Reporte de Proyecto Final: Estructuras de Datos Avanzadas

Nombre del Alumno: Fernando Osuna Manzo
Fecha: 2 de Diciembre de 2025
Materia: Estructuras de Datos Avanzada

1. Introducción

En este proyecto se implementaron y compararon tres estructuras de datos fundamentales para el manejo eficiente de información: Árboles Binarios de Búsqueda (BST), Árboles AVL y el algoritmo de compresión de Huffman. El objetivo principal fue desarrollar un sistema capaz de indexar texto para búsquedas rápidas y comprimir archivos para ahorrar espacio de almacenamiento.

2. Metodología

2.1. Indexación con BST (Binary Search Tree)

Se implementó un BST donde cada nodo almacena una palabra y una lista de sus posiciones (línea, columna) en el texto.

- **Inserción:** Se recorre el árbol comparando la palabra a insertar. Si ya existe, se añade la nueva posición a la lista. Si no, se crea un nuevo nodo.
- **Búsqueda:** Se recorre el árbol de manera similar a la inserción.
- **Complejidad:** En el peor caso (árbol degenerado), la complejidad es O(n). En el caso promedio es O(log n).

2.2. Indexación con AVL (Adelson-Velsky and Landis)

Se implementó un árbol AVL, que es un BST auto-balanceado.

- **Balanceo:** Cada nodo mantiene un factor de balance. Si este factor sale del rango [-1, 1] tras una inserción o eliminación, se realizan rotaciones (simple o doble) para restaurar el equilibrio.
- **Ventaja:** Garantiza una altura logarítmica, asegurando que las operaciones de búsqueda, inserción y eliminación sean siempre O(log n).

2.3. Compresión con Huffman

Se implementó el algoritmo de Huffman para la compresión de archivos de texto.

- **Frecuencias:** Se cuenta la frecuencia de cada caracter en el texto.
- **Árbol de Huffman:** Se construye un árbol binario donde las hojas son los caracteres y los nodos internos representan la suma de frecuencias de sus hijos. Los caracteres más frecuentes quedan más cerca de la raíz.
- **Codificación:** Se asignan códigos binarios (0 para izquierda, 1 para derecha). Los caracteres frecuentes obtienen códigos más cortos.
- **Persistencia:** El archivo comprimido (.huff) almacena la tabla de frecuencias (para reconstruir el árbol) y los bits comprimidos.

3. Resultados y Análisis

3.1. Comparativa de Tiempos de Construcción (BST vs AVL)

Pruebas realizadas con `test_data.txt (~1000 palabras).`

Estructura	Tiempo de Construcción (ms)	Observaciones
BST	9.6474 ms	Más rápido al no requerir balanceo.
AVL	27.2293 ms	Más lento debido al cálculo de factores de balance y rotaciones.

Análisis:
Como se esperaba, la construcción del árbol AVL tomó aproximadamente 3 veces más tiempo que el BST (27.22 ms vs 9.64 ms). Esto se debe al costo computacional adicional de verificar el factor de equilibrio en cada inserción y realizar las rotaciones necesarias para mantener el árbol balanceado. Para una carga inicial masiva de datos donde el tiempo de indexación es crítico, el BST simple tiene ventaja, aunque esto podría comprometer el rendimiento de búsquedas futuras si los datos entran ordenados.

3.2. Comparativa de Tiempos de Búsqueda

Palabra Buscada	Tiempo BST (ms)	Tiempo AVL (ms)
hidalgo	0.019073	0.010014
caballero	0.018358	0.007868
dulcinea	0.007629	0.005960

Análisis:
Los resultados muestran consistentemente que el árbol AVL ofrece tiempos de

búsqueda más rápidos. Por ejemplo, para la palabra "caballero", el AVL fue más del doble de rápido que el BST (0.0078 ms vs 0.0183 ms). Esto confirma la teoría de que mantener el árbol balanceado (altura logarítmica) optimiza significativamente las operaciones de lectura, justificando el costo extra pagado durante la construcción.

3.3. Eficiencia de Compresión Huffman

- **Tamaño Original:** 6892 Bytes
- **Tamaño Comprimido:** 4048 Bytes
- **Porcentaje de Ahorro:** 41.27%

Análisis:
El algoritmo de Huffman logró reducir el tamaño del archivo en un 41.27%, lo cual es una compresión significativa para un texto en español antiguo. Este ahorro se logra asignando códigos binarios más cortos a letras muy frecuentes como 'a', 'e', 'o', mientras que caracteres raros reciben códigos más largos. Es una técnica muy efectiva para almacenamiento y transmisión de texto.

4. Conclusiones

El proyecto demostró claramente el compromiso (trade-off) clásico en estructuras de datos: el AVL requiere más tiempo de procesamiento durante la inserción (casi el triple que el BST en nuestras pruebas) para garantizar una estructura óptima, lo que resulta en búsquedas mucho más rápidas (hasta el doble de velocidad). Por lo tanto, AVL es ideal para bases de datos de lectura intensiva, mientras que un BST simple podría bastar para datos aleatorios o escritura intensiva donde la búsqueda no es crítica. Adicionalmente, la implementación de Huffman validó su eficacia para reducir el almacenamiento necesario en más de un 40% sin pérdida de información.

5. Contribucion de la AI

1. Mejora de la redactabilidad de los documentos para lograr una redacción profesional.
2. Limpieza y refactorizacion de codigo para un codigo mas limpio y legible
3. Consulta de dudas y sugerencias.
4. Archivos de testing para la creación del proyecto y el reporte.

6. Referencias

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms*. MIT Press.
- Documentación de Python.
- Gemini 3.0 High.