INF-134 Estructuras de Datos, 2024-2

Tarea 2

Profesores: Andrés Navarro, Ricardo Salas, Juan-Pablo Castillo
andres.navarrog@usm.cl, ricardo.salas@usm.cl, juan.castillog@sansano.usm.cl

Ayudantes:
Gabriel Escalona (gabriel.escalona@usm.cl)
Tomás Barros (tomas.barros@sansano.usm.cl)
Jaime Inostroza (jinostroza@usm.cl)
Juan Alegría (juan.alegria@usm.cl)
Carlos Lagos (carlos.lagosc@usm.cl)
Damian Rojas (damian.rojas@usm.cl)

Fecha de entrega: 20 de Octubre, 2024.
Plazo máximo de entrega: 1 día.

1. Reglas

- La presente tarea debe hacerse en equipos de dos personas. Toda excepción a esta regla está sujeta a autorización del ayudante coordinador Gabriel Escalona.
- 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.

2. Objetivos de aprendizaje

Entender el funcionamiento de los árboles de búsqueda binaria.

- Entender el funcionamiento de listas enlazadas.
- Aprender cómo funcionan los recorridos de árboles de búsqueda binaria.
- Comprender el uso de punteros para definir estructuras de tamaño dinámico.
- Entender sobre los posibles usos de los árboles de búsqueda binaria.

Además se fomentará el uso de las buenas prácticas y el orden en la programación de los problemas correspondientes.

3. Problema a resolver

La correcta organización de la información permite mejorar los tiempos de respuestas en labores cruciales de algunos sistemas de información. En esta tarea trabajaremos con un sistema que gestiona la información de películas y directores.

La información a utilizar está almacenada de forma persistente en archivos de datos. El archivo ASCII `Peliculas.txt` contiene toda la información a incorporar en su sistema. El archivo de entrada posee en total `n+1` líneas donde el primer valor indica la cantidad de líneas y las siguientes `n` líneas poseen la información de una película como se muestra a continuación:

```
numero_peliculas
titulo1;director1;rating1
titulo2;director2;rating2
...
titulo_n;director_n;rating_n
```

Para cada película se presenta su título, director y rating separados por el carácter especial `;`. No hay películas con el mismo título, pero sí muy parecidas cuando se tienen sagas como "Rápido y furioso".

3.1. La Tarea

El contenido del archivo debe leerse y almacenarse utilizando estructuras de datos tipo árbol binario de búsqueda donde cada nodo corresponde a un director. El árbol debe ordenarse alfabéticamente de acuerdo al nombre de los directores. Cada director posee una lista enlazada de sus películas. Los listados de películas deben ordenarse por orden alfabético de acuerdo al título. Así, cada vez que se encuentra un nuevo director en el archivo debe crearse un nodo nuevo con su correspondiente lista nueva e insertarse en la posición debida del árbol. A su vez, cuando se encuentra un director repetido, debe buscarse en el árbol e insertar la película ordenadamente en el listado.

Una vez leído el archivo completo, i.e., listo el árbol de directores, se debe crear un segundo árbol ordenado según el promedio de ratings de las películas del director. Para esto, debe implementarse una copia del árbol recorriendo el árbol de directores en pre-orden.

Una vez creados los dos árboles, el árbol de directores y el árbol de ratings se deben implementar las siguientes acciones de usuario:

1. `sd director` : buscar `director` en el árbol de nombres y mostrar sus películas.

   ```
   sd director
   pelicula1 pelicula2 ...
   ```

2. `sm pelicula`: buscar `película` en el árbol de directores y mostrar información de esta en consola (título, director, rating)

   ```
   sm pelicula
   titulo / director / rating
   ```

3. `br n`: mostrar `n` directores con mejores ratings en el árbol de ratings. `n` no puede tomar valores mayores que la cantidad de directores.

   ```
   br 3
   (1) director1
   (2) director2
   (3) director3
   ```

4. `wr n`: mostrar `n` directores con peores ratings en el árbol de ratings. `n` no puede tomar valores mayores que la cantidad de directores. Considerando un árbol de `m` directores la salida sería como se muestra a continuación.

```
wr 3
(m) director1
(m-1) director2
(m-2) director3
```

5. `e`: terminar ejecución.

3.2. Implementación

Para la implementación se la tarea se deberán implementar las clases `Director` y `Arboles` como se muestra a continuación:

```
struct Pelicula{
  string nombre;
  string director;
  float rating;
}

class Director{
private:
  struct lNodo{
    Pelicula * val;
    lNodo *sig;
  };

  Director();  // constructor
  ~Director(); // destructor
  lNodo* head;
  lNodo* curr;
  lNodo* tail;
  size_t size; // longitud lista
  string nombre_director;
public:

  void agregar_pelicula(Peliculas* pelicula); // agrega pelicula al final de la lista
      enlazada
  void ordenar();          // ordena la lista
  void calcular_rating_promedio();
  void mostrar_peliculas();
}

class Arboles{
private:
  struct aNodo{
    Director * val;
    aNodo *izq;
    aNodo *der;
  };

  aNodo* root_1; // raiz arbol ordenado por directores
  aNodo* curr_1;
  size_t size_1;
  aNodo* root_2; // raiz arbol ordenado por rating
  aNodo* curr_2;
  size_t size_2;
public:

  Arboles();  // constructor
  ~Arboles(); // destructor
  void insertar_pelicula(Pelicula* pelicula);
  void copiar_arbol() // hace copia de arbol 1 en arbol 2 ordenado respecto de rating
  Director* buscar_director(string director); // retorna arreglo de peliculas
  Pelicula* buscar_pelicula(string pelicula);  // retorna peliculas
```

```
  void mejores_directores(int n); // Muestra por pantalla los mejores n directores.
      Enumerando de 1 a n.
  void peores_directores(int n);  // Muestra por pantalla los peores n directores.
      Enumerando desde m (cantidad de directores) hasta m-n.
}
```

Para ambas clases se pueden agregar atributos extra y crear métodos auxiliares si es que son necesarios. A los métodos que sí deben implementar no se les pueden agregar parámetros adicionales a los definidos.

También se pueden crear clases, estructuras y funciones auxiliares si es que son necesarias.

3.3. Ejemplos de entrada y salida

Ejemplo 1:

A continuación se presenta un ejemplo de entrada y salida de los datos. Primero se muestra el contenido del archivo "Peliculas.txt". Observe que los directores `Martin Scorsese` y `Quentin Tarantino` aparecen con dos películas en el archivo por lo que el tamaño de ambos árboles debe ser 5.

```
7
The Departed;Martin Scorsese;8.5
Kill Bill: Volume 1;Quentin Tarantino;8.2
Good Fellas;Martin Scorsese;8.7
Django Unchained;Quentin Tarantino;8.5
The Dark Knight;Christopher Noland;9.0
The Lord of the Rings: The Fellowship of the Ring;Peter Jackson;8.9
Arrival;Denis Villeneuve;7.9
```

Luego se muestra un ejemplo de ejecución, utilizando el archivo de ejemplo (>significa input en la terminal/consola).

```
> sd Quentin Tarantino
Django Unchained / 8.5
Kill Bill: Volume 1 / 8.2
> sm Arrival
Arrival / Denis Villeneuve / 8.0
> br 3
(1) Christopher Noland
(2) Peter Jackson
(3) Martin Scorsese
> wr 3
(5) Denis Villeneuve
(4) Quentin Tarantino
(3) Martin Scorsese
```

4. Entrega de la Tarea

Entregue la tarea enviando un archivo comprimido denominada `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:

- Los archivos con los códigos fuentes necesarios para el funcionamiento de la tarea. ¡Los archivos deben compilar!
- El archivo **nombres.txt** que debe contener Nombre, ROL y Paralelo de cada miembro del equipo detallando qué programó cada uno.
- **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).

Considere además que:

- El **apellido1** es el primer apellido de uno 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.

5. Entrega mínima

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

- Se leen los archivos en listas separadas por director

6. 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 deberán ser re-corregidas por 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 en 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. Habrá descuentos si alguno de estos ítems no se cumple.

7. 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.

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