Beschreibung der rekursiven Funktionen:

FindMin():

Funktion: Findet den minimalen Schlüsselwert im Baum.

Rekursion: Die Rekursion erfolgt, indem die Funktion sich selbst mit dem linken Kindknoten als Argument aufruft, bis ein Blatt erreicht wird.

Basisfall: Wenn der linke Kindknoten null ist, wird der Schlüsselwert des aktuellen Knotens zurückgegeben.

FindMax():

Funktion: Findet den maximalen Schlüsselwert im Baum.

Rekursion: Die Rekursion erfolgt, indem die Funktion sich selbst mit dem rechten Kindknoten als Argument aufruft, bis ein Blatt erreicht wird.

Basisfall: Wenn der rechte Kindknoten null ist, wird der Schlüsselwert des aktuellen Knotens zurückgegeben.

FindAverage():

Funktion: Berechnet den Durchschnittswert der Schlüssel im Baum.

Rekursion: Die Rekursion erfolgt, indem die Funktion die Teilsummen und die Anzahl der Schlüssel für die linken und rechten Teilbäume berechnet und diese zusammen mit dem Wert des aktuellen Knotens kombiniert.

Search():

Funktion: Sucht nach einem Schlüsselwert im Baum.

Rekursion: Die Rekursion erfolgt, indem die Funktion sich selbst mit dem entsprechenden linken oder rechten Kindknoten als Argument aufruft, basierend darauf, ob der gesuchte Schlüssel kleiner oder größer als der aktuelle Knotenschlüssel ist.

Basisfall: Wenn der aktuelle Knoten null ist oder der gesuchte Schlüssel dem Schlüssel des aktuellen Knotens entspricht, wird der aktuelle Knoten zurückgegeben.

IsSubtree():

Funktion: Überprüft, ob ein gegebener Teilbaum innerhalb eines anderen Baumes vorhanden ist.

Rekursion: Die Rekursion erfolgt, indem die Funktion sich selbst mit den linken und rechten Kindknoten des Hauptbaums aufruft, um zu überprüfen, ob der Teilbaum in einem der Teilbäume gefunden werden kann.

Basisfall: Wenn der Teilbaum null ist, wird true zurückgegeben; wenn der Hauptbaum null ist, wird false zurückgegeben.

O-Notation und Laufzeitvergleich:

FindMin() und FindMax():

Zeitkomplexität: O(h), wobei h die Höhe des Baumes ist. Im schlimmsten Fall (ungleichmäßiger Baum) kann die Laufzeit O(n) betragen, wobei n die Anzahl der Knoten im Baum ist.

Raumkomplexität: O(h) für die Rekursionstiefe.

FindAverage():

Zeitkomplexität: O(n), da jeder Knoten im Baum genau einmal besucht wird.

Raumkomplexität: O(h) für die Rekursionstiefe.

Search():

Zeitkomplexität: O(h), wobei h die Höhe des Baumes ist. Im schlimmsten Fall (ungleichmäßiger Baum) kann die Laufzeit O(n) betragen, wobei n die Anzahl der Knoten im Baum ist.

Raumkomplexität: O(h) für die Rekursionstiefe.

IsSubtree():

Zeitkomplexität: O(m \* n), wobei m die Anzahl der Knoten im Teilbaum und n die Anzahl der Knoten im Hauptbaum ist. Im schlimmsten Fall kann es zu einer Laufzeit von O(n^2) kommen, wenn der Teilbaum und der Hauptbaum sehr ähnlich sind.

Raumkomplexität: O(h1 + h2), wobei h1 die Höhe des Hauptbaums und h2 die Höhe des Teilbaums ist. Im schlimmsten Fall kann dies O(n) betragen.

Zusammenfassend sind die rekursiven Funktionen FindMin(), FindMax() und Search() in Bezug auf die Zeitkomplexität effizienter, wenn der Baum relativ ausgewogen ist. Die Funktionen FindAverage() und IsSubtree() haben eine höhere Zeitkomplexität, da sie mehr Knoten im Baum besuchen müssen. In Bezug auf den Speicherplatz haben alle Funktionen eine Raumkomplexität von O(h) oder besser, was bedeutet, dass sie in Bezug auf den Speicherplatz effizient sind, insbesondere wenn der Baum ausgewogen ist.