Ludwig-Maximilians-Universität München
Institut für Informatik
Prof. Dr. Thomas Seidl
Janina Sontheim, Florian Richter

München, 02.05.2019

Algorithmen und Datenstrukturen

SS 2019

Übungsblatt Global 1: Grundlagen

Aufgabe Global 1-1 *Knobelei: Robo-Seiltanz*

Gegeben ist ein 10 Meter langes Seil und eine Menge von n identischen Robotern. Jeder Roboter verhält sich gemäß folgender Anweisungen:

- Jeder Roboter kann sich je nach Ausrichtung entweder nach links oder rechts bewegen.
- Die Geschwindigkeit ist stets 10 Meter pro Minute.
- Wenn zwei Roboter kollidieren, drehen sich beide ohne Zeitverlust herum und bewegen sich jeweils in die entgegengesetzte Richtung.

Geben Sie eine obere Schranke für die Laufzeit der Roboter an (wann fällt der letzte Roboter vom Seil), wenn die Roboter an beliebiger Position und mit beliebiger Ausrichtung starten können.

Lösungsvorschlag:
Angenommen, wir haben bloß einen einzelnen Roboter. Dieser kann maximal die gesamte Strecke fahren und am anderen Ende herunterfallen. Er braucht dafür 1 Minute. Nun schauen wir uns an, was bei einer Kollision passiert:

Abbildung 1: Vor Kollision: Beide Roboter schauen sich gegenseitig an.

Abbildung 2: Nach Kollision: Beide Roboter haben ihre Richtung gedreht und schauen entgegengesetzt.

Wir verändern nun die „Welt“, in der sich die Roboter befinden und nehmen an, sie würden nicht kollidieren, sondern geschickt aneinander vorbei fahren. Dann sieht es nach der Kollision wie folgt aus:

Abbildung 3: Aneinder vorbeifahren: Auch hier schauen die Roboter in entgegengesetzte Richtungen.

Der Ausgang ist bis auf die Bezifferung gleich. Da aber die Roboter alle identisch sind, ist der Zustand nach einer Kollision mit dem Zustand nach Vorbeifahren identisch. Damit brauchen viele Roboter nur maximal so lange wie ein einzelner Roboter und nach spätestens einer Minute ist der letzte Roboter vom Seil gefallen.

Aufgabe Global 1-2 *Stack und Queue*

Gegeben sei ein Array ganzer Zahlen der Länge $n \in \mathbb{N}$.

(a) Erläutern Sie, wie mit Hilfe dieses Arrays zwei Stacks implementiert werden könnten. Dabei soll es immer nur dann zum Überlauf kommen, wenn die Summe der Elemente beider Stacks zusammen gleich n ist. Die Operationen Push und Pop sollen dabei die Komplexität $\mathcal{O}(1)$ haben.

Lösungsvorschlag:
Ein Stack erlaubt es, Objekte zu speichern und immer das zuletzt abgespeicherte wieder abzurufen. Einen Stack in einem Array zu realisieren, benötigt daher bloß einen Zeiger auf das zuletzt eingefügte Element. Dieser Zeiger kann verschoben werden, falls eine der beiden Zugriffsoperationen ausgeführt wird.

Um nun zwei Stacks in einem Array zu realisieren, lässt man den zweiten Stack am Ende beginnen und führt einen weiteren Zeiger auf sein Top-Element ein. So können beide Stacks auf den jeweiligen Seiten des Arrays in Richtung Mitte wachsen.

Wenn ein Stack über die Mitte hinauswächst, ist dies kein Problem, solange der gegenüberliegende Stack genügend Platz bietet. Berühren sich beide Stacks, so ist kein Platz für neue Elemente mehr vorhanden.

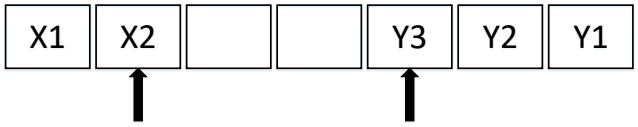


Abbildung 4: Erster Stack hat schon zwei Elemente, Zweiter Stack drei. Noch ist Platz zwischen beiden Stacks.

(b) Erläutern Sie, wie mit Hilfe dieses Arrays zwei Queues implementiert werden könnten. Dabei soll es immer nur dann zum Überlauf kommen, wenn eine Queue mindestens $n/2 - 1$ Elemente beinhaltet.

Lösungsvorschlag:
Bei Queues kann man sich Arrays gut als zyklische Formen vorstellen, bei denen Anfang und Ende verbunden ist. In der Praxis benutzt man für den Zugriff einfach eine Modulo-Operation, so dass

$$a_n = a_{n \bmod n} = a_0$$

Wenn man nun zwei Queues jeweils eine Hälfte des Arrays überlässt, so können auch beide Teilarrays als zwei Zykel gelesen werden. Die erste Queue benutzt dann die Positionen $0 \ldots n/2 - 1$, die zweite Queue die Positionen $n/2 \ldots n - 1$.

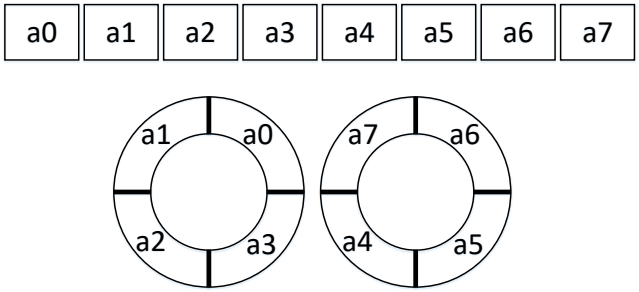


Abbildung 5: Hier wird das Array nicht nur einmal zusammen gerollt, sondern in zwei Kreise zerlegt.

Aufgabe Global 1-3 *Traversieren*

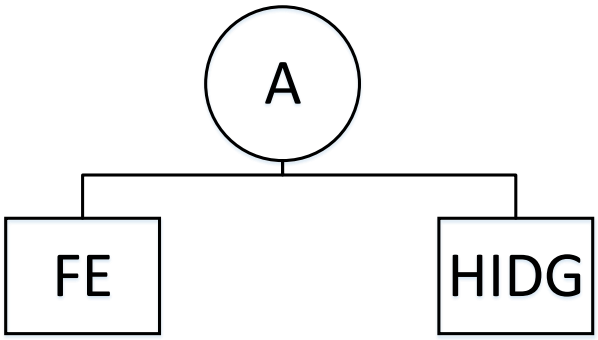
Gegeben ist das Ergebnis einer Preorder- und Inorder-Traversierung für ein und denselben binären Baum.

PreOrder-Traversierung = A E F D I H G
InOrder-Traversierung = F E A H I D G

Die Knotenbezeichnungen sollen in dieser Aufgabe immer eindeutig sein.

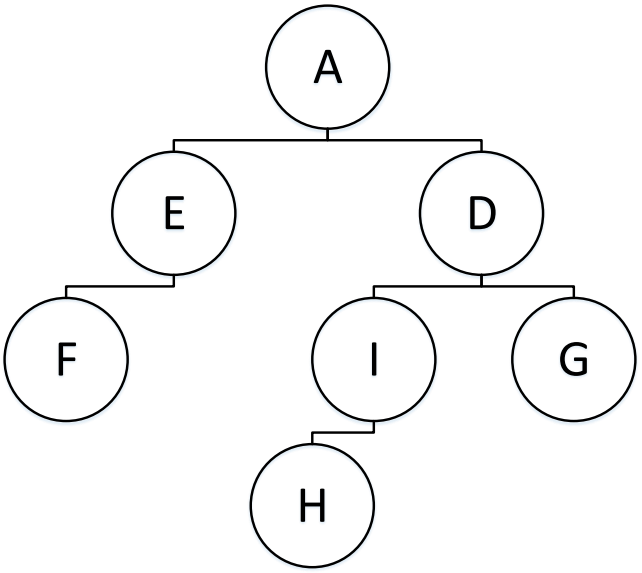
(a) Geben Sie einen Baum an, zu dem die PreOrder- und InOrder-Traversierung passt.

Lösungsvorschlag:
Da die PreOrder-Traversierung eine Reihenfolge der Knoten von oben nach unten vorgibt, muss A die Wurzel des Baums sein. Die InOrder-Traversierung enthält keine Informationen über diese Dimension des Baums, gibt aber andererseits die Reihenfolge der Knoten in horizontaler Richtung vor. Alle Knoten links von A müssen daher im linken Teilbaum, alle Knoten rechts von A müssen daher im rechten Teilbaum sein. Es ergibt sich folgender Pseudobaum, der erst Mengen von Kandidaten in den Blättern stehen hat:



Für FE bestimmt nun die PreOrder-Traversierung die vertikale Reihenfolge. Also muss E die Wurzel sein. Die InOrder-Traversierung schreibt erst F, dann E, somit muss F im linken Teilbaum zu finden sein.

Im rechten Teilbaum folgt analog: D muss Wurzel sein, weil es zuerst in der PreOrder-Traversierung vorkommt. HI sind dann im linken Teilbaum, G im rechten Teilbaum wegen der InOrder-Reihenfolge. Insgesamt ergibt sich der eindeutige Baum:



(b) Geben Sie zwei Binärbäume $T_1, T_2, T_1 \neq T_2$ an, für die gilt:
PreOrder(T_1) = PreOrder(T_2) aber InOrder(T_1) $\neq$ InOrder(T_2)

Lösungsvorschlag:
Dies ist relativ einfach. Mit einem Knoten schafft man es nicht, da dann direkt die Gleichheit der Bäume folgen muss. Mit Wurzel 1 und Kindknoten 2 ist es für die PreOrder-Traversierung egal, ob 2 linker oder rechter Kindknoten ist. Für die InOrder-Traversierung ist dies aber unterschiedlich.

(c) Gibt es zwei Binärbäume T_1, T_2 mit $T_1 \neq T_2$, für die gilt:
PreOrder(T_1) = PreOrder(T_2) und InOrder(T_1) = InOrder(T_2).
Begründen Sie Ihre Antwort. Ein formaler Beweis ist nicht notwendig.

Lösungsvorschlag:
Eine intuitive Antwort liefert uns schon die Tatsache, dass wir bei der ersten Teilaufgabe keinerlei Wahlmöglichkeiten hatten, um den Baum aufzubauen. Die zweite Teilaufgabe lieferte auch sehr leicht eine Lösung falls die InOrder-Traversierung unterschiedlich ist.

Angenommen, wir haben zwei Bäume T_1 und T_2, und beide haben jeweils folgende Traversierungen:

$$Pre = P_1 P_2 \dots P_n$$
$$In = I_1 I_2 \dots I_n$$

P_1 muss dann analog zu a) die Wurzel sein. Es gibt nun ein Element $I_r = P_1$ und alle Elemente vor I_r sind im linken Teilbaum, alle Elemente nach I_r sind im rechten Teilbaum. Es ergeben sich nun zwei Teilbäume, die wir untersuchen müssen. T_1 und T_2 können nur dann unterschiedlich sein, wenn mindestens einer der Teilbäume unterschiedlich ist.

Diese Argumentation können wir sukzessive fortsetzen, bis die Kandidaten für die Knoten zu ein-elementigen Mengen geschrumpft sind. T_1 und T_2 können also nur dann unterschiedlich sein, falls wir am Ende ein-elementige Mengen finden, die die gleichen Elemente enthalten aber unterschiedlich sind. Da dies ein direkter Widerspruch ist, kann es keine solche Bäume T_1 und T_2 geben.