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

Sommersemester 2014
Übungsblatt 5
19. Mai 2014

Grundlagen: Algorithmen und Datenstrukturen

Abgabetermin: Jeweilige Tutorübung in der Woche vom 26. bis 30. Mai

Tutoraufgabe 1

Konstruieren Sie eine statische, perfekte Hashtabelle für die Elemente:

$$(16,10,11) \quad (8,2,15) \quad (7,12,8) \quad (1,10,3) \quad (13,11,14) \quad (6,11,14)$$
$$(7,3,16) \quad (2,2,8) \quad (10,5,15) \quad (7,3,14) \quad (2,10,1) \quad (14,11,6)$$

Jedes Element x besteht aus den Stellen (x_0, x_1, x_2). Verwenden Sie jeweils passend eine der Hashfunktionen:

$$\left(\sum_{i=0}^{2} 2^i x_i\right) \bmod 17$$
$$\left(\sum_{i=0}^{2} a_i x_i\right) \bmod 7 \text{ mit } \mathbf{a} = (0,0,1) \text{ oder } \mathbf{a} = (6,6,2)$$
$$\left(\sum_{i=0}^{2} a_i x_i\right) \bmod 3 \text{ mit } \mathbf{a} = (1,0,0) \text{ oder } \mathbf{a} = (0,2,2).$$

Zur Erinnerung: Beim statischen perfekten Hashing wird in der ersten Stufe so lange eine Hashfunktion $h : \text{Key} \to \{0, \ldots, \lceil\sqrt{2}cn\rceil - 1\}$ aus einer c-universellen Familie $H_{\lceil\sqrt{2}cn\rceil}$ gezogen, bis für die gezogene Hashfunktion die Zahl $C(h)$ der Kollisionen maximal $\sqrt{2}n$ beträgt. Letzteres ist eine hinreichende Voraussetzung dafür, dass die zweite Stufe in erwartet linearer Laufzeit ablaufen kann. Die Laufzeit für die erste Stufe ist ebenfalls erwartet linear.

Alle Schlüssel, die durch h auf dieselbe Position abgebildet werden, gehören zum selben Bucket. Wenn b_ℓ Schlüssel zu Bucket B_ℓ mit $\ell \in \{0, \ldots, \lceil\sqrt{2}cn\rceil - 1\}$ gehören, dann hat das Bucket die Größe $m_\ell = cb_\ell(b_\ell - 1) + 1$.

In der zweiten Stufe wird für jedes Bucket B_ℓ so lange eine Hashfunktion h_ℓ einer c-universellen Familie H_{m_ℓ} gezogen, bis h_ℓ die Schlüssel von Bucket B_ℓ injektiv abbildet.

Bei dieser Aufgabe sind die Hashfunktionen bereits gegeben und müssen nicht erst gezogen werden.

Tutoraufgabe 2

Sortieren Sie die Zahlenfolge $523, 126, 67, 1, 500, 34, 21, 229, 9, 123, 13$ mit MergeSort. Geben Sie für jede Rekursionsebene jeweils für das Aufspalten der Teilsequenzen und für das Verschmelzen der sortierten Teilsequenzen einen Zwischenschritt an (d.h. bei dieser Eingabesequenz insgesamt circa acht Zwischenschritte), sodass Ihr Vorgehen nachvollzogen werden kann.

Wie viele Rekursionsebenen gibt es im Allgemeinen bei MergeSort (wobei wir den initiale Aufruf von MergeSort nicht als eigene Rekursionsebene zählen)? In welcher Größenordnung liegt asymptotisch der Aufwand für jede Rekursionsebene?

Tutoraufgabe 3

Ein Hotelmanager hat n Buchungen für die nächste Saison. Sein Hotel hat k identische Räume. Die Buchungen enthalten ein Ankunfts- und ein Abreisedatum. Er will herausfinden, ob er zu allen Zeiten genügend Räume für die Buchungen zur Verfügung hat. Entwickeln Sie einen Algorithmus, der dieses Problem in Zeit $\mathcal{O}(n + \text{sort}(n))$ löst, wobei $\text{sort}(n)$ die Worst-Case-Laufzeit eines beliebigen Algorithmus für das Sortieren von n Zahlen ist.

Zusatzaufgabe 1

Wir betrachten ein Negativbeispiel für eine Hashfunktion h, die auf einem String $s = s_1 \dots s_n$ der Länge n bestehend aus ASCII-Zeichen arbeitet:

$$h(s) := \sum_{i=1}^{n} \text{Anzahl der Einsen in der Binärdarstellung von } s_i.$$

Nehmen Sie an, dass jedes der 256 Zeichen in dem Text gleichwahrscheinlich vorkommt. Begründen Sie, warum Sie die Hashfunktion nicht für geeignet halten.

Hausaufgabe 1

Veranschaulichen Sie Double Hashing. Die Größe der Hash-Tabelle ist dabei $m = 13$. Führen Sie die folgenden Operationen aus.

$$\text{Insert} \quad 15, 2, 12, 16, 28, 9, 23, 4, 7, 13, 8, 1, 5$$

Verwenden Sie die Hashfunktion

$h(k, i) = [h(k) + i \cdot h'(k)] \bmod m$, wobei $h(k) = 5k \bmod m$ und $h'(k) = 1 + (3k \bmod (m-1))$

Die Schlüssel der Elemente sind (im Kontext dieser Aufgabe) die Elemente selbst.

Hausaufgabe 2

Geben Sie ein Verfahren an, um 5 Elemente mit 7 Vergleichen zu sortieren.

Hausaufgabe 3

Wir betrachten eine Datenstruktur mit einer Liste L, auf der die folgenden Operationen definiert sind.

Prozedur pushBack(int i)

```
1 L.pushBack(i) /* Hänge i an das Ende der Liste L. */
```

Funktion findMin(int k)

```
1 int min := ∞
2 int counter := 0
3 while counter < k und L ist nicht leer do
4 |   int nextElement := L.popFront() /* Entferne das erste Element aus L
  |   und weise nextElement den Wert dieses Elements zu */
5 |   if nextElement < min then
6 |   |   min := nextElement
7 |   counter := counter + 1
8 return min
```

Wir nehmen an, dass jede Elementar-Operation (d.h. Zuweisungen, Vergleiche, etc.), die innerhalb von `pushBack` oder `findMin` verwendet wird, konstante Laufzeit hat.

(a) Ermitteln Sie die jeweilige Worst-Case-Laufzeit der Operationen `pushBack` und `findMin` bei einer Operationsfolge der Länge m. Berechnen Sie daraus eine pessimistische Abschätzung für die Worst-Case-Laufzeit von Operationsfolgen der Länge m.

(b) Zeigen Sie mithilfe einer amortisierten Analyse, dass die amortisierte Laufzeit der Operationen `pushBack` und `findMin` konstant (d.h. in $\mathcal{O}(1)$) ist, und bestimmen Sie die Laufzeit für Operationsfolgen der Länge m.

Hausaufgabe 4

Implementieren Sie die Methoden `insert`, `remove` und `find` für Cuckoo-Hashing. Als Hashfunktionen werden Funktionen vom Typ

$$\left(\left(\sum_{j=0}^{k-1} a_j x^j\right) \bmod p\right) \bmod n$$

mit einem Vektor $a = (a_0, ..., a_{k-1})$ und ganzen Zahlen k, p und n verwendet, die bei der Initialisierung übergeben werden. Beachten Sie, dass x hier nur eine ganze Zahl und *kein* Vektor ist. Die Zahl n gibt hier die Größe der Hashtabelle an. Außerdem soll bei der Initialisierung der Hashtabelle ein Wert `max` übergeben werden, der einen Höchstwert für die Anzahl der Verschiebungen angibt (in der Vorlesung sind dies $2 \log n$). Wenn dieser Wert der Verschiebungen erreicht wird, so dürfen Sie das Programm sofort beenden.

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

Achten Sie bei der Abgabe Ihrer Aufgabe darauf, dass Ihre Klasse `DynHash` heißt und auf den Rechnern der Linuxhalle (`lxhalle.informatik.tu-muenchen.de`) mit der bereitgestellten Datei `main_d` 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.