UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
CENTRO UNIVERSITARIO DE OCCIDENTE
DIVISIÓN DE CIENCIAS DE LA INGENIERÍA
ESTRUCTURA DE DATOS

MANUAL DE TECNICO

DIEGO ALESSANDRO GONZALEZ BRAVO / 202230393

Objetivo del Sistema
Desarrollar un sistema de gestion de biblioteca que utilice multiples estructuras de datos avanzadas para optimizar operaciones de busqueda, insercion y eliminación de libros, comparando el rendimiento entre diferentes metodos.

Alcance Tecnico

Implementación desde cero de todas las estructuras de datos
Sin uso de librerias STL para estructuras (std::vector, std::map, etc.)
Programación Orientada a Objetos (POO)

Tecnologías Utilizadas

Lenguaje: C++20
Herramienta de compilacion: Make
Visualizacion: Graphviz (DOT)

Tipos abstractos de datos

1. Libro
   TAD Libro
      Datos:
         titulo: String
         isbn: String (clave unica)
         genero: String
         anio: Entero
         autor: String

      Operaciones:
         Libro(t: String, i: String, g: String, a: Entero, au: String)
            Pre: isbn debe ser unico en el sistema
            Post: Crea una instancia de Libro con los atributos dados

         getTitulo(): String
            Pre: Libro inicializado
            Post: Retorna el titulo del libro

         getIsbn(): String
            Pre: Libro inicializado
            Post: Retorna el ISBN del libro

         getGenero(): String
            Pre: Libro inicializado
            Post: Retorna el genero del libro

         getAño(): Entero
            Pre: Libro inicializado
            Post: Retorna el año de publicacion

getAutor(): String
Pre: Libro inicializado
Post: Retorna el autor del libro

mostrarInfo(): void
Pre: Libro inicializado
Post: Imprime informacion del libro en consola

Invariantes:
- isbn != ""
- titulo != ""
- -3000 <= anio <= 2100

Fin TAD

2. Lista enlazada

TAD ListaEnlazada
Datos:
cabeza: NodoLista*
tamaño: Entero

Operaciones:
ListaEnlazada()
Pre: Ninguna
Post: Crea lista vacia con cabeza = NULL, tamaño = 0

insertar(libro: Libro*): void
Pre: libro != NULL
Post: Agrega libro al final, tamaño++
Complejidad: O(n)

insertarAlInicio(libro: Libro*): void
Pre: libro != NULL
Post: Agrega libro al inicio, tamaño++
Complejidad: O(1)

buscarPorTitulo(titulo: String): Libro*
Pre: Ninguna
Post: Retorna libro con titulo dado o NULL
Complejidad: O(n)

buscarPorISBN(isbn: String): Libro*
Pre: Ninguna
Post: Retorna libro con ISBN dado o NULL
Complejidad: O(n)

eliminar(isbn: String): Boolean
Pre: Ninguna
Post: Elimina libro con ISBN dado, retorna true si existe

Complejidad: O(n)

estaVacia(): Boolean
Pre: Ninguna
Post: Retorna true si tamaño = 0
Complejidad: O(1)

obtenerTamaño(): Entero
Pre: Ninguna
Post: Retorna tamaño
Complejidad: O(1)

Invariantes:
- tamaño >= 0
- Si tamaño = 0 entonces cabeza = NULL
- Si tamaño > 0 entonces cabeza != NULL

Fin TAD

3. Arbol AVL

TAD ArbolAVL
Datos:
raiz: NodoAVL*
tamaño: Entero

Operaciones:
ArbolAVL()
Pre: Ninguna
Post: Crea arbol vacio con raiz = NULL, tamaño = 0

insertar(libro: Libro*): void
Pre: libro != NULL
Post: Inserta libro manteniendo balance AVL, tamaño++
Complejidad: O(log n)

buscarPorTitulo(titulo: String): Libro*
Pre: Ninguna
Post: Retorna libro con titulo dado o NULL
Complejidad: O(log n)

eliminar(titulo: String): Boolean
Pre: Ninguna
Post: Elimina libro manteniendo balance AVL
Complejidad: O(log n)

obtenerLibrosOrdenados(): ListaEnlazada*
Pre: Ninguna
Post: Retorna lista con libros en orden alfabetico
Complejidad: O(n)

obtenerAltura(): Entero
Pre: Ninguna
Post: Retorna altura del arbol
Complejidad: O(1)

estaVacio(): Boolean
Pre: Ninguna
Post: Retorna true si raiz = NULL
Complejidad: O(1)

Invariantes:
- Para todo nodo n: |altura(n.izq) - altura(n.der)| <= 1
- Para todo nodo n: n.izq.titulo < n.titulo < n.der.titulo
- altura >= -1
- tamaño >= 0

Fin TAD

4. Arbol B

TAD ArbolB

Datos:
raiz: NodoB*
grado: Entero (orden minimo)
tamaño: Entero

Operaciones:
ArbolB(g: Entero)
Pre: g >= 2
Post: Crea arbol B con grado = g, raiz = NULL, tamaño = 0

insertar(libro: Libro*): void
Pre: libro != NULL
Post: Inserta libro usando anio como clave, tamaño++
Complejidad: O(log n)

buscarPorAño(anio: Entero): Libro*
Pre: Ninguna
Post: Retorna libro con anio dado o NULL
Complejidad: O(log n)

buscarPorRangoFechas(inicio: Entero, fin: Entero): ListaEnlazada*
Pre: inicio <= fin
Post: Retorna libros con inicio <= anio <= fin
Complejidad: O(log n + k) donde k = cantidad de resultados

eliminar(anio: Entero): Boolean
Pre: Ninguna
Post: Elimina libro manteniendo propiedades de Arbol B
Complejidad: O(log n)

Invariantes:
- Todos los nodos hoja estan al mismo nivel
- Nodo interno con k claves tiene k+1 hijos
- Nodo raiz: 1 <= claves <= 2*grado-1
- Nodo no-raiz: grado-1 <= claves <= 2*grado-1
- Claves ordenadas en cada nodo

Fin TAD

5. Arbol B+

TAD ArbolBPlus

Datos:
raiz: NodoBPlus*
grado: Entero
tamaño: Entero
primeraHoja: NodoBPlus*

Operaciones:

ArbolBPlus(g: Entero)
Pre: g >= 2
Post: Crea arbol B+ con grado = g, raiz = NULL

insertar(libro: Libro*): void
Pre: libro != NULL
Post: Inserta libro usando genero como clave, tamaño++
Complejidad: O(log n)

buscarPorGenero(genero: String): ListaEnlazada*
Pre: Ninguna
Post: Retorna todos los libros del genero dado
Complejidad: O(log n + m) donde m = libros del genero

eliminar(genero: String, isbn: String): Boolean
Pre: Ninguna
Post: Elimina libro del genero manteniendo propiedades
Complejidad: O(log n)

recorridoSecuencial(): void
Pre: Ninguna
Post: Muestra todos los generos usando enlaces entre hojas
Complejidad: O(k) donde k = cantidad de generos

Invariantes:
- Datos solo en nodos hoja
- Nodos internos solo contienen claves guia
- Todas las hojas estan al mismo nivel
- Hojas enlazadas secuencialmente
- primeraHoja apunta a la hoja mas izquierda

Fin TAD

Complejidades:

Complejidad Temporal Lista enlazada:

Operacion	Mejor Caso	Caso Promedio	Peor Caso
Insertar al inicio	O(1)	O(1)	O(1)
Insertar al final	O(n)	O(n)	O(n)
Buscar	O(1)	O(n/2)	O(n)
Eliminar	O(1)	O(n/2)	O(n)
Acceso por indice	O(1)	O(n/2)	O(n)

Ventajas:

- Insercion al inicio muy rapida
- Tamaño dinamico
- No requiere memoria contigua

Desventajas:

- Busqueda secuencial lenta
- No permite acceso directo por indice
- Overhead de punteros

Complejidad Temporal Arbol AVL:

Operacion	Mejor Caso	Caso Promedio	Peor Caso
Insertar	O(log n)	O(log n)	O(log n)
Buscar	O(1)	O(log n)	O(log n)
Eliminar	O(log n)	O(log n)	O(log n)
Recorrido In-order	O(n)	O(n)	O(n)
Obtener altura	O(1)	O(1)	O(1)

Ventajas:

- Garantiza O(log n) para busqueda
- Auto-balanceo
- Recorrido in-order da elementos ordenados

Desventajas:

- Overhead de almacenar altura
- Rotaciones frecuentes
- Mas complejo que BST simple

Complejidad Temporal Arbol B:

Operacion	Mejor Caso	Caso Promedio	Peor Caso
Insertar	O(log n)	O(log n)	O(log n)
Buscar	O(1)	O(log n)	O(log n)
Buscar Rango	O(log n + k)	O(log n + k)	O(log n + k)
Eliminar	O(log n)	O(log n)	O(log n)
Division de nodo	O(1)	O(1)	O(1)

Complejidad Espacial: O(n)

Ventajas:

- Optimizado para busquedas por rango
- Menos profundidad que arbol binario
- Ideal para sistemas con disco
- Multiples claves por nodo reducen accesos

Desventajas:

- Mas complejo de implementar
- Overhead de mantener propiedades
- Operaciones de fusion complejas

Complejidad Temporal Arbol B+:

Operacion	Mejor Caso	Caso Promedio	Peor Caso
Insertar	O(log n)	O(log n)	O(log n)
Buscar por clave	O(1)	O(log n)	O(log n)
Buscar todos de categoria	O(log n + m)	O(log n + m)	O(log n + m)
Eliminar	O(log n)	O(log n)	O(log n)
Recorrido secuencial	O(k)	O(k)	O(k)

donde m = libros en la categoria, k = numero de categorias

Ventajas:

- Recorrido secuencial muy rapido
- Busquedas por categoria eficientes
- Hojas enlazadas para rangos
- Nodos internos optimizados (solo claves)

Desventajas:

- Duplicacion de claves (espacio extra)
- Insercion mas compleja
- Mantenimiento de enlaces adicionales