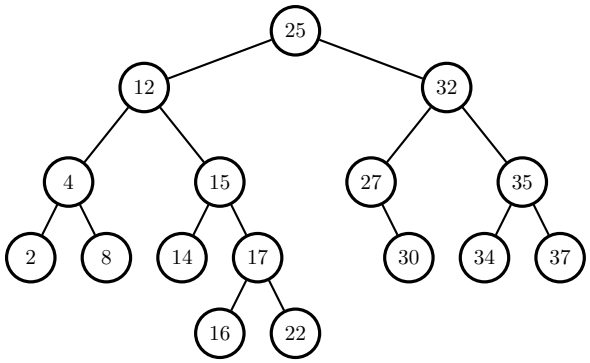
Präsenzübung 8

Besprechung: 17.12.18 – 21.12.18

Aufgabe 1: Suchbäume (keine Punkte)

a) Gegeben sind die Zahlen $[2, 8, 55, 13, 21, 3, 1, 5, 34]$. Geben Sie eine Eingabereihenfolge dieser Zahlen an, welche einen Suchbaum produziert, der so unbalanciert wie möglich ist. Geben Sie dann eine Eingabereihenfolge an, welche einen möglichst balancierten Suchbaum erzeugt. Zeichnen Sie die beiden resultierenden Bäume. Traversieren Sie die Knoten des balancierten Baumes jeweils in Preorder, Inorder und Posterorder-Reihenfolge und geben Sie die resultierende Reihenfolge der Schlüsselwerte an.

b) Gegeben ist folgender Suchbaum. Löschen Sie nacheinander die Knoten 15 und 25 und zeichnen Sie den Suchbaum nach jeder Operation.



c) Sei $d(v)$ die Anzahl der Kanten von einem Knoten v bis zur Wurzel. Sei $d_l(v)$ der Wert $d(v)$, den v nach einer Links-Rotation um einen beliebigen Knoten hätte. Betrachten Sie im folgenden eine Links-Rotation um einen Knoten x. Bestimmen Sie die Werte $d_l(x), d_l(x.left), d_l(x.right)$ und $d_l(x.p)$ unter Verwendung von d. Geben Sie nun in analoger Weise die Werte d_r der vier oben genannten Knoten an, wenn eine Rechts-Rotation um den Knoten x durchgeführt wurde.

d) Betrachten Sie den Suchbaum aus Teilaufgabe b) in seiner ursprünglichen Form. Führen Sie die minimale Anzahl von Rotationsschritten durch, sodass der Knoten 17 zur Wurzel des Suchbaums wird. Zeichnen Sie den Baum nach jeder Rotationsoperation.

Aufgabe 2: Radixsort (keine Punkte)

a) Sortieren Sie die folgenden Zahlen mit Radixsort: $[421, 117, 73, 42, 681, 35]$. Schreiben Sie dabei pro Iteration jeweils zwei Zwischenergebnisse auf: Die Verteilung der Zahlen auf die (10) Buckets und die anschließende Konkatenation der Zahlen.

Aufgabe 3: Ternäre Suche (keine Punkte)

Bei der binären Suche ist der Input ein sortiertes Array A und eine Zahl x (die nicht notwendig im Array A vorkommen muss). Dabei wird A in zwei gleich große Teile A_1 und A_2 geteilt und ermittelt, in welchem der beiden Teile sich x befinden müsste. Dieses Verfahren wird *rekursiv* fortgesetzt. Betrachten Sie nun die *ternäre Suche*, bei der A statt in zwei Teile, in *drei* etwa gleich große Teile A_1, A_2 und A_3 geteilt wird.

(a) Geben Sie ein Array A und ein zu suchendes Element x an, so dass die binäre Suche mit weniger Vergleichen auskommt als die ternäre Suche.

(b) Geben Sie ein Array A und ein zu suchendes Element x an, so dass die ternäre Suche mit weniger Vergleichen auskommt als die binäre Suche.

(c) Geben Sie Pseudocode für die ternäre Suche an. Verwenden Sie dabei Rekursion. Nummerieren Sie die Zeilen in Ihrem Pseudocode und erklären Sie detailliert jede Zeile Ihres Codes.

(d) Analysieren Sie die Zeitkomplexität der ternären Suche. Was können Sie über die asymptotische Laufzeit der ternären Suche im Vergleich zur binären Suche sagen?

(e) Bei jedem Rekursionsschritt werden ein oder zwei Vergleiche benötigt, um zu entscheiden, in welchem Teil des Arrays A das Element x liegt. Was ist die minimale, die durchschnittliche und die maximale Anzahl an Vergleichen die benötigt wird, wenn x nicht in A liegt?

Hinweis: Sie dürfen für die average-case Analyse davon ausgehen, dass x nur Werte im Intervall von $A[1]$ bis $A[n]$ annehmen kann.