Übungsblatt 7

Abgabe via Moodle.
Deadline Fr. 23ter Juni

Aufgabe 1 (***Implizite binäre Suchbäume,*** $2 + 1 + 2 + 4 + 1$ ***Punkte***)

Ein binärer Suchbaum mit n Datenelementen, bei dem auch die inneren Knoten Datenelemente enthalten, heiße *balanciert*, wenn für jeden Knoten gilt:
Die Höhen des rechten und linken Teilbaumes unterscheiden sich maximal um 1.
Die inneren Knoten des Baumes sind Knoten vom Grad größer als 1.

1. Zeichnen Sie einen balancierten binären Suchbaum, der die Elemente 2, 3, 4, 10, 14, 15, 21, 22, 23, 25 enthält und bei dem auch die inneren Knoten des Baumes Elemente tragen.

Wir betrachten nun die *implizite* Darstellung von balancierten binären Suchbäumen, bei denen auch die innere Knoten Elemente tragen. Diese Darstellung erfolgt mit Hilfe von Arrays (analog zur impliziten Darstellung binärer Heaps). Dabei dürfen nur weniger als die Hälfte aller Arrayeinträge leer sein.

2. Geben Sie die Formeln für den Index des linken und rechten Kindes des Elements mit Index i an (nehmen Sie an, dass beide Kinder existieren).

1. Der Suchbaum aus 1. soll nun implizit mit Hilfe eines Arrays dargestellt werden. Geben Sie den Inhalt des entsprechenden Arrays an. Leere Arrayeinträge werden mit dem Symbol $\perp$ bezeichnet.

3. Gegeben sei eine sortierte Folge von n Elementen. Geben Sie einen Algorithmus an, der in $\mathrm{O}(n)$ Zeit die implizite Darstellung eines balancierten binären Suchbaumes erzeugt. Der dargestellte Baum enthalte alle Elemente der Folge. Wieder dürfen nur weniger als die Hälfte aller Arrayeinträge leer sein.

4. Argumentieren Sie warum Ihr Algorithmus aus 3. das gewünschte Laufzeitverhalten aufweist.

Aufgabe 2 (***Suchen in sortierten Arrays bei mehrfachen Vorkommen,*** $1 + 1 + 4 + 2$ ***Punkte***)

Betrachten Sie den Fall eines sortierten Arrays, bei dem Elemente auch mehrfach auftreten dürfen.

1. *Warmup:* Gegeben sei ein sortiertes Array A der Länge n von ganzen Zahlen. Geben Sie einen Algorithmus an, der in schlimmstenfalls $\Theta(k + \log n)$ Zeit die Positionen des ersten und des letzten Auftretens eines Elementes in A ermittelt. Dabei sei k die Anzahl der Vorkommen des gesuchten Elementes.

2. Argumentieren Sie warum Ihr Algorithmus aus 1. das gewünschte Laufzeitverhalten aufweist.

3. Geben Sie einen weiteren Algorithmus an, der in $\mathrm{O}(\log n)$ Zeit die Positionen des ersten und des letzten Auftretens eines Elementes in A ermittelt.

4. Argumentieren Sie warum Ihr Algorithmus aus 3. das gewünschte Laufzeitverhalten aufweist.

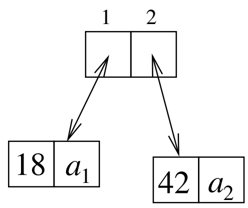
Aufgabe 3 (*Adressierbare Heaps,* $2+2+1+1$ *Punkte*)

Gegeben sei ein leerer, adressierbarer binärer Heap, der implizit als Array realisiert ist. D.h. in diesem Fall speichert das Array Handles auf Paare der Form (*Schlüssel, Datenelement*). Außerdem seien 6 Datenelemente $a_1, \ldots, a_6$ gegeben.

1. Stellen Sie den Zustand des adressierbaren Heaps graphisch dar, wie er nach Ausführen der Operationsfolge

   $$\textit{insert}(a_1, 18), \textit{insert}(a_2, 42), \textit{insert}(a_3, 11), \textit{insert}(a_4, 19), \textit{insert}(a_5, 7), \textit{insert}(a_6, 13)$$

   aussieht. Z.B. kann man den Zustand nach Ausführen der beiden ersten Operationen wie folgt darstellen:

   Die Doppelpfeile symbolisieren Handles für jeweils beide Richtungen.

2. Nun werde *decreaseKey*$(a_6, 5)$ ausgeführt. Wie sieht der Zustand des Arrays danach aus?
3. Im Anschluss werde *deleteMin* ausgeführt. Wie sieht der Zustand des Arrays nun aus?
4. Zuletzt werde *decreaseKey*$(a_2, 10)$ ausgeführt. Wie sieht der Zustand des Arrays nun aus?

Aufgabe P7 (*Is this a Binary Search Tree?, optional*)

Für die praktischen Übungen verwenden wir die Plattform `www.hackerrank.com`. Hier müssen Sie sich registrieren um an den Übungen teilzunehmen. Unter dem Link

`https://www.hackerrank.com/adsi-2023`

finden die praktischen Übungen in der Form eines Programmierwettbewerbs statt.

In der siebten Challenge geht es um binäre Suchbäume. Ihnen wird ein Baum als Eingabe gegeben und Sie müssen überprüfen, ob es sich dabei um einen binären Suchbaum handelt.

Wichtig in dieser Aufgabe ist, dass keine Gleichheit zwischen Elter- und Kindknoten auftreten darf. Hier wird der binäre Suchbaum so definiert, dass zu jedem Knoten die Werte der Knoten des linken Teilbaumes echt kleiner und die Werte der Knoten des rechten Teilbaumes echt größer sein müssen.

Eine genauere Beschreibung, sowie ein Beispiel finden Sie auf HackerRank.