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

Sommersemester 2012
Übungsblatt 10
25. Juni 2012

Grundlagen: Algorithmen und Datenstrukturen

Abgabetermin: Jeweilige Tutorübung in der Woche vom 2. bis 6. Juli

Tutoraufgabe 1

Wir betrachten die Anzahl der Blocktransfers die beim externen Sortieren auftreten. Diese ist in der Vorlesung mit $\mathcal{O}(\frac{n}{B} \log_{M/B} \frac{n}{M})$ hergeleitet worden.

(a) Zeigen Sie: Der Algorithmus für das externe Sortieren muss unter normalen Umständen (realistische Werte für M und B) für noch nicht einmal jedes Element eine Lese- oder Schreiboperation ausführen.

Gehen Sie dabei wie folgt vor: Gehen Sie von der asymptotischen Anzahl der Blocktransfers aus und geben Sie eine Bedingung für n an, so dass mindestens n Lese- oder Schreiboperation auf dem Hauptspeicher ausgeführt werden müssen. Argumentieren Sie dann warum diese Bedingung meist nicht zu erfüllen ist, oder wie M und B zu wählen ist, damit diese Bedingung realistisch erfüllbar ist.

(b) Man kann für externes Sortieren eine untere Schranke von $\Omega(\frac{n}{B} \log_{M/B} \frac{n}{B})$ benötigten Blocktransfers zeigen. Ist damit der in der in der Vorlesung gezeigte Algorithmus asymptotisch optimal?

Tutoraufgabe 2

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

- build({15, 20, 9, 1, 11, 8, 4, 13, 17}),
- insert(7),
- delMin(),
- decreaseKey(20, 3),
- delete(8).

Tutoraufgabe 3

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

Analysieren Sie die Anzahl der Element-Vergleiche, die der in der Vorlesung angegebene `siftDown`-Algorithmus im Worst-Case benötigt.
Geben Sie ausgehend von dem Algorithmus in der Vorlesung einen Algorithmus an, der mit $\log n + \mathcal{O}(\log \log n)$ Vergleichen auskommt.

Hausaufgabe 1

Implementieren Sie in der Klasse `UIaHeap` einen adressierbaren binären Heap, der neben den Basisoperationen auch die folgenden Operationen bereitstellt:

- `insertH(`e`)`: Fügt ein Element e in den Heap ein und gibt ein Integer als Handle zurück
- `removeH(`h`)`: Entfernt das Element mit Handle h
- `decreaseKeyH(`h, k`)`: Verringert das Element mit Handle h auf den Wert k.

Verwenden Sie für Ihre Implementierung die auf der Übungswebseite bereitgestellten Klassen und verändern Sie für Ihre Implementierung *ausschließlich* die Klasse `UIaHeap`.
Achten Sie bei der Abgabe Ihrer Aufgabe darauf, dass Ihre Klasse `UIaHeap` heißt und auf den Rechnern der Rayhalle (`rayhalle.informatik.tu-muenchen.de`) mit der bereitgestellten Datei `main_ah` 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.

Hinweis: Hier ist die Verwendung eines Dynamischen Arrays sinnvoll. Falls Sie die früher benutzte Klasse `UIsArray` benutzen wollen, so fügen Sie den Quellcode Ihrer Abgabe hinzu. Sie können aber auch das in Java implemntierte Dynamische Array benutzen, welches deutlich flexibler ist.

Anmerkung: Handles müssen im Allgemeinen keine Integers sein. Oft ist es sinnvoller, Objekte als Handles zu verwenden, die Referenzen auf die tatsächlichen Daten in der Datenstruktur bzw. deren Wrapper besitzen.

Hausaufgabe 2

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

- build({24, 27, 13, 2, 17, 8, 18, 7, 28, 1}),
- delMin(),
- delMin(),
- decreaseKey(27, 10),
- delete(10).
- insert(11),
- insert(1)

Hausaufgabe 3

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.