Beschreibung der rekursiven Funktionen

1. REKURSIVE FUNKTIONEN ZUR BERECHNUNG STATISTISCHER DATEN

Diese Funktionen berechnen den minimalen, maximalen und durchschnittlichen Schlüsselwert im Baum.

findMinimum und findMaximum:
Diese Funktionen finden rekursiv den minimalen bzw. maximalen Schlüsselwert im Baum.

Rekursiver Aufruf:
Wenn der linke (bzw. rechte) Knoten eines Knotens nicht `nullptr` ist, rufen sie sich selbst mit diesem Knoten als Argument auf.

sumAndCount:
Diese Funktion durchläuft den Baum rekursiv und summiert alle Schlüsselwerte, während sie die Anzahl der Knoten zählt.

Rekursiver Aufruf:
Ruft sich selbst für den linken und rechten Kindknoten auf, um die Summe und Anzahl der Knoten zu aggregieren.

2. REKURSIVE FUNKTION ZUR PRÜFUNG DER AVL-BEDINGUNGEN

Diese Funktionen prüfen, ob der Baum die Balance-Bedingungen eines AVL-Baums erfüllt.

printBalanceAndHeight:
Diese Funktion prüft rekursiv die Balance jedes Knotens und gibt den Balance-Faktor aus.

Rekursiver Aufruf:
Ruft sich selbst für den linken und rechten Kindknoten auf und berechnet anschließend den Balance-Faktor des aktuellen Knotens.

printInOrder (AVLTreeNode* node):
durchläuft den Baum rekursiv in In-Order-Reihenfolge und gibt die Schlüssel aus. Dies wird oft verwendet, um die Elemente des Baums in aufsteigender Reihenfolge anzuzeigen, da es zuerst das linke Kind besucht, dann den Knoten selbst und schließlich das rechte Kind.

Rekursiver Aufruf:
Diese Funktion führt zwei rekursive Aufrufe durch: einen zum linken Kind und einen zum rechten Kind des aktuellen Knotens, um sicherzustellen, dass der Baum in der richtigen Reihenfolge durchlaufen wird.

3. REKURSIVE FUNKTIONEN ZUR SUBTREE-SUCHE

Diese Funktionen überprüfen, ob ein bestimmter Subtree in einem Baum existiert.

areIdentical:
Überprüft, ob zwei Bäume (oder Subbäume) identisch sind, indem sie rekursiv jeden Knoten vergleichen.

Rekursiver Aufruf:
Vergleicht die aktuellen Knoten und ruft sich dann rekursiv für die linken und rechten Kindknoten auf.

isSubtreeHelper:
Bestimmt, ob ein Subtree in einem anderen Baum existiert, indem es rekursiv den Hauptbaum und den Subtree vergleicht.

Rekursiver Aufruf:

Ruft `areIdentical` für die aktuellen Knoten auf und prüft dann rekursiv, ob der Subtree in den linken oder rechten Unterbäumen des Hauptbaums vorkommt.

Laufzeitkomplexität

findMinimum` und findMaximum - Laufzeitkomplexität: O(h)

Diese Funktionen besuchen rekursiv den jeweils tiefsten Knoten des linken bzw. rechten Unterbaums, wobei `h` die Höhe des Baumes ist. Die Laufzeit hängt von der Tiefe des Baumes ab, wobei im schlimmsten Fall (bei einem degenerierten Baum) `h = n` (mit `n` als Anzahl der Knoten) sein kann.

sumAndCount - Laufzeitkomplexität: O(n)

Die Funktion durchläuft jeden Knoten des Baums genau einmal, um die Summe aller Schlüsselwerte zu berechnen und die Anzahl der Knoten zu zählen. Dabei ist `n` die Gesamtzahl der Knoten im Baum.

printBalanceAndHeight - Laufzeitkomplexität: O(n)

Diese Funktion muss jeden Knoten des Baums besuchen, um den Balance-Faktor zu berechnen und die Höhe jedes Unterbaums zu bestimmen. Da jeder Knoten genau einmal besucht wird, ist die Laufzeit proportional zur Anzahl der Knoten im Baum.

areIdentical - Laufzeitkomplexität: O(m)

Diese Funktion vergleicht zwei Bäume Knoten für Knoten. Dabei ist `m` die Anzahl der Knoten im kleineren Baum oder Subtree. Im schlimmsten Fall müssen alle Knoten des Subtrees besucht werden.

isSubtreeHelper - Laufzeitkomplexität: O(n * m)

Für jeden Knoten im Hauptbaum (`n` Knoten) wird potenziell die `areIdentical`-Funktion aufgerufen, die bis zu `m` Operationen (die Größe des Subtrees) durchführen kann. In jedem Schritt des Hauptbaums muss möglicherweise ein vollständiger Vergleich des Subtrees durchgeführt werden, was zu einer Laufzeit von O(n * m) führt.