	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación
salas A y B

Profesor: Ing. Jonathan Roberto Torres Castillo

Asignatura: Estructura De Datos Y Algoritmos II

Grupo: 5

No de Práctica(s): 5

Integrante(s): Rosas Martínez Jesús Adrián

Semestre: 2018-I

Fecha de entrega: 21 de marzo de 2018

Observaciones:

CALIFICACIÓN: _____

PRACTICA 5
ALGORITMOS DE BÚSQUEDA PARTE II
ESTRUCTURA DE DATOS Y ALGORITMOS II
UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Profesor: Ing. Jonathan Roberto Torres Castillo

Alumno: Rosas Martínez Jesús Adrián
Correo: jesus_olimpo@outlook.com

1. Reglas de las prácticas de laboratorio

- Deberá respetar la estructura general del documento proporcionado para la respectiva práctica y llenar cada una de las secciones que aparecen en él.
- El desarrollo de la práctica deberá ser autentico. Aquellos alumnos que presenten los mismos cálculos, código fuente, etcétera, se les será anulada inapelablemente la práctica correspondiente con una calificación de 0.
- Deberá subir el informe y los scripts al link de DropBox correspondiente a la práctica que se encuentra en la carpeta del curso. Los archivos deberán tener las características descritas en el archivo "Reglas_Laboratorio.pdf" o se anularan son calificación de 0.
- El día de entrega establecido deberá ser respetado por todos los alumnos, la práctica debe ser terminada parcialmente al finalizar la clase, completada máximo a los 7 días siguientes a la sesión de laboratorio tiempo en el cual debe subir la practica al link correspondiente y sustentada en la primera hora de clase de la siguiente sesión de laboratorio.
- No se recibirán informes ni actividades si no asiste a la sesión de laboratorio y no presenta justificación médica, por lo cual no tendrá derecho a la calificación de dicha práctica y la nota será de 0.
- La inasistencia a 3 sesiones de laboratorio en el semestre será causal de reprobación de las prácticas y del curso.

2. Objetivos

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

3. Introducción

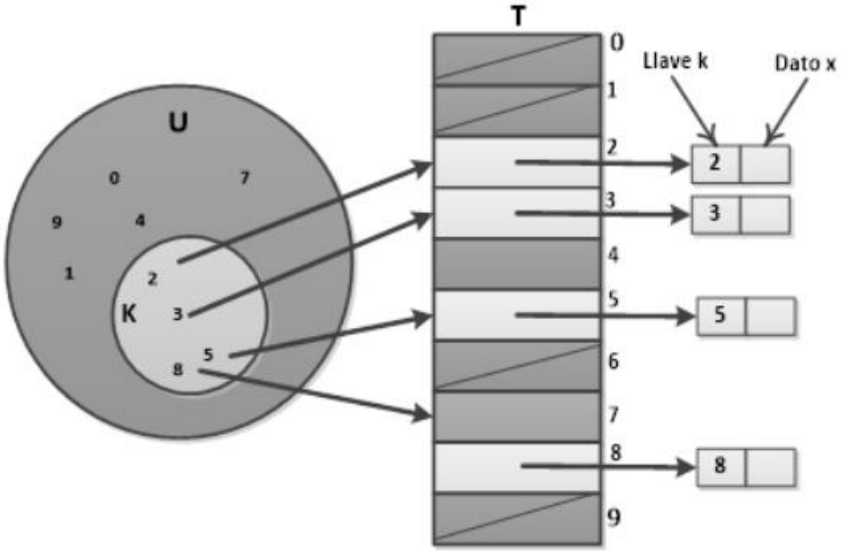
Una tabla hash es una estructura de datos para implementar un tipo de dato abstracto (TDA) diccionario.

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 y se obtiene la dirección del elemento en la tabla. En ocasiones se puede generar una colisión, que se define como una misma dirección para dos o más llaves distintas. 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.

Para trabajar con este método de búsqueda se necesita 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.



4. Planteamiento del Problema y Desarrollo

Realizar un programa en cualquier lenguaje de programación donde dada una llave o código formado por una cadena de caracteres (letras y dígitos), se pueda insertar o buscar el valor o dato (ej. posición) correspondiente a esa llave en la tabla hash. En esta práctica el estudiante debe realizar el mapeo a la tabla hash mediante el método de división o **función modulo** y para tratar las colisiones se debe usar el **método de direccionamiento abierto**.

Actividad 1

1.1. Desarrollar una función que permita crear una tabla o arreglo de datos de dimensión 'm', la cual debe tener la siguiente estructura.

Función_Tabla(m)
Crear arreglo de datos nulos (None) de tamaño m.
Retornar arreglo
fin

En este apartado muestre **únicamente** la función implementada y el funcionamiento de la misma:

```
def forma_tabla(m):
    return arreglo

print(forma_tabla(13))
```

```
C:\Users\jesus\Desktop>py act1.py
[None, None, None, None, None, None, None, None, None, None, None, None, None]
```

1.2. Desarrollar una función que permita transformar una llave o clave alfanumérica tipo string a un índice numérico. La función debe obtener el valor numérico al que corresponde cada carácter de la cadena de texto que forma la llave y realizar una sumatoria de estos valores, luego usar dicho resultado para obtener el índice usando la función modular. La función debe tener la siguiente estructura.

Funcion_hash(llave,m)
Obtener el valor numérico de cada carácter de 'llave'
Sumar esos valores
Obtener modulo m de la suma anterior
Retornar resultado
fin

El parámetro '*llave'* es una cadena de caracteres y el parámetro '*m'* es un valor numérico.

En este apartado muestre **únicamente** la función implementada y el funcionamiento de la misma:

```python
def hash(llave,m):
    suma=0
    for i in range(len(llave)):
        suma=suma+ord(llave[i])
    print("Suma: ",suma)
    return suma%m

#llave=[74+69+83+85+83+57]=[451]
print("Resultado: ",hash("JESUS9",len("JESUS9")))
#451%6=1
```

```
C:\Users\jesus\Desktop>py act2.py
Suma:  451
Resultado:  1
```

Actividad 2

2.1. Desarrollar una función que permita agregar en una tabla o arreglo un valor o información, para obtener la posición o índice de la tabla en el cual se debe guardar este valor se debe hacer una transformación a la llave que contiene dicha información. La función debe tener la siguiente estructura.

*funcion_agregar(llave,valor,tabla) Ej: Funcion_agregar('*XFRT65*',0,10)*

Donde '*llave*' es la llave o id (como ejemplo se da esta la llave XFRT65, que es una referencia de una reserva aérea). El parámetro '*valor'* hace referencia a los datos o información que contiene la llave, como ejemplo se da la posición 0 que en este caso establece que esa llave se encuentra en la primera posición, pero este parámetro también podría contener otro tipo de información. Por ultimo está el parámetro '*tabla'* que es el arreglo en el cual se va a guardar *'valor'*. Los subprocesos de la función son deben ser los siguiente.

funcion_agregar(llave,valor,tabla)

Dimensión de la tabla (m)
Obtener el índice donde se colocara 'valor' (usar Funcion_hash)
Si la tabla en la posición 'índice' está vacía colocar 'valor' en esa posición y finalizar proceso.
Sino
Si la llave dada ya fue agregada, actualizar con el nuevo valor en el mismo sitio y finalizar proceso.
Sino usar direccionamiento abierto para solucionar la colisión.
Si se logra solucionar retornar 'true' y finalizar proceso.
Sino retornar mensaje indicando que llave y valor no se pudo agregar a la tabla.

fin

En este apartado muestre **únicamente** la función implementada y el funcionamiento de la misma:

```
def agregar(llave,valor,tabla):
    llave_hash=1
    ParllaveValor=[llave,valor]
    if tabla[llave_hash] is None:
        tabla[llave_hash]=list(ParllaveValor)
        return True
    else:
        i=0
        for par in tabla[llave_hash]:
            if par==llave:
                i+=1
                continue
            if par==valor:
                i+=1
            if i==2:
                return True
    for j in range(len(tabla)):
        llaveh=(llave_hash+j)%13
        if llaveh>=len(tabla):
            print("Tabla llena",llave_hash)
            break
        else:
            if tabla[llaveh] is None:
                tabla[llaveh]=list(ParllaveValor)
                return True

tabla=[None,None,None,None,None,None,None,None,None,
None,None,None,None]
agregar("JESUS9",0,tabla)
print(tabla)
```

```
C:\Users\jesus\Desktop>py act3.py
[None, ['JESUS9', 0], None, None, None, None, None, None, None, None, None, None, None]
```

2.2. Desarrollar una función que permita buscar en una tabla o arreglo una llave y obtener el valor o información que contiene dicha llave. La función debe tener la siguiente estructura.

funcion_buscar (llave,tabla)

Dimensión de la tabla (m)

Obtener el índice donde es posible que se encuentre la llave (usar Funcion_hash)

Si tabla en la posición de 'índice' está vacía retornar 'None' y finalizar proceso.
Sino
Obtener el valor de la llave que guarda tabla en 'índice' y compararlo con la llave ingresada en la función.
Si son iguales retornar el valor que se encuentra en la tabla y finalizar proceso.
Sino comparar la llave ingresada con cada una de las llaves que guarda la tabla.
Si es exitosa alguna de las comparaciones retornar el valor que se encuentra en la tabla y finaliza el proceso.
Sino retornar 'None'.

fin

En este apartado muestre **únicamente** la función implementada y el funcionamiento de la misma:

```
def buscar(llave,tabla):
    llave_hash=1
    if tabla[llave_hash] is not None:
        for par in tabla[llave_hash]:
            if par==llave:
                return (tabla[llave_hash][1])
            else:
                for j in range(len(tabla)):
                    llaveh=(llave_hash+j)%13
                    if llaveh==len(tabla):
                        break
                    for par1 in tabla[llaveh]:
                        if par1==llave:
                            return (tabla[llaveh][1])
    return None

tabla=[None,["JESUS9",5],None,None,None,None,None,None,
None,None,None,None,None]
print("Resultado: ",buscar("JESUS9",tabla))
```

```
C:\Users\jesus\Desktop>py act4.py
Resultado:  5
```

Actividad 3

3.1. Desarrollar un programa agrupando todas las funciones de la actividad 1 y actividad 2 y ejecute las siguientes líneas de código.

```
vec_inf=['987632','453462','453452','564753','568679','345476','879737','456479'
  ,'436564','123346','645645','296372']

m=m
tabla_H=Funcion_Tabla(m)
for i in range(len(vec_inf)):
   agregar(vec_inf[i],i,tabla_H)

print("\n",tabla_H)
```

Muestre los resultados y describa lo que sucede.

```
C:\Users\jesus\Desktop>py act5.py
 [['453462', 1], ['564753', 3], ['123346', 9], ['645645', 10], ['436564', 8], ['987632', 0],
['453452', 2], ['568679', 4], ['345476', 5], ['879737', 6], ['456479', 7], ['296372', 11]]
```

Al agrupar las funciones mostradas anteriormente, se ejecuta la primera función que se encarga de crear un arreglo del tamaño o dimensión "m". Como la lista de llaves o elementos se encuentran una detrás de otra se usa un ciclo for para recorrer el arreglo que las contiene. En cada iteración o vuelta se manda a llamar la función agregar en la cual se pasan como parámetros cada una de las llaves (una por iteración), su posición o valor y la tabla donde se guardarán. Al final se imprime la tabla con las llaves, las cuales presentaron colisiones ya que están en un orden distinto al que tenían en el arreglo inicial.

3.2. Siguiendo el ejercicio anterior ahora se debe buscar la llave '345476' y obtener su información (posición) usando la tabla creada.

```
Resultado=función_buscar ("345476",tabla_H)
```

Muestre los resultados y describa lo que sucede.

```
C:\Users\jesus\Desktop>py act5.py
 [['453462', 1], ['564753', 3], ['123346', 9], ['645645', 10], ['436564', 8], ['987632', 0],
['453452', 2], ['568679', 4], ['345476', 5], ['879737', 6], ['456479', 7], ['296372', 11]]
Resultado:  5
```

Al tener la tabla con las llaves guardadas, se llama a la función buscar la cual recibe como parámetros la llave a encontrar y la tabla en la que se hará la búsqueda. Dentro de la función

buscar, se vuelve a hacer uso de la función hash la cual obtiene el índice en el que posiblemente se encuentra la llave; se hace la búsqueda y si coincide la llave en dicho índice con la llave a buscar se muestra en pantalla la posición o valor que tiene la llave, sino se recorre todo el arreglo y se vuelve a realizar el direccionamiento abierto (colisión) el cual se usará como nuevo índice para buscar la llave. En este caso, como la llave dada estaba en la tabla se mostró su posición, en caso contrario se mostraría que la llave buscada no está en la tabla.

3.3. Siguiendo el ejercicio anterior ahora se debe buscar la llave '296372' y obtener su información (posición) usando la tabla creada.

*Resultado=función_buscar ("*296372*",tabla_H)*

Muestre los resultados y describa lo que sucede.

```
C:\Users\jesus\Desktop>py act5.py
 [['453462', 1], ['564753', 3], ['123346', 9], ['645645', 10], ['436564', 8], ['987632', 0],
['453452', 2], ['568679', 4], ['345476', 5], ['879737', 6], ['456479', 7], ['296372', 11]]
Resultado:  11
```

Al tener la tabla con las llaves guardadas, se llama a la función buscar la cual recibe como parámetros la llave a encontrar y la tabla en la que se hará la búsqueda. Dentro de la función buscar, se vuelve a hacer uso de la función hash la cual obtiene el índice en el que posiblemente se encuentra la llave; se hace la búsqueda y si coincide la llave en dicho índice con la llave a buscar se muestra en pantalla la posición o valor que tiene la llave, sino se recorre todo el arreglo y se vuelve a realizar el direccionamiento abierto (colisión) el cual se usará como nuevo índice para buscar la llave. En este caso, como la llave dada estaba en la tabla se mostró su posición, en caso contrario se mostraría que la llave buscada no está en la tabla.

5. Código

En esta sección se presenta el código fuente del programa que permitió cumplir los objetivos propuestos.

```
#Actividad1.1

def forma_tabla(m):
        return arreglo

print(forma_tabla(13))
```

```
#Actividad1.2

def hash(llave,m):
        suma=0
        for i in range(len(llave)):
                suma=suma+ord(llave[i])
        print("Suma: ",suma)
        return suma%m

#llave=[74+69+83+85+83+57]=[451]
print("Resultado: ",hash("JESUS9",len("JESUS9")))
#451%6=1
```

```
#Actividad2.1

def agregar(llave,valor,tabla):
   llave_hash=1
   ParllaveValor=[llave,valor]
   if tabla[llave_hash] is None:
      tabla[llave_hash]=list(ParllaveValor)
      return True
   else:
      i=0
      for par in tabla[llave_hash]:
         if par==llave:
            i+=1
            continue
         if par==valor:
            i+=1
         if i==2:
            return True
```

```
    for j in range(len(tabla)):
        llaveh=(llave_hash+j)%13
        if llaveh>=len(tabla):
            print("Tabla llena",llave_hash)
            break
        else:
            if tabla[llaveh] is None:
                tabla[llaveh]=list(ParllaveValor)
                return True

tabla=[None,None,None,None,None,None,None,None,None,
None,None,None,None]
agregar("JESUS9",0,tabla)
print(tabla)
```

```
#Actividad2.2

def buscar(llave,tabla):
    llave_hash=1
    if tabla[llave_hash] is not None:
        for par in tabla[llave_hash]:
            if par==llave:
                return (tabla[llave_hash][1])
            else:
                for j in range(len(tabla)):
                    llaveh=(llave_hash+j)%13
                    if llaveh==len(tabla):
                        break
                    for par1 in tabla[llaveh]:
                        if par1==llave:
                            return (tabla[llaveh][1])
    return None

tabla=[None,["JESUS9",5],None,None,None,None,None,None,
None,None,None,None,None]
print("Resultado: ",buscar("JESUS9",tabla))
```

```
#Actividad3

def forma_tabla(m): #funcion que crea un arreglo vacio de tamano m
        arreglo=[None]*m
        return arreglo

def hash(llave,m):
        suma=0
        for i in range(len(llave)):
                suma=suma+ord(llave[i])
        return suma%m

def agregar(llave,valor,tabla):
   llave_hash=hash(llave,len(llave))
   ParllaveValor=[llave,valor]
   if tabla[llave_hash] is None:
      tabla[llave_hash]=list(ParllaveValor)
      return True
   else:
      i=0
      for par in tabla[llave_hash]:
         if par==llave:
            i+=1
            continue
         if par==valor:
            i+=1
         if i==2:
            return True
   for j in range(len(tabla)):
      llaveh=(llave_hash+j)%13
      if llaveh==len(tabla):
         print("Tabla llena")
         break
      else:
         if tabla[llaveh] is None:
            tabla[llaveh]=list(ParllaveValor)
            return True
```

```
def buscar(llave,tabla):
    llave_hash=hash(llave,len(llave))
    if tabla[llave_hash] is not None:
        for par in tabla[llave_hash]:
            if par==llave:
                return (tabla[llave_hash][1])
            else:
                for j in range(len(tabla)):
                    llaveh=(llave_hash+j)%13
                    if llaveh==len(tabla):
                        break
                    for par1 in tabla[llaveh]:
                        if par1==llave:
                            return (tabla[llaveh][1])
    return None

vec_inf=['987632','453462','453452','564753','568679','345476',
'879737','456479','436564','123346','645645','296372']

m=len(vec_inf)
tabla_H=forma_tabla(m)
for i in range(len(vec_inf)):
    agregar(vec_inf[i],i,tabla_H)

print("\n",tabla_H)

Resultado=buscar("296372",tabla_H)
print("\nResultado: ",Resultado)
```

6. Conclusiones

En conclusión, los algoritmos de búsqueda son importantes para recuperar y/o buscar información. Anteriormente se había trabajo con los algoritmos de búsqueda por comparación de llaves los cuales mejoraban o no su eficiencia dependiendo las características de la lista (ordenada, etc). En este caso, el algoritmo de búsqueda por transformación de llaves es mucho más eficiente que los anteriores ya que no se hacen comparaciones entre todos los elementos de la lista, lo cual lleva a este algoritmo a tener una complejidad O(1).
Este algoritmo de búsqueda es mucho más abstracto y difícil de entender ya que se trabaja con una tabla hash en la cual se pueden presentar colisiones al tratar de guardar dos

direcciones en un mismo lugar, pero existen opciones para evitar esta situación (en este caso se solucionó mediante direccionamiento directo).
Finalmente se cumplieron los objetivos ya que se usó un método para solucionar las colisiones entre direcciones, y además se usó una función hash para transformar una llave en una dirección para trabajar con ella en una tabla.

7. Referencias

[1] Sáenz García, Elba Karen, Guía práctica de estudio 5, Algoritmos de búsqueda parte 2, Facultad de ingeniería, UNAM.

[2] CORMEN, Thomas, LEISERSON, Charles, et al. Introduction to Algorithms 3rd edition MA, USA The MIT Press, 2009

[3] KNUTH, Donald The Art of Computer Programming New Jersey Addison-Wesley Professional, 2011 Volumen 1.