Contenido Extra. Opcional

Programa 3

El objeto de este programar es programar un Tabla Hash. Se facilita al alumno una primera versión de la clase Tabla Hash simple:

```
class SimpleHashTable(object):
    def __init__(self,size):
        # Set up size and slots and data
        self.size = size
        self.slots = [None] * self.size
        self.data  = [None] * self.size
    def put(self,key,data):
        preHash = hash(key)
        hashvalue = self.hashfunction(preHash,len(self.slots))
        if DEBUG: print (key,  preHash, hashvalue)
        if self.slots[hashvalue]:
            raise ValueError(f"Collision found with key {key}",hashvalue)
        self.slots[hashvalue] = key
        self.data[hashvalue] = data
    def hashfunction(self,key,size):
        # Remainder Method
        return key%size
    def get(self,key):
        # Getting items given a key
        # Set up variables for our search
        startslot = self.hashfunction(hash(key),len(self.slots))
        data = None
        position = startslot
        if self.slots[position] == key:
            data = self.data[position]
        return data
    # Special Methods for use with Python indexing
    def __getitem__(self,key):
        return self.get(key)
    def __setitem__(self,key,data):
        self.put(key,data)
```

En esta Tablas Hash surge el problema de las **colisiones**. Las colisiones ocurren cuando en la convergencia entre el gran espacio de las claves, definido por el valor preHash, se hace colapsar sobre el más limitado espacio de hashvalue que, no puede exceder del tamaño de la tabla hash.

Veamos un experimento. Intentemos meter 10 entradas con claves aleatorias en una tabla de tamaño 10, con el modelo simple definido más arriba.

Como los hashvalues que queremos usar como índices en la tabla se encuentran en el rango [0,*size*] es probable que dos claves diferentes proporcionen el mismo hashvalue y eso es una colisión.

Para resolverlo, añadiremos, una lista que acumule las colisiones. La búsqueda en esta lista, no es óptima, pero sólo la utilizaremos en caso de colisión. Para ello, definimos una SimpleHashTableV2 que hereda de lo anterior, añade un atributo adicional, colList y sobrescribimos el método de escritura en la tabla, put,

Una vez finalizado este punto, debemos crea una nueva clase SimpleHashTableV3, reescribimos el método get para gestionar las colisiones.

¿Qué está funcionando mal? Aunque de manera amortizadas (esto es, tomando el valor medio de muchas llamadas) la búsqueda por clave en esta tabla hash es buena (sólo en una fracción delas llamadas, incurrimos en búsquedas secuenciales en la lista de colisiones) el tener una única lista de colisión aumenta la longitud de ésta y el "worst case" empieza a ser del orden del número de elementos a guardar

¿Cómo lo arreglamos? Una opción es tener una lista por cada slot en el que hay colisión. Creamos la versión Veamos la V4 heredando de la versión anterior sobre escribiendo el método.

Informe de la práctica

1. A través de Moodle de la Asignatura de Algorítmicos y Estructura de Datos. Se deberá entregar el viernes 03/03/2023.