INF-134 Estructuras de Datos, 2020-1

Tarea 3

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

Ayudantes:
Anastasiia Fedorova anastasiia.fedorova@sansano.usm.cl,
Gonzalo Fernandez gonzalo.fernandezc@sansano.usm.cl,
Tomás Guttman tomas.guttman@sansano.usm.cl,
Héctor Larrañaga hector.larranaga@sansano.usm.cl,
Martín Salinas martin.salinass@sansano.usm.cl.

Fecha de inicio: 21 de julio, 2020
Fecha de entrega: 07 de agosto, 2020
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. Debe usarse el lenguaje de programación C. Alternativamente, se puede usar C++, aunque en ese caso no se pueden usar las librerías `stl`. En cualquiera de los dos casos, las estructuras de datos usadas en esta tarea deben ser implementadas por usted mismo. Al evaluarlas, las tareas serán compiladas usando el compilador `gcc` (o `g++` en el caso de C++), usando la línea de comando `gcc archivo.c -o output -Wall`. Alternativamente, se aceptan variantes o implementaciones particulares de `gcc`, como el usado por MinGW (que está asociado a la IDE code::blocks). Se deben seguir los tutoriales que están en Aula USM, cualquier alternativa explicada allí es válida. Recordar que una única tarea en el semestre puede tener nota menor a 30. El no cumplimiento de esta regla implica reprobar el curso.

2. Objetivos

Entender y familiarizarse con estructuras de datos de tipo heap y hashing.

3. Boleta de Supermercado

Una cadena de supermercados necesita un sistema que permita analizar las compras de sus clientes para decidir sus políticas y estrategias a futuro. Se debe tener 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:

```
typedef struct {
   int  codigo_producto;
   char nombre_producto[31];
   int  precio;
```

```
} producto;
```

Dado que la cantidad de productos a manejar es grande, se decide usar un hashing cerrado para almacenarlos. El campo `codigo_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:

```
typedef struct {
   int codigo_producto;
   int cantidad_descuento;
   int monto_descuento;
} oferta;
```

El campo `codigo_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 `monto_descuento` indica la cantidad de pesos a descontar en total (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.

Su programa deberá:

- Leer un conjunto de compras realizadas por clientes
- Mantener un registro de las ventas de cada producto y el monto total obtenido producto de estas.
- Generar un ranking de los productos con mayor monto total obtenido. Para la generación eficiente del ranking se debe hacer uso de una estructura de datos tipo heap.

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 de texto `compras.txt`. Éste es un archivo con formato ASCII que tiene la siguiente forma: la primera línea contiene un único entero `R` ($1 \leq$ `R` $\leq$ 1,000,000), indicando la cantidad de productos a incluir en el ranking, la segunda 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 productos que va a comprar el cliente. Luego, le siguen `C` líneas, cada una de ellas conteniendo un único entero que indica el código de un 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 del archivo `compras.txt` sería:

```
5
3
4
2020
```

```
1918
1981
2020
6
2020
2020
1346
2020
1981
1991
3
430
1981
430
```

En este caso se pide un ranking de los 5 productos con mayor monto obtenido a partir de las compras realizadas por 3 clientes. El primer cliente ha comprado 4 productos (con códigos 2020, 1918, 1981, y 2020), el segundo ha comprado 6 productos (con códigos 2020, 2020, 1346, 2020, 1981 y 1991), y finalmente el tercer cliente ha comprado 3 productos (con códigos 430, 1981 y 430).

Tenga en cuenta que puede haber compras muy grandes (de hasta 1 millón de productos). Es responsabilidad de cada grupo 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 de texto `ranking.txt` que tiene la siguiente estructura: `R` líneas, donde cada línea contiene información sobre cada uno de los productos con mayor monto obtenido por sus compras. Cada línea debe contener, separados por espacios, la siguiente información de cada producto: código, nombre, cantidad de unidades vendidas y monto total obtenido. El archivo debe estar ordenado de mayor a menor en base al monto total obtenido.

Un ejemplo del archivo `ranking.txt` sería:

```
2020 Papel Higienico 5 5170
430 Aceite 2 1758
1981 Arroz 3 1710
1346 Lata de Atun 1 1450
1918 Azucar 1 770
```

6. Entrega de la Tarea

La entrega de la tarea debe realizarse enviando un archivo comprimido llamado

`tarea3-apellido1-apellido2-apellido3.tar.gz`

(reemplazando sus apellidos según corresponda) en el sitio Aulas USM del curso, a más tardar el día 07 de agosto, 2020, 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.

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 **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 indicada en los tutoriales entregados en 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**.