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

Sommersemester 2013
Übungsblatt 6
27. Mai 2013

Grundlagen: Algorithmen und Datenstrukturen

Abgabetermin: Jeweilige Tutorübung in der Woche vom 3. bis 7. Juni

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:

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

Tutoraufgabe 2

Diese Tutoraufgabe ist ein Kurztutorial für die Bankkonto-Methode. Die darauf folgende Tutoraufgabe 3 veranschaulicht die Theorie an einem Beispiel.

Sei T eine Menge von Operationen, und bezeichne $\ell(\tau)$ die Laufzeit einer Operation $\tau \in T$ (diese Laufzeit kann vom aktuellen Zustand des Objekts, auf dem die Operation wirkt, sowie von Argumenten abhängen). Weiterhin bezeichne $\ell(\tau_1, \tau_2, \ldots, \tau_m) := \sum_{i=1}^{m} \ell(\tau_i)$ die Laufzeit der Operationsfolge $(\tau_1, \tau_2, \ldots, \tau_m)$.

Zweck einer amortisierten Analyse ist, eine möglichst gute obere Schranke für $\ell(\tau_1, \tau_2, \ldots, \tau_m)$ zu bestimmen. Bei der Konto-Methode bestimmen wir dazu eine Funktion $\Delta : T \to \mathbb{Z}$, die folgende Eigenschaften besitzt:

(i) Für alle legalen, also ausführbaren Operationsfolgen $(\tau_1, \ldots, \tau_m)$ gilt $\sum_{i=1}^{m} \Delta(\tau_i) \geq 0$.

(ii) Δ ist möglichst gut gewählt.

Was Eigenschaft (ii) bedeutet, wird später noch spezifiziert. Für $\tau \in T$ ist $\Delta(\tau)$ die *Veränderung des Tokenkontos* durch die Operation τ. Wenn $\Delta(\tau) > 0$, dann zahlt die Operation auf das Konto ein, falls $\Delta(\tau) < 0$, dann hebt sie vom Konto ab. $\alpha(\tau) := \ell(\tau) + \Delta(\tau)$ nennen wir dann die *amortisierte Laufzeit* von τ. Entsprechend ist $\alpha(\tau_1, \ldots, \tau_m) := \sum_{i=1}^{m} \alpha(\tau_i)$ die amortisierte Laufzeit der Operationsfolge $(\tau_1, \ldots, \tau_m)$.

Eigenschaft (i) sagt aus, dass das Tokenkonto nie negativ ist. Die Sinnhaftigkeit eines stets nichtnegativen Kontos ergibt sich aus folgender Beobachtung

$$\alpha(\tau_1, \ldots, \tau_m) = \sum_{i=1}^{m} \alpha(\tau_i) = \sum_{i=1}^{m} (\ell(\tau_i) + \Delta(\tau_i)) = \ell(\tau_1, \ldots, \tau_m) + \sum_{i=1}^{m} \Delta(\tau_i) \geq \ell(\tau_1, \ldots, \tau_m)$$

Die amortisierte Laufzeit der Operationsfolge ist also eine obere Schranke für ihre tatsächliche Laufzeit. Jedes einzelne Token steht für eine gewisse konstante Menge an Laufzeit. Damit wird auch klar, was die Tokenzahl auf dem Konto aussagt: Es ist genau der Wert, um den die amortisierte Laufzeit die tatsächliche Laufzeit übersteigt.

Nun zu Eigenschaft (ii). Meistens versucht man, $\max_{\tau \in T}(\alpha(\tau))$ zu minimieren, d.h. die Operation mit der schlechtesten amortisierte Laufzeit soll möglichst geringe amortisierte Laufzeit besitzen. Oft ist es dabei nicht so wichtig, ob beispielsweise $\max_{\tau \in T}(\alpha(\tau)) = \log(m)$ oder $\max_{\tau \in T}(\alpha(\tau)) = 5 \log m$ gilt, solange dieser Wert im asymptotischen Sinne (also insbesondere nach Vernachlässigung konstanter Faktoren) möglichst gut ist.

Wenn dieses Ziel erreicht ist, ist $\mathcal{O}(m \cdot \max_{\tau \in T}(\alpha(\tau)))$ eine oftmals nicht schlechte obere Schranke für die asymptotische Laufzeit von Worst-Case-Operationsfolgen (wobei die Länge m der Operationsfolgen die asymptotische Variable ist). Ein Nebenziel ist häufig, zusätzlich noch die asymptotische Worst-Case-Komplexität der einzelnen Operationen zu minimieren. (Dies ist in vielen Fällen allerdings nicht für alle Operationen gleichzeitig möglich, da möglicherweise eine Operation amortisiert lang dauern muss, wenn eine andere Operation amortisiert kurze Laufzeit besitzt, und umgekehrt.)

Tutoraufgabe 3

Wir betrachten eine FIFO-Queue (first in, first out), die mit Hilfe zweier Listen L_1 und L_2 realisiert wird. Beide Listen können nur an der ersten Listenposition modifiziert werden. Das Entfernen eines Elements von der ersten Listenposition bzw. das Anhängen eines Elements an der ersten Listenposition kostet konstant viel Zeit. Beide Listen sind zu Beginn leer. Die FIFO-Queue bietet die folgenden beiden Operationen an.

Prozedur pushFirst(Element e)

1 Hänge e an den Anfang der Liste L_1.

Funktion popLast

1 Falls L_2 leer ist, führe reverse() aus.
2 Falls L_2 nicht leer ist, entferne das erste Element von L_2 und gib es aus.

Die Hilfsoperation reverse ist wie folgt definiert:

Prozedur reverse

1 **while** L_1 *ist nicht leer* **do**
2 | Entferne das erste Element von L_1 und hänge es an den Anfang von L_2.

(a) Ermitteln Sie die jeweilige Worst-Case-Laufzeit der Operationen `pushFirst` und `popLast` 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 `pushFirst` und `popLast` konstant (d.h. in $\mathcal{O}(1)$) ist. Was bedeutet dies für die Laufzeit von Operationsfolgen der Länge m?

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

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.

Hausaufgabe 2

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

Funktion 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 der innerhalb den Funktionen `pushBack` und `findMin` verwendeten Operationen 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 3

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.