INF-134 Estructuras de datos 2024-1

Tarea 3: Registro de cuentas

Profesores

Elizabeth Montero
elizabeth.montero@usm.cl

Andrés Navarro
andres.navarrog@usm.cl

Roberto Díaz
roberto.diazu@usm.cl

José Miguel Cazorla
jcazorla@usm.cl

Ricardo Salas
ricardo.salas@usm.cl

Ayudantes Casa Central

Sebastián Torrealba
sebastian.torrealba@usm.cl

Tomás Barros
tomas.barros@sansano.usm.cl

Bastián Jimenez
bjimenez@usm.cl

Franco Cerda
franco.cerda@usm.cl

Ayudantes San Joaquín

Carlos Lagos
carlos.lagosc@usm.cl

Gabriel Escalona
gabriel.escalona@alumnos.inf.utfsm.cl

Juan Alegria
juan.alegria@usm.cl

Lucas Morrison
lucas.morrison@sansano.usm.cl

Giuliana Zanetti
giuliana.zanetti@sansano.usm.cl

Reglas

- La presente tarea debe hacerse en *equipos de dos personas*. Toda excepción a esta regla está sujeta a autorización del ayudante *coordinador* Sebastián Torrealba.
- Debe usarse el lenguaje de programación **C++**. Al realizar la evaluación, las tareas serán compiladas usando el compilador **g++**, usando la línea de comando `g++ archivo.cpp -o output -Wall`. No se aceptarán variantes o implementaciones particulares de **g++**, como el usado por **MinGW** (normalmente ocupado en **Dev C++**). Se deben seguir los tutoriales disponibles en Aula USM.
- No se permite usar la biblioteca STL, así como ninguno de los contenedores y algoritmos definidos en ella (e.g. `vector`, `list`, etc.). Está permitido usar otras funciones de utilidad definidas en bibliotecas estándar, como por ejemplo `math.h`, `string`, `fstream`, `iostream`, etc.
- Recordar que una única tarea en el semestre puede tener nota menor a 30. El incumplimiento de esta regla implica reprobación del curso.

Objetivos de aprendizaje

Entender el funcionamiento del hashing

- Aprender como crear funciones de hashing eficientes
- Aprender como gestionar el manejo de colisiones dentro de una tabla hash
- Aprender a redimensionar una tabla hash

Problema a resolver: Hashing el registro de estudiantes

Contexto y problema a resolver

La Universidad Técnica Federico Santa María (UTFSM) está desarrollando un nuevo sistema de registro de estudiantes que permita gestionar su información de manera eficiente y segura. Así entonces, ha solicitado a los estudiantes de Estructuras de Datos realizar una primera versión de la implementación del sistema. El objetivo entonces es crear una tabla hash que almacene el rol, nombre y una breve descripción de cada estudiante. La tabla hash deberá gestionar colisiones mediante hashing cerrado. Este sistema debe permitir la inserción, búsqueda, modificación y eliminación de registros de estudiantes, garantizando un manejo eficiente de colisiones. Además, deberá realizar redimensionamiento dinámico de la tabla cuando sea necesario.

Para probar si la tabla hash funciona, se usarán distintos casos de prueba. A continuación, se presenta la interfaz de entrada y salida:

- `AGREGAR rol nombre descripcion`: Se agrega una cuenta de usuario con el respectivo rol, nombre y descripción. En caso de que exista, escribir, `Rol ya existente`
- `QUITAR rol`: Se quita la cuenta que tenga ese `rol`, en caso de que no exista debe decir, `Rol no existente`
- `MODIFICAR rol descripcion`: Se modifica la descripción del rol correspondiente. En caso de que no exista el rol, debe mostrar `Rol no existente`
- `OBTENER rol`: Muestra el nombre y la descripción dado un rol. En caso de que no exista se debe mostrar `Rol no existente`.
- `ESTADISTICAS`: Mostrar `RANURAS OCUPADAS`, `RANURAS TOTALES` y el `FACTOR DE CARGA`

IMPORTANTE: Se asegura que los caracteres que se probarán serán los del alfabeto inglés, con guión y guiones bajos.

Ejemplo de entrada y salida

Entrada

```
AGREGAR 99999999-9 Pablo Un_estudiante_de_historia
AGREGAR 00000000-0 Laura Una_estudiante_de_arte
AGREGAR 99999999-9 Roberto Un_estudiante_de_diseño
OBTENER 99999999-9
AGREGAR 12341234-2 Ana Un_estudiante_de_matematicas
AGREGAR 12341234-2 Lucia Una_estudiante_de_biologia
OBTENER 12341234-2
ESTADISTICAS
QUITAR 00000000-0
ESTADISTICAS
```

Salida

```
Rol ya existente
Pablo Un_estudiante_de_historia
Rol ya existente
Ana Un_estudiante_de_matematicas
RANURAS OCUPADAS: 3
RANURAS TOTALES: 15
FACTOR DE CARGA: 0.2
RANURAS OCUPADAS: 2
RANURAS TOTALES: 15
FACTOR DE CARGA: 0.1333
```

Caso de prueba de ejemplo Nº2

Entrada

```
AGREGAR 10101010-1 Alicia Una_estudiante_de_musica
AGREGAR 11111111-1 Sofia Una_estudiante_de_matematicas
AGREGAR 12121212-2 Bastian Un_estudiante_de_fisica
ESTADISTICAS
QUITAR 11111111-1
ESTADISTICAS
QUITAR 10101010-1
ESTADISTICAS
```

Salida

```
RANURAS OCUPADAS: 3
RANURAS TOTALES: 15
FACTOR DE CARGA: 0.2
RANURAS OCUPADAS: 2
RANURAS TOTALES: 15
FACTOR DE CARGA: 0.1333
RANURAS OCUPADAS: 1
RANURAS TOTALES: 15
FACTOR DE CARGA: 0.0667
```

Caso de ejemplo Nº3

Entrada

```
AGREGAR 13131313-3 Carla Una_estudiante_de_literatura
AGREGAR 14141414-4 Diego Un_estudiante_de_filosofia
AGREGAR 15151515-5 Eduardo Un_estudiante_de_ingenieria
ESTADISTICAS
MODIFICAR 13131313-3 Una_estudiante_de_sociologia
ESTADISTICAS
```

Salida

```
RANURAS OCUPADAS: 3
RANURAS TOTALES: 15
FACTOR DE CARGA: 0.2
RANURAS OCUPADAS: 3
RANURAS TOTALES: 15
FACTOR DE CARGA: 0.2
```

Caso de ejemplo Nº4

Entrada

```
AGREGAR 16161616-6 Fernanda Una_estudiante_de_arquitectura
AGREGAR 17171717-7 Gustavo Un_estudiante_de_medicina
AGREGAR 18181818-8 Helena Una_estudiante_de_psicologia
AGREGAR 19191919-9 Ignacio Un_estudiante_de_periodismo
ESTADISTICAS
QUITAR 17171717-7
ESTADISTICAS
```

Salida

```
RANURAS OCUPADAS: 4
RANURAS TOTALES: 15
FACTOR DE CARGA: 0.2667
RANURAS OCUPADAS: 3
RANURAS TOTALES: 15
FACTOR DE CARGA: 0.2
```

Caso de ejemplo Nº5

Entrada

```
AGREGAR 20202020-2 Javier Un_estudiante_de_derecho
AGREGAR 21212121-1 Karen Una_estudiante_de_biologia
AGREGAR 22222222-2 Luis Un_estudiante_de_fisica
AGREGAR 23232323-3 Monica Una_estudiante_de_linguistica
AGREGAR 24242424-4 Nicolas Un_estudiante_de_ingenieria
ESTADISTICAS
QUITAR 23232323-3
QUITAR 20202020-2
ESTADISTICAS
```

Salida

```
RANURAS OCUPADAS: 5
RANURAS TOTALES: 15
FACTOR DE CARGA: 0.3333
RANURAS OCUPADAS: 3
RANURAS TOTALES: 15
FACTOR DE CARGA: 0.2
```

Requisitos al resolver el problema

Se debe usar la siguiente plantilla como **registro_cuentas** y **cuenta**:

```
struct cuenta {
  // El rol es el identificador de la persona.
  // El nombre y la descripcion son valores asociados al rol
  string rol, nombre, descripcion;
};

class registro_cuentas {
  private:
    float factor_de_carga = 0.0;
    cuenta tabla*; // Aca se almacenaran los elementos de la tabla
    int ranuras = 15; // Cuantas ranuras tiene la tabla hash (inicialmente)
    int hash(string rol); // Se obtiene el hash dado el rol
    int p(string rol, int i); // Se otiene la ranura a revisar en caso de colisión
dado el rol y el intento i
  public:
    registro_cuentas() {} // (Recuerde que puede crear con distintos parametros)
    cuenta obtener(string rol); // Dado el rol, devuelve la cuenta con ese rol
    void agregar(cuenta c); // Se agrega una cuenta a la tabla
    void eliminar(string rol); // Se elimina la cuenta
    void modificar(string rol, string descripcion); // Se modifica la descripcion del
rol
    void redimensionar(int n); // Se redimensiona la tabla a n espacios
    void estadisticas(); // Debe mostrar las estadisticas
};
```

Tener en cuenta que se pueden crear nuevos métodos, no así, modificar lo que ya está preestablecido. En caso de una modificación, consultar al ayudante coordinador en los medios de comunicación establecidos.

IMPORTANTE: En caso de que la tabla hash se quede sin espacios libres, se debe redimensionar. La función debe iniciar con `15` espacios, si se cumplen los requisitos para aumentar los espacios, se debe aumentar esta cantidad. La función hash, la función de gestión de colisiones y la técnica de redimensionamiento de la tabla la define cada equipo. Incluya una clara descripción de cada uno de los procesos (función hash, función de gestión de colisiones y técnica de redimensionamiento de la tabla) en los comentarios de su código.

Entrega de la tarea

Entregue la tarea enviando un archivo comprimido zip llamado **tarea1-apellido1-apellido2.zip** (reemplazando sus apellidos según corresponda), en cuyo interior debe estar la tarea dentro de una carpeta también llamada **tarea1-apellido1-apellido2**, a la página *aula.usm.cl* del curso, el cual contenga:

- El **apellido1** es el primer apellido de alguno de los integrantes y el **apellido2** es el primer apellido del otro integrante. Esto significa que el **apellido1** y **apellido2** no son los dos apellidos de alguno de los integrantes.
- Solo **una** persona del grupo debe subir la tarea al aula, no hacer esto, puede conllevar un descuento.
- Los archivos con los códigos fuentes necesarios para el funcionamiento de la tarea. ¡Los archivos deben compilar!
- **nombres.txt**, que debe indicar Nombre, ROL, Paralelo y detalle de qué programó cada integrante del equipo.
- **README.txt**, que debe indicar las instrucciones de compilación en caso de ser necesarias, y la forma de compilación que usó (debe ser alguna de las indicadas en los tutoriales entregados en Aula USM).

Entrega mínima

La entrega mínima permite obtener una nota 30 si cumple con los siguientes requisitos:

- Se es capaz de insertar un valor en la tabla y mostrarlo correctamente

Restricciones y consideraciones

- Se permite un único día de atraso en la entrega de la tarea. La nota máxima que se puede obtener en caso de entregar con un día de atraso es nota 50.
- Las tareas que no compilen no serán revisadas y tendrán que ser re-corregidas con el ayudante coordinador.
- Debe usar obligatoriamente alguna de las formas de compilación indicadas en los tutoriales entregados en Aula USM.
- Por cada Warning en la compilación se descontarán 5 puntos de la nota.
- Si se detecta COPIA la situación será revisada por ayudante y profesor coordinador.
- La prolijidad, orden y legibilidad del código fuente es obligatoria. Se aplicarán descuentos en la nota si alguno de estos ítems no se cumple.

Directrices de programación

Las directrices corresponden a buenas prácticas de programación, las cuales serán tomadas en consideración para la evaluación de la tarea. El código fuente del programa debe estar estructurado adecuadamente en archivos (separados de ser necesario). Si el código fuente está desordenado, se pueden descontar hasta 20 puntos de la nota. Cada función programada debe tener comentarios de la siguiente forma:

```
/*****
* TipoFunción NombreFunción
******
* Resumen Función
******
* Input:
* tipoParámetro NombreParámetro : Descripción Parámetro
* .......
******
* Returns:
* TipoRetorno, Descripción retorno
*****/
```

- Por cada comentario faltante, se restarán 5 puntos.
- La indentación (1 TAB o 4 espacios), es muy importante. Por cada bloque mal indentado, se quitarán 10 puntos.