Pontificia Universidad Javeriana
Departamento de Ingeniería de Sistemas
Estructuras de Datos
Taller 3: Árboles de búsqueda, 2025-30

1. Objetivo

Evaluar la eficiencia de los árboles binarios ordenados en operaciones de búsqueda de elementos. En especial, se desea evaluar las habilidades del estudiante en el uso y análisis de las operaciones de inserción, búsqueda y eliminación de datos en árboles AVL, rojinegros (RN) y montículos.

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++ -std=c++11 -c *.c *.cxx *.cpp`
2. **Encadenamiento**: de todo el código de bajo nivel en el archivo ejecutable
   `g++ -std=c++11 -o nombre_de_mi_programa *.o`

   Nota: Estos dos pasos (compilación y encadenamiento) pueden abreviarse en un sólo comando:
   `g++ -std=c++11 -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 fábrica de televisores almacena la información de sus productos utilizando como índice un código numérico(identificación) de 6 dígitos (0-9). La empresa busca una estrategia de almacenamiento de esta información para que la búsqueda de un televisor particular o la extracción de grupos de televisores producidos de acuerdo con rangos de códigos sea fácil y eficiente.

El desarrollo del taller consistirá en utilizar tres implementaciones diferentes de árboles binarios ordenados para organizar los códigos de identificación de los televisores. Las tres implementaciones que usar y comparar serán: implementación propia del árbol AVL, la estructura *std::set<T>* de la STL (implementada internamente como un árbol rojinegro (RN)) y la implementación de montículos (*heap*). Por simplicidad, se asumirá que sólo se dispone del código numérico de los televisores como información asociada.

La información necesaria para poblar los árboles con los códigos de los televisores producidos se encuentra en un archivo de texto, donde línea por línea se indica la operación a realizar (*add: agregar,del: eliminar*) y el número de identificación del televisor. Por ejemplo, el siguiente archivo:

```
add 243535
add 546384
del 243535
```

Agregaría al árbol dos nuevos televisores con códigos 243535 y 546384, para luego eliminar del árbol el televisor identificado con código 243535. En el archivo, se garantiza que todos los códigos tienen longitud de 6 dígitos, por lo que pueden existir códigos de televisores con ceros a la izquierda.

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

1. **(25%)** Utilizar su implementación propia del árbol AVL para cargar la información de los códigos de los televisores en un árbol binario ordenado. Además de que la información quede adecuadamente distribuida 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 #01, #02, #03, #04, #05 en el código fuente *taller_3_ordenamiento_busqueda.cxx*): *Favor de no borrar el comentario "TODO" para facilitar el proceso calificativo*.
    a. **(3 %)** TODO #01: Incluir cabecera de la implementación propia del árbol AVL.
    b. **(3 %)** TODO #02: Definir árbol AVL de tipo std::string.
    c. **(3 %)** TODO #03: Definir variable tipo árbol AVL.
    d. **(9 %)** TODO #04: Usar el archivo para llenar el árbol AVL a través del uso de la función genérica provista para ello. El desarrollo de este TODO estaría en la implementación propia del árbol AVL.
    e. **(4 %)** TODO #05: Llamar la función que genera el recorrido en inorden del árbol AVL y lo guarda en una lista dada como parámetro.
2. **(25%)** Utilizar la implementación dada por la STL de C++ para el árbol rojinegro para cargar la información de los códigos de los televisores. Además de que la información quede adecuadamente distribuida 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 #06, #07, #08, #09, #10 en el código fuente *taller_3_ordenamiento_busqueda.cxx*): *Favor de no borrar el comentario "TODO" para facilitar el proceso calificativo*.
    a. **(3 %)** TODO #06: Incluir cabecera de la STL correspondiente al árbol rojinegro.
    b. **(3 %)** TODO #07: Definir árbol rojinegro de tipo std::string.
    c. **(3 %)** TODO #08: Definir variable tipo árbol rojinegro.
    d. **(9 %)** TODO #09: Usar el archivo para llenar el árbol AVL a través del uso de la función genérica provista para ello. El desarrollo de este TODO estaría en la implementación del árbol RN.
    e. **(4 %)** TODO #10: Llamar la función que genera el recorrido en inorden del árbol rojinegro y lo guarda en una lista dada como parámetro.
3. **(25%)** Utilizar la implementación dada por la STL de C++ para el montículo para cargar la información de los códigos de los televisores. Además de que la información quede adecuadamente distribuida 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 #11, #12, #13, #14, #15 en el código fuente *taller_3_ordenamiento_busqueda.cxx*) Recuerde que el montículo tiene un funcionamiento diferente a los árboles, por lo que la estrategia a seguir para la inserción y eliminación de los códigos debe ser diferente. *Favor de no borrar el comentario "TODO" para facilitar el proceso calificativo*.
    a. **(3 %)** TODO #11: Incluir cabecera de la STL correspondiente al montículo.
    b. **(3 %)** TODO #12: Definir Montículo de tipo std::string.
    c. **(3 %)** TODO #13: Definir variable tipo Montículo.
    d. **(9 %)** TODO #14: Usar el archivo para llenar el montículo a través del uso de la función genérica provista para ello. El desarrollo de este TODO estaría en la implementación propia del árbol AVL.
    e. **(4 %)** TODO #15: Llamar la función que genera el recorrido en inorden del montículo y lo guarda en una lista dada como parámetro.
4. **(2.5 %)** TODO #16: Crear iteradores para recorrer cada una de las estructuras lineales.
5. **(5 %)** TODO #17: Recorrer las estructuras lineales y comparar elemento a elemento la igualdad o desigualdad.
6. **(2.5 %)** TODO #18: Informar si los árboles coinciden en la totalidad de los elementos teniendo en cuenta su posición.

En este punto del taller, el enfoque debe desplazarse de la implementación del código hacia el análisis crítico y la evaluación de los resultados obtenidos. El objetivo es que el grupo de trabajo desarrolle sus propios casos de prueba, basándose en los archivos entregados o generando conjuntos de datos personalizados, con el fin de obtener conclusiones fundamentadas sobre el comportamiento de las estructuras analizadas.

A partir de los experimentos realizados, analicen y discutan las siguientes preguntas:

7. **(5 %)** ¿Cuál de las estructuras analizadas (AVL, Árbol Rojo-Negro, Montículo) mostró un mejor desempeño en operaciones de inserción?
    a. **(2.5 %)** ¿Cómo varía el desempeño de la inserción cuando se trabaja con pequeños volúmenes de datos frente a conjuntos de datos más grandes?
    b. **(2.5 %)** ¿Se puede identificar un punto de inflexión donde una estructura comienza a ser más eficiente que otra en el caso de la inserción?
8. **(5 %)** ¿Cuál de las estructuras analizadas (AVL, Árbol Rojo-Negro, Montículo) mostró un mejor desempeño en operaciones de eliminación?
    a. **(2.5 %)** ¿Cómo varía el desempeño de la eliminación cuando se trabaja con pequeños volúmenes de datos frente a conjuntos de datos más grandes?
    b. **(2.5 %)** ¿Se puede identificar un punto de inflexión donde una estructura comienza a ser más eficiente que otra en el caso de la eliminación?
9. **(7 %)** Considerando un escenario donde la eficiencia de búsqueda es crítica (por ejemplo, en sistemas de videojuegos para la gestión de colisiones o en simulaciones físicas), ¿qué estructura sería más adecuada y por qué?
10. **(7 %)** En el caso de sistemas con grandes volúmenes de datos dinámicos (como una base de datos en tiempo real), ¿cuál sería la mejor elección y bajo qué circunstancias?

4. Evaluación

La entrega se hará a través de la correspondiente actividad de Brightspace, antes de la media noche del **viernes 26 de septiembre de 2025**. Se debe entregar el **documento reporte (.pdf)** y un **archivo comprimido** (únicos formatos aceptados: .zip, .tar, .tar.gz, .tar.bz2, .tgz) que contenga dentro de un mismo directorio (sin estructura de carpetas interna) el código fuente modificado (.h, .hxx, .hpp, .c, .cxx, .cpp).

- Si la entrega contiene archivos en cualquier otro formato o más archivos de los solicitados en esta descripción, será descartada y no será evaluada, es decir, la nota definitiva de la entrega será de 0 (cero) sobre 5 (cinco).
- Si el archivo comprimido no cuenta con las especificaciones descritas (sin estructura de carpetas interna), la nota definitiva de la entrega será de 0 (cero) sobre 5 (cinco).
- Si el código fuente no compila, la nota definitiva de la entrega será de 0.0 (cero) sobre 5 (cinco).

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

- **Excelente** (**5.0/5.0**): El código es correcto, compila sin advertencias y/o la respuesta es correcta y concisa.
- **Bueno** (**3.5/5.0**): El código es correcto, pero compila con advertencias y/o la respuesta es correcta pero no es concisa.
- **No fue un trabajo formal de ingeniería** (**3.0/5.0**): El código es parcialmente correcto y/o compila con advertencias y/o la respuesta es aproximada y/o no es concisa.
- **Necesita mejoras sustanciales** (**2.0/5.0**): El código no es correcto y/o la respuesta no es clara.
- **Malo** (**1.0/5.0**): El código entregado por el estudiante no compila en el compilador g++
- **No entregó** (**0.0/5.0**).

Para el caso de las secciones escritas correspondientes al reporte se manejará la siguiente escala:

- **Excelente** (**5.0/5.0**): El estudiante presenta análisis y explicaciones organizados (información) que permiten comprender en su totalidad los procedimientos, funciones o errores.
- **Bueno** (**4.0/5.0**): El estudiante tiene una idea parcial de los procedimientos, funciones o errores, pero falla en el formalismo de la presentación escrita. No presenta las secciones escritas de manera organizada lo que dificulta la comprensión de las ideas.
- **Regular** (**3.0/5.0**): El estudiante logra esbozar una idea de los procedimientos, funciones o errores, falla en presentarla de manera clara y contundente. No presenta claridad en sus explicaciones.
- **Malo** (**1.5/5.0**): El estudiante no logra entender las instrucciones presentadas, y su análisis es vago o ambiguo.
- **No entregó** (**0.0/5.0**): No entregó el archivo del informe, o el archivo entregado no está en formato PDF.

Evaluación individual mediante quiz (viernes 15 de septiembre)

El **viernes 26 de septiembre de 2025** se llevará a cabo un ***quiz individual obligatorio*** como parte del proceso de evaluación del Taller 2. Este quiz tiene como objetivo verificar la comprensión individual de los conceptos, estructuras y operaciones implementadas durante el desarrollo del taller.

- El quiz constará de 4 preguntas de tipo conceptual, técnico o práctico.
- Cada estudiante deberá responder solo 3 preguntas de su elección.
- Las preguntas serán elaboradas con base en el contenido del taller y en los temas vistos en clase.

Criterio de evaluación individual

La calificación individual del taller dependerá de la cantidad de respuestas correctas en el quiz, de la siguiente manera:

Preguntas correctas	Nota individual respecto a la nota grupal
3	**100% de la nota grupal**
2	**85% de la nota grupal**
1	**65% de la nota grupal**
0	**45% de la nota grupal**

Este mecanismo busca promover la ***responsabilidad individual***, ***el trabajo colaborativo con comprensión real*** y la ***participación activa*** en el desarrollo del taller. Cada estudiante debe estar en capacidad de explicar e interpretar el funcionamiento del código entregado por su grupo.

Recomendaciones: uso de herramientas de IA generativa

Para la implementación de los componentes de este taller, se espera que los estudiantes evidencien la apropiación y aplicación de los conceptos estudiados en clase. No obstante, se permite el uso de herramientas de inteligencia artificial generativa (como ChatGPT, GitHub Copilot, Copilot Chat, entre otras), bajo las siguientes condiciones:

1. **Identificación clara**: Todo fragmento de código, texto, explicación o análisis que haya sido generado con el apoyo de una herramienta de IA generativa debe estar explícitamente identificado mediante un comentario o nota aclaratoria en el informe o código fuente.
2. **Referencia del prompt**: Si se genera un contenido textual o de código completo con IA, se debe incluir el prompt (instrucción) utilizado para obtener dicho contenido, ya sea directamente o en un comentario adjunto al fragmento generado.
3. **Transformaciones propias**: Si el contenido generado por IA ha sido posteriormente editado, modificado o adaptado por el estudiante, esta transformación también debe dejarse explícita, indicando qué se cambió o adaptó y con qué fin.
4. **Límite de contenido generado**: Se sugiere que el contenido generado por herramientas de IA no supere un tercio del total del trabajo entregado, de manera que se garantice el dominio propio del estudiante sobre los conceptos desarrollados.

El no cumplimiento de estas recomendaciones podrá ser considerado como falta de honestidad académica si se detecta un uso encubierto de herramientas automatizadas en la elaboración del trabajo.