75.41 Algoritmos y Programación II Curso 4
TDA Hash

1 de julio de 2021

1. Enunciado

Se pide implementar una **Tabla de Hash cerrada** (direccionamiento abierto) en C. 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 de la **Tabla de Hash** cumpliendo con las buenas prácticas de programación. La función de **Hash** a utilizar será interna de la implementación y también debe ser decidida por el alumno. Para esta implementación las claves admitidas por la tabla serán solamente strings. Adicionalmente se pide la creación de un iterador interno que sea capaz de recorrer las claves almacenadas en la tabla.

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.

A **modo de ejemplo**, se brindará al alumno un archivo simple de "minidemo". **Este archivo no es mas que un programa mínimo de ejemplo de utilización del TDA a implementar** y es provisto sólo a fines ilustrativos como una ayuda extra para entender el funcionamiento del mismo. No es necesario modificar ni entregar el archivo de minidemo, pero si la implementación es correcta, debería correr con valgrind sin errores de memoria.

Para la resolución de este trabajo se recomienda utilizar una **metodología orientada a pruebas**. A tal fín, se incluye un archivo **pruebas.c** que debe ser completado con las pruebas pertinentes de cada una de las diferentes primitivas del TDA. **El archivo de pruebas forma parte de la entrega y por lo tanto de la nota final**. Aún mas importante, las pruebas van a resultar fundamentales para lograr no sólamente una implementación correcta, si no también una experiencia de desarrollo menos turbulenta.

2. Consideraciones para la implementación

- Recuerde que en un hash cerrado las colisiones no se encadenan, si no que se almacenan dentro de la tabla utilizando algún criterio de resolución de colisiones. **IMPORTANTE: dos claves identicas no colisionan.**
- Al eliminar elementos de la tabla, debe tener en cuenta el efecto que puede causar esta operación en la búsqueda y futuras inserciones en la tabla.
- A medida que la tabla se va llenando, la cantidad de colisiones aumenta. Esto produce una degradación en el rendimiento de las operaciones sobre la tabla. Para evitar que esto suceda, debe decidir un criterio para decidir en qué momento se debe producir una reorganización y redimensionamiento de la tabla.

3. Consejos para la elaboración del trabajo

Intente comprender primero el funcionamiento de la tabla y del mecanismo de resolución de colisiones. Utilice lápiz y papel para dibujar algunas tablas y realice varias operaciones de inserción, eliminación y búsqueda. Asegúrese de entender bien cómo se relacionan las operaciones y cómo los datos se almacenan en la tabla antes de comenzar la implementación.

Y como siempre, las recomendaciones generales que no pueden faltar: recuerde que al escribir pruebas **no se busca en el código de pruebas la encapsulación ni simplificación de las mismas**. No es incorrecto tener pruebas con código repetitivo. Las pruebas son una **especificación** del comportamiento deseado de las primitivas. Como tal, deben ser fáciles de leer y entender su objetivo.

En general, para todo el código: utilice nombres claros de variables y funciones auxiliares. Una variable con el nombre **cantidad_elementos_recorridos** o incluso **cantidad** es mucho mas claro que una variable con el nombre **n**, **cant**, o **rec**. Intente darle un significado el nombre de cada variable, dentro de lo posible (por supuesto, quedan excluidos casos como i, j, etc para bucles y cosas así).

NO escriba código a lo loco sin compilar cada tanto. Implemente la solución de a poco y compilando a cada paso. Dejar la compilación para el final es uno de los peores errores que puede cometer. Para la compilación del trabajo se provee un **Makefile**. Utilice el comando **make** frecuentemente para compilar y correr su programa.

NO avance en la implementación si le quedan errores sin resolver en alguna prueba. Cada vez que escriba una prueba implemente toda la funcionalidad necesaria para que funcione correctamente. Esto incluye liberar memoria y accesos inválidos a la misma. Sólamente una vez que haya logrado que la prueba pase exitosamente es que puede comenzar a escribir la próxima prueba para continuar el trabajo.

NO está permitido modificar los archivos **.h**. Puede hacer modificaciones al **makefile** si lo desea, pero recuerde que **el trabajo será compilado por el sistema de entregas con el makefile original**.

4. Entrega

La entrega deberá contar con todos los archivos que se adjuntan con este enunciado (todo lo necesario para compilar y correr correctamente). Dichos archivos deberán formar parte de un único archivo **.zip** el cual será entregado a través de la plataforma de corrección automática **Chanutron2021**.

El archivo comprimido deberá contar, además de los archivos del TDA con:

- Un **Readme.txt** donde se explique la definición de tabla de hash. Enumere los distintos tipos de tablas que conoce y explique cómo se diferencian.
- El **Readme.txt** deberá contener también una explicación de la solución implementada. Este archivo le sirve a su corrector para saber si entendió o no el tema. Explique la implementación como si se lo estuviese explicando a alguien que no sabe del tema. Explique las decisiones que tomó al implementar las primitivas (esto incluye estrategia de resolución de colisiones, criterio de redimensionamiento, etc). Si hizo extra alguna suposición para resolver el TP (ya sea por enunciado poco claro o cualquier otro motivo) también explíquela.
- Este enunciado.