Karlsruher Institut für Technologie — Prof. Dr. Jörn Müller-Quade
Institut für Theoretische Informatik — Björn Kaidel, Sebastian Schlag, Sascha Witt

4. Übungsblatt zu Algorithmen I im SoSe 2017

http://crypto.iti.kit.edu/index.php?id=799
{bjoern.kaidel,sascha.witt}@kit.edu

Musterlösungen

Aufgabe 1 (*Relationen,* $3 + 3 = 6$ *Punkte*)

Gegeben sei eine endliche Teilmenge $M \subseteq \mathbb{N}$ und eine Menge $R_M \subseteq \{(a, b) \ : \ a, b \in M\}$. R_M definiert eine zweistellige Relation über $M \times M$: zwei Zahlen $a, b \in M$ stehen in Relation R_M zueinander, wenn $(a, b) \in R_M$ gilt. Für diese Aufgabe können Sie davon ausgehen, dass $|M| \leq |R_M|$ und weder M noch R_M Duplikate enthalten.
Hinweis: Eine übliche Relation ist beispielsweise „$\leq$".

a) Die Relation R_M ist *symmetrisch*, wenn für jedes $(a, b) \in R_M$ auch $(b, a) \in R_M$ ist. Geben Sie einen Algorithmus an, der in *erwarteter* Laufzeit $\mathcal{O}(|R_M|)$ überprüft, ob R_M symmetrisch ist. Begründen Sie die Laufzeit ihres Algorithmus.

b) Die Relation R_M ist *reflexiv*, wenn für jedes $a \in M$ das Paar $(a, a) \in R_M$ ist. Geben Sie einen Algorithmus an, der in *erwarteter* Laufzeit $\mathcal{O}(|R_M|)$ überprüft, ob R_M reflexiv ist. Begründen Sie die Laufzeit ihres Algorithmus.

Musterlösung:

a) Der Algorithmus funktioniert folgendermaßen: Er speichert zuerst alle Paare $(a, b) \in R_M$ in eine Hashtabelle H mit (a, b) als Schlüssel. Danach geht er alle Paare erneut durch und überprüft für jedes (a, b), ob auch (b, a) in der Hashtabelle enthalten ist. Ist dieser Test für alle Elemente erfolgreich, dann ist R_M symmetrisch.

   foreach $(a, b) \in R_M$ **do**
       H.insert$((a, b))$
   foreach $(a, b) \in R_M$ **do**
       if H.find$((b, a)) = \perp$ **then**
           return false
   return true

   Jede Hashtabellen-Operation hat erwartete Laufzeit $\mathcal{O}(1)$, womit die beiden Schleifen je in erwarteter Laufzeit $\mathcal{O}(|R_M|)$ sind. Die erwartete Gesamtlaufzeit ist somit in $\mathcal{O}(|R_M|)$.

b) Der Algorithmus funktioniert folgendermaßen: Er speichert zuerst alle Paare $(a, b) \in R_M$ in eine Hashtabelle H mit (a, b) als Schlüssel. Danach geht er alle Zahlen a aus M durch und überprüft, ob (a, a) in der Hashtabelle enthalten ist. Ist dieser Test für alle Zahlen erfolgreich, dann ist R_M reflexiv.

   foreach $(a, b) \in R_M$ **do**
       H.insert$((a, b))$
   foreach $a \in M$ **do**
       if H.find$((a, a)) = \perp$ **then**
           return false
   return true

Jede Hashtabellen-Operation hat erwartete Laufzeit $\mathcal{O}(1)$. Die erste Schleife liegt somit in erwarteter Laufzeit $\mathcal{O}(|R_M|)$. Da nach Aufgabenstellung $|M| \leq |R_M|$ gilt, hat auch die zweite Schleife diese erwartete Laufzeit. Somit ist erwartete Gesamtlaufzeit in $\mathcal{O}(|R_M|)$.

Aufgabe 2 (*Hashing,* $2+2+2=6$ *Punkte*)

Gegeben sei eine Hashtabelle mit 11 Einträgen. Dabei sei die Hashfunktion h definiert über $h(x) = x \bmod 11$.

a) Verwenden Sie Hashing mit verketteten Listen und Hashing mit linearer Suche, um folgende Zahlen in der gegebenen Reihenfolge (von links nach rechts) in zwei Hashtabellen (eine Tabelle pro Verfahren) einzufügen:

$$56, 32, 102, 3, 63, 51, 22, 13, 23.$$

Geben Sie jeweils die Tabellen nach dem Einfügen von 3 und nach dem Einfügen von 23 an. Verwenden Sie beim Hashing mit verketteten Listen das Symbol „$\multimap$“, um Nullzeiger darzustellen und „$\rightarrow$“ für Zeiger von einem Element zum nächsten. Verwenden Sie dazu folgendes Schema um die Hashtabelle darzustellen (im Beispiel für die Variante mit verkettete Listen, bei Arrays werden keine Nullzeiger benötigt):

0	1	2	3	4	5	6	7	8	9	10
$\multimap$	$\multimap$	$\multimap$	$\multimap$	$\multimap$	$\multimap$	$\multimap$	$\multimap$	$\multimap$	$\multimap$	$\multimap$

b) Geben Sie die Anzahl der beim Einfügen aller Zahlen aus a) betrachteten Hashtabellenplätze für beide Verfahren an! Begründen Sie ihre Antwort!

c) Wie groß ist der Speicherverbrauch, wenn sowohl Zeiger als auch Element ein Maschinenwort benötigen und mit einem Nullzeiger das Ende einer Liste markiert werden kann?

Musterlösung:

a) Es ist $h(56) = 1$, $h(32) = 10$, $h(102) = 3$, $h(3) = 3$, $h(63) = 8$, $h(51) = 7$, $h(22) = 0$, $h(13) = 2$, $h(23) = 1$.

Nach dem Einfügen von 3

Hashing mit verketteten Listen

0	1	2	3	4	5	6	7	8	9	10
$\multimap$	56 $\multimap$	$\multimap$	3 $\rightarrow$ 102 $\multimap$	$\multimap$	$\multimap$	$\multimap$	$\multimap$	$\multimap$	$\multimap$	32 $\multimap$

Hashing mit linearer Suche:

0	1	2	3	4	5	6	7	8	9	10
	56		102	3						32

Nach dem Einfügen von 23

Hashing mit verketteten Listen ($\multimap$ stellt einen Nullzeiger dar.):

0	1	2	3	4	5	6	7	8	9	10
22 $\multimap$	23 $\rightarrow$ 56 $\multimap$	13 $\multimap$	3 $\rightarrow$ 102 $\multimap$	$\multimap$	$\multimap$	$\multimap$	51 $\multimap$	63 $\multimap$	$\multimap$	32 $\multimap$

Hashing mit linearer Suche:

0	1	2	3	4	5	6	7	8	9	10
22	56	13	102	3	23		51	63		32

b) Beim Hashing mit verketteten Listen muss pro Einfügeoperation immer genau ein Platz in der Hashtabelle betrachtet werden, hier also insgesamt 9 Stück. Bei linearer Suche sind dies unter Umständen mehr, da man bei bereits belegten Plätzen weitersuchen muss - in diesem konkreten Fall werden 14 Plätze betrachtet, da es je zwei Werte mit den Hashwerten 1 und 3 gibt.

c) Hashing mit linearer Suche belegt hier genau ein Maschinenwort pro Tabellenplatz, also 11 Maschinenworte. Beim Hashing mit verketteten Listen gestaltet sich die Sache etwas schwieriger: Hier enthält jeder Tabellenplatz einen (potentiell genullten) Zeiger (belegt also ein Maschinenwort), und jeder Eintrag belegt zwei Maschinenworte (einen für das eigentliche Element, einen für den next-Zeiger). Damit ergeben sich $11 + 2 \cdot 9 = 29$ Maschinenworte Speicherverbrauch.

Aufgabe 3 (*Kollisionsresistenz, 1+5=6 Punkte*)

Hashfunktionen spielen in der Kryptographie eine zentrale Rolle, beispielsweise zum sicheren Speichern von Passwörtern oder zum Signieren digitaler Daten, wie etwa E-Mails oder Webseiten. Um bei diesen Anwendungen Sicherheit bieten zu können, benötigen Hashfunktionen die Eigenschaft der *Kollisionsresistenz*: Umgangssprachlich (und vereinfacht) ist eine Hashfunktion h kollisionsresistent, wenn kein Algorithmus existiert, der mit hoher Wahrscheinlichkeit zwei unterschiedliche Nachrichten M_1 und M_2 berechnen kann, sodass $h(M_1) = h(M_2)$. Ein solches Paar von Nachrichten $M_1 \neq M_2$ nennen wir *Kollision*.
Betrachten Sie die folgende (stark vereinfachte) Definition von Kollisionsresistenz: Eine Hashfunktion h ist kollisionsresistent, wenn für alle Algorithmen $\mathcal{A}$ gilt, dass

$$\Pr[\mathcal{A} \text{ gibt zwei Nachrichten } M_1 \neq M_2 \text{ mit } h(M_1) = h(M_2) \text{ aus.}] \leq \frac{1}{2^{1024}}.$$

a) Aus der Vorlesung ist die folgende universelle Familie bekannt: Es sei p eine Primzahl, Key $x = (x_1, ..., x_k) \in \{0, ..., p-1\}^k$ $(k \in \mathbb{N})$ und $a = (a_1, ..., a_k) \in \{0, ..., p-1\}^k$. Definiere $h_a(x) = \sum_{i=1}^{k} a_i \cdot x_i \bmod p$ und $\mathcal{H}_p := \{h_a \ : \ a \in \{0, ..., p-1\}^k\}$. Geben Sie für $p = 3$ eine Hashfunktion h_a aus $\mathcal{H}_p$ und eine Kollision für h_a an.

b) Enthält die Familie $\mathcal{H}_p$ für jede Primzahl p und alle $k \geq 2$ mindestens eine Hashfunktion h_a, welche die obige Definition der Kollisionsresistenz erfüllt? Beweisen Sie Ihre Antwort!

Musterlösung:

a) Setze $a = (1, 1, 1)$. Dann ist $M_1 = (1, 0, 0)$ und $M_2 = (0, 1, 0)$ eine Kollision, denn es gilt

$$h_a(M_1) = 1 = h_a(M_2).$$

b) Nein, keine Hashfunktionen aus $\mathcal{H}_p$ erfüllt die vereinfachte Definition der Kollisionsresistenz, egal welche Primzahl verwendet wird.

Um dies zu beweisen, sei nun p eine beliebige Primzahl und h_a eine beliebige Hashfunktion aus H_a. Wir geben einen Algorithmus $\mathcal{A}$ an, der mit Wahrscheinlichkeit $1 > 1/2^{1024}$ eine Kollision berechnet. $\mathcal{A}$ wählt zuerst eine beliebige Nachricht $M = (m_1, ..., m_k)$. Für die Nachricht $M' = (m'_1, ..., m'_k)$ setzt $\mathcal{A}$ m'_2 auf einen beliebigen Wert ungleich m_2, die Nachrichtenteile $m'_3, ..., m'_k$ auf beliebige Werte und

$$m'_1 := m_1 + a_1^{-1} \sum_{i=2}^{k} a_i(m_i - m'_i).$$

M und M' sind eine Kollision, denn es gilt

$$\begin{aligned}
h_a(M') &= \sum_{i=1}^{k} a_i \cdot m'_i \\
&= a_1 \cdot m'_1 + \sum_{i=2}^{k} a_i \cdot m'_i \\
&= a_1 \cdot (m_1 + a_1^{-1} \sum_{i=2}^{k} a_i(m_i - m'_i)) + \sum_{i=2}^{k} a_i \cdot m'_i \\
&= a_1 \cdot m_1 + \sum_{i=2}^{k} a_i(m_i - m'_i + m'_i) \\
&= h_a(M).
\end{aligned}$$

Der Algorithmus $\mathcal{A}$ gibt also mit Wahrscheinlichkeit 1 eine Kollision aus. Da wir keine Einschränkungen an die Hashfunktion gestellt haben, ist somit keine der Hashfunktionen aus der universellen Familie H kollisionsresistent, was die Behauptung beweist.

Anmerkung: Die hier vorgestellte Definition der Kollisionsresistenz ist vereinfacht und wesentlich abgeschwächt, gibt aber einen Einblick darin, wie die richtige Definition aussieht. Universelle Hashfunktionen aus H_a erfüllen auch den richtigen Kollisionsresistenzbegriff nicht (die Argumentation ist exakt die gleiche, wie in der Musterlösung). Mehr über kollisionsresistente Hashfunktionen und Kryptographie können Sie beispielsweise in der Vorlesung „Sicherheit“ lernen.