Aufgabe 1: AVL-Bäume

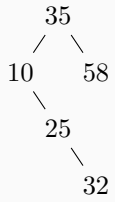
Fügen Sie die folgenden Zahlen nacheinander in einen *AVL-Baum* ein:

$$35 \quad 58 \quad 10 \quad 25 \quad 32 \quad 27$$

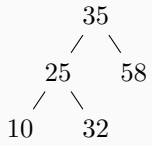
Zeichnen Sie den Baum vor und nach jeder durchgeführten Rotation. Geben Sie auch jeweils an, was für Rotationen Sie durchführen.

Lösung

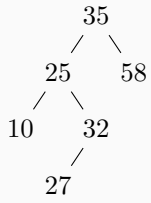
Einfügen von $35, 58, 10, 25, 32$



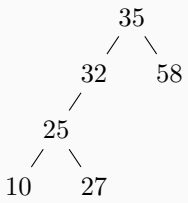
L-Rotation um 10



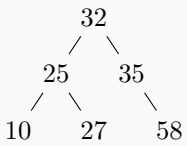
Einfügen von 27



LR-Rotation um 35 (1)



LR-Rotation um 35 (2)



Aufgabe 2: AVL-Bäume

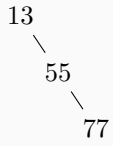
Fügen Sie die folgenden Zahlen nacheinander in einen *AVL-Baum* ein:

$$13 \quad 55 \quad 77$$

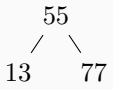
Zeichnen Sie den Baum vor und nach jeder durchgeführten Rotation. Geben Sie auch jeweils an, was für Rotationen Sie durchführen.

Lösung

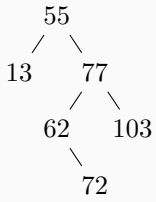
Einfügen von $13, 55, 77$



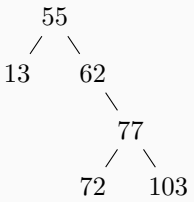
L-Rotation um 13



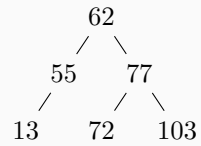
Einfügen von $62, 103, 72$



RL-Rotation um 55 (1)



RL-Rotation um 55 (2)



Aufgabe 3: Algorithmen (Entwurf)

Gegeben eine Liste von Zahlen, soll eine Liste von *Leadern* erstellt werden. Ein Element der Liste ist ein Leader, wenn es größer oder gleich allen Elementen rechts von ihm ist.
Bestimmen Sie auch die Laufzeit-Komplexität Ihrer Lösung.

Beispiele:

Eingabe: $[5, 3, 8, 2, 7, 4]$
Ausgabe: $[8, 7, 4]$
Eingabe: $[1, 2, 3, 4, 5]$
Ausgabe: $[5]$
Eingabe: $[5, 4, 3, 4, 1]$
Ausgabe: $[5, 4, 4, 1]$

Anmerkung: Ihre Lösung wird sowohl anhand der Korrektheit als auch der Laufzeitkomplexität bewertet.

Hinweis: Sie müssen keinen Code für den Algorithmus angeben, eine Erklärung der Idee genügt. Eine optimale Lösung sollte in $\mathcal{O}(n)$ Zeit laufen. Genauer ist es sogar möglich, die Liste mit einem einzigen Durchlauf zu sortieren.

Lösung

Lösung: Durchlauf mit elementweisem Vergleich
Eine mögliche Lösung ist, die Liste zu durchlaufen und dabei jedes Element mit allen seinen rechten Nachbarn zu vergleichen. Diese Lösung hätte eine Laufzeitkomplexität von $O(n^2)$, eine typische Implementierung würde eine geschachtelte Schleife verwenden.

Bessere Lösung: Durchlauf von rechts nach links
Es genügt, die Liste nur einmal von rechts nach links zu durchlaufen, und sich dabei den größten Wert zu merken, der bisher gefunden wurde. So kann die innere Schleife entfallen und die Laufzeitkomplexität auf $O(n)$ reduziert werden.

Aufgabe 4: Algorithmen (Entwurf)

Gegeben eine Liste, die $n-1$ *verschiedene* Zahlen zwischen 1 und n enthält, soll die Zahl zwischen 1 und n bestimmt werden, die nicht in der Liste enthalten ist.
Bestimmen Sie auch die Laufzeit-Komplexität Ihrer Lösung.

Beispiele:

Eingabe: $[1, 3, 4, 5, 6]$
Ausgabe: 2
Eingabe: $[4, 2, 3, 6, 5]$
Ausgabe: 1

Anmerkung: Ihre Lösung wird sowohl anhand der Korrektheit als auch der Laufzeitkomplexität bewertet.

Hinweis: Sie müssen keinen Code für den Algorithmus angeben, eine Erklärung der Idee genügt. Eine optimale Lösung sollte in $\mathcal{O}(n)$ Zeit laufen. Genauer ist es sogar möglich, die Liste mit einem einzigen Durchlauf zu sortieren.

Lösung

Lösung: Durchlauf mit elementweisem Vergleich
Eine mögliche Lösung ist, für jede Zahl zwichen 1 und n zu prüfen, ob sie in der Liste enthalten ist. Diese Lösung hätte eine Laufzeitkomplexität von $O(n^2)$, da für jede Zahl die gesamte Liste durchsucht werden müsste.

Bessere Lösung: Elemente aufsummieren und vergleichen
Es genügt, die Liste nur einmal zu durchlaufen und die Summe aller Zahlen zu bilden. Dieser Durchlauf hat eine Laufzeitkomplexität von $O(n)$. Die Summe der Zahlen zwischen 1 und n ist $\frac{n(n+1)}{2}$, was in konstanter Zeit berechnet werden kann. Alternativ (etwas schlechter) kann auch die Summe der Zahlen durch eine Schleife berechnet werden, was ebenfalls in $O(n)$ Zeit möglich ist.

Aufgabe 5: Algorithmen (Entwurf)

Gegeben eine Liste von Zahlen sowie eine Zahl n, soll die erste Teilliste der Liste bestimmt werden, deren Summe gleich n ist.

Beispiele:

Eingabe: $[1, 5, 2, 3, 4]$, Ziel: 10
Ausgabe: $[5, 2]$

Eingabe: $[1, 4, 3, 2, 5, 3]$, Ziel: 5
Ausgabe: $[1, 4]$

Eingabe: $[1, 5, 6, 2, 4]$, Ziel: 13
Ausgabe: $[5, 6, 2]$

Eingabe: $[1, 5, 6, 2, 4]$, Ziel: 23
Ausgabe: []

Anmerkung: Ihre Lösung wird sowohl anhand der Korrektheit als auch der Laufzeitkomplexität bewertet.

Hinweis: Sie müssen keinen Code für den Algorithmus angeben, eine Erklärung der Idee genügt. Eine optimale Lösung sollte in $\mathcal{O}(n)$ Zeit laufen. Genauer ist es sogar möglich, die Liste mit einem einzigen Durchlauf zu sortieren.

Lösung

Lösung: Durchlauf mit Berechnung vieler Einzel-Summen
Eine mögliche Lösung ist, für jede Position in der Liste ab dort die Summe der folgenden Elemente zu bilden und zu prüfen, ob diese Summe gleich n ist bzw. ob sie n übersteigt. Diese Lösung hätte eine Laufzeitkomplexität von $O(n^2)$, da ab jeder Position potenziell alle weiteren Elemente summiert werden müssten.

Bessere Lösung: Sliding Window mit variabler Länge
Eine schnellere Lösung ist, sich zwei Positionen für Anfang und Ende der Teilliste, sowie die aktuelle Summe zu merken. Beide sind am Anfang auf der ersten Position, und die Summe ist auf dem Wert des ersten Elements.
Nun kann man die Positionen je nach Wert der Summe anpassen:

- Ist die Summe kleiner als n, wird das Ende um eins weiter gesetzt und der Wert des neuen Endes zur Summe addiert.
- Ist die Summe größer als n, wird der Anfang um eins weiter gesetzt und der Wert des alten Anfangs von der Summe subtrahiert.
- Ist die Summe gleich n, ist die Teilliste gefunden.

Es müssen noch Details beachtet werden, wie etwa, dass das Ende nicht über die Länge der Liste hinausgehen und nicht kleiner als der Anfang sein darf. Diese Details spielen aber keine Rolle für die Laufzeitkomplexität. Die Komplexität ist $O(n)$, da jeder Wert der Liste höchstens

einmal addiert und höchstens einmal subtrahiert wird. Die Teilliste kann dann aus den Werten zwischen Anfang und Ende gebildet werden, was eine weitere Schleife erfordert, aber ebenfalls in Zeit $O(n)$ möglich ist. Die Gesamtlaufzeit ist also $O(n)$.