Programa 3:
Tablas de hash

Estructura de Datos y Algoritmos II

Autor: José Mauricio Matamoros de Maria y Campos

Entrega: Lunes 23 de Marzo, 2020

1 Introducción

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

2 Desarrollo:

Con base en el código de los Apéndices A y B, realice los apartados que se muestran en este documento.

- [1 punto] Ejecute el programa `p05.py`. ¿Qué sucede al agregar el sexto elemento?
- [1 punto] Ejecute el programa `p05.py`. ¿Qué pasa si se busca un elemento que no esté en la tabla?
- [3 puntos] Ejecute el programa `hashtables.py` variando el parámetro m de la clase *HashTable* y con los valores obtenidos llene el Cuadro 1:
- [5 puntos] Cargue el archivo `words.txt` a un diccionario nativo de Python reporte el tiempo promedio de búsqueda de las palabras *elegant*, *fork*, *kawaii* y *plant*. ¿Es más rápido que la implementación de `hashtables.py`? Explique.

Table 1: Tiempo de búsqueda con diferentes tamaños de tabla hash

m	Elegant	Fork	Kawaii	Plant
5				
50				
500				
5000				
50000				

A Archivo `p05.py`

p05.py

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

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

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

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

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


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)
  print(buscar(T, m, "Hola5", "12213295"))


if __name__ == "__main__":
  main()
```

B Archivo **hashtables.py**

hashtables.py

```
class HashTable:

  def __init__(self, m = 5):
    self.T = [None]*m
    for i in range(0, m):
      self.T[i] = []
    self.m = m
    self.longest_list = 0

  def _hash(self, value):
    i = len(value) -1
    wsum = 0
    while i >= 0:
      wsum += ord(value[i]) * i
      i -= 1
    return wsum % self.m

  def insert(self, value):
    if not isinstance(value, str) or len(value.strip()) < 1:
      return
    h = self._hash(value)
    lst = self.T[h]
    # print('      Adding {} with hash {} to {}'.format(value, h, lst))
    i = 0
    while i < len(lst):
      if lst[i] == value:
        print('{} is already in the dictionary'.format(value))
        return
      i += 1
    lst.append(value)
    self.longest_list = max(self.longest_list, i)

  def delete(self, value):
    if not isinstance(value, str) or len(value.strip()) < 1:
      return False
    h = self._hash(value)
    lst = self.T[h]
    i = 0
    while i < len(lst):
      if lst[i] == value:
        del lst[i]
        return True
      i += 1
    return False

  def search(self, value):
    if not isinstance(value, str) or len(value.strip()) < 1:
      return False
    h = self._hash(value)
    lst = self.T[h]
    i = 0
    while i < len(lst):
      if lst[i] == value:
        return True
      i += 1
    return False


def main():
  hashTable = HashTable(5)
  with open('words.txt', 'r') as fp:
    lines = fp.readlines()

  print('Populating dictionary with radix={}'.format(hashTable.m))
```

```
  start_time = time.time()
  for line in lines:
    hashTable.insert(line.strip())
    elapsed = 1000 * (time.time() - start_time)
  print('    Longest list size: {} elements'.format( hashTable.longest_list ))
  print('    Elapsed: {} milliseconds'.format( elapsed ))
  print()

  while True:
    try:
      word = input('Type a word to search: ').strip().lower()
      print('    Searching for {}...'.format(word), end='')
      start_time = time.time()
      found = hashTable.search(word)
      elapsed = 1000 * (time.time() - start_time)
      print('\r    {}. Search took {} milliseconds'.format('Found' if found else 'Not found'
    , elapsed ))

    except KeyboardInterrupt:
      print()
      return



if __name__ == "__main__":
  main()
```