Institut für Informatik
Prof. Dr. Heiko Röglin
Tobias Brunsch
Clemens Rösner

universitätbonn

Algorithmen und Berechnungskomplexität I
Wintersemester 2013/14
Abgabe: 03.12.2013, 12:30 Uhr

Übungsblatt 7

Allgemeiner Hinweis zu den Übungsaufgaben in dieser Vorlesung: Wenn nach einem Algorithmus, der ein bestimmtes Problem löst, gefragt ist, genügt es nicht, unkommentierten Quelltext anzugeben. Sie sollten zusätzlich die Idee Ihres Algorithmus beschreiben und argumentieren, warum er korrekt ist, auch wenn dies nicht explizit gefordert ist. Ist nach der Laufzeit eines Algorithmus gefragt, so gehört zu einer vollständigen Lösung ebenfalls ein Beweis.

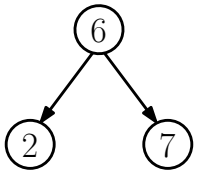
Aufgabe 7.1

4+2 Punkte

Gegeben sei der in der Abbildung dargestellte AVL-Baum.

(a) Fügen Sie die Schlüssel $9, 10, 8, 1, 4, 5, 3$ in der angegebenen Reihenfolge in den Baum ein. Zeichnen Sie den Baum nach jeder Änderung (einfügen, rotieren).

(b) Löschen Sie anschließend die Schlüssel $6, 5, 1$ in der angegebenen Reihenfolge. Zeichnen Sie den Baum nach jeder Änderung (ggf. vertauschen + löschen, rotieren).



Aufgabe 7.2

1+1+6 Punkte

Wir betrachten eine Datenstruktur, die Mengen von Elementen mit geordneten Schlüsseln verwaltet und die folgenden zwei Operationen anbietet:

- INSERT(x): Fügt das Element x unter dem Schlüssel x.key in die Menge ein.
- FINDSMALLEST(k): Gibt das Element x mit dem k-t kleinsten Schlüssel zurück.

Der Einfachheit halber nehmen wir an, dass beim Aufruf von INSERT(x) kein Element mit dem Schlüssel x.key in der Datenstruktur vorhanden ist und dass die Datenstruktur beim Aufruf von FINDSMALLEST(k) mindestens k Elemente enthält.

Geben Sie jeweils eine Datenstruktur an, bei der die Ausführung von INSERT(x) und FINDSMALLEST(k) im Worst Case

(a) in Zeit $O(1)$ und $O(n)$,

(b) in Zeit $O(n)$ und $O(1)$ bzw.

(c) in Zeit $O(\log n)$ und $O(\log n)$

durchgeführt wird, wenn die Datenstruktur zum Zeitpunkt des Aufrufs n Elemente besitzt.

Aufgabe 7.3

6 Punkte

Wir wollen die Laufzeiten von *Quicksort*, *Mergesort* und einer optimierten Variante von *Radixsort* experimentell miteinander vergleichen. Erzeugen Sie dazu für $n = 2, \ldots, 500$ n-mal hintereinander ein Feld mit n zufälligen Einträgen aus der Menge $\{0, \ldots, 2^{32} - 1\}$ und führen Sie *Quicksort*, *Mergesort* und *Radixsort* auf diesem Feld aus. Bestimmen Sie für jedes n die durchschnittliche Laufzeit der drei Sortierverfahren und stellen Sie diese als Graphen in Abhängigkeit von n dar. Verwenden sie dabei für *Quicksort* die `Partition`-Funktion aus der Vorlesung und für *Radixsort* die folgende Variante: Zerlegen Sie die Binärdarstellung jeder der n 32-Bit-Zahlen von rechts nach links in Blöcke der Länge $r := \lceil \log_2(n) \rceil$ und interpretieren Sie diese Blöcke als Zahlen zwischen 0 und $M - 1$ mit $M := 2^r$, auf die Sie MSort anwenden.

Aufgabe 7.4

4 Punkte

Geben Sie ein auf AVL-Bäumen basierendes Sortierverfahren an, das jedes Feld mit n Einträgen in Zeit $O(n \log n)$ sortiert. Verwenden Sie kein anderes Sortierverfahren als Baustein.

Aufgabe 7.5

6 Zusatzpunkte

In einem Algorithmus soll ein Integer-Feld der Länge n für eine sehr große natürliche Zahl n verwendet werden. Dieses sei fortlaufend in einem riesigen Speicher angelegt. Wir betrachten ein typisches Szenario, in dem Speicherzugriffe sehr langsam sind und wir gehen davon aus, dass der Algorithmus nur einen Bruchteil ($o(n)$ Einträge) des Feldes benutzen wird. Die Indizes, auf die der Algorithmus zugreifen wird, sind jedoch nicht a priori bekannt und können beliebig über das gesamte Feld verteilt sein. Das Feld ist am Anfang uninitialisiert und zu Beginn steht beliebiger „Bitmüll“ in den Einträgen. Gesucht ist eine Datenstruktur zur Verwaltung des Feldes, die in konstanter Zeit initialisiert werden kann und die die folgenden Methoden bereitstellt.

- Write(i, v) schreibt den Wert v an Position i des Feldes und markiert Eintrag i als gesetzt.
- Delete(i) markiert Eintrag i als ungesetzt.
- Read(i) liest Eintrag i aus. Ist dieser nicht als gesetzt markiert (also Bitmüll), wird ein Fehler ausgegeben. Ansonsten wird Eintrag i zurückgeliefert.

Beschreiben Sie den Aufbau und die Funktionsweise einer Datenstruktur, die obige Methoden bereitstellt und bei der jede Methode nur $O(1)$ Zeit benötigt. Quelltext ist nicht nötig. Eine schematische Darstellung mit Erklärungen genügt. Sie können dabei annehmen,

- dass das Lesen und Schreiben einer Speicheradresse in Zeit $O(1)$ möglich ist,
- dass in jeder Einheit des Speichers eine beliebig große Zahl gespeichert werden kann,
- dass beliebig viel zusammenhängender Speicher in Zeit $O(1)$ allokiert werden kann und
- dass genügend freier Speicher vorhanden ist.

Begründen Sie, warum Ihre Datenstruktur den Anforderungen genügt.