Prof. Dr. Leif Kobbelt
Thomas Ströder, Fabian Emmes
Sven Middelberg, Michael Kremer

Informatik 8
RWTH Aachen

Datenstrukturen und Algorithmen (SS 2013)

Übungsblatt 7

Abgabe: Montag, **17.06.2013**, 14:00 Uhr

- Die Übungen sollen in Gruppen von zwei bis drei Personen bearbeitet werden.
- Schreiben Sie die Namen jedes Gruppenmitglieds sowie alle Matrikelnummern auf die abgegebenen Lösungen.
- Schreiben Sie die Namen jedes Gruppenmitglieds sowie alle Matrikelnummern auch in die Quellcode-Dateien.
- Geben Sie Ihre Lösungen am **Anfang** der Globalübung, montags, 14:00 Uhr, ab.
- Schicken Sie den jeweiligen Quellcode bitte per **E-Mail** direkt an Ihre/n Tutor/in.
- Geben Sie außerdem den ausgedruckten Quellcode zusammen mit den schriftlichen Lösungen ab.
- Zu spät abgegebene Lösungen werden nicht bewertet.
- Sofern nicht anders gefordert, müssen alle Lösungen und Zwischenschritte kommentiert werden.

Prof. Dr. Leif Kobbelt
Thomas Ströder, Fabian Emmes
Sven Middelberg, Michael Kremer

Informatik 8
RWTH Aachen

Aufgabe 1 (*Hashing* [**10 Punkte**])

(a) Güte von Hash-Funktionen
Betrachten Sie die folgenden Funktionen, die ein Wort $s = a_1 \dots a_n$ auf einen Hash-Wert zwischen 0 und m abbilden. Dabei sind die a_i als ASCII-Werte gegeben, also $0 \leq a_i \leq 127$ für alle i.

- $h_1(s) = n \mod m$
- $h_2(s) = (\sum_{k=1}^{n} a_k) \mod m$
- $h_3(s) = (\sum_{k=1}^{n} k \cdot a_k^2) \mod m$
- $h_4(s) = ((h_{good}(s)^{p-1} \mod p) \mod m)$

wobei $h_{good}(s)$ eine Hash-Funktion ist, die alle wichtigen an eine Hash-Funktion gestellten Eigenschaften (einfache Berechenbarkeit, Surjektivität und Streuung) erfüllt und p eine Primzahl ist. Erläutern Sie, inwiefern die Funktionen h_i ($1 \leq i \leq 4$) die drei genannten Eigenschaften einer guten Hash-Funktion erfüllen. [2 Punkte]

Hinweis: Wenn Sie sich nicht sicher sind, wie sich die Hash-Funktionen verhalten, ist es ratsam, sie auf ein paar Beispiel-Wörter anzuwenden. Für die vierte Hash-Funktion können Sie dabei beispielsweise die drei anderen Hash-Funktionen anstelle von h_{good} verwenden. Dies dient jedoch lediglich Ihrer Intuition und ist für die Bearbeitung dieser Aufgabe nicht zwingend erforderlich.

(b) Einfaches Hashing
Gegeben sei eine Hash-Tabelle der Größe 19 und die folgenden beiden Hash-Funktionen über der Universalmenge $U = \{0, \dots, 99\}$:

- $h_1(x) =$ Quersumme von x
- $h_2(x) = x \mod 19$

1. Fügen Sie die Werte $12, 99, 21, 76, 23, 30$ sowohl mit h_1 als auch h_2 mittels
   (i) Hashing mit Verkettung
   (ii) Hashing mit linearem Sondieren

   in jeweils eine Tabelle ein (es sind also 4 Tabellen zu erstellen). Geben Sie die nicht-leeren Teile der Tabellen nach jedem Einfügeschritt an. [2 Punkte]
2. Löschen Sie anschließend nacheinander die Werte 21, 12 und 76 aus allen Tabellen aus dem vorigen Aufgabenteil. Geben Sie die nicht-leeren Teile der Tabellen nach jedem Löschschritt an. Erläutern Sie, welches Vorgehen dafür jeweils nötig ist. [2 Punkte]
3. Suchen Sie in den Tabellen, die aus der letzten Teilaufgabe (nach dem Löschen) resultierten, nach dem Wert 12. Erläutern Sie, welches Vorgehen dafür jeweils nötig ist. [1 Punkt]

(c) Doppeltes Hashing
In der Vorlesung haben Sie unter anderem auch doppeltes Hashing kennen gelernt, bei dem eine zusätzliche Hash-Funktion verwendet wird, um Sondierungsschritte durchzuführen. In dieser Teilaufgabe sollen Sie dieses Verfahren und eine Variante davon implementieren. Dazu sollen Sie den von uns bereit gestellten Code vervollständigen. Letzterer enthält drei Klassen. Die Datei `MainClass.java` enthält die Hauptklasse, die zum Testen genutzt werden kann und nicht modifiziert werden soll. Ebenso soll die Datei `Hash.java` nicht modifiziert werden. Diese stellt eine einfache Datenstruktur für Hash-Werte mit einer zusätzlichen Markierung bereit, um angeben zu können, ob ein Hash-Eintrag frei (free), belegt (used) oder entfernt worden (deleted) ist. In der Datei `Hashing.java` soll von Ihnen die Methode `insert2` vervollständigt werden. Die Methode `insert1` implementiert bereits eine Einfüge-Operation nach doppeltem Hashing mit der Sondierungs-Funktion $h(k, 0) = h_1(k) \mod m$ und $h(k, i) = (h_1(k) + h_2(k) + i - 1) \mod m$ für $i > 0$, wobei h_1 und h_2 Hash-Funktionen sind (**Achtung: Die Folien zum doppelten Hashing wurden geändert – die frühere Version** $h(k, i) = (h_1(k) + h_2(k) \times i) \mod m$ **führt zu Nicht-Terminierung der Kollisionsbehandlung, falls** $h_2(k) = 0$). Die von Ihnen zu vervollständigende Methode soll im Gegensatz dazu bei einer Kollision die nächste Sondierungsposition sowohl für das neu einzufügende als auch für das bereits enthaltene Element an der Kollisionsposition berechnen. Sollte das alte Element an seiner neuen Position ohne weitere Sondierungen einzufügen sein, während das neue Element weitere Sondierungen benötigen würde, wird das alte Element an seine neue Position verschoben und das neue Element an der ursprünglichen Position des alten Elementes eingefügt. Ansonsten wird rekursiv versucht, das neue Element an der nächsten Sondierungsposition einzufügen.

Etwas formaler ausgedrückt funktioniert die von Ihnen zu implementierende Hashing-Variante für ein Element k_1 im i-ten Sondierungsschritt wie folgt:

- Ist $h(k_1, i)$ frei, so füge k_1 an dieser Position ein.
- Ist $h(k_1, i)$ belegt mit einem Element k_2, $h(k_1, i + 1)$ ebenfalls belegt, k_2 wurde mit j Sondierungsschritten eingefügt (d. h. $h(k_1, i) = h(k_2, j)$) und $h(k_2, j + 1)$ ist frei, so füge k_2 an der Position $h(k_2, j + 1)$ ein und k_1 an der bisherigen Position von k_2.
- Ansonsten fahre mit der Sondierung für k_1 und $i + 1$ fort.

Die von Ihnen zu ergänzenden Teile sind im Code durch Kommentare markiert. [2 Punkte]

(d) Experimenteller Vergleich
Die Testklasse gibt für das bereits vorgegebene und das von Ihnen ergänzte Hash-Verfahren aus, wie viele Kollisionen beim Einfügen von 500000 zufällig gewählten Elementen in eine anfangs leere Hash-Tabelle der Größe 1000000 entstehen. Messen Sie einige Male die Kollisionszahlen und erklären Sie den gemessenen Unterschied. [1 Punkt]

Aufgabe 2 (*Selektion* [**10 Punkte**])

Gegeben sei eine Folge von n unsortierten Integer-Zahlen.

(a) Entwerfen Sie einen Algorithmus, welcher das kleinste Element der Folge mit $n-1$ Vergleichen findet. Vergleiche, welche nicht die Folgenelemente selbst betreffen, z. B. Vergleiche zwischen den Folgenindizes, werden nicht dazugezählt. Außerdem soll jedes einzelne Element maximal $\lceil \log_2 n \rceil$ mal verglichen werden. Zeigen Sie, dass Ihr Algorithmus höchstens die geforderte Anzahl von Vergleichen benötigt. [5 Punkte]

(b) Erweitern Sie das Verfahren so, dass es das zweitkleinste Element von n Elementen im Worst-Case mit $n + \lceil \log_2 n \rceil - 2$ Vergleichen bestimmt. Zeigen Sie, dass Ihr Algorithmus höchstens die geforderte Anzahl von Vergleichen benötigt. Wenden Sie Ihr Verfahren auf die Elemente

$$10, 5, 3, 2, 8, 7, 1, 6, 9$$

an. Stellen Sie dabei die notwendigen Schritte in geeigneter Form dar. [5 Punkte]