INF-134 Estructuras de Datos, 2021-1

Tarea 3

Profesores: Diego Arroyuelo B., Roberto Díaz U.
darroyue@inf.utfsm.cl, roberto.diazu@usm.cl

Ayudantes:
Diego Beltrán (diego.beltran@sansano.usm.cl)
Martin Crisóstomo (martin.crisostomo@sansano.usm.cl)
Joaquín Gatica (joaquin.gatica@sansano.usm.cl)
Diana Gil (diana.gil@sansano.usm.cl)
Martín Salinas (martin.salinass@sansano.usm.cl)
Sebastián Sepúlveda (sebastian.sepulvedab@sansano.usm.cl)
Daniel Tapia (daniel.tapiara@sansano.usm.cl)

Fecha de entrega: 05 de agosto de 2021
Plazo máximo de entrega: 5 horas.

1. Reglas del Juego

La presente tarea debe hacerse en grupos de 3 personas. Toda excepción a esta regla debe ser conversada con los ayudantes ANTES de comenzar la tarea. No se permiten de ninguna manera grupos de más de 3 personas. Las tareas deben compilar en los computadores que se encuentran en el laboratorio LDS – B-032. Deben usarse los lenguajes de programación C o C++. Se recomienda compilar en el terminal usando `gcc archivo.c -o output -Wall` (en el caso de lenguaje C) o `g++ archivo.cpp -o output -Wall` (en el caso de C++).

2. Objetivos

Entender y familiarizarse con estructuras de datos de tipo hashing.

3. Boleta de Supermercado

Una cadena de supermercados necesita un sistema que permita generar boletas de venta para las compras de sus clientes, teniendo en cuenta que varios de los productos en venta están en oferta. Suponga que un producto en venta está definido por el siguiente tipo en C++:

```
struct producto {
   int  cod_producto;
   char nombre_producto[31];
   int  precio;
};
```

Dado que la cantidad de productos a manejar es grande, se decide usar un hashing cerrado para almacenarlos. El campo `cod_producto` será la clave de búsqueda en esta tabla.

Además, el supermercado tiene algunos productos en oferta, en las que se aplican descuentos si es que se compra una cierta cantidad de algunos productos. Dicha cantidad para aplicar descuentos varía de producto a producto, por lo que se decide mantener otro hashing cerrado que almacena elementos del tipo:

```
struct oferta {
   int cod_producto;
   int cantidad_descuento;
   int descuento;
   int productos_equivalentes[10];
};
```

El campo `cod_producto` es el mismo que en la tabla anterior. El campo `cantidad_descuento` indica la cantidad exacta de unidades (del producto en cuestión) que deben comprarse para aplicar el descuento. Finalmente, el campo `descuento` indica la cantidad de pesos a descontar (si aplica el descuento). Es importante destacar que las ofertas sólo se aplican por cantidades exactas. Por ejemplo, si una oferta de un cierto producto aplica si se compran 3 unidades del producto, y un cliente compra 7 unidades, se aplica el descuento a 2 tríos del producto, y se cobra la unidad restante a precio normal.

El último campo `productos_equivalentes` es un arreglo que contiene los códigos de aquellos productos que se consideran equivalentes al producto original para la oferta. Es decir, aportan a la cantidad de producto para la aplicación de la oferta. Aquellos productos con menos de 10 productos equivalentes contendrán enteros con un valor igual a `-1` hacia el final del arreglo. Notar que si un producto a está en el arreglo de productos equivalentes de b, entonces b también estará en el arreglo de productos equivalentes de a. Además, las ofertas serán iguales en `cantidad_descuento` y `descuento` para cualquier par de productos equivalentes.

Su programa deberá leer un conjunto de compras realizadas por clientes, aplicar los descuentos correspondientes en cada caso, y mostrar los montos a pagar.

4. Entrada de Datos

La entrada de datos será a través de los archivos `productos.dat` y `ofertas.dat`. El primero es un archivo binario que contiene structs de tipo `producto`, mientras que el segundo contiene structs del tipo `oferta`. Ambos archivos tienen como primer elemento un número entero que indica la cantidad de structs que contiene el archivo (recuerde que ambos archivos son binarios). Las dos estructuras de hashing cerrado (mencionadas anteriormente) deben construirse usando estos datos. El factor de carga a emplear debe ser $\alpha = 0{,}7$ en ambos casos. La política de resolución de colisiones a emplear debe ser la de doble hashing en ambos casos.

Luego de construir estas dos estructuras de datos, se deben leer datos de compras desde el archivo `compras.txt`. Éste es un archivo con formato ASCII que tiene la siguiente forma: la primera línea contiene un único entero `N` ($1 \leq$ `N` $\leq 100{,}000$), indicando la cantidad de clientes a atender. A continuación, por cada cliente se tienen los siguientes datos: una línea que contiene un único entero `C` ($1 \leq$ `C` $\leq 1{,}000{,}000$), indicando la cantidad de compras realizadas por cada cliente. Luego, le siguen `C` líneas, cada una de ellas conteniendo un único entero que indica el código de producto comprado. Los productos están en orden arbitrario dentro de este listado (después de todo, eso es lo que hacemos en una caja de supermercado). Además, pueden haber productos repetidos en cada compra.

Un ejemplo de entrada válida es:

```
3
4
12
4
8
4
```

```
5
12
12
18
12
18
2
5
4
```

el cual representa las compras realizadas por 3 clientes. El primer cliente compró 4 productos (con códigos 12, 4, 8 y 4), el segundo compró 5 productos (con códigos 12, 12, 18, 12 y 18), y finalmente el tercer cliente compró 2 productos (con códigos 5 y 4).

Tenga en cuenta que puede haber compras muy grandes (de hasta 1 millón de productos). Es responsabilidad suya manejar de forma eficiente cada compra. Use las estructuras de datos que crea adecuadas para esto.

5. Salida de Datos

La salida de datos se realizará a través del archivo `boletas.txt`. La primera línea del mismo debe contener un único entero `N`, que indica la cantidad de boletas generadas (es el mismo valor `N` del archivo `compras.txt`, explicado anteriormente). Luego deben seguirle `N` líneas, cada una conteniendo un único número entero `T`, que es el total a pagar por el cliente por los productos que está comprando. Este total debe incluir todos los descuentos que corresponda aplicar. El orden en que se muestran los totales debe coincidir con el de los clientes del archivo `compras.txt`.

6. Entrega de la Tarea

Entregue la tarea enviando un archivo comprimido `tarea2-apellido1-apellido2-apellido3.zip` o `tarea2-apellido1-apellido2-apellido3.tar.gz` (reemplazando sus apellidos según corresponda) a la página aula.usm del curso, a más tardar el día 05 de agosto de 2021, a las 23:59:00 hs (Chile Continental), el cual contenga:

- Los archivos con los códigos fuentes necesarios para el funcionamiento de la tarea. ¡Los archivos deben compilar!
- **nombres.txt**, Nombre, ROL, Paralelo y qué programó cada integrante del grupo.
- **README.txt**, 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).

7. Restricciones y Consideraciones

- Por **cada hora de atraso** en la entrega de la tarea **se descontarán 10 puntos** en la nota.
- El plazo máximo de entrega es de **5 horas** después de la fecha original de entrega.
- **Las tareas que no compilen no serán revisadas y serán calificadas con nota 0**.
- Debe usar **obligatoriamente** alguna de las formas de compilación indicadas en los tutoriales entregados en Aula USM.
- Solo se aceptarán dudas de enunciado y solo vía Aula USM.

- Por cada *Warning* en la compilación se descontarán 5 puntos.
- Si se detecta **COPIA** la nota automáticamente sera 0 (CERO), para todos los grupos involucrados. El incidente será reportado al jefe de carrera.
- La prolijidad, orden y legibilidad del código fuente es obligatoria. Habrá descuentos si alguno de estos items no se cumple.

8. Consejos de Programación

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.

Por último, la indentación (1 TAB o 4 espacios), es muy importante. Por **cada bloque mal indentado, se quitarán 10 puntos**.