Protokoll – Algorithmen & Datenstrukturen – Programmieraufgabe 2

Rekursives Einfügen in den Binärbaum:

Die Methode insert ist in zwei Versionen vorhanden. Die nicht-rekursive Variante wird ausschließlich beim ersten Aufruf genutzt, um die Wurzel (root) des Baumes zu initialisieren, falls diese noch leer ist. Sobald ein Wurzelknoten vorhanden ist, wird automatisch die rekursive Variante verwendet, die das Einfügen eines Wertes übernimmt.

Die rekursive insert-Funktion prüft dabei, ob der einzufügende Wert größer oder kleiner als der aktuelle Knotenwert ist. Abhängig vom Vergleich wird entweder der linke oder rechte Teilbaum weiter betrachtet. Ist die entsprechende Stelle frei (nullptr), wird der neue Wert dort eingefügt. Ist der Platz belegt, erfolgt ein weiterer rekursiver Aufruf mit dem nächsten Knoten.

Laufzeitanalyse:

- Best Case: **O(1)**
- Average/Worst Case: **O(log(n))** bei balanciertem Baum

Höhenberechnung:

Die Methode height berechnet die Höhe eines Teilbaumes unterhalb eines bestimmten Knotens. Diese Information wird für die spätere Balance-Faktor-Berechnung im Rahmen der AVL-Prüfung benötigt.

Struktur:

- **Abbruchbedingung:** if (!node) return 0; – ein leerer Teilbaum hat die Höhe 0
- **Parameter:** Zeiger auf den aktuellen Knoten (Node*)
- **Rückgabewert:** Ganzzahliger Wert der Höhe
- **Laufzeit:** Worst Case **O(n)** – bei vollständig einseitigem Baum werden alle Knoten durchlaufen

---

AVL-Bedingung prüfen:

Die Funktion checkAVL durchläuft den kompletten Baum rekursiv und berechnet für jeden Knoten den Balance-Faktor nach folgender Formel:

bal(k) = Höhe(rechter Teilbaum) - Höhe(linker Teilbaum)

Ist der Betrag dieses Wertes größer als 1, so liegt eine Verletzung der AVL-Bedingung vor. In diesem Fall wird eine Ausgabe auf der Konsole erzeugt und der boolesche Referenzwert isAVL entsprechend gesetzt.

Struktur:

- **Abbruchbedingung:** if (!node) return;
- **Parameter:** Knotenzeiger (Node*) und Referenz auf bool isAVL
- **Rückgabewert:** keine (nur Veränderung über Referenz & Konsole)
- **Laufzeit:** Worst Case **$O(n^2)$** – da für jeden Knoten height() erneut vollständig aufgerufen wird

---

Berechnung von Statistikdaten (Minimum, Maximum, Summe, Durchschnitt):

Die Methode getStats durchquert den gesamten Baum in Inorder-Reihenfolge und sammelt dabei Informationen über:

- den kleinsten Schlüssel
- den größten Schlüssel
- die Summe aller Werte
- die Anzahl der Knoten

Auf Basis dieser Daten lässt sich am Ende der Durchschnitt berechnen.

Struktur:

- **Abbruchbedingung:** if (!node) return;
- **Parameter:** Knotenzeiger sowie Referenzen auf minVal, maxVal, sum, count
- **Rückgabewert:** keine (Verarbeitung über Referenzen)
- **Laufzeit: O(n)** – jeder Knoten wird genau einmal besucht

---

Rekursive Einzelsuche:

Die Methode simpleSearch durchläuft den Baum rekursiv und vergleicht den gesuchten Wert mit dem aktuellen Knoten. Wird der Wert gefunden, wird der Pfad über die Konsole ausgegeben. Ist der gesuchte Wert nicht vorhanden, wird dies ebenfalls ausgegeben.

Struktur:

- **Abbruch:** entweder bei Fund oder bei leerem Teilbaum (nullptr)
- **Entscheidung:** Links oder rechts weiter je nach Vergleich des Wertes
- **Laufzeit:**
    - Best Case: **O(1)** (direkter Treffer bei Root)
    - Average/Worst Case: **O(log(n))** bei balanciertem Baum

---

Rekursive Teilbaumsuche:

Zur Überprüfung, ob ein Teilbaum im Hauptbaum enthalten ist, wird zunächst mittels simpleSearch nach dem Wurzelwert des Subtrees gesucht. Wird dieser nicht gefunden, endet die Suche sofort.

Ist der Wurzelknoten vorhanden, folgt ein struktureller Vergleich der beiden Bäume mit der Methode subTreeSearch. Dabei werden die linke und rechte Seite jedes Knotens rekursiv miteinander verglichen.

Struktur:

- Suche erfolgt zuerst mit einfacher Suche, dann rekursiver Vergleich
- **Laufzeit:**
    - Best Case: **O(1)** – z. B. bei leerem Baum
    - Average/Worst Case: **O(log(n))**