TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Fachbereich Informatik
AG Algorithmen und Komplexität

Ausgabedatum: 28.11.2013
Version: 2013-11-27 19:53

7. Übungsblatt für Track AI zur Vorlesung Entwurf und Analyse von Algorithmen, WS 13/14

Abgabe: *Bis* ***Freitag****, 06.12.2013, 12:00 Uhr, Kasten im Treppenhaus 48-6.*

Bitte denken Sie an die Anmeldung zur Zwischenklausur!

Der Fehler der Woche

$f \in \omega(n^\alpha)$ für alle $\alpha \in \mathbb{R}^+$, also ist f exponentiell.

Was ist falsch?

Basisaufgaben

B1: Rekursionsgleichungen und Satz 3.3

2 Punkte

Verwenden Sie Satz 3.3, um folgende homogene lineare Rekursionsgleichungen zu lösen:

a) $a_1 = 1$,
$a_2 = 4$,
$a_i = 2 \cdot a_{i-1} + 3 \cdot a_{i-2}, \quad i \geq 3$.

b) $b_0 = 5$,
$b_1 = 7$,
$b_2 = 9$,
$b_i = 12 \cdot b_{i-2} - 16 \cdot b_{i-3}, \quad i \geq 3$.

B2: Subtraktionsmethode

3 Punkte

Bestimmen Sie mit Hilfe der Subtraktionsmethode (diese haben wir bei der Analyse der Suchzeiten binärer Suchbäume kennengelernt) eine Rekursionsgleichung, die auf nur konstant viele Vorgänger zugreift und dieselbe Zahlenfolge erzeugt wie die nachfolgende Rekursionsgleichung mit Full History:

$$X_0 = 1,$$
$$X_n = 3 + \frac{2}{n+1} \sum_{k=0}^{n-1} X_k, \quad n \geq 1 .$$

B3: Sortierte Teilfolgen und Balanciertheit

3 Punkte

Wir haben gesehen, dass das Einfügen einer sortierten Folge in einen binären Suchbaum diesen zu einer Liste entarten lässt. Wir untersuchen, ob dieser Effekt bereits durch das Vorkommen sortierter Teilfolgen in der Einfügefolge erzwungen wird.

Sei $(x_i)_i = x_1, \ldots, x_k$ eine Folge, dann ist $x_{i_1}, \ldots, x_{i_l}$ eine Teilfolge von $(x_i)_i$ der Länge $l \leq k$, falls $i_j < i_{j+1}$ für $1 \leq j \leq l-1$ und $1 \leq i_1$ sowie $i_l \leq k$ gilt. Eine Teilfolge ist also die ursprüngliche Folge oder sie entsteht durch Weglassen von Einträgen der ursprünglichen Folge.

Eine Folge $y_1, \ldots, y_l$ heißt monoton, falls entweder $y_i \leq y_{i+1}$ für alle i mit $1 \leq i \leq l-1$ gilt, oder $y_i \geq y_{i+1}$ für alle i mit $1 \leq i \leq l-1$ gilt.

Die Einfügefolge $(x_i)_i = x_1, \ldots, x_k$, die den binären Suchbaum T erzeugt, bestehe aus paarweise verschiedenen natürlichen Zahlen. Es sei H die Höhe von T und es sei l die Länge einer längsten monotonen Teilfolge von $(x_i)_i$.

Zeigen oder widerlegen Sie die folgenden Aussagen:

i) Es gilt stets $H \leq l$.

ii) Es gilt stets $H \geq l$.

iii) Es gilt stets $H = l$.

B4: Rekursionsgleichungen und OGF

4 Punkte

a) Lösen Sie die inhomogene lineare Rekursionsgleichung

$$\begin{aligned} a_0 &= 1, \\ a_i &= 3 \cdot a_{i-1} + 2^i, \quad i \geq 1, \end{aligned}$$

mittels Erzeugendenfunktionen.

b) Lösen Sie die inhomogene lineare Rekursionsgleichung

$$\begin{aligned} b_0 &= 1, \\ b_1 &= 1, \\ b_{i+2} &= 3 \cdot b_{i+1} - 2 \cdot b_i + i, \quad i \geq 0, \end{aligned}$$

mittels Erzeugendenfunktionen.

Hinweis: Sie *können* die Rekursionsgleichungen natürlich homogenisieren, *müssen* es jedoch nicht.

Aufbauaufgaben

18. Aufgabe

3 Punkte

Wir bezeichnen mit *Durchmesser* d eines Graphen $G = (V, E)$ den längsten kürzesten Pfad in G, also

$$d(G) := \max \{\text{dist}(u, v) \mid u, v \in V\}$$

mit $\text{dist}(u, v)$ der Länge eines kürzesten Pfades (wir zählen Kanten) zwischen u und v in G.

Entwerfen Sie einen Algorithmus, der in Zeit $\mathcal{O}(|V|)$ den Durchmesser eines beliebigen zusammenhängenden und kreisfreien Graphen[1] berechnet.

19. Aufgabe

$2 + 1 + 2$ Punkte

Bei der Baumdarstellung der Partitionen zur Lösung des UNION/FIND-Problems sind wir davon ausgegangen, dass wir die Größen eines jeden Baumes kennen, und zur Implementierung der UNION-Operation den kleineren Baum zum Sohn der Wurzel des größeren machen.

In dieser Aufgabe gehen wir davon aus, dass wir keine Größen kennen, sondern einen jeden Knoten x mit einem Gewicht $g(x)$ versehen. Wird eine Partition mit nur einem Element als Einknotenbaum initialisiert, so wird das Gewicht dieses Knotens auf 0 gesetzt. Die UNION-Operation macht nun die Wurzel mit einem höheren Gewicht zum Vater der Wurzel mit dem kleineren Gewicht; haben beide Wurzelknoten dasselbe Gewicht, so wählen wir zufällig einen der Wurzelknoten als Vater aus und erhöhen sein Gewicht um 1. FIND-Operationen belassen alle Gewichte unverändert.

a) Beweisen Sie, dass für $|T_x|$ die Anzahl der Knoten in einem derart konstruierten Baum mit Wurzel x stets $|T_x| \geq 2^{g(x)}$ gilt.

b) Zeigen Sie, dass in der so verwalteten Datenstruktur die Operation FIND(x) in Zeit $\mathcal{O}(\log |P_x|)$ läuft.

c) Erweitern Sie die aus der Vorlesung bekannte Datenstruktur für Partitionen, die Bäume und Pfadkomprimierung benutzt, um eine Operation `SET(x)`. Für ein beliebiges Element `x` soll `SET(x)` die Partition P_x, in der `x` enthalten ist, als (neue) lineare Liste zurückgeben; die Reihenfolge der Elemente im Ergebnis spielt keine Rolle.

   Die Laufzeit von `SET(x)` soll in $\mathcal{O}(|P_x|)$ liegen. Weiterhin sollen die asymptotischen Laufzeiten der anderen Operationen nicht beeinträchtigt werden.

   Begründen Sie Korrektheit und Effizienz Ihres Entwurfs.

   Hinweis: Es genügt, jeden Knoten der Datenstruktur um ein Attribut zu erweitern.

[1] In der Literatur heißt ein solcher Graph auch „Baum“, aber unsere Definition passt nicht.

20. Aufgabe

3 + 2 + 2 Punkte

a) Entwerfen Sie eine Datenstruktur, die Paare aus $\mathbb{N} \times \mathbb{N}$ so speichert, dass ein beliebiges Element in $\mathcal{O}(\log n)$ Zeit gefunden bzw. eingefügt werden kann, wenn n Elemente bereits enthalten sind.

Begründen Sie Korrektheit und Effizienz Ihres Entwurfs.

b) Beschreiben Sie, wie man aus Ihrer Datenstruktur aus a) – evtl. mit kleinen Modifikationen, die die in Teil a) geforderten Eigenschaften nicht zerstören – in Zeit $\mathcal{O}(m + \log n)$ eine Liste aller m Elemente $(k, \_)$ in beliebiger Reihenfolge für festes k erzeugen kann.

Begründen Sie Ihre Behauptungen.

c) Nehmen Sie an, dass auf einer festen Instanz eines Wörterbuchs eine (lange) Folge von Suchen nach den Schlüsseln $(k_0, l_0), (k_1, l_1), \ldots$ ausgeführt werden soll. Die Elemente sind nicht – wie üblicherweise angenommen – uniform gezogen, sondern verhalten sich lokal in der ersten Komponente, oder formal

$$\Pr[k_i = k_{i+1}] = 1 - \varepsilon$$

für ein $\varepsilon \in (0, 1)$, wobei wir kleine ε, also das Szenario $\varepsilon \to 0$ untersuchen wollen. Sie dürfen außerdem annehmen, dass es sowohl „genügend viele“ Schlüssel also auch „genügend viele“ verschiedene erste bzw. zweite Komponenten gibt.

Ermöglicht es Ihre Datenstruktur aus a) bzw. b) – evtl. wieder mit kleinen, den allgemeinen Fall nicht störenden Änderungen – aus diesem Wissen Kapital zu schlagen und die mittlere Suchzeit zu verbessern?

Begründen Sie Ihre Behauptung.