Aufgabe 02 - Sortieren und Bäume

Sortieren mittels Quicksort

Implementieren Sie den Divide-And-Conquer Sortieralgorithmus *Quicksort* auf rekursive Art. Dabei wird ein Element, das *Pivot-Element*, verwendet, um die noch nicht sortierte Liste in zwei Teile aufzuteilen - ein Teil in dem die Elemente kleiner als das Pivot-Element und ein Teil in dem die Elemente größer als das Pivot-Element sind. Diese Teillisten werden dann anschließend in den rekursiven Aufrufen erneut geteilt, bis nur noch ein Element übrig ist (oder der Teil der Liste bereits sortiert ist). Das Prinzip ist in Abbildung 1 dargestellt. Anschließend werden die sortierten Ergebnisse zusammengefügt und als Ergebnis die gesamte, sortierte Liste zurückgegeben.

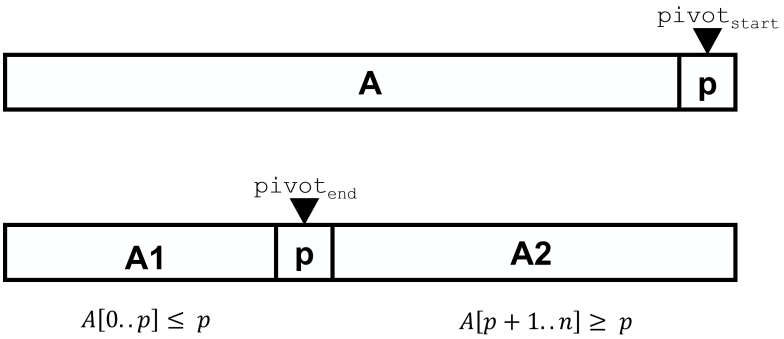

Abbildung 1: Divide-And-Conquer sortierverfahren Quicksort

Die Position des Pivot-Elements kann die Durchlaufzeiten des Algorithmus beeinflussen. Vergleichen Sie drei mögliche Varianten - Pivot-Element am *Anfang* der Liste, Pivot-Element in der *Mitte* der Liste und Pivot-Element am *Ende* der Liste. Abbildung 2 zeigt ein Beispiel für die Positionierung des Pivot-Elements in der Mitte der Liste. Sie sollten für dieselben Folgen von Zahlen immer dasselbe (korrekte) Ergebnis (sortierte Liste) nach Ausführung erhalten, aber die dafür notwendigen Schritte sollten sich unterscheiden bzw. können einmal mehr und einmal weniger Schritte notwendig sein.

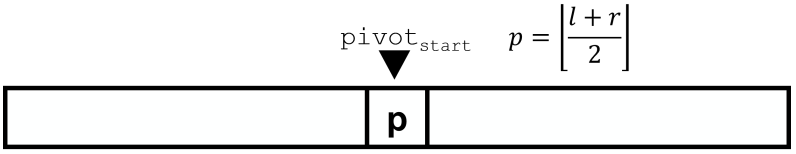

Abbildung 2: Quicksort mit Pivot-Element in der Mitte der Liste

Für Ihre Implementierung können Sie Arrays, Listen… verwenden. Testen Sie Ihre Algorithmen mit verschiedenen Inputs.

Binärer Suchbaum

Implementieren Sie einen *binären Suchbaum* in Java (oder in einer anderen objekt-orientierten Sprache).In einem binären Suchbaum werden die Werte geordnet abgelegt. Dabei ist immer der Vaterknoten größer als alle Knoten seines linken Teilbaums und kleiner als alle Knoten seines rechten Teilbaums. Abbildung 3 zeigt den Ablauf während des Einfügens von Werten. Duplikate Einträge im Baum werden von unserer Implementierung nicht unterstützt. Implementieren Sie die Methoden *insert*, *search*, *minimum*, *maximum*, *predecessor*, *successor* und *remove*. Überlegen Sie sich wie Sie den Baum visualisieren können (in der Konsole genügt).

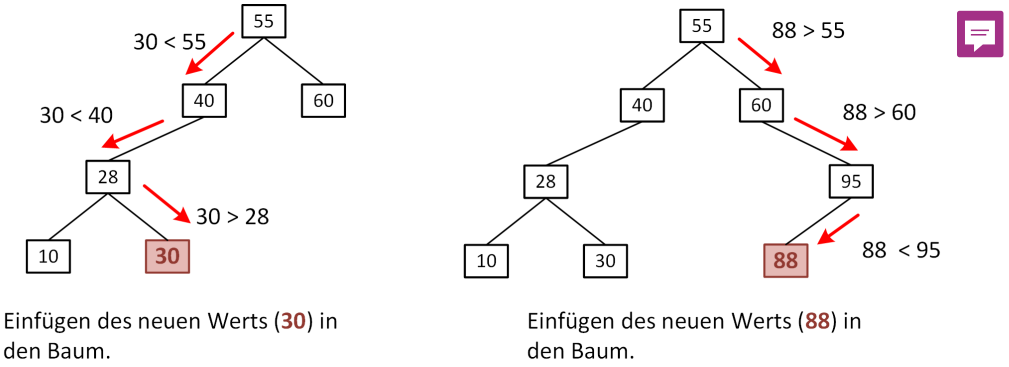

Abbildung 3: Einfügen in einen binären Suchbaum

Schaffen Sie eine Möglichkeit den Wert eines bestimmten Knotens zu aktualisieren. Dabei müssen Sie die Grenzen für eine zulässige Aktualisierung beachten und dürfen die Struktur des binären Suchbaumes nicht verletzen. Wenn Sie einen Wert aktualisieren darf er sich daher nur zwischen dem Predecessor und Successor befinden. Abbildung 4 veranschaulicht das Aktualisieren von Knoten.

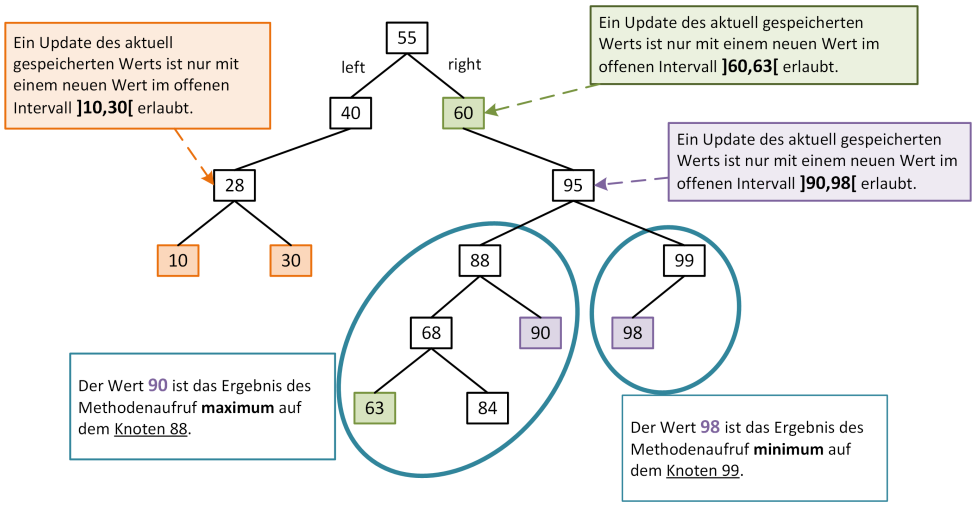

Abbildung 4: Aktualisieren in einen binären Suchbaum

Implementieren Sie eine Traversierung auf dem Baum welcher diesen in einer geordneten List ausgibt.

AVL-Baum

Erweitern Sie Ihre Implementierung auf einen AVL-Baum. Beim Einfügen müssen Sie in diesem Fall, falls notwendig, eine Rebalancierung durchführen. Implementieren Sie dazu das Einfügen als rekursive Funktion und prüfen Sie auf jeder Ebene, nachdem der Wert eingefügt wurde, ob die Bedingungen eines AVL-Baums verletzt sind. Dies ist der Fall wenn sich die Höhe des linken und rechten Teilbaums um mehr als 1 unterscheiden. Ist dies der Fall, führen Sie eine Rebalancierung durch. Dabei müssen Sie die zwei in der Vorlesung vorgestellten Fälle beachten und eine Rotation (einfach oder doppelt) durchführen.

Im einfacheren Fall muss eine einzelne Rotation nach links oder rechts durchgeführt werden. Abbildung 5 veranschaulicht diese Rotation.

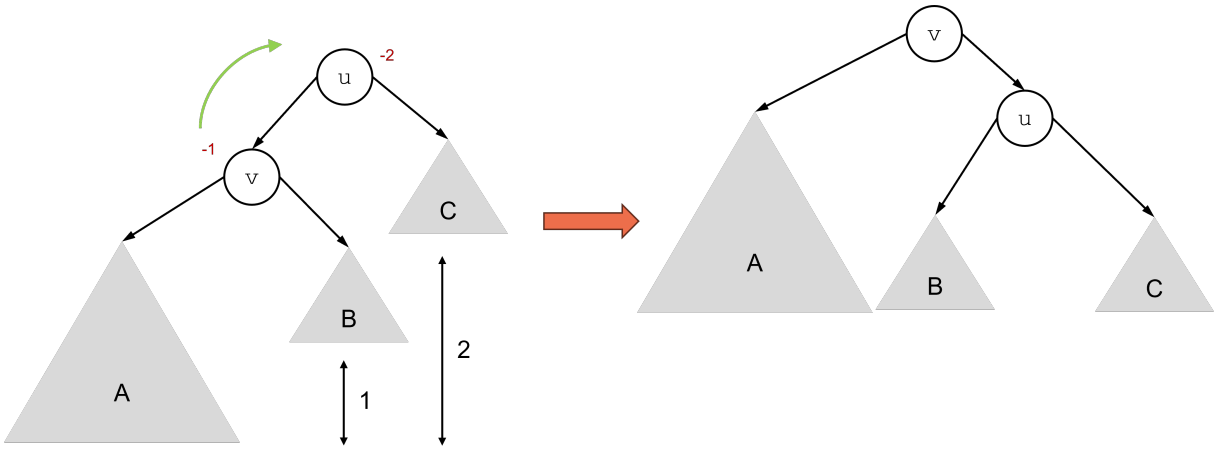

Abbildung 5: Einfache Rotation in einem AVL-Baum

Im anderen Fall muss eine doppelte Rotation (Links-Rechts oder Rechts-Links) durchgeführt werden. Abbildung 6 veranschaulicht diese Rotation.

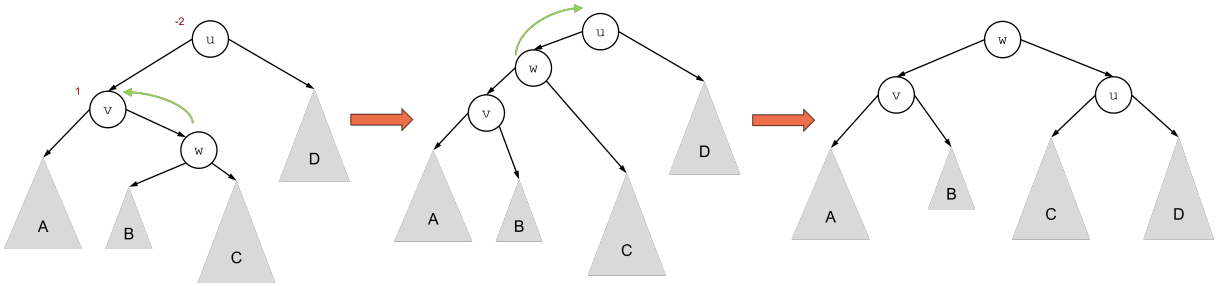

Abbildung 6: Doppelte Rotation in einem AVL-Baum

Optional: Implementieren Sie das Löschen eines Knotens in einem AVL-Baum.

Aufgabe

Start: 2024-04-02

Ende: 2024-05-11

Basierend auf den Beschreibungen oben:

1. Implementierung des Quicksort-Algorithmus
2. Implementierung eines binären Suchbaums (insert, search, remove, minimum, maximum, predecessor und successor)
3. Erweiterung des binären Suchbaums zu einem AVL-Baum (remove optional)

Die Abgabe erfolgt in Ilias bis 2024-05-11. Für eine Abgabe erhalten Sie *zwei Bonuspunkte*.