Technische Universität München
Fakultät für Informatik
Lehrstuhl für Effiziente Algorithmen
Dr. Hanjo Täubig
Jeremias Weihmann

Sommersemester 2014
Übungsblatt 8
9. Juni 2014

Grundlagen: Algorithmen und Datenstrukturen

Abgabetermin: Jeweilige Tutorübung in der Woche vom 16. bis 20. Juni

Tutoraufgabe 1

In dieser Aufgabe betrachten wir Operationen auf einem sortierten Feld $F = [0, ..., n-1]$. Wir nehmen an, dass dieses Feld zu jedem Zeitpunkt maximal $n > 0$ Elemente enthält und zu Beginn bereits bezüglich der Schlüssel der Elemente sortiert ist. Die Schlüssel der Elemente sind der Einfachheit halber natürliche Zahlen.

Auf F sollen nun folgende Operationen ausgeführt werden: `insert(e)` (Füge Element e ein), `remove(k)` (Entferne Element mit Schlüssel k) und `find(k)` (Gib Element mit Schlüssel k aus).

Nehmen Sie an, es gibt ein weiteres Feld $B = [0, ..., \lceil\sqrt{n}\rceil - 1]$ der Größe $\lceil\sqrt{n}\rceil$. Die Operationen `insert(e)` und `remove(k)` auf F werden *lazy* bearbeitet: solange das Feld B noch nicht voll ist, merken wir uns die Operationen der Reihe nach im Feld B, anstatt diese Operationen sofort auf F anzuwenden – das Feld F bleibt also unverändert. Wir nehmen an, dass eine solche Einfügeoperation auf B konstante Laufzeit $\mathcal{O}(1)$ hat. Sobald das B Feld aber voll ist, muss ein Update auf F ausgeführt werden, bei welchem alle in B gespeicherten Operationen auf das Feld F angewendet werden und B geleert wird. Natürlich soll das Feld danach wieder sortiert sein.

Wir nehmen an, ein solches Update hat Laufzeit maximal cn für eine (genügend groß gewählte) Konstante c. Die `find()` Operation sucht immer zuerst in F nach einem Element; danach wird in B überprüft, ob das Element bereits gelöscht oder noch nicht in F eingefügt wurde.

Zeigen Sie, dass die amortisierte Laufzeit für die Operationen `insert(e)`, `remove(k)` und `find(k)` durch diese *lazy* Bearbeitung durch $\mathcal{O}(\sqrt{n})$ beschränkt ist.

Tutoraufgabe 2

Führen Sie auf einem anfangs leeren Binomialheap folgende Operationen aus und stellen Sie die Zwischenergebnisse graphisch dar:

- build({15, 20, 9, 1, 11, 4, 13, 17, 42, 7}),
- delMin(),
- delMin(),
- delMin(),

- delMin()

Die build-Operation ist dabei durch iterative Ausführung von der insert-Operation auf den Elementen realisiert. Die insert-Operation wiederum ist eine merge-Operation auf dem zu erstellenden Binomialheap und einem einelementigen Binomialheap, wobei letzterer das einzufügende Element enthält.

Zusatzaufgabe 1

Zeigen Sie, dass es für jedes $b \geq 1$ einen fast vollständigen echten Binärbaum mit b Blättern gibt.

Hausaufgabe 1

In der Tutoraufgabe haben wir die *lazy* Bearbeitung eines Feldes der Größe n mittels einem Buffer der Größe $\lceil \sqrt{n} \rceil$ betrachtet. Wir haben mittels amortisierter Analyse gezeigt, dass die Laufzeit für die Operationen `insert`, `find` und `remove` in $\mathcal{O}(\sqrt{n})$ liegt. Dabei haben wir vereinfachend vorausgesetzt, dass ein Update, also das Synchronisieren des Feldes mit dem Buffer, in Zeit maximal cn erfolgen kann, wobei $c > 0$ eine Konstante ist.

(a) Zeigen sie nun etwas allgemeiner: Ein Buffer der Größe k kann mit einem Feld der Größe $n > 0$ immer in $\mathcal{O}(n + \text{sort}(k))$ Schritten synchronisiert werden, wobei sort(k) die Worst-Case-Laufzeit eines beliebigen vergleichsbasierten Sortieralgorithmus zum Sortieren eines Feldes der Größe k ist. Nehmen Sie dabei an, dass das Feld groß genug ist um alle Elemente aufnehmen zu können.

(b) Ist damit die Forderung aus der ersten Tutoraufgabe erfüllt, dass eine Konstante c existiert, sodass die Synchronisierung in Zeit maximal cn durchgeführt werden kann? Begründen Sie Ihre Antwort.

Hausaufgabe 2

Führen Sie auf zwei anfangs leeren Binomialheap folgende Operationen aus und stellen Sie die Zwischenergebnisse graphisch dar:

- build({99,98,97,96,95}) des ersten Binomialheaps,
- build({1,2,3}) des zweiten Binomialheaps,
- merge der beiden Binomialheaps,
- delMin(),
- delMin(),
- delMin()

Die build-Operation ist dabei durch iterative Ausführung von der insert-Operation auf den Elementen realisiert. Die insert-Operation wiederum ist eine merge-Operation auf dem zu erstellenden Binomialheap und einem einelementigen Binomialheap, wobei letzterer das einzufügende Element enthält.

Hausaufgabe 3

Implementieren Sie in der Klasse `UIsqsArray` den QuickSort-Algorithmus, in der Funktion `sort`, der die Elemente in dem Feld `A` sortiert.

Verwenden Sie für Ihre Implementierung die auf der übungswebseite bereitgestellten Klassen und verändern Sie für Ihre Implementierung *ausschließlich* die Klasse `UIsqsArray`.

Achten Sie bei der Abgabe Ihrer Aufgabe darauf, dass Ihre Klasse `UIsqsArray` heißt und auf den Rechnern der Linuxhalle (`lxhalle.informatik.tu-muenchen.de`) mit der bereitgestellten Datei `main_qs` kompiliert werden kann. Anderenfalls kann eine Korrektur nicht garantiert werden. Achten Sie darauf, dass Ihr Quelltext ausreichend kommentiert ist.

Schicken Sie die Lösung per Email mit dem Betreff `[GAD] Gruppe <Gruppennummer>` an Ihren Tutor.