TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Fachbereich Informatik
AG Algorithmen und Komplexität

Ausgabedatum: 13.11.2014
Version: 2014-11-11 19:04

3. Übungsblatt für Track AI zur Vorlesung Entwurf und Analyse von Algorithmen, WS 14/15

Abgabe: *Bis Freitag, 21.11.2014, 12:00 Uhr, Kasten im Treppenhaus 48-6.*

Basisaufgaben

B3.1: Rekursionsgleichungen mit Iteration

3 Punkte

Lösen Sie folgende Rekursionsgleichungen mittels Iteration und Korrektheitsbeweis; bringen Sie Ihre Ergebnisse außerdem in möglichst geschlossene Form.

a) $A(0) = 1,$
$A(1) = 1,$
$A(n) = 3 \cdot A(n-2) + 5, \quad n \geq 2\,.$

b) $C(0) = 2,$
$C(n) = n \cdot C(n-1) + n, \quad n \geq 1\,.$

B3.2: Rekursionsgleichungen mit Mastertheorem

3 Punkte

Ermitteln Sie Θ-Asymptotiken für die durch die folgenden Rekursionsgleichungen beschriebenen Zahlenfolgen! Treffen Sie dafür geeignete Annahmen an n und beweisen Sie Ihre Behauptungen.

a) $A(1) = 1,$
$$A(n) = \frac{5}{2} \cdot A\left(\frac{n}{2}\right) + \frac{n}{2}, \quad n \geq 2.$$

b) $B(1) = 6,$
$$B(n) = B\left(\frac{n}{3}\right) + n - 1, \quad n \geq 3\,.$$

c) $C(1) = 1,$
$$C(n) = 4 \cdot C\left(\frac{n}{2}\right) + 7 \cdot \left(\frac{n}{2}\right)^2, \quad n \geq 2\,.$$

B3.3: Funktionen Sortieren

3 Punkte

Betrachten Sie alle möglichen Paare (g_i, g_j) mit $0 \leq i, j \leq 9$ aus den untenstehenden Funktionen und geben Sie die jeweils stärkste gültige Beziehung aus $\mathcal{O}$, Ω, Θ, o, ω und $\sim$ an.

$$
\begin{array}{llll}
g_0(n) = \ln(\ln(n)) & g_1(n) = \ln^2(n) & g_2(n) = \frac{n}{\ln(n)} & g_3(n) = \frac{\ln(n)}{n} \\
g_4(n) = \sqrt{n}\ln^2(n) & g_5(n) = \left(\frac{2}{3}\right)^n & g_6(n) = \left(\frac{3}{2}\right)^n & g_7(n) = n \\
g_8(n) = \sqrt{n} & g_9(n) = \ln(n) &&
\end{array}
$$

Zeigen Sie exemplarisch (pro Beziehungstyp einmal), wie die Beziehungen bewiesen werden können.

Hinweis: Können Sie sich Schreib- und Beweisarbeit sparen?

B3.4: Mengen als sortierte Listen

3 Punkte

Entwerfen Sie für die folgenden Operationen auf Mengen in verketteter Listendarstellung je einen Algorithmus in detailliertem Pseudocode analog zu der Implementierung des Durchschnitts aus der Vorlesung. Insbesondere soll auch hier die resultierende Liste wieder eine sortierte Liste sein.

Wie für alle Entwurfsaufgaben soll ihre Abgabe (sofern nicht anders vermerkt) aus folgenden Teilen bestehen:

- einer kurzen Beschreibung der (algorithmischen) Idee,
- dem Algorithmus selbst, in Form von (detailliertem) Pseudocode,
- einer Begründung der Korrektheit des Algorithmus, sowie
- einer (groben) Laufzeit-Analyse in Θ-Klassen.

Gestalten Sie Ihren Code *lesbar*, d. h. versehen Sie komplizierte Codestellen mit sinnvollen Kommentaren und verwenden Sie Strukturierungselemente wie z. B. Hilfsprozeduren!

a) Vereinigung $(A \cup B)$

b) Differenz $(A \setminus B)$

Aufbauaufgaben

A3.1: Zu Lemma 1.4

3 + 5 Punkte

a) Beweisen Sie Teil e) von Lemma 1.4 im Buch (Seite 15), natürlich ohne die anderen Teile zu verwenden.

b) Teile a) – c) von Lemma 1.4 gelten in beide Richtungen – dort könnte also „genau dann, wenn“ stehen – da sie genau zu Punkten 4. – 6. in Definition 1.8 passen.

Wie kann man Lemma 1.4 modifizieren, damit auch d) – f) zu Definition 1.8 äquivalent werden? Skizzieren Sie einen Beweis für die behauptete Äquivalenz.

Hinweis: Die Existenz von $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ ist nicht vorauszusetzen.

Tipp: Finden Sie zwei Funktionen f, g mit $f \in \mathcal{O}(g)$, sodass $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ nicht existiert?

A3.2: Mengen als Sparse Bitvectors

7 + 2 Punkte

In der Vorlesung haben wir gesehen, wie man Mengen mit Bitvektoren – Arrays aus Booleans – implementieren kann, was schnelle Operationen auf Kosten von Speicher erlaubt. Ein unangenehmer Nachteil entsteht bei sehr großen Universen und kleinen Mengen, auf denen nur wenige Operationen ausgeführt werden; in diesem Fall ist die Initialisierungszeit – jeder Eintrag im Array muss explizit auf `0` gesetzt werden – dominant, nicht die eigentlichen Operationen.

a) Seien eine beliebige Menge $\mathcal{U}$ mit $|\mathcal{U}| = N \in \mathbb{N}$ und eine Bijektion $\varphi : \mathcal{U} \to [0..N-1]$ gegeben. Geben Sie eine Implementierung für Teilmengen von $\mathcal{U}$ auf Basis von Arrays an, die

- die Basisoperationen – also Prüfung, ob ein Element in der Menge enthalten ist, sowie Entfernen und Hinzufügen von Elementen – ebenso in Zeit $\mathcal{O}(T_\varphi)$ ermöglicht, $T_\varphi : \mathcal{U} \to \mathbb{R}^+$ die Laufzeit von φ,
- dabei $\mathcal{O}(mN)$ Speicher benötigt, aber
- in Zeit $\mathcal{O}(1)$ initialisiert werden kann – was das eigentliche Ziel ist.

m ist hierbei die *Wortbreite*, also die Anzahl Bits, die zur Speicherung eines Integers benötigt wird. Sie können annehmen, dass $m \in \Theta(\log N)$.

Wir nehmen an, dass Speicherallokation in konstanter Zeit vom Betriebssystem durchgeführt wird. Sie können jedoch keine Annahmen darüber machen, welchen Inhalt eine nicht initialisierte Speicherstelle direkt nach der Allokation hat.

Begründen Sie, warum die gewünschten Kriterien – also Korrektheit, Speicherbedarf und Laufzeiten – von Ihrer Datenstruktur erfüllt werden.

Hinweis: Die Landausymbole sind hier für $N \to \infty$ zu lesen.

b) Skizzieren Sie, wie man mit Ihrer Datenstruktur aus a) die Vereinigungsoperation, also $C = A \cup B$, in Zeit $\mathcal{O}(|A| + |B|)$ durchführen kann. Sollte das nicht gehen, begründen Sie, warum nicht.

A3.3: Asymptotische Entwicklung

3 Punkte

Beweisen oder widerlegen Sie:

a) $\sqrt{n} - \sqrt{n-1} \sim \frac{1}{2\sqrt{n}}$

A3.4: Stacks und Queues

3 + 1 Punkte

a) Nehmen Sie an, Sie dürfen ausschließlich die Datenstruktur `Stack` verwenden, ohne daran Veränderungen vorzunehmen. Wie können Sie dann unter Verwendung zweier Stacks *effizient* eine Queue implementieren?

Effizienz heißt hier, dass jedes Element zwischen Eintritt in die Queue bis zum Verlassen der Queue nur konstant oft *gepusht* oder *gepopt* wird, und zwar unabhängig davon, wie viele Elemente dazwischen in die Queue eingefügt oder aus der Queue entfernt werden.

Formulieren Sie die wesentliche(n) Invarianten, die die Korrektheit ihres Entwurfs sicherstellen. Geben Sie weiterhin Pseudocode für die Initialisierung und Queueoperationen ihrer Datenstruktur an. Zeigen Sie weiterhin, dass Ihre Lösung tatsächlich effizient ist.

b) Lässt sich auch umgekehrt ein Stack durch zwei Queues *effizient* implementieren? Wie bzw. warum nicht?