Resumen de Rotaciones AVL y Balanceo

1. Causas y Tipos de Rotaciones

Se aplican rotaciones cuando un nodo tiene **factor de balance (FB) = ±2** (desbalanceado).

A. Rotación Simple Derecha (LL)

- **Causa**: FB(nodo) = +2 y FB(hijo izquierdo) ≥ 0.
- **Qué pasa**:
  - El hijo izquierdo del nodo desbalanceado se convierte en la nueva raíz del subárbol.
  - El nodo desbalanceado pasa a ser hijo derecho de la nueva raíz.
  - El subárbol derecho del hijo izquierdo se mueve como izquierdo del nodo desbalanceado.

B. Rotación Simple Izquierda (RR)

- **Causa**: FB(nodo) = -2 y FB(hijo derecho) ≤ 0.
- **Qué pasa**:
  - El hijo derecho del nodo desbalanceado se convierte en la nueva raíz del subárbol.
  - El nodo desbalanceado pasa a ser hijo izquierdo de la nueva raíz.
  - El subárbol izquierdo del hijo derecho se mueve como derecho del nodo desbalanceado.

C. Rotación Doble Izquierda-Derecha (LR)

- **Causa**: FB(nodo) = +2 y FB(hijo izquierdo) = -1.
- **Qué pasa**:
  1. **Rotación izquierda** en el hijo izquierdo (convierte el caso en LL).
  2. **Rotación derecha** en el nodo desbalanceado.

D. Rotación Doble Derecha-Izquierda (RL)

- **Causa**: FB(nodo) = -2 y FB(hijo derecho) = +1.
- **Qué pasa**:
  1. **Rotación derecha** en el hijo derecho (convierte el caso en RR).
  2. **Rotación izquierda** en el nodo desbalanceado.

---

2. Proceso de Balanceo

El balanceo ocurre después de **inserciones** o **eliminaciones**:

A. Durante Inserción

1. Se inserta el nodo como en un BST normal.
2. Se actualizan las alturas desde el nodo insertado hacia la raíz.
3. Si algún nodo tiene FB = ±2, se aplica la rotación correspondiente.

B. Durante Eliminación

1. Se elimina el nodo como en un BST.
2. Se actualizan alturas desde el nodo eliminado hacia arriba.
3. Si algún nodo queda con FB = ±2, se rebalancea con rotaciones.

---

3. Funcionamiento del Balanceo

- **Objetivo**: Mantener **|FB| ≤ 1** en todos los nodos.
- **Efecto**:
  - Las rotaciones **reducen la altura** del árbol.
  - Garantizan que el árbol **siempre esté equilibrado**, manteniendo operaciones en **O(log n)**.
- **Propagación**: El balanceo puede requerir **múltiples rotaciones** desde el nodo modificado hasta la raíz.

Conclusión

Las rotaciones **LL, RR, LR y RL** corrigen desbalances locales, mientras que el balanceo **propaga** estos ajustes hacia arriba, asegurando que el árbol AVL **siempre mantenga su estructura equilibrada**.

Explicación de las Operaciones de Rotación en el Código AVL

El código implementa las 4 rotaciones fundamentales del árbol AVL. A continuación, se muestra **exactamente dónde y cómo se ejecuta cada una**:

1. Rotación Simple Derecha (LL)

Ubicación en código:

```
if (balance > 1 && valor < nodo->izquierda->valor) {
    return rotacionDerecha(nodo);  // <-- LL
}
```

Flujo de ejecución:

1. Se detecta FB = +2 en un nodo (`balance > 1`)
2. El hijo izquierdo tiene FB ≥ 0 (`valor < nodo->izquierda->valor`)
3. Se llama a `rotacionDerecha(nodo)`

Qué hace `rotacionDerecha`:

```
pNodoAVL x = y->izquierda;  // Hijo izquierdo se convierte en nueva raíz
pNodoAVL T2 = x->derecha;   // Subárbol derecho del hijo izquierdo

x->derecha = y;  // El nodo desbalanceado pasa a ser hijo derecho
y->izquierda = T2;  // T2 se mueve como izquierdo del nodo desbalanceado
```

2. Rotación Simple Izquierda (RR)

Ubicación en código:

```
if (balance < -1 && valor > nodo->derecha->valor) {
    return rotacionIzquierda(nodo);  // <-- RR
}
```

Flujo de ejecución:

1. FB = -2 en un nodo (`balance < -1`)
2. Hijo derecho tiene FB ≤ 0 (`valor > nodo->derecha->valor`)
3. Se llama a `rotacionIzquierda(nodo)`

Qué hace `rotacionIzquierda`:

```
pNodoAVL y = x->derecha;  // Hijo derecho se convierte en nueva raíz
pNodoAVL T2 = y->izquierda;  // Subárbol izquierdo del hijo derecho

y->izquierda = x;  // El nodo desbalanceado pasa a ser hijo izquierdo
x->derecha = T2;   // T2 se mueve como derecho del nodo desbalanceado
```

3. Rotación Doble Izquierda-Derecha (LR)

Ubicación en código:

```
if (balance > 1 && valor > nodo->izquierda->valor) {
    nodo->izquierda = rotacionIzquierda(nodo->izquierda);  // Paso 1
    return rotacionDerecha(nodo);  // Paso 2  <-- LR
}
```

Flujo de ejecución:

1. FB = +2 en un nodo (`balance > 1`)
2. Hijo izquierdo tiene FB = -1 (`valor > nodo->izquierda->valor`)
3. **Paso 1**: Rotación izquierda en el hijo izquierdo
4. **Paso 2**: Rotación derecha en el nodo desbalanceado

4. Rotación Doble Derecha-Izquierda (RL)

Ubicación en código:

```
if (balance < -1 && valor < nodo->derecha->valor) {
    nodo->derecha = rotacionDerecha(nodo->derecha);  // Paso 1
```

```
    return rotacionIzquierda(nodo);  // Paso 2  <-- RL
}
```

Flujo de ejecución:

1. FB = -2 en un nodo (`balance < -1`)
2. Hijo derecho tiene FB = +1 (`valor < nodo->derecha->valor`)
3. **Paso 1**: Rotación derecha en el hijo derecho
4. **Paso 2**: Rotación izquierda en el nodo desbalanceado

Proceso de Balanceo Durante Inserción

1. **Inserción BST normal**:

```
if (valor < nodo->valor) {
    nodo->izquierda = insertar(nodo->izquierda, valor);
} else if (valor > nodo->valor) {
    nodo->derecha = insertar(nodo->derecha, valor);
}
```

2. **Actualización de alturas**:

```
nodo->altura = 1 + max(obtenerAltura(nodo->izquierda), obtenerAltura(nodo-
>derecha));
```

3. **Verificación de balance**:

```
int balance = factorBalance(nodo);  // altura(izq) - altura(der)
```

4. **Aplicación de rotaciones** (como se explicó arriba).

Proceso de Balanceo Durante Eliminación

1. **Eliminación BST estándar** (3 casos):
    - Nodo sin hijos.
    - Nodo con un hijo.
    - Nodo con dos hijos (usa sucesor inorden).
2. **Actualización de alturas** (igual que en inserción).
3. **Verificación de balance**:

```
int balance = factorBalance(nodo);
```

4. **Rotaciones** (idénticas a las de inserción, pero con condiciones adicionales):

```
if (balance > 1 && factorBalance(nodo->izquierda) >= 0) { ... }  // LL
if (balance > 1 && factorBalance(nodo->izquierda) < 0) { ... }   // LR
// ... (análogo para RR y RL)
```

Conclusión

- Cada rotación **corrige un desbalance específico** detectado por el `factorBalance`.
- El balanceo es **propagado recursivamente** desde el nodo modificado hasta la raíz.
- Las condiciones `if` en inserción/eliminación **determinan qué rotación aplicar**.
- Las funciones `rotacionDerecha`/`rotacionIzquierda` **reestructuran el árbol** manteniendo el orden BST.