Institut für Theoretische Informatik
Lehrstuhl Prof. Dr. D. Wagner

Übungsblatt 1

Vorlesung Algorithmentechnik im WS 09/10

Ausgabe 22. Oktober 2009
Abgabe 5. November, 15:30 Uhr in der Übung
(oder im Kasten vor Zimmer 319, Informatik-Hauptgebäude, 3. OG)

Bitte schreiben Sie Ihren Namen und Ihre E-Mailadresse aber <u>keine</u> Matrikelnummer auf Ihr Übungsblatt. Die Bearbeitung in Zweiergruppen ist ausdrücklich erwünscht. Die Kapitelangaben können in zukünftigen Skriptversionen abweichen.

Problem 1: Dynamisches Array (Amortisierte Analyse) [vgl. Kapitel 0.3 im Skript]

Ein dynamisches Array wird zunächst als leeres Array A der Länge $l(A) = 0$ initialisiert. Falls Insert (x) ein bereits volles Array A vorfindet, wird ein neues Array A' mit doppelter Länge angelegt und die bereits vorhandenen Elemente werden aus dem vorherigen Array in das neue Array kopiert. Schließlich wird auch das eigentlich einzufügende Element x nach A' kopiert. Das Array A' hat dann also die Länge $l(A') = 2\,l(A)$ und enthält $l(A) + 1$ Elemente. Falls nach Delete (x) das Array A nur noch zu maximal einem Viertel gefüllt ist, so wird ein neues Array A' mit halber Länge angelegt und die übrigen Elemente aus dem vorherigen Array werden in das neue Array kopiert. Das Array A' hat dann also die Länge $l(A') = \lfloor \frac{1}{2}\, l(A) \rfloor$ und enthält $\lfloor \frac{1}{4}\, l(A) \rfloor$ Elemente (Hinweis: Da die Arraylängen Zweierpotenzen sind, ist jede Länge $l(A) \geq 4$ durch 4 teilbar). In allen anderen Fällen wird x einfach mit konstanten Kosten eingefügt bzw. gelöscht. Gelöschte Elemente geben Platz frei, der durch Einfügungen wieder gefüllt wird, bevor es zu einer Verdoppelung der Arraylänge kommt. Eine entsprechende Speicherplatzverwaltung wird als gegeben vorausgesetzt.

Analysieren Sie die amortisierten Kosten einer Folge von Insert (x) und Delete (x). Setzen Sie für das Kopieren eines Elements konstante Kosten voraus.

Problem 2: Master-Theorem (Laufzeit rekursiver Funktionen) [vgl. Kapitel 0.4.5 im Skript]

Prüfen Sie die Anwendbarkeit der beiden Master-Theorem-Versionen (Master-Theorem und seine allgemeinere Form) auf die folgenden Rekurrenzgleichungen *(auch Rekursionsgleichungen genannt)*:

(a) $T(n) = 2\,T(\lfloor n/2 \rfloor) + 2\,T(\lceil n/2 \rceil) + n^2$

(b) $T(n) = 7\,T(n/7) + n\,\ln(n)$

Begründen Sie Ihre Antwort.

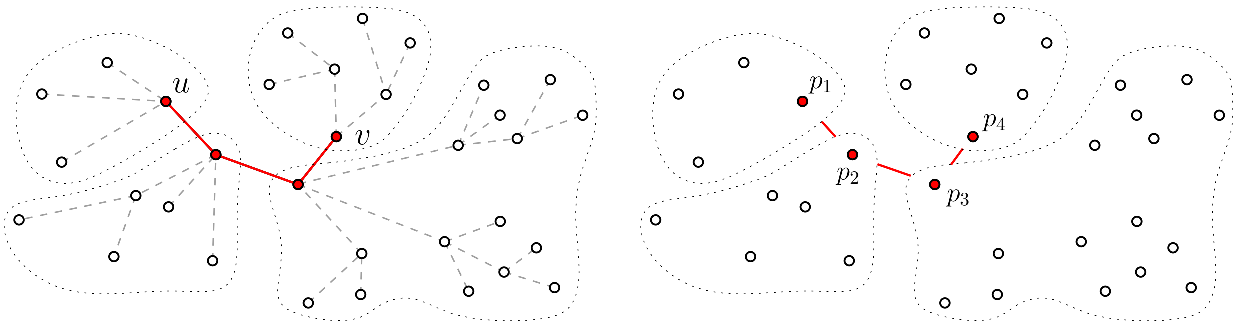
Problem 3: Binäre Suche (Laufzeit rekursiver Funktionen) [vgl. Kapitel 0.4.5 im Skript]

Gegeben sei eine sortierte Sequenz $A := (a_1, \ldots, a_n)$ von n Elementen. Die Binäre Suche sucht nun nach einem Element x in A, indem sie x mit dem mittleren Element in A vergleicht und so entscheidet, ob und wenn ja in welcher Hälfte die Suche rekursiv fortgesetzt wird.

Modellieren Sie die Binäre Suche in Pseudocode und erstellen Sie die zugehörige Rekurrenzgleichung. Bestimmen Sie eine asymptotisch scharfe obere und eine asymptotisch scharfe untere Schranke der Laufzeit.

Problem 4: Pfad aus Superknoten (Union-Find) [vgl. Kapitel 1.1 und 1.2 im Skript]

Gegeben sei ein Baum T mit n Knoten und ein Knotenpaar $\{u, v\}$. Schreiben Sie einen Algorithmus in Pseudocode, der den (eindeutigen) Pfad zwischen u und v in T berechnet und für jeden Pfadknoten p_i die Teilbäume mit Wurzel p_i kontrahiert (siehe untenstehende Abbildung). Benutzen Sie dazu eine modifizierte Tiefensuche. Speichern Sie besuchte Knoten in einer Union-Find-Struktur, in der Knotenmengen vereinigt werden, die zu Teilbäumen des selben Pfadknotens gehören.



(a) Pfadknoten mit Teilbäumen

(b) Knotenmengen nach Kontraktion der Teilbäume.

Problem 5: Heap-Varianten (Heap-Datenstruktur) [vgl. Kapitel 1.3 im Skript]

(a) Die klassische Heap-Eigenschaft $A[\mathit{Vorgänger}[i]] \geq A[i]$ kann auch als Max-Heap-Eigenschaft bezeichnet werden. Der klassische Heap wird somit zu einem Max-Heap. Wie sieht konsequenterweise die Heap-Eigenschaft eines Min-Heaps aus?

(b) Sei A ein Max-Heap (gespeichert in einem Array) und A' enthalte die Elemente aus A in umgekehrter Reihenfolge. Ist A' ein Min-Heap? Begründen Sie Ihre Antwort.

(c) Ergänzen Sie an untenstehendem Heap A (Max-Heap in Baumform) die zugehörigen Array-Indizes. Wenden Sie Delete $(A, 2)$ an, indem Sie die einzelnen Zustände, die der Baum durchläuft, darstellen.