Randomisierte Algorithmen WS'04
Vorlesung: T. Worsch und A. Wolff
Übungen: M. Benkert, T. Schank und É. Schramm

UNIVERSITÄT KARLSRUHE – FAKULTÄT FÜR INFORMATIK
INSTITUT FÜR ALGORITHMEN UND KOGNITIVE SYSTEME
INSTITUT FÜR LOGIK, KOMPLEXITÄT UND DEDUKTIONSSYSTEME

7. Übungsblatt

Abgabe: Montag, 31. Januar 2005, *zu Beginn* der Vorlesung

Besprechung: Donnerstag, 3. Februar 2005, Raum -101, 11:30 Uhr

Vergessen Sie bitte nicht die Online-Befragung!
`http://www.evaluation-online.de`

Aufgabe 1
Wir betrachten die `add`-Methode der Java-Datenstruktur `Vector`.

Die `add`-Methode geht folgendermaßen vor: Anfangs werden konstant viele Speicherstellen zur Verfügung gestellt. Wird nun ein Element eingefügt, so besetzt es die erste freie Stelle, falls die Speicherkapazität damit nicht überschritten wird. Ansonsten wird die Speicherkapazität verdoppelt, und die bisherigen Einträge werden an eine freie Stelle im Speicher kopiert. Der neue Eintrag wird nun wie im Normalfall an die erste freie Stelle geschrieben.

Zeigen Sie, dass eine Folge von n Aufrufen der `add`-Methode $O(n)$ Zeit benötigt. Mit anderen Worten: Ein `add`-Aufruf benötigt amortisiert konstante Zeit.

Aufgabe 2
Sei $M \subset \mathbb{R}_+^2$ eine Menge von Punkten und $p, q \in M$. Der Punkt $p = (x_p, y_p)$ *dominiert* $q = (x_q, y_q)$, falls $p \neq q$ sowie $x_p \leq x_q$ und $y_p \leq y_q$ gilt. Ein Punkt aus M, der von keinen anderen dominiert ist, heißt *pareto-optimal* (bzgl. M). Die *Pareto-Menge* $P(M)$ ist die Menge aller bzgl. M pareto-optimalen Punkte. M erfülle nun folgende Bedingung: Für zwei beliebige Punkte $p \neq q$ gelte $x_p \neq x_q$ und $y_p \neq y_q$.

Geben Sie für solche Mengen einen randomisiert-inkrementellen semi-dynamischen Algorithmus an, der $P(M)$ in $O(n \log n)$ Zeit berechnet.

Welche Schritte Ihres Algorithmus müssen geändert werden, falls man auf obige Restriktion verzichtet.

Aufgabe 3
Betrachten Sie den dynamischen Algorithmus zur Bestimmung der Trapezzerlegung aus der Vorlesung. Zu einem Zeitpunkt i sei N_i die Menge aktiver Strecken und n_i deren Kardinalität. Zu jedem beliebigen Zeitpunkt i ist die Anzahl der toten Strecken in $\mathrm{history}(i)$ durch $2n_i$ beschränkt. Dies wird dadurch gewährleistet, dass $\mathrm{history}(i)$ durch Einfügen aller Strecken in N_i neu aufgebaut wird, falls die Bedingung durch die i. Operation verletzt würde.

Sei $a(l)$ die Anzahl der aktiven Strecken zum Zeitpunkt des l. Wiederaufbaus. Sei $\mathrm{span}(l)$ die Anzahl der Operationen zwischen dem l. und dem $(l+1)$. Wiederaufbau. Sei W die Anzahl aller Wiederaufbaus.

(a) Zeigen Sie, dass für $l < W$ gilt $a(l) \leq c\,\mathrm{span}(l)$, wobei $c > 1$ eine Konstante ist.

(b) Der Algorithmus aus der Vorlesung hat eine Laufzeit von $O(I + n \log^2 n)$, wobei n die Anzahl aller eingefügten Strecken und I die Anzahl ihrer Schnitte ist. Allerdings kann beim Einfügen oder Löschen eines einzelnen Segments eine lange Wartezeit entstehen, falls diese Operation einen Wiederaufbau erfordert. Dieses Problem kann wie folgt behoben werden: Für den l. Wiederaufbau wird $\mathrm{history}$ nicht sofort mit allen $a(l)$ Strecken erstellt. Stattdessen wird während der nächsten $a(l)/c \leq \mathrm{span}(l)$ Operationen mit der alten $\mathrm{history}$ weitergearbeitet. Parallel dazu wird eine neue $\mathrm{history}$ aufgebaut, in der pro Operation jeweils eine konstante Anzahl von Änderungen durchgeführt wird. Danach wird die alte $\mathrm{history}$ verworfen und mit der neuen weitergearbeitet. Arbeiten Sie die Details dieses Ansatzes aus. Skizzieren Sie, warum der modifzierte Algorithmus nach wie vor eine Laufzeit von $O(I + n \log^2 n)$ hat.