto de llaves que produzcan todas las mismas salidas o le sea asignado la misma dirección en la tabla. Una forma de minimizar esto es, en lugar de usar una sola función hash ya definida, se puede seleccionar una función *hash* de forma aleatoria de una familia de funciones H . A esto se le denomina *hashing universal*.


Definición: Sea U el 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 $x \neq y$

$$|\{h \in H: h(x) = h(y)\}| = \frac{|H|}{m}$$

En otras palabras, la probabilidad de una colisión para dos llaves diferentes x e y dada una función hash aleatoria seleccionada del conjunto H es $\frac{1}{m}$.

Construcción de un conjunto universal de funciones Hash.

Para formar un conjunto H de funciones hash se llevan a cabo los siguientes pasos:

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	55/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Paso 1

Seleccionar el tamaño de la tabla m tal que sea primo.

Paso 2

Descomponer la llave x en $r + 1$ bytes esto es $x = \langle x_0, x_1, x_2, \dots, x_r \rangle$, donde $x_i \in \{0, 1, 2, \dots, m - 1\}$
Equivalente a escribir x en base m .

Paso 3

Sea $a = \langle a_0, a_1, a_2, \dots, a_r \rangle$ una secuencia de $r + 1$ elementos seleccionados aleatoriamente tal que $a_i \in \{0, 1, 2, \dots, m - 1\}$. Hay m^{r+1} posibles secuencias.

Paso 4

Definir a la función hash $h_a = \sum_{i=0}^r a_i x_i \mod m$.

Paso 5

El conjunto H de funciones hash es:

$$H = \bigcup_a h_a$$

Con m^{r+1} miembros, uno para cada posible secuencia a .

Ejemplo:


Se tiene que el universo de claves es el conjunto de direcciones IP, y cada dirección IP es una 4-tupla de 32 bits $\langle x_1, x_2, x_3, x_4 \rangle$, donde $x_i \in \{0, 1, 2, \dots, 255\}$

Sea m un número primo, por ejemplo $m=997$ si se desean almacenar 500 IPs

Se define h_a para 4-tupla $a = \langle a_0, a_1, a_2, a_4 \rangle$ donde $a_i \in \{0, 1, 2, \dots, m - 1\}$ como:

$$h_a: \text{DireccionIP} \rightarrow \text{Posicion Tabla}$$

$$h_a(x_0, x_1, x_2, x_4) = (a_1 x_1 + a_2 x_2 + a_3 x_3 + a_4 x_4) \mod m$$

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	56/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Resolución de Colisiones

Las técnicas de resolución de colisiones se clasifican en dos categorías

- 1) Direccionamiento abierto (*Open Addressing*) o también llamado *closed hashing*.
- 2) Enlazamiento (*Chaining*) o también llamado *open hashing*.

Direccionamiento abierto

Cuando el número n de elementos en la tabla se puede estimar con anticipación, existen diferentes métodos para almacenar n registros en una tabla de tamaño m donde $m > n$, los cuales utilizan las entradas vacías de la tabla para resolver colisiones, esos métodos se denominan métodos de direccionamiento abierto.

Aquí cuando una llave x se direcciona a una entrada de la tabla que ya está ocupada, se elige una secuencia de localizaciones alternativas $h_1(x), h_2(x) \dots$ dentro de la tabla. Si ninguna de las $h_1(x), h_2(x) \dots$ posiciones se encuentra vacía, entonces la tabla está llena y x no se puede insertar.

Existen diferentes propuestas para elegir las localizaciones alternativas. Las más sencillas se denominan *hashing* lineal, en el que la posición $h_j(x)$ de la tabla viene dada por:

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

$m = \text{tamaño de la tabla}$


Ejemplo

Retomando el ejemplo de los ocho estudiantes donde se sabe que:

$h(197354864) = 5 = h(132489973)$	$h(134152056) = 12 = h(216510306)$	$h(106500306) = 3$
$h(933185952) = 933185952 \% 13 = 10$	$h(216500306) = 9$	$h(197354865) = 6$

Utilizando el *hashing* o prueba lineal se tiene.

Número de Cuenta(NC)	$h(\text{NC})$	$(h(\text{NC})+1)\%13$	$(h(\text{NC})+2)\%13$
197354864	5		
933185952	10		
132489973	5	6	
134152056	12		
216500306	9		
106500306	3		
216500306	12	0	
197354865	6	7	

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	57/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Enlazamiento

Consiste en colocar los elementos mapeados a la misma dirección de la tabla hash en una lista ligada. Figura 5.3 La posición o dirección j contiene un apuntador al inicio de la una lista ligada que contiene todos los elementos que son mapeados a la posición j . Si no hay elementos entonces la localidad tendrá un NULL.

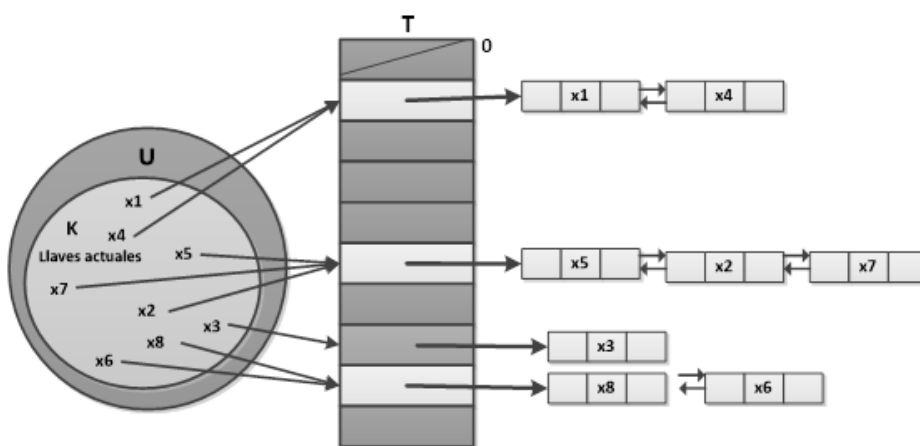


Figura 5.3 [1]

La búsqueda en la tabla cuando las colisiones se trabajan con enlazamiento se puede escribir como sigue:

Busqueda_Enlazamiento(T, k)

Inicio


Busca para un elemento con llave k en la lista $T[h(k)]$

Fin

Desarrollo

Actividad 1

Ir realizando un programa en Python donde dada una llave formada por una cadena de caracteres (letras y dígitos), se pueda insertar y buscar el valor o dato correspondiente a esa llave en la tabla hash. Para este desarrollo se tratarán las colisiones utilizando el direccionamiento abierto.

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	58/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Aunque en Python ya se cuenta con una estructura de datos de diccionario, en esta guía se hará el mapeo a la tabla hash mediante listas para entender lo explicado antes.

Lo primero que se hará es crear la lista vacía de tamaño n , lo cual se puede hacer con la siguiente función

```
#crear arreglo indexado 0-tamaño
def formaArreglo(tamaño):
    Arr=[None]*tamaño
    return Arr
```


Ahora para representar la llave formada por una cadena de caracteres como un valor entero, se usa una función que suma el valor de cada carácter en ASCII y lo que retorna representará a la llave. La función en Python queda:

```
# Convertir la llave a valor numérico
def obtenerLlaveNumerica(llave):
    hash=0
    for char in str(llave):
        hash+=ord(char)
    return hash
```

Como función hash se utiliza $h(x) = x \bmod m$, la función en Python es:

```
#Funcion Hash
def H(llaveN):
    return llaveN%5
```

Para agregar un elemento se tiene la siguiente función en Python que considera el manejo de colisiones utilizando el *hashing* Lineal. La función recibe información acerca de la llave y el valor a insertar; la tabla hash y su tamaño correspondiente.

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	59/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

```
# Función donde dada una llave, se genera la dirección o índice
# en la tabla llamada map de tamaño n donde se agrega un valor.
def agregar(llave, valor, map, tamaño):


    # Dada la llave obtener el lugar donde se colocara valor (dirección)
    llave_hash = H(obtenerLlaveNumerica(llave))
    # Datos a colocar
    ParllaveValor = [llave, valor]

    # Si la dirección o posición esta vacía colocar datos
    if map[llave_hash] is None:
        map[llave_hash] = list([ParllaveValor])
        return True
    else:
        # Si la llave dada ya fue agregada, colocar valor en el mismo sitio

        for par in map[llave_hash]:
            if par[0] == llave:
                par[1] = valor
                return True

        # Si la llave genera una dirección ya ocupada (colisión),
        # se busca otra dirección
        # manejo de colisión con hash lineal
        for j in range(tamaño):
            llaveh = (llave_hash + j) % 13
            # Si la tabla ya esta llena
            if (llaveh == len(map)):
                print("Tabla llena", llave_hash)
                break
            else:
                # Si ya se encuentra una dirección vacía, se coloca el valor
                if map[llaveh] is None:
                    map[llaveh] = list([ParllaveValor])
                    return True
```

Una vez que se introduzcan elementos, será posible localizar datos en la tabla hash. La siguiente función en Python realiza una búsqueda, la cual recibe como información la llave y el tamaño de la tabla hash.

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	60/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

```
#Función que localiza el valor dada una llave
def buscar(llave,tamaño):
    #Dada la llave obtener el lugar donde probablemente
    #se encuentre el valor buscado
    llave_hash=H(obtenerLlaveNumerica(llave))
    #Si la posición no esta vacia
    if map[llave_hash] is not None:

        for par in map[llave_hash]:
            #Si la llave está en la posición generada por la función
            #se retorna el valor buscado
            if par[0] == llave:
                return par[1]
            #Si la llave generó una entrada que no contiene lo buscado
            # ¡Colisión!.Buscar posición alternativa
            else:
                for j in range(tamaño):
                    llaveh=(llave_hash+j)%13
                    #Si ya se busco en toda la tabla
                    if (llaveh==len(map)):
                        break

                for parl in map[llaveh]:
                    #Si ya se localizó la dirección donde esta lo buscado
                    #retornar valor
                    if parl[0]== llave:
                        return parl[1]


    return None
```

Para visualizar qué pasa, primero se forma la tabla, en este ejemplo con 10 elementos, lo cual se consigue llamando a la función formaArreglo() como sigue:

```
map=formaArreglo(10);
```

Ahora, se agregan elementos (llave, valor) de la siguiente manera:

```
agregar("Hola9","12213299",map,10)
agregar("Hola4",12213214,map,10)
agregar("Hola1",1221321,map,10)
agregar("Hola2",1221322,map,10)
agregar("Hola3",1221323,map,10)
agregar("Hola5",1221325,map,10)
agregar("Hola6",1221326,map,10)
agregar("Hola7",1221327,map,10)
agregar("Hola8",1221328,map,10)
agregar("Hola10",1221310,map,10)
```

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	61/183
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Hasta aquí se han agregado 10 elementos, agregar uno más y ver qué sucede. Describirlo_____

Antes de buscar un elemento se verá qué hay en la tabla.

```
print (map)
```

Ahora, para buscar un elemento, se llama a la función correspondiente de la siguiente forma:

```
print (buscar("Hola1",10))
```

¿Qué pasa si se busca un elemento que no esté en la tabla? _____

Actividad 2

Ejercicios propuestos por el profesor.

Referencias

[1] CORMEN, Thomas, LEISERSON, Charles, et al.

Introduction to Algorithms

3rd edition

MA, USA

The MIT Press, 2009

[2] D.S.Malik

Data Structure Using C++

Course Technology, Cengage Learning

Second Edition

[3] Ziviani, Nivio

Diseño de algoritmos con implementaciones en Pascal y C

Ediciones paraninfo /Thomson Learning

2007