	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	47/180
		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

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	48/180
		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 sirve para obtener la posición donde están guardados los datos que se buscan. Se utiliza una función que transforma la llave o dato clave en una dirección o posición 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.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	49/180
		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			

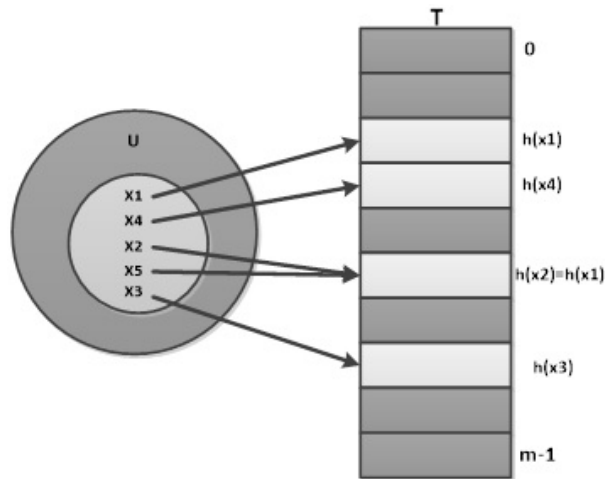


Figura 5.1 [1]

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*

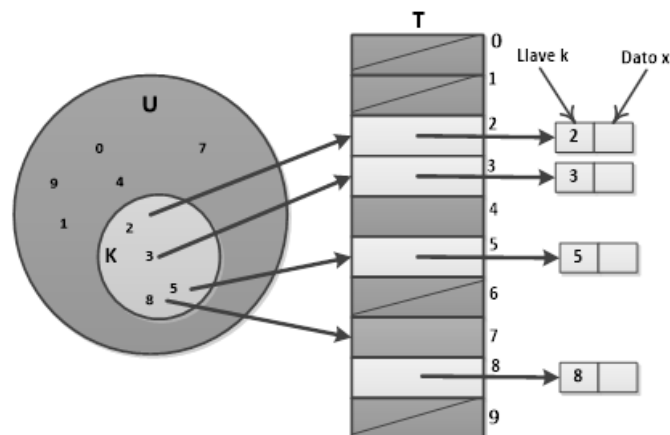



Figura 5.2

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	50/180
		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:

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

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	51/180
		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 Tabla 5.1:

$h(x_1) = h(197354864) = 197354864 \% 13 = 5$	$h(x_5) = h(216500306) = 216500306 \% 13 = 9$
$h(x_2) = h(933185952) = 933185952 \% 13 = 10$	$h(x_6) = h(106500306) = 106500306 \% 13 = 3$
$h(x_3) = h(132489973) = 132489973 \% 13 = 5$	$h(x_7) = h(216510306) = 216510306 \% 13 = 12$
$h(x_4) = h(134152056) = 134152056 \% 13 = 12$	$h(x_8) = h(197354865) = 197354865 \% 13 = 6$

Tabla 5.1

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	52/180
		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 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  $\{0,1,2,3,\dots,m-1\}$ . 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:

#### Paso 1

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

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	53/180
		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 2

Descomponer la llave  $x$  en  $r + 1$  elementos 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:

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

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

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	54/180
		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 (*Chainhing*) 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 en el cálculo de las direcciones en la tabla se obtuvieron colisiones Tabla 5.2:

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


Tabla 5.2

Utilizando el *hashing* o prueba lineal se tiene: Tabla 5.3.

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	

Tabla 5.3



	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	55/180
		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			

### Operaciones de Búsqueda e Inserción con manejo de colisiones utilizando hashing lineal

Cuando se busca algún dato o registro en una tabla hash por lo general está ya contiene información, por lo que primero se muestra un algoritmo en pseudocódigo que permite realizar la inserción de un par  $(x, valor)$  en una tabla  $T$  de tamaño  $m$ , donde  $x$  es la llave y  $valor$  el dato o registro correspondiente a la llave  $x$ . Tabla 5.4

Índice	(Llave,valor)
0	$(x_4, valor)$
1	
2	
3	
4	
$m-1$	

Tabla 5.4

En el algoritmo, si la dirección en la tabla ya está ocupada, se vuelve a calcular otra utilizando hashing lineal.

Insertar( $T, m, x, valor$ )

Inicio

$j=0$

$h=h(x)$

Mientras  $j < m$

$indice=(h+j) \bmod m$

$par=(x, valor)$

    Si  $T[indice]$  está vacía

$T[indice]=par$

        retornar indice

    Si no


$j=j+1$

    Fin sino

Fin Mientras

retornar -1

Fin

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	56/180
		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			

Para el algoritmo correspondiente a la búsqueda de un dato dada una llave  $x$ , si en la dirección o índice calculado de la tabla no se encuentra el valor buscado entonces se considera una colisión y se vuelve a calcular el índice utilizando hashing lineal hasta que exista coincidencia o se determine que no existe esa llave en la tabla.

Buscar( $T, x, \text{valor}, m$ )

Inicio

$j=0$

$h=h(x)$

Mientras  $j < m$

$\text{indice}=(h+j)\text{mod } m$

    Si  $T[\text{indice}]$  no está vacía

        Si  $T[\text{indice}][0]==x$

            retornar  $T[\text{indice}][1]$

        Si no

$j=j+1$

    Fin si no

Fin Si

No existe esa llave

    retornar -1


Fin Mientras

retornar -1

Fin

## 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.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	57/180
		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			

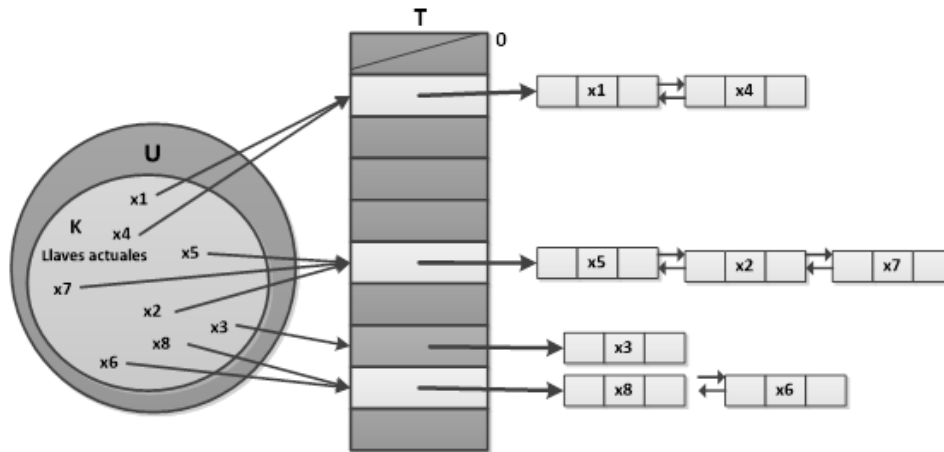


Figura 5.3 [1]

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.

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 tabla de tamaño  $m$  utilizando una lista vacía de tamaño  $n$ , lo cual se puede hacer con la siguiente función

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	58/180
		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			

```
def formarTabla(m):
    T=[None]*m
    return T
```

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:


```
def convertirLlave(x):
    keyNum=0
    i=0
    for char in x:
        keyNum += ord(char)*i
        i+=1
    return keyNum
```

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

```
def h(x,m):
    return x%m
```

Para agregar un elemento se tiene la siguiente función en Python que considera el manejo de colisiones utilizando el *hashing* lineal y se basa en el pseudocódigo mostrado anteriormente en esta guía. La función recibe información acerca tabla T, su tamaño m, la llave “x” y el valor a insertar.

```
def insertar(T,m,x,valor):
    j=0
    h1=h(convertirLlave(x),m)
    while (j<m):
        indice =(h1+j)%m
        par=[x,valor]
        if(T[indice]==None):
            T[indice] = par
            return indice
        else:
            j +=1
    print ("No hay lugar")
    return -1
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	59/180
		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			

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 la misma información que la función insertar().

```
def buscar(T,m,x,valor):
    j=0
    h1=h(convertirLlave(x),m)
    while(j<m):
        indice = (h1+j)%m
        if(T[indice]!= None):
            if(T[indice][0]==x):
                return indice
            else:
                j+=1
        return -1
    return -1
```

Para visualizar qué pasa se define la función main() donde primero se forma la tabla de tamaño m=5, después se agregan 6 elementos (llave, valor) y por último se busca un elemento de los insertados.

```
def main():
    m= 5
    T=formarTabla(m)
    print(T)
    insertar(T,m,"Hola1","12213291")
    insertar(T,m,"Hola2","12213292")
    insertar(T,m,"Hola3","12213293")
    insertar(T,m,"Hola4","12213294")
    insertar(T,m,"Hola5","12213295")
    insertar(T,m,"Hola6","12213296")


    print(T)
    #retorna el indice de la tabla donde localizo el valor dada la llave
    print (buscar(T,m,"Hola5","12213295"))
main()
```

¿Qué sucede al agregar el sexto elemento? \_\_\_\_\_

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

## Actividad 2

Ejercicios propuestos por el profesor.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II</b>	Código:	MADO-20
		Versión:	01
		Página	60/180
		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			

## 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