75.41 Algortimos y Programación II Curso 4
TDA Hash

15 de junio de 2019

1. Enunciado

Se pide implementar un Hash. Para ello se brindan las firmas de las funciones públicas a implementar y se deja a criterio del alumno la creación de las funciones privadas del TDA para el correcto funcionamiento del Hash cumpliendo con las buenas prácticas de programación.

2. hash.h

```c
#ifndef __HASH_H__
#define __HASH_H__

#include <stdlib.h>

typedef struct hash hash_t;
typedef struct hash_iter hash_iter_t;
typedef void (*destruir_dato_t)(void*);

/*
* Creará el hash, reservando la memoria necesaria para el.
* Devolverá el hash creado o NULL en caso de no poder crearlo.
*/
hash_t* crear_hash(hash_destruir_dato_t destruir_elemento);

/*
* Guardara un elemento reservando la memoria necesaria para este.
* Devolverá 0 si pudo guardarlo o -1 si no pudo.
* La función de hasheo queda a criterio del alumno.
*/
int guardar_elemento(hash_t* hash, const char* clave, void* elemento);

/*
* Eliminará un elemento del hash.
* Devolverá 0 si pudo eliminar o -1 si no pudo.
*/
int quitar_elemento(hash_t* hash, const char* clave);

/*
* Devolverá un elemento del hash con la clave pasada por parámetro o NULL si no existe.
*/
void* obtener_elemento(const hash_t* hash, const char* clave);

/*
* Devolverla true si existe una clave dentro del hash o false si no existe.
*/
bool existe_clave(const hash_t* hash, const char* clave);

/*
* Devolverá la cantidad de elementos almacenados en el hash.
*/
size_t cantidad(const hash_t* hash);

/*
* Destruirá el hash liberando la memoria reservada por este.
* Devolverá 0 si pudo destruirlo o -1 si no pudo.
*/
int destruir_hash(hash_t* hash);
```

```c

/* Iterador externo para el HASH */
/*
* Creará el hash reservando la memoria necesaria para el mismo.
* Devolverá NULL en caso de no poder crearlo o el iterador si pudo.
*/
hash_iterador_t* crear_iterador(const hash_t* hash);

/*
* Avanza un elemento en el hash.
* Devolverá true si pudo avanzar o false si no pudo.
*/
bool avanzar_iterador(hash_iter_t* iterador);

/*
* Devolverá el elemento actual en el que esta parado el iterador.
*/
const char* elemento_actual(const hash_iter_t* iterador);

/*
* Devolverá true si el iterador llegó al final del hash o false en otro caso.
*/
bool esta_al_final(const hash_iter_t* iterador);

/*
* Destruirá el iterador del hash.
* Devolverá 0 si pudo destruirlo o -1 si no pudo.
*/
int hash_iterador_destruir(hash_iter_t* iterador);

#endif /* __HASH_H__ */
```

3. Compilación y Ejecución

El TDA entregado deberá compilar y pasar las pruebas dispuestas por la cátedra sin errores, adicionalmente estas pruebas deberán ser ejecutadas sin pérdida de memoria.

Compilación:

```
gcc *.c -o hash -g -std=c99 -Wall -Wconversion -Wtype-limits -pedantic -Werror -O0
```

Ejecución:

```
valgrind --leak-check=full --track-origins=yes --show-reachable=yes ./hash
```

4. Minipruebas

Se les brindará un lote de minipruebas, las cuales recomendamos fuertemente sean ampliadas ya que no son exhaustivas y no prueban todos los casos borde, solo son un ejemplo de como agregar, eliminar, obtener y buscar elementos dentro del árbol y qué debería verse en la terminal en el **caso feliz**.

Minipruebas:

La salida por pantalla luego de correrlas con valgrind debería ser:

5. Entrega

La entrega deberá contar con todos los archivos necesarios para compilar y ejecutar correctamente el TDA.

Dichos archivos deberán formar parte de un único archivo **.zip** el cual será entregado a través de la plataforma de corrección automática **Kwyjibo**.

El archivo comprimido deberá contar, además del TDA con:

- El archivo con las pruebas agregadas para comprobar el correcto funcionamiento del TDA.
- Un **Readme.txt** donde se deberá explicar qué es lo entregado, como compilarlo (línea de compilación), como ejecutarlo (línea de ejecución) y todo lo que crea necesario aclarar.
- El enunciado.