Universidad Rafael Landívar
Facultad de Ingeniería.
Ingeniería en informática y sistemas.
Estructura de datos I - Sección: 01
Docente:René Daniel Mejía Alvarado.

PROYECTO DE APLICACIÓN I

"Libreta de contactos"

Estudiante: Jose Rodrigo Peñate Ortiz.
Carné: 1045324.

Estudiante: Nery Samuel Hernández Herrera.
Carné: 1098824.

Guatemala, 25 de abril de 2025.

INDICE

I. INTRODUCCIÓN

El propósito de esté proyecto es realizar una aplicación de consola en java capaz de gestionar contactos de una eficiente manera con el fin de emplear estructuras de datos avanzadas como arboles BST, AVL, entre otras. Dicha aplicación permite realizar operaciones como crear, eliminar, consultar, y eliminar contactos, y optimiza las busquedes utilizando índices con el uso de arboles binarios de búsqueda (BST) y arboles AVL.

En sistemas modernos, es fundamental buscar la gestión eficiente de datos, especialmente en casos donde los datos deben organizarse y buscarse de manera rápida, este proyecto aplica conceptos donde se utilizan estructuras de datos para situaciones como esta. Integrando manejo de archivos y estructuras balanceadas para búsquedas rápidas.

II. ANÁLISIS DEL SISTEMA

2.1 REQUISITOS FUNCIONALES:

1. Almacenamiento ediciente de contactos en un archivo contacts.csv
2. Operaciones básicas de gestión de contactos (Crear, Leer, Actualizar, Eliminar)
3. Creación de índices en campos específicos (nombre, apellido, correo, etc.) utilizando AVL y BST.
4. Exportación de estructuras de índice a archivos de texto para reconstrucción posterior.
5. Soporte para búsquedas basadas en índices.

2.2 ESTRUCTURA DEL CÓDIGO:

Tabla No. 01

Clases utilizadas

Clase	Descripción
Main,java	- Punto de entrada del programa - Captura las acciones del usuario (agregar contactos, eliminarlos, buscar, actualizar, etc) - Delega operaciones específicas a otras clases como ArchCSV e implementaciones de arboles. - Presenta el flujo lógico por un switch – case -
Contacto.java	- Define las propiedades de un contacto (id, nombre, apodo, numeroTelefónico, etc.) - Centraliza el manejo de datos de un contacto mediante getters y setters. - Facilita la persistencia en formato CSV a través del método toCSV()
ArchCSV.java	- Clase encargada de la gestión de archivo contacts.csv, Centraliza las operaciones CRUD en el archivo CSV, como las siguientes: - **Agregar Contacto:** Genera un nuevo ID único automáticamente - **Buscar contacto:** Localiza un contacto por su ID y lo imprime en la consola. - **Eliminar contacto:** Usa un archivo temporal para eliminar un contacto sin perder los demás datos. - **Actualizar contacto**: Reemplaza un contacto existente en el archivo. - **Organizar por campo**: Devuelve datos organizados por un campo específico, útil para crear índices.
ArbolBST.java	- Proporciona una estructura para almacenar contactos indexados por un campo como nombre o apellido - Inserta nodos en el árbol en la posici[on correcta seg[un el orden indicado por un comparator. - Realiza un recorrido BFS para generar índices persistentes

ArbolAVL.java	- Extiende la funcionalidad del BST agregando la lógica del balanceo - Inserta nodos manteniendo el árbol balanceado mediante rotaciones (rotateLeft, rotateRight) - Realiza un recorrido por niveles para exportar la estructura de un archivo
Nodo.java	- Representa la estructura de un elemento de un árbol, encapsulando tanto el ID único de un contacto como el campo que se está utilizando para ordenar e indexar. - Define la estructura básica de un nodo (int id, string dato, int height, Nodo left y Nodo right) - Actua como base que las clases ArbolAVL y ÁrbolBST utilizan para implementar las operaciones de inserción, búsqueda y recorrido.
ArchivoIndice.java	- Genera y guarda la representación de los árboles en archivos de texto, preservando su estructura completa (incluyendo nodos null). - Serializa la estructura de un árbol (AVL o BST) con un recorrido por niveles. - Guarda los resultados en un archivo nombrado según el campo y el tipo de árbol (por ejemplo, nombre-avl.txt).

III. ALGORITMOS UTILIZADOS

3.1 INSERCIÓN EN BST

El algoritmo sigue las reglas del ordenamiento binario:

- Si el nuevo dato es menor que el nodo actual, se recorre hacia la izquierda.
- Si el nuevo dato es mayor, se recorre hacia la derecha.
- Cuando se encuentra un espacio vacío (hijo nulo), se inserta el nuevo nodo.

* No incluye lógica de balanceo, por lo que el árbol puede desbalancearse si los datos están ordenados de forma ascendente o descendente.

Pasos del algoritmo:

1. Comienza en la raíz.
2. Compara el dato del nuevo nodo (nuevo.dato) con el del nodo actual (node.dato):
   Si es menor, recorre hacia node.left.
   Si es mayor, recorre hacia node.right.
3. Cuando encuentra un nodo null, inserta el nuevo nodo allí.
4. Retorna el nodo actualizado.

Codigo del Proyecto:

```
private Nodo insert(Nodo node, Nodo nuevo) {
    if (node == null) {
        size++;
        return nuevo;
    }

    int cmp = comparator.compare(nuevo.dato, node.dato);
    if (cmp < 0) {
        node.left = insert(node.left, nuevo);
    } else if (cmp > 0) {
        node.right = insert(node.right, nuevo);
    } else {
        // Si el dato es igual, podés decidir ignorarlo o permitir duplicados
por id
        return node;
    }
```

3.2 INSERCIÓN EN AVL

- Es similar a la inserción en BST, pero incluye lógica adicional para asegurar que el árbol permanezca balanceado mediante rotaciones.
- Después de insertar un nodo:
  Calcula el factor de balance de cada nodo afectado.
  Si el factor de balance está fuera del rango [-1, 1], realiza una rotación (izquierda, derecha o doble) para equilibrar el árbol.

  Pasos del algoritmo:
  1. Inserta el nodo siguiendo las reglas del BST.

2. Actualiza la altura del nodo.
3. Calcula el factor de balance: $Factor\ de\ Balance = Altura\ izquierda - Altura\ derecha$
4. Si el factor de balance > 1 o < -1:

Realiza rotaciones (rotateLeft, rotateRight).

Retorna el nodo balanceado.

Codigo del proyecto:

```
private Nodo insert(Nodo node, Nodo nuevo) {
    if (node == null) {
        size++;
        return nuevo;
    }

    int cmp = comparator.compare(nuevo.dato, node.dato);
    if (cmp < 0) {
        node.left = insert(node.left, nuevo);
    } else if (cmp > 0) {
        node.right = insert(node.right, nuevo);
    } else {
        return node;
    }

    return balance(node);
}
```

3.3 SERIALIZACIÓN DE RECORRIDOS POR NIVELES:

Recorrido por niveles (BFS):

- Este algoritmo recorre el árbol nivel por nivel, comenzando desde la raíz y avanzando hacia los hijos usando una cola.Si es mayor, recorre hacia la derecha.
- Es utilizado para generar archivos que representen la estructura de los árboles, incluyendo nodos null para preservar la estructura completa.

Pasos del algoritmo:

1. Crea una cola y agrega la raíz del árbol.
2. Mientras la cola no esté vacia:
   Extrae el nodo en el frente de la cola.
   Agrega su id al resultado si el nodo no es null.
   Si el nodo es null, agrega una representación de null al resultado.
   Agrega los hijos izquierdo y derecho del nodo a la cola.
3. Retorna el recorrido en nivel como una lista.

Código en el proyecto:

```java
public List<Integer> recorridoPorNiveles() {
    List<Integer> resultado = new ArrayList<>();
    if (root == null) return resultado;

    Queue<Nodo> cola = new LinkedList<>();
    cola.add(root);

    while (!cola.isEmpty()) {
        Nodo actual = cola.poll();
        resultado.add((Integer) actual.id); // Solo guardás el ID

        if (actual.left != null) cola.add(actual.left);
        if (actual.right != null) cola.add(actual.right);
    }

    return resultado;
```

IV. ARCHIVOS GENERADOS POR EL SISTEMA

4.1 ARCHIVO DE CONTACTOS: <contacts.csv>:

Este archivo actúa como el almacenamiento principal de la información de los contactos. Todas las operaciones CRUD (Crear, Leer, Actualizar y Eliminar) se reflejan directamente en este archivo.

Estructura:

El archivo contiene los datos de los contactos organizados en formato CSV (Comma Separated Values). Su primera línea es una cabecera con los nombres de los campos, y cada línea subsiguiente representa un contacto.

Ejemplo de contenido:

id,nombre,apellido,apodo,numeroTelefono,correo,direccion,fechaNacimiento
1,Rodri,Peñate,R.P,5551234,rodri.peñate@example.com,Zona 1,6-6-2005
2,Samuel,Hernandez,S.H5555678,samu.her@example.com,Zona 2-27-08-2005

4.1 ARCHIVO DE INDICES POR ÁRBOLES:

Los índices generados por los árboles AVL y BST se almacenan en archivos de texto independientes, nombrados según el campo utilizado para indexar y el tipo de estructura.

Nombre de archivo:

`Formato: <campo>-<tipo>-completo.txt`

Ejemplos:

`nombre-avl-completo.txt`

`apellido-bst-completo.txt`

Estos archivos sirven para preservar la estructura completa de los árboles generados, incluidas las posiciones de los nodos nulos (null). Esto permite reconstruir los árboles más adelante si es necesario.z

Contenido:

Cada archivo incluye un recorrido por niveles de la estructura del árbol. Los nodos null también se representan explícitamente para mantener la integridad de la estructura.

Ejemplo:

```
1
2
null
3
null
null
4
```