Aufgabensammlung zum Buch „Algorithmen und Datenstrukturen“ (Kapitel 6)

1 Aufgaben aus dem Buch

Zu folgenden Aufgaben, die direkt aus dem Buch entnommen sind, gibt es an der Universität Freiburg am Lehrstuhl Ottmann Musterlösungen. In der Version mit Lösungen sind diese angegeben. Hinter der fortlaufenden Aufgabennummer steht in Klammern die Nummer der Aufgabe im Buch.

Aufgabe 1 (Aufgabe 6.6):

Bei der Kompressionsmethode zur Pfadverkürzung haben nach Erledigung einer Operation *Find*(x) alle ursprünglich auf dem Pfad von x zur Wurzel des Baumes gelegenen Knoten die Entfernung 1 zur Wurzel. Entwerfen und implementieren Sie eine Pfadverkürzungsmethode, bei der diese Entfernung höchstens 2 beträgt, bei der man aber den Pfad von x zur Wurzel nur einmal durchläuft.

2 Ähnlich Aufgaben

Bei den folgenden Aufgaben handelt es sich um Aufgaben die an der ETH Zürich, am Institut für Theoretische Informatik und an der Universität Freiburg im Institut für Informatik in diversen Vorlesungen gestellt wurden. Inhaltlich sind diese Aufgaben mit dem behandelten Stoff im Buch verwandt. Zu allen Aufgaben gibt es Musterlösungen, die allerdings nur in der Version mit Lösungen enthalten sind.

Aufgabe 2:

Eine Priority Queue (*Vorrang-Warteschlange*) ist eine Verallgemeinerung einer Queue, in der jedem Element eine Priorität zugeordnet wird. Bei der Operation `dequeue(Q)` wird ein Element x aus der Priority Queue entfernt, das die höchste Priorität hat.

Eine Priority Queue mit beschränkter Grösse bietet folgende Operationen an:

init(Q): Initialisiert Q als leere Priority Queue.

enqueue(Q,x,p): Fügt das Objekt x mit der Priorität p in die Priority Queue Q ein. Falls Q bereits voll ist, gibt es eine Fehlermeldung.

dequeue(Q): Liefert ein Objekt x mit der höchsten Priorität in der Priority Queue Q und entfernt es aus Q. Falls Q leer ist, gibt es eine Fehlermeldung.

isempty(Q): Liefert *TRUE*, falls die Priority Queue Q leer ist, andernfalls *FALSE*.

1. Begründe, dass eine Queue eine spezielle Priority Queue ist.
2. Beschreibe eine Datenstruktur, die die Operationen einer Priority Queue (beschränkter Grösse) ermöglicht. Gib an, wie jede der Operationen durchgeführt wird.
3. Gib für die in Teilaufgabe *b)* beschriebene Datenstruktur die worst-case-Laufzeit für jede der vier Operationen an.
4. Schreibe ein Programm, das eine Priority Queue mit der Datenstruktur aus Teilaufgabe *b)* implementiert.

Aufgabe 3:

Beschreibe, wie die Operationen einer Priority Queue mit Hilfe von Heaps realisiert werden können. Welche Laufzeiten haben die einzelnen Operationen?

Aufgabe 4:

Heaps sind einfache Datenstrukturen mit einem hohen Freiheitsgrad bezüglich der Anordnung der Elemente.

1. Zeigen Sie, daß und wie der Pfad zur Stelle n in einem Heap durch die Binärdarstellung von n bestimmt ist.

2. Heaps lassen sich als Vorrangwarteschlangen nutzen. Überlegen Sie, wie man die Datenstruktur anreichern kann, um eine beidseitige Vorrangwarteschlange zu erhalten, die sowohl die Extraktion des Minimum als auch des Maximums in logarithmischer Zeit erlaubt.

3. Die harte Nuß (freiwillige Zusatzaufgabe): Sei $f(n)$ die Anzahl der Heaps mit n paarweise verschiedenen Schlüsseln und s_i die Größe des Teilbaumes zur Wurzel i, $1 \leq i \leq n$. Zeigen Sie, daß $f(n) = n! / \prod_{i=1}^{n} s_i$ gilt. Nutzen Sie dabei die Rekursion $f(n) = \binom{n-1}{|T_1|} f(|T_1|) f(|T_2|)$ mit T_1 bzw. T_2 als der rechte bzw. linke Teilbaum der Wurzel.