Pontificia Universidad Javeriana
Departamento de Ingeniería de Sistemas
Estructuras de Datos
Taller 3: Árboles Binarios Ordenados, 2030-30

1. Objetivo

Evaluar la eficiencia de los árboles binarios ordenados en operaciones de inserción de elementos. En especial, se desea evaluar las habilidades del estudiante en el uso y análisis de las operaciones de inserción y búsqueda de datos en árboles binarios ordenados, árboles AVL y árboles Rojo-Negro (*Red-Black*).

2. Recordatorio: compilación con g++

La compilación con `g++` (compilador estándar que será usado en este curso para evaluar y calificar las entregas) se realiza con los siguientes pasos:

1. **Compilación**: de todo el código fuente compilable (**ÚNICAMENTE LOS ARCHIVOS CON EXTENSIONES** `*.c`, `*.cpp`, `*.cxx`)
   `g++ -c *.c *.cxx *.cpp`
2. **Encadenamiento**: de todo el código de bajo nivel en el archivo ejecutable
   `g++ -o nombre_de_mi_programa *.o`

   Nota: Estos dos pasos (compilación y encadenamiento) pueden abreviarse en un sólo comando:
   `g++ -o nombre_de_mi_programa *.c *.cxx *.cpp`
3. **Ejecución**: del programa ejecutable anteriormente generado
   `./nombre_de_mi_programa`

ATENCIÓN: Los archivos de encabezados (`*.h`, `*.hpp`, `*.hxx`) **NO SE COMPILAN**, se incluyen en otros archivos (encabezados o código). Así mismo, los archivos de código fuente (`*.c`, `*.cpp`, `*.cxx`) **NO SE INCLUYEN**, se compilan. Si el programa entregado como respuesta a este Taller no atiende estas recomendaciones, automáticamente se calificará la entrega sobre un 25% menos de la calificación máxima.

3. Desarrollo del taller

Una de las actividades comunes más dispendiosa que las personas suelen realizar es acudir al banco (o institución financiera) a realizar alguna transacción, como pagos, consignaciones o retiros de dinero. Para mejorar la atención a los clientes, y evitar que deban esperar de pie durante largos períodos de tiempo en la fila de atención, algunas entidades han optado por incorporar el uso del digiturno, un código alfanumérico digital consecutivo que es asignado a cada usuario a medida que va llegando al banco. El código utiliza letras para distinguir las diferentes operaciones que el usuario puede realizar (las cuales se representan con prefijos de dos o tres letras) y números consecutivos de 3 o 4 cifras para indicar el orden en el que han llegado los usuarios a realizar la transacción específica. Para complementar la implementación de esta estrategia de atención a usuarios, se busca una estrategia de almacenamiento de los digiturnos para que la búsqueda de un cliente particular o la extracción de grupos de clientes de acuerdo a rangos de turnos sea fácil y eficiente.

El desarrollo del taller consistirá en utilizar tres implementaciones diferentes de árboles binarios ordenados para organizar los códigos de turnos. Las implementaciones a usar y comparar serán: implementación propia del árbol binario ordenado, implementación propia del árbol AVL y la estructura `std::set<T>` de la STL (implementada internamente como un árbol Rojo-Negro (*Red-Black*)). Por simplicidad, se asumirá que sólo se dispone del código alfanumérico de los turnos como información asociada.

La información necesaria para poblar el árbol con los turnos ya asignados a los clientes se encuentra en un archivo de texto, donde línea por línea se indica la operación a realizar (`add`: agregar, `del`: eliminar) y el código de cada turno. Por ejemplo, el siguiente archivo:

```
add VEN145
add SOP091
del SOP091
```

Agregaría al árbol dos turnos con códigos `VEN145` y `SOP091`, para luego eliminar del árbol el turno con código `SOP091`. En el archivo, se garantiza que todos los códigos de turno tienen longitud de 6 caracteres, 3 letras y 3 dígitos.

Las tareas puntuales que hacen parte del desarrollo del taller son las siguientes:

1. **(20 %)** Utilizar la implementación propia del árbol binario ordenado para cargar la información de los turnos. Es de especial importancia realizar la medición del tiempo de ejecución del proceso de carga en el árbol, para lo cual se hará uso de comandos de medición de tiempo (Ver `TODOs #1, #2, #3, #4, #5` en el código fuente `taller_3_ordenamiento_busqueda.cxx`).

2. **(20 %)** Utilizar la implementación propia del árbol AVL para cargar la información de los turnos. Además de que la información quede adecuadamente distribuída y organizada dentro del árbol, es de especial importancia realizar la medición del tiempo de ejecución del proceso de carga en el árbol, para lo cual se hará uso de comandos de medición de tiempo (Ver `TODOs #1, #2, #3, #6, #7` en el código fuente `taller_3_ordenamiento_busqueda.cxx`).

3. **(15 %)** Implementar una función que le permita extraer el recorrido en inorden del árbol binario ordenado en una lista (`std::list<T>`). El resultado de esta función permitirá verificar si luego del proceso de carga las estructuras almacenan exactamente la misma información (Ver `TODO #8` en el código fuente `taller_3_ordenamiento_busqueda.cxx`).

4. **(15 %)** Implementar una función que le permita extraer el recorrido en inorden del árbol AVL en una lista (`std::list<T>`). El resultado de esta función permitirá verificar si luego del proceso de carga las estructuras almacenan exactamente la misma información (Ver `TODO #9` en el código fuente `taller_3_ordenamiento_busqueda.cxx`).

5. **(30 %)** Redacte un informe en el cual describa los pasos llevados a cabo para la carga de la información en las estructuras y para la verificación de los datos almacenados en cada una. Utilice además las mediciones de tiempo para redactar un párrafo en el cual describa cuál estructura es la más adecuada para este problema. Debe adicionar la plantilla de diseño de los TADs propios y el respectivo diagrama de TADs.

4. Evaluación

Como parte del desarrollo del taller, se debe entregar:

1. Código fuente funcional que realiza la carga del archivo en los respectivos árboles (con medición de tiempo) y que, para cada implementación propia de los árboles, incluye la función de la extracción, en una lista, del recorrido en inorden de un árbol.
2. Informe escrito que describa los pasos llevados a cabo y que genere una recomendación sobre cuál estructura es la mejor para resolver el problema planteado en términos de criterios como el tiempo de ejecución y la facilidad de uso. Debe adicionar la plantilla de diseño de los TADs propios y el respectivo diagrama de TADs.

La entrega se hará a través de la correspondiente actividad de BrigthSpace, antes de la media noche del 20 de septiembre de 2023. Se debe entregar un único archivo comprimido (únicos formatos aceptados: .zip, .tar, .tar.gz, .tar.bz2, .tgz) que contenga dentro de un mismo directorio (sin estructura de carpetas interna), documentos (único formato aceptado: .pdf) y código fuente (.h, .hxx, .hpp, .c, .cxx, .cpp). Si la entrega contiene archivos en cualquier otro formato, será descartada y no será evaluada, es decir, la nota definitiva de la entrega será de 0 (cero) sobre 5 (cinco).

La evaluación del taller tendrá la siguiente escala para cada una de las actividades o secciones de código a completar:

- **Excelente** (**5.0/5.0**): El código es correcto o la respuesta es correcta y concisa.
- **Bueno** (**3.5/5.0**): El código es parcialmente correcto o la respuesta es aproximada y/o no es concisa.
- **Necesita mejoras sustanciales** (**2.0/5.0**): El código no es correcto o la respuesta es incorrecta.
- **No entregó** (**0.0/5.0**).

5. Penalizaciones.

1. El informe debe estar en formato IEEE para informes (1 columna). El incumplimiento de esta regla generará una calificación del taller sobre 3.0