Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI-2692 - Laboratorio de Algoritmos y Estructuras de Datos II
Trimestre Enero-Marzo 2020

Implementación de una Tabla de Hash usando el Cuco Hashing (Versión 1.2)

1. Introducción

El objetivo del laboratorio es realizar la implementación de la estructura de datos Tabla de Hash que usa el método de Cuco hashing [1] para la resolución de colisiones. También se quiere que haga un estudio experimental, en el que se comparen los rendimientos de esta Tabla de Hash, con las que se implementó en la semana 9, basada en el método de encadenamiento.

2. Descripción de la Tabla de Hash

Usted debe implementar una tabla que usa Cuco hashing en donde las claves son de tipo *String*, y el valor que esta asociado a cada clave, es un elemento de tipo *String*. La Tabla de Hash estática. Esto es, la tabla se crea con un tamaño determinado y este no cambia durante su vida de ejecución. La figura 1 muestra un ejemplo de la Tabla.

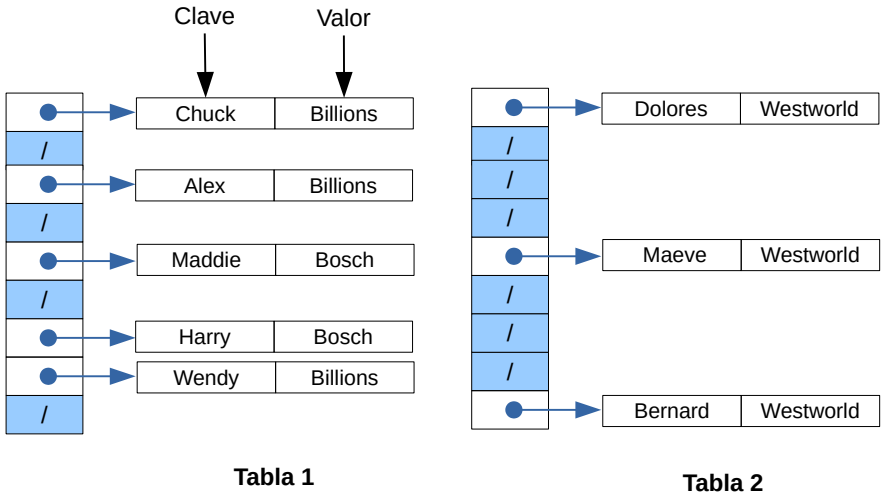

Figura 1: Ejemplo de la Tabla de Hash usando Cuco hashing. Las claves son elementos de tipo *String*, y el valor asociado a cada clave es de tipo *String*.

2.1. Clase `CucoEntry`

El tipo `CucoEntry` debe poseer dos campos: el primero se llama *clave* y el segundo se llama *valor* y ambos son del tipo *String*. El tipo `CucoEntry` debe ser implementado como una clase de Python, en el archivo `CucoEntry.py`. El constructor de la clase *CucoEntry* recibe como entrada un clave, de tipo *String* y un valor asociado con la clave, que también es de tipo `String`. A continuación la firma del constructor.

crearCucoEntry(String *clave*, String *valor*)

2.2. Clase `CucoTabla`

La Tabla de Hash usando Cuco hashing debe ser implementada como una clase pública de Python, llamada `CucoTabla`. El desarrollo de tabla debe tener como base el pseudocódigo presentado en la clase del curso de teoría de Algoritmos y Estructuras de Datos 2 [2]. La tabla almacenará elementos de tipo `CucoEntry`. La `CucoTabla` debe poseer dos arreglos, que contienen a los elementos de tipo `CucoEntry`. Las operaciones de la Tabla de Hash son:

Crear tabla: Constructor de la clase, que crea una tabla con un tamaño inicial de n.

crearCucoTabla(int n)

Agregar clave y valor: Se agrega a la Tabla de Hash una clave c, que tiene asociada un valor v. Si la clave a agregar se encuentra en la Tabla de Hash, entonces se sustituye el valor del dato en la Tabla de Hash por el valor de v. Usted debe prevenir que este método quede en un ciclo infinito, utilizando la condición adecuada para terminar el mismo. Si para agregar un elemento es necesario hacer rehshing, entonces las tablas aumentan su tamaño en un factor de 3/2. Si un elemento ha sido agregado sin necesidad de hacer rehashing, entonces se retorna el valor de *True*, y en caso contrario el método retorna *False*.

agregar(String c, String v)

Eliminar con clave: Dada un clave c, si algún elemento en la Tabla de Hash tiene una clave igual a c, entonces el elemento se elimina de la tabla y retorna el valor asociado a esa clave. En caso de que no haya ninguna clave c en la Tabla de Hash, se retorna `None`.

eliminar(String c) → String

Buscar con clave: Dada un clave c, se busca el elemento en la Tabla de Hash que posea la clave igual a c. Si el elemento se encuentra en la tabla, entonces se retorna el valor asociado a esa clave. En caso de que no haya ninguna clave c en la Tabla de Hash, se retorna `None`.

BUSCAR(String c) → String

Mostrar el contenido: Se muestra por la salida estándar todos los elementos de la Tabla de Hash, en forma de pares clave y valor.

MOSTRAR(void)

2.3. Funciones de hash

El método de Cuco hashing hace uso de dos funciones de hash h_1 y h_2. Para la función h_1 se empleará la función de hash SHA256 [1] . Para la función h_2 se utilizará el algoritmo MD5 [2]. En específico se debe hacer uso de las implementaciones de SHA256 y MD5 que vienen en en el módulo de Python3 `hashlib`. Para este laboratorio se les dará la implementación de las funciones h_1 y h_2. El listado 1 muestra el código de h_1 y h_2.

```python
import hashlib

def h1(clave):
    h = hashlib.sha256(clave.encode())
    return int(h.hexdigest(),base=16)

def h2(clave):
    h = hashlib.md5(clave.encode())
    return int(h.hexdigest(),base=16)
```

Listado 1: Funciones de hash h1 y h2.

3. Actividades a realizar

3.1. Implementación del Cuco Hashing

Usted debe implementar en `Python3` la Tabla de Hash que usa el método de Cuco hashing, descrita en la sección anterior. Además de la Tabla de Hash, debe hacer un cliente que muestre el correcto funcionamiento de las operaciones que haya implementado. Los archivos que debe realizar son los siguientes:

`cuco_entry.py`: Contiene la implementación de la clase `CucoEntry`.

`cuco_tabla.py`: En este archivo contiene la implementación de la Tabla de Hash por medio de la clase `CucoTabla`.

`test_cucoTabla.py`: Cliente con el que se prueba y se muestra el correcto funcionamiento de las operaciones de la Tabla de Hash.

3.2. Evaluación de las Tablas de Hash

Se desea hacer una comparación experimental del rendimiento de la Tabla de Hash basada en Cuco Hashing y la Tabla Hash basada en encadenamiento, implementada en la

[1] `https://en.wikipedia.org/wiki/Secure_Hash_Algorithm`
[2] `https://en.wikipedia.org/wiki/MD5`

semana 9. El objetivo es comparar el rendimiento de ambas tablas, bajo un conjunto de operaciones, y usando diferente cantidad de datos.

A la Tabla de Hash basada en el método de encadenamiento debe realizársele las siguientes modificaciones.

- La clase `HashEntry` debe ser modificada para que el atributo de la clave sea del tipo *String*.
- La Tabla de Hash realiza automáticamente la operación de `rehashing`, en el caso de que si al agregar un nuevo elemento a la tabla, la misma posea un *factor de carga* igual o mayor a 0.7.
- En la operación de `rehashing`, el tamaño de la tabla aumenta en un factor de 3/2.
- La función de hash a utilizar es SHA256.

Una vez efectuadas las modificaciones a la Tabla Hash basada en encadenamiento, entonces es posible hacer la comparación del rendimiento de las dos tablas. Ambas tablas deben procesar el mismo arreglo generado.

Debe realizar un programa cliente `testing_hash_tabla.py`, el cual compara las dos tablas de hash mediante la siguiente prueba:

1. Se crea un arreglo, el cual va a contener n números enteros generados aleatoriamente, de los cuales solo el 20 % son distintos.
2. El arreglo de enteros se convierte en arreglo de *Strings*.
3. El arreglo de *Strings* contienen la clave y el valor, que son iguales, que se van a insertar en la Tabla de Hash.
4. Se crea una Tabla de Hash con un tamaño inicial de 11.
5. Para cada uno de los elementos del arreglo, se consulta si el elemento existe en la Tabla de Hash, entonces si existe el elemento se elimina, de lo contrario se agrega. La idea es usar las operaciones de `buscar`, `agregar` e `eliminar` de las tablas de hash.
6. Se debe medir el tiempo usado por la Tabla de Hash para procesar todos los elementos del arreglo de *Strings*. No se debe tomar en cuenta el tiempo usado para la creación del arreglo de *Strings*.
7. Se debe medir la memoria usada por la Tabla de Hash.

El procedimiento anterior debe ser aplicado a cada Tabla de Hash. El programa `testing_hash_tabla.py` debe ser ejecutado con la siguiente línea de comando:

```
>./testing_hash_tabla.py <n>
```

Donde n es el número de elementos que contiene el arreglo de tipo *String*, que va a ser procesado por las tablas de hash. Como resultado el programa debe mostrar por la salida estándar, el tiempo de ejecución y la memoria usada por cada tabla.

Se debe presentar un informe con estudio sobre el rendimiento de las tablas de hash. Se debe ejecutar el programa `testing_hash_tabla.py` con los siguientes valores, 250000, 500000, 1000000, 2000000 y 4000000. El informe debe contener dos tablas y por cada tabla

una gráfica. La primera tabla contiene un los tiempos de las dos tablas de hash para cada uno de los valores indicados anteriormente. Debe realizar entonces un gráfica tiempo versus tamaño del arreglo. La segunda tabla muestra la memoria usada por cada Tabla de Hash, para cada tamaño de arreglo. La gráfica asociada a esta tabla refleja la memoria usada por cada tamaño de arreglo. Las unidades de memoria y tiempo debe ser indicadas en la tabla y las gráficas. Puede usar el programa que desee para realizar las gráficas, incluyendo `graficar_puntos.py`. Finalmente, para cada tabla debe hacer un análisis de los resultados. El informe debe estar en formato PDF.

4. Condiciones de entrega

La versión final del código del laboratorio y el informe, deben estar contenidos en un archivo comprimido, con formato *tar.xz*, llamado *LabSem10_X.tar.xz*, donde X es el número de carné del estudiante. La entrega del archivo *LabSem10_X.tar.xz*, debe hacerse al profesor del laboratorio por email, antes de las 9:00 pm del día viernes 27 de marzo de 2020.

Referencias

[1] Pagh, R., and Rodler, F. F. Cuckoo hashing. *Journal of Algorithms 51*, 2 (2004), 122–144.

[2] Palma, G. Ci2612: Algoritmos y estructuras de datos ii. `https://gpalma.github.io/courses/ci2612em20/`, 2020.

Guillermo Palma / gvpalma@usb.ve / Marzo 2020