INF-134 Estructuras de Datos, 2019-1

Tarea 2

Profesores: Diego Arroyuelo, Natalia González, Roberto Díaz
darroyue@inf.utfsm.cl, natalia.gonzalezg@usm.cl, robertodiazurra@gmail.com

Ayudantes:
Gabriel Carmona (gabriel.carmonat@sansano.usm.cl)
María Paz Morales (maria.moralesll@sansano.usm.cl)
Abdel Sandoval (abdel.sandoval@sansano.usm.cl)
Alejandro Vilches (alejandro.vilches@sansano.usm.cl)

Fecha de entrega: 29 de mayo 2019
Plazo máximo de entrega: 5 días.

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 la implementación de y uso de estructuras de datos de tipo listas y árboles.

3. Problema 1: Administrador de Memoria Dinámica

Se desea implementar un administrador de memoria dinámica. Se tiene una memoria de M bytes, inicialmente disponible como un único bloque de memoria contigua. A lo largo de la ejecución, existirán requerimientos de bloques de memoria contigua de un tamaño variable m bytes (operación `malloc`), y además se liberarán bloques de memoria asignados anteriormente (operación `free`).

Se pide administrar los bloques de memoria usando listas enlazadas. En particular, se necesitan dos listas:

- Lista de bloques disponibles (L_1): cada nodo de la lista almacenará un bloque de memoria disponible, representado por dos números enteros. Estos indican el byte de comienzo y el byte de final ocupados por el bloque. Inicialmente esta lista contiene un único nodo, que representa el bloque de memoria $[1..M]$. Los nodos de la lista se mantienen ordenados por el byte de comienzo de los bloques libres.
- Lista de bloques asignados (L_2): esta lista se mantiene de manera similar a la anterior, salvo que no es necesario mantener el orden de los nodos. Inicialmente, la lista está vacía (no hay bloques asignados).

Dado un requerimiento de memoria de m bytes, se debe recorrer la lista L_1 hasta encontrar el primer bloque de tamaño m' capaz de satisfacer el requerimiento (estrategia conocida como "first-fit"). Es decir, $m \leq m'$. En caso de que $m < m'$, se debe agregar el bloque de tamaño m a la lista L_2, y se debe modificar el tamaño del bloque disponible en la lista L_1 (el tamaño ahora es $m' - m$ bytes).

La liberación de memoria, por otro lado, se hará indicando el byte de comienzo del bloque a liberar. Dicho bloque debe ser buscado en la lista L_2, será eliminado de la misma, y asignado a la lista L_1 (en la posición adecuada). Se debe tener en cuenta que si existen bloques libres contiguos en L_1, los mismos deben ser unificados en un único gran bloque. Por ejemplo, supongamos que la lista L_1 contiene los bloques $[i..j]$ y $[k..l]$, y luego se libera el bloque $[j+1..l-1]$ de L_2. Entonces, los tres nodos contiguos de L_1 $[i..j]$, $[j+1..l-1]$ $[k..l]$ de la lista de bloques disponibles deben unificarse en un único bloque disponible $[i..l]$.

3.1. Entrada de Datos

La entrada de datos se hará mediante el archivo `input1.dat`, el cual tiene el siguiente formato:

```
M
N
OP1
...
OPN
```

En donde:

- `M` es la cantidad de bytes total de la memoria.
- `N` es la cantidad de operaciones que se harán sobre la memoria dinámica.
- `OPi` son las operaciones sobre la memoria dinámica, las cuales pueden ser de dos tipos:
  - `malloc m`: solicita la asignación de un bloque contiguo de `m` bytes.
  - `free b`: libera el bloque de memoria que comienza en el byte `b`. Se asume que los valores de `b` serán tales que el bloque correspondiente ya ha sido asignado.

Un posible ejemplo es el siguiente:

```
100
9
malloc 10
malloc 20
malloc 10
malloc 40
malloc 21
free 31
malloc 5
malloc 10
malloc 5
```

3.2. Salida de Datos

La salida de datos se hará mediante el archivo `output1.dat`. Por cada una de las operaciones del archivo `input2.dat`, este archivo tendrá una línea que indique alguna de las siguientes opciones:

- `Bloque de m bytes asignado a partir del byte B`: indica que una operación "`malloc m`" ha sido exitosa. El valor `B` indica el byte de comienzo del bloque asignado.

- `Bloque de m bytes NO puede ser asignado`: cuando una operación "`malloc m`" no puede ser satisfecha, dado que no hay un bloque disponible de tamaño suficiente.
- `Bloque de m bytes liberado`: indica que una operación "`free b`" liberó un bloque de m bytes.

De acuerdo a esto, la salida correspondiente al ejemplo visto anteriormente es:

```
Bloque de 10 bytes asignado a partir del byte 1
Bloque de 20 bytes asignado a partir del byte 11
Bloque de 10 bytes asignado a partir del byte 31
Bloque de 40 bytes asignado a partir del byte 41
Bloque de 21 bytes NO puede ser asignado
Bloque de 10 bytes liberado
Bloque de 5 bytes asignado a partir del byte 31
Bloque de 10 bytes asignado a partir del byte 81
Bloque de 5 bytes asignado a partir del byte 36
```

4. Problema 2: Árboles Binarios de Búsqueda con Operación Rank

Además de las operaciones típicas Insert y Buscar sobre un árbol binario de búsqueda (ABB), se necesita implementar la siguiente operación:

- Rank(x): dado un elemento x (no necesariamente almacenado en el ABB), retorna la cantidad de elementos del ABB que son menores o iguales a x.

La implementación de dicha función debe ser eficiente: si buscar por x en el ABB toma ℓ comparaciones, la operación Rank debe calcularse en ese mismo tiempo. De ser necesario, modifique la estructura del árbol y/o los nodos del árbol para resolver esta operación de forma eficiente.

4.1. Entrada de Datos

La entrada de datos se hará mediante el archivo `input2.dat`. Cada línea del archivo contiene una operación a realizarse sobre el ABB (inicialmente vacío). Las operaciones pueden ser Insert y Rank. Un ejemplo de dicho archivo de entrada es:

```
Insert 5
Insert 12
Insert 4
Rank 7
Insert 6
Rank 7
Insert 3
Rank 7
Insert 8
Rank 7
Rank 12
```

4.2. Salida de Datos

La salida de datos se hará mediante el archivo `output2.txt`. Por cada operación Rank del archivo de entrada, se debe imprimir un entero con la respuesta a dicha operación. Para el ejemplo anterior, la salida es:

```
2
3
4
4
6
```

5. Entrega de la Tarea

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

`tarea2-apellido1-apellido2-apellido3.tar.gz`

(reemplazando sus apellidos según corresponda) en el sitio Aulas USM del curso, a más tardar el día 29 de mayo 2019, a las 23:55:00 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.

6. Restricciones y Consideraciones

- Por cada día de atraso en la entrega de la tarea se descontarán 10 puntos en la nota.
- El plazo máximo de entrega es 5 días después de la fecha original de entrega.
- Pueden programar la tarea en C o C++ según ustedes consideren conveniente. Al programar en C++ queda prohibido utilizar la librería STL.
- Las tareas deben compilar en los computadores que se encuentran en el laboratorio B-032. **Las tareas que no compilen no serán revisadas y serán calificadas con nota 0**.
- 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.

7. 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**.