Reporte Semana 7: Árboles Binarios de Búsqueda

Estructuras de Datos Jerárquicas

📋 Resumen Ejecutivo

Semana: 7
Tema: Árboles Binarios de Búsqueda (BST)
Fecha: Diciembre 2025

🎯 Objetivos Cumplidos

- ☑ Implementar Árbol Binario de Búsqueda (BST)
- ☑ Operaciones CRUD (Inserción, Búsqueda, Eliminación)
- ☑ Recorridos (In-Order, Pre-Order, Post-Order)
- ☑ Cálculo de altura del árbol
- ☑ Análisis de complejidad

💻 Implementación

Árbol Binario de Búsqueda

Propiedades:

- Cada nodo tiene máximo 2 hijos
- Subárbol izquierdo: valores menores
- Subárbol derecho: valores mayores
- Permite búsqueda eficiente O(log n)

Operaciones Implementadas:

1. Inserción
2. Búsqueda
3. Eliminación (3 casos)
4. Recorridos (In-Order, Pre-Order, Post-Order)
5. Cálculo de altura

📊 Resultados de Ejecución

Árbol de Prueba

```
Valores insertados: 50, 30, 70, 20, 40, 60, 80

Estructura del árbol:
```

```
        50
       /  \
      30   70
     / \   / \
    20 40 60 80
```

Salida del Programa

```
🌳 Semana 7 - Árboles Binarios de Búsqueda

📥 Insertando elementos: 50, 30, 70, 20, 40, 60, 80

🔍 Recorridos del árbol:
In-Order (ordenado):   20, 30, 40, 50, 60, 70, 80
Pre-Order (raíz primero): 50, 30, 20, 40, 70, 60, 80
Post-Order (raíz último): 20, 40, 30, 60, 80, 70, 50

📏 Altura del árbol: 3

🔎 Búsquedas:
¿Existe 40? True
¿Existe 25? False
¿Existe 70? True

🗑️ Eliminando 30...
In-Order después de eliminar: 20, 40, 50, 60, 70, 80

🗑️ Eliminando 50 (raíz)...
In-Order después de eliminar: 20, 40, 60, 70, 80

📏 Nueva altura: 3

☑️ Programa completado!
```

📈 Análisis de Complejidad

Operaciones en BST

Operación	Mejor Caso	Caso Promedio	Peor Caso
Búsqueda	O(1)	O(log n)	O(n)
Inserción	O(1)	O(log n)	O(n)
Eliminación	O(1)	O(log n)	O(n)
Recorrido	O(n)	O(n)	O(n)
Altura	O(n)	O(n)	O(n)

Nota: El peor caso O(n) ocurre cuando el árbol está desbalanceado (se convierte en lista enlazada).

Complejidad Espacial

Aspecto	Complejidad
Almacenamiento	O(n)
Recursión	O(h) donde h = altura
Mejor caso	O(log n)
Peor caso	O(n)

🔍 Operaciones Detalladas

1. Inserción

Algoritmo:

```
1. Si el árbol está vacío, crear raíz
2. Si valor < nodo actual, ir a izquierda
3. Si valor > nodo actual, ir a derecha
4. Repetir hasta encontrar posición vacía
```

Ejemplo:

```
Insertar 35 en árbol:
50 → 30 (35 > 30) → 40 (35 < 40) → Insertar a la izquierda de 40
```

2. Búsqueda

Algoritmo:

```
1. Si nodo es null, no existe
2. Si valor == nodo, encontrado
3. Si valor < nodo, buscar en izquierda
4. Si valor > nodo, buscar en derecha
```

Complejidad: O(log n) promedio, O(n) peor caso

3. Eliminación (3 Casos)

Caso 1: Nodo sin hijos (hoja)

```
Simplemente eliminar el nodo
```

Caso 2: Nodo con un hijo

```
Reemplazar nodo con su único hijo
```

Caso 3: Nodo con dos hijos

```
1. Encontrar sucesor in-order (mínimo del subárbol derecho)
2. Reemplazar valor del nodo con el sucesor
3. Eliminar el sucesor
```

Ejemplo:

```
Eliminar 30 (tiene dos hijos):
1. Sucesor = 40 (mínimo de subárbol derecho)
2. Reemplazar 30 con 40
3. Eliminar 40 original
```

🎓 Recorridos del Árbol

1. In-Order (Izquierda-Raíz-Derecha)

```
Resultado: 20, 30, 40, 50, 60, 70, 80
Uso: Obtener elementos ordenados
```

2. Pre-Order (Raíz-Izquierda-Derecha)

```
Resultado: 50, 30, 20, 40, 70, 60, 80
Uso: Copiar árbol, expresiones prefijas
```

3. Post-Order (Izquierda-Derecha-Raíz)

```
Resultado: 20, 40, 30, 60, 80, 70, 50
Uso: Eliminar árbol, expresiones postfijas
```

💡 Conceptos Clave Aprendidos

1. Propiedad BST

- Subárbol izquierdo < Raíz < Subárbol derecho
- Permite búsqueda binaria
- Recorrido in-order produce secuencia ordenada

2. Balanceo

- Árbol balanceado: altura = O(log n)
- Árbol desbalanceado: altura = O(n)
- Solución: AVL, Red-Black Trees

3. Recursión

- Todas las operaciones son naturalmente recursivas
- Caso base: nodo null
- Caso recursivo: procesar subárboles

4. Aplicaciones

- Bases de datos (índices)
- Sistemas de archivos
- Compiladores (árboles de sintaxis)
- Diccionarios y sets

Casos de Prueba

Caso 1: Inserción Ordenada

```
Entrada: 10, 20, 30, 40, 50
Resultado: Árbol desbalanceado (lista enlazada)
Altura: 5
Complejidad: O(n) - ¡Peor caso!
```

Caso 2: Inserción Balanceada

```
Entrada: 50, 30, 70, 20, 40, 60, 80
Resultado: Árbol balanceado
Altura: 3
Complejidad: O(log n) - ¡Óptimo!
```

Caso 3: Eliminación de Raíz

```
Entrada: Eliminar 50
Proceso: Reemplazar con 60 (sucesor)
```

```
Resultado: Árbol válido
Verificación: ☑ Propiedad BST mantenida
```

🎯 Conclusiones

Logros

1. ☑ Implementación completa de BST
2. ☑ Todas las operaciones CRUD funcionando
3. ☑ Tres tipos de recorridos implementados
4. ☑ Manejo correcto de los 3 casos de eliminación

Ventajas de BST

- Búsqueda eficiente O(log n) en promedio
- Inserción y eliminación eficientes
- Recorrido in-order produce secuencia ordenada
- Estructura simple y elegante

Limitaciones

- Puede desbalancearse (peor caso O(n))
- Requiere balanceo para garantizar eficiencia
- No es óptimo para datos ordenados

Mejoras Posibles

- Implementar AVL (auto-balanceo)
- Implementar Red-Black Tree
- Agregar operaciones adicionales (min, max, rango)

📚 Aplicaciones Prácticas

1. Bases de Datos

- Índices B-Tree (variante de BST)
- Búsqueda rápida de registros
- Ordenamiento eficiente

2. Compiladores

- Árboles de sintaxis abstracta (AST)
- Tablas de símbolos
- Análisis de expresiones

3. Sistemas Operativos

- Gestión de memoria

- Planificación de procesos
- Sistemas de archivos

4. Aplicaciones

- Diccionarios
- Sets (conjuntos)
- Mapas (clave-valor)

📖 Referencias

- Cormen, T. H., et al. (2009). *Introduction to Algorithms* (3rd ed.)
- Knuth, D. E. (1998). *The Art of Computer Programming, Vol. 3*
- Material del curso - Semana 7

Fecha: Diciembre 2025
Curso: Estructuras de Datos Avanzadas
Semana: 7 - Árboles BST
Estado: ☑ Completado