13 Hashing

13.1 Grundlagen

Gegeben ist ein Universum M der Kardinalität $|M| = m$. Davon soll eine Untermenge $S \subseteq M$ von Schlüsseln verwaltet werden. Dies soll mit Hilfe einer Hashtabelle T geschehen. Ihre Einträge werden mit Indizes aus einer Menge N mit $|N| = n < m$ adressiert. Dazu wird eine Hashfunktion $h : M \to N$ benutzt. Die Idee besteht darin, die zu einem Schlüssel $s \in S$ gehörende Information über den Eintrag $T[h(s)] = (s, \ldots)$ zugänglich zu machen.

13.1 Definition Eine Hashfunktion heißt *perfekt* für eine Schlüsselmenge S, falls für alle $x, y \in S$ gilt: $x \neq y \implies h(x) \neq h(y)$. ◇

Ist $|S| > |N|$, dann muss es nach dem Schubfachprinzip zu einer *Kollision* kommen, d. h. es existieren verschiedene Schlüssel x und y mit $h(x) = h(y)$ und eine perfekte Hashfunktion kann in diesem Fall nicht existieren.

Ist $|M| > |N|$, dann kann es keine Hashfunktion geben, die für alle Schlüsselmengen perfekt ist.

Zur Auflösung von Kollisionen gibt es verschiedene Strategien. Eine Methode besteht darin, jeweils alle Schlüssel mit gleichem Hashwert in einer verketteten Liste zu speichern. Eine Alternative ist es, solche Schlüssel mit einer weiteren Hashfunktion auf jeweils eine (kleinere) zweite Hashtabelle abzubilden.

Es gibt mehrere Anwendungen von Hashfunktionen im Bereich randomisierter Algorithmen. Ausgangspunkt ist im Allgemeinen eine ganze sogenannte universelle Familie H von Hashfunktionen. Das ist Thema der Abschnitte 13.2 und 13.3.

13.2 Beim Entwurf randomisierter Algorithmen wird mitunter aus H zufällig eine Hashfunktion ausgewählt und benutzt. Das werden wir beim dynamischen und beim statischen Wörterbuchproblem in den Abschnitten 13.4 bzw. 13.5 genauer sehen.

Eine weitere Anwendung für universelle Familien von Hashfunktionen ist die Einsparung von Zufallsbits bei randomisierten Algorithmen. Darauf werden wir in Abschnitt 13.6 zu sprechen kommen.

13.2 Universelle Familien von Hashfunktionen

13.3 Definition Es seien M und N wie oben mit $|M| = m \geqslant |N| = n$. Eine Familie H von Hashfunktionen $h : M \to N$ heißt 2-*universell*, wenn für alle $x, y \in M$ mit $x \neq y$ gilt: Für ein zufällig gleichverteilt aus H ausgewähltes h ist $\mathbf{P}(h(x) = h(y)) \leqslant 1/n$. ◇

Man beachte, dass $1/n$ auch die Wahrscheinlichkeit ist, dass $h(x)$ und $h(y)$ gleich sind, wenn man sie unabhängig zufällig gleichverteilt wählt.

13.4 Ein einfaches Beispiel einer 2-universellen Familie ist die Menge *aller* Hashfunktionen $H = N^M$. In diesem Fall ist $\mathbf{P}(h(x) = h(y)) = 1/n$, denn zu jedem h mit $h(x) = h(y)$ gibt es $n-1$ andere Hashfunktionen, die sich von h nur an der Stelle y unterscheiden.

Diese Familie von Hashfunktionen hat aber für Anwendungen noch zwei Nachteile. Zum einen benötigt man $\Theta(m \log n)$ Zufallsbits, um ein $h \in H$ auszuwählen, zum anderen ist nicht sichergestellt, dass man jedes h „leicht" berechnen kann.

13.5 Definition Mit den Bezeichnungen wie bisher und für $X, Y \subseteq M$ definieren wir:

$$\begin{aligned}
\delta(x,y,h) &= \begin{cases} 1 & \text{falls } x \neq y \wedge h(x) = h(y) \\ 0 & \text{sonst} \end{cases} \\
\delta(x,y,H) &= \sum_{h \in H} \delta(x,y,h) \\
\delta(x,Y,h) &= \sum_{y \in Y} \delta(x,y,h) \\
\delta(X,Y,h) &= \sum_{x \in X} \delta(x,Y,h) \\
\delta(X,Y,H) &= \sum_{h \in H} \delta(X,Y,h)
\end{aligned}$$

◇

13.6 Ist H eine 2-universelle Familie, so ist für $x \neq y$ stets $\delta(x,y,H) \leqslant |H|/n$. Denn andernfalls müsste bei zufälliger gleichverteilter Wahl eines $h \in H$ von $|H|$ vielen offensichtlich $\mathbf{P}(h(x) = h(y)) > 1/n$ sein.

Die Abschätzung $\delta(x,y,H) \leqslant |H|/n$ für eine 2-universelle Familie ist recht gut, wie das folgende Ergebnis zeigt.

13.7 Satz. *Für jede Familie H von Funktionen $h : M \to N$ existieren x und y aus M, so dass $\delta(x,y,H) \geqslant |H|/n - |H|/m$.*

Vor dem Beweis sei noch die folgende kurze Rechnung eingeschoben.

13.8 Es seien k und g zwei positive ganze Zahlen und $A(k,g) = k(k-1) + g(g-1)$. Für $k \leqslant g-1$ gilt dann $A(k,g) \geqslant A(k+1, g-1)$, denn es ist

$$\begin{aligned}
A(k,g) - A(k+1,g-1) &= k(k-1) + g(g-1) - \big((k+1)k + (g-1)(g-2)\big) \\
&= k^2 - k + g^2 - g - \big(k^2 + k + g^2 - 3g + 2\big) \\
&= -k - g - k + 3g - 2 \\
&= 2(g-1-k) \\
&\geqslant 0\,.
\end{aligned}$$

13.9 Beweis. (von Satz 13.7) Es sei $h \in H$ beliebig. Für $z \in N$ sei $A_z = \{x \in M \mid h(x) = z\}$.
Für $w, z \in N$ gilt:

$$\delta(A_w, A_z, h) = \begin{cases} 0 & \text{falls } w \neq z \\ |A_z|\,(|A_z| - 1) & \text{falls } w = z \end{cases},$$

 © Th. Worsch 2000-2020

da alle Paare (x, y) mit $x \neq y$ genau 1 zur Summe beitragen.

Wegen der Rechnung in Punkt 13.8 wird die Gesamtzahl der Kollisionen minimiert, wenn die Mengen A_z gleich groß sind. Also ist

$$\delta(M, M, h) = \sum_{z \in N} |A_z| \, (|A_z| - 1) \geqslant n \frac{m}{n} \left(\frac{m}{n} - 1 \right) = m^2 \left(\frac{1}{n} - \frac{1}{m} \right) .$$

Folglich ist $\delta(M, M, H) \geqslant |H| \, m^2 \, (1/n - 1/m)$. Daher existieren nach dem Schubfachprinzip $x, y \in M$ mit $\delta(x, y, H) \geqslant \delta(M, M, H)/m^2 \geqslant |H| \, (1/n - 1/m)$. ∎

13.3 Zwei 2-universelle Familien von Hashfunktionen

O. B. d. A. sei nun $M = \{0, \ldots, m-1\}$ und $N = \{0, \ldots, n-1\}$.

In der weiter unten beschriebenen Konstruktion wird eine Primzahl $p \geqslant m$ benötigt. Der folgende Satz, dessen Aussage auch als „Bertrands Postulat“ bezeichnet wird, stellt sicher, dass es stets eine relativ kleine solche Zahl gibt.

13.10 Satz. (Chebyshev) *Zu jedem $m > 1$ gibt es eine Primzahl p mit $m \leqslant p < 2m$.*

13.11 Definition Es seien $m \geqslant n$ und $p \geqslant m$ eine Primzahl. Die Menge $H = \{h_{a,b} \mid a, b \in \mathbb{Z}_p \wedge a \neq 0\}$ von Hashfunktionen ist wie folgt definiert:

$$\begin{array}{rrcl} & h_{a,b}(x) & = & g(f_{a,b}(x)) \\ \text{mit} & f_{a,b} : \mathbb{Z}_p & \to & \mathbb{Z}_p \\ & x & \mapsto & ax + b \bmod p \\ & g : \mathbb{Z}_p & \to & N \\ & x & \mapsto & x \bmod n \end{array}$$

Benutzt wird beim Hashen eigentlich nur die Einschränkung der $h_{a,b} : \mathbb{Z}_p \to N$ auf $M \subseteq \mathbb{Z}_p$. Hierdurch kann die Zahl der Kollisionen offensichtlich nicht größer werden. ◇

Ziel ist nun der Nachweis des folgenden Satzes.

13.12 Satz. *Die Familie H aus Definition 13.11 ist 2-universell.*

13.13 Beweis. Zu zeigen ist, dass bei zufällig gewähltem $h_{a,b} \in H$ gilt: $\mathbf{P}\big(h_{a,b}(x) = h_{a,b}(y)\big) \leqslant 1/n$. Dies kann auch formuliert werden als $\delta(x, y, H) \leqslant |H|/n$.

Der Beweis wird in zwei Schritten geführt. Zunächst wird gezeigt, dass für $x \neq y$ gilt: $\delta(x, y, H) = \delta(\mathbb{Z}_p, \mathbb{Z}_p, g)$. In einem zweiten Schritt wird $\delta(\mathbb{Z}_p, \mathbb{Z}_p, g)$ nach oben abgeschätzt.

1. Es seien x und y so, dass es zu einer Kollision $h_{a,b}(x) = h_{a,b}(y)$ für ein $h_{a,b}$ kommt. Es ist also $g(f_{a,b}(x)) = g(f_{a,b}(y))$.

   Da $a \neq 0$ und p prim ist, sind alle $f_{a,b}$ bijektiv. Also ist $r = f_{a,b}(x) \neq f_{a,b}(y) = s$. Eine Kollision passiert also immer genau dann, wenn $g(r) = g(s)$, d. h. $r \equiv s \pmod{n}$.

   Umgekehrt legt die Wahl von Werten $x \neq y$ und $r \neq s$ *eindeutig* genau eine Hashfunktion $h_{a,b}$ fest, so dass $f_{a,b}(x) = r$ und $f_{a,b}(y) = s$, denn es muss das lineare Gleichungssystem

   $$\begin{array}{rcl} ax + b & \equiv & r \pmod{p} \\ ay + b & \equiv & s \pmod{p} \end{array}$$

 © Th. Worsch 2000-2020

über $\mathbb{Z}_p$ erfüllt werden.

Also ist die Anzahl $\delta(x, y, H)$ der Hashfunktionen, für die x und y zu einer Kollision führen, gleich der Anzahl $\delta(\mathbb{Z}_p, \mathbb{Z}_p, g)$ von Paaren (r, s) mit $r \neq s$ und $r \equiv s \mod n$.

2. Damit bleibt noch abzuschätzen, wieviele solche Paare es höchstens gibt.

   Für $z \in N$ sei dazu $A_z = \{x \in \mathbb{Z}_p \mid g(x) = z\}$. Für jedes z ist $|A_z| \leqslant \lceil p/n \rceil$. Folglich gibt es zu jedem r höchstens $\lceil p/n \rceil$ Werte s, so dass r und s alle Bedingungen erfüllen. Also ist (wegen der Forderung $r \neq s$)

$$\delta(\mathbb{Z}_p, \mathbb{Z}_p, g) \leqslant p\left(\left\lceil \frac{p}{n} \right\rceil - 1\right) = p\left(\left\lfloor \frac{p-1}{n} \right\rfloor + 1 - 1\right) \leqslant \frac{p(p-1)}{n} = \frac{|H|}{n} .$$

■

Wegen der großen Bedeutung von (universellen Familien von) Hashfunktionen gehen wir noch auf eine zweite ein.

13.14 Definition Es sei nun n eine Primzahl. (Falls das zunächst nicht der Fall ist, vergrößere man die Hashtabelle. Satz 13.10 sichert zu, dass man dafür nicht „nicht allzu viel“ zusätzlichen Speicherplatz benötigt.

Ein Schlüssel x werde dargestellt als Folge $\langle x_0, x_1, \ldots, x_r \rangle$ von $r+1$ Werten x_i, für die alle gilt: $0 \leqslant x_i \leqslant n-1$. Analog beschreiben wir Hashfunktionen h_a durch eine Folge $a = \langle a_0, a_1, \ldots, a_r \rangle$ von $r+1$ Werten a_i mit $0 \leqslant a_i \leqslant n-1$. Es sei

$$h_a(x) = \sum_{i=0}^{r} a_i x_i \bmod n .$$

Die interessierende Familie von Hashfunktionen ist die Menge aller n^{r+1} solchen h_a. ◇

13.15 Satz. *Die in Definition 13.14 festgelegte Familie von Hashfunktionen ist 2-universell.*

13.16 Beweis. Es sei $x \neq y$ und etwa $x_0 \neq y_0$. (Der Fall $x_i \neq y_i$ für ein $i > 0$ kann analog behandelt werden.)

Es seien $a_1, \ldots, a_r$ beliebig aber fest. Es ist

$$a_0(x_0 - y_0) = -\sum_{i=1}^{r} a_i(x_i - y_i) \bmod n .$$

Da n prim ist, ist $x_0 - y_0$ invertierbar modulo n und daher a_0 eindeutig bestimmt. Also kollidieren zwei Werte x und y für genau n^r der n^{r+1} möglichen a bzw. h_a. Also ist die Wahrscheinlichkeit für eine Kollision gerade $1/n$. ■

13.4 Das dynamische Wörterbuchproblem

13.17 Bei den Wörterbuchproblemen besteht die Aufgabe darin, für eine Reihe von „Anfragen“ der Form find(x) jeweils festzustellen, ob x in einer Menge von Schlüsseln S enthalten ist. Man betrachtet üblicherweise zwei Problemvarianten:

 © Th. Worsch 2000-2020

1. das *dynamische Wörterbuchproblem*: Hier liegt S nicht von vorneherein fest, sondern ergibt sich durch Operationen INSERT(x) und DELETE(x), die zwischen die FIND-Operationen eingestreut sind.
2. das *statische Wörterbuchproblem*: Die Menge S ist hier von vorneherein festgelegt. Die Aufgabe besteht darin, eine Folge von Anfragen FIND(x_1), FIND(x_2), usw. zu bearbeiten und jedes Mal festzustellen, ob x_i in der Hashtabelle eingetragen ist.

Die grundsätzliche Idee für randomisierte Algorithmen zur „Wörterbuchverwaltung“ besteht darin, aus einer Familie H von Hashfunktionen zu Beginn zufällig ein h auszuwählen und damit zu arbeiten. Bei geeigneter Wahl von H sind nicht nur die $h \in H$ leicht zu berechnen und mit wenigen Zufallsbits auszuwählen, sondern die Wahrscheinlichkeit für Kollisionen kann auch auf ein annehmbares Maß reduziert werden.

Wir wollen zunächst die folgende naheliegende Vorgehensweise für das dynamische Wörterbuchproblem analysieren.

13.18 ALGORITHMUS. Zu Beginn wird aus einer 2-universellen Familie H zufällig gleichverteilt eine Hashfunktion h ausgewählt und dann bei der Abarbeitung einer (vorher unbekannten) Liste $R = R_1, R_2, \ldots, R_r$ von Operationen benutzt.

Bei einer INSERT-Operation für Schlüssel x wird er in der Hashtabelle an Stelle $h(x)$ eingefügt. Im Falle einer Kollision wird eine lineare verkette Liste aufgebaut.

Bei einer FIND-Operation wird in der Hashtabelle an Stelle $h(x)$ (bzw. ggf. in der dort angehängten linearen Liste) nach x gesucht.

13.19 Ist zu einem Zeitpunkt S die Menge der gerade gespeicherten Schlüssel und soll ein weiterer Schlüssel x eingefügt werden, so ist $\delta(x, S, h)$ die Länge der lineare Liste, in die x eingefügt wird.

Den Erwartungswert für ihre Länge wollen wir nun abschätzen.

13.20 LEMMA. Für alle $x \in M$ und $S \subseteq M$ und ein zufällig gewähltes h aus einer 2-universellen Familie H gilt:

$$\mathbf{E}\,[\delta(x, S, h)] \leqslant \frac{|S|}{n}\,.$$

13.21 BEWEIS.

$$\mathbf{E}\,[\delta(x, S, h)] = \sum_{h\in H} \frac{\delta(x, S, h)}{|H|} = \frac{1}{|H|} \sum_{h\in H} \sum_{y\in S} \delta(x, y, h) = \frac{1}{|H|} \sum_{y\in S} \delta(x, y, H) \leqslant \frac{1}{|H|} \sum_{y\in S} \frac{|H|}{n} = \frac{|S|}{n}\,.$$

Dabei rührt das „$\leqslant$“ von der Überlegung in Punkt 13.6 her. ∎

13.22 SATZ. *Mit den Bezeichnungen wie in Algorithmus 13.18 ist der Erwartungswert für den Gesamtzeitaufwand $t(R, h)$ zur Abarbeitung einer Liste R mit s INSERT-Operationen bei zufällig gewähltem $h \in H$*

$$\mathbf{E}\,[t(R, h)] \leqslant r\left(1 + \frac{s}{n}\right)\,.$$

13.23 BEWEIS. $\mathbf{E}\,[t(R, h)]$ ist kleiner oder gleich dem r-fachen des maximal zu erwartenden Zeitbedarfs für die Bearbeitung einer Operation. Dieser ist im wesentlichen beschränkt durch den Erwartungswert für die Länge der jeweils zu untersuchenden linearen Liste, der nach Punkt 13.20 maximal s/n ist. ∎

13.24 Der Erwartungswert für den Zeitaufwand pro Operation ist also kleiner oder gleich $1 + s/n$. Kann man sich erlauben, $n \geqslant s$ zu wählen, dann ist der Erwartungswert durch die Konstante 2 beschränkt.

Allerdings kann im schlimmsten Fall immer noch ein Zeitaufwand proportional zu s pro FIND-Operation auftreten.

 © Th. Worsch 2000-2020

13.5 Das statische Wörterbuchproblem

Im Folgenden betrachten wir das statische Wörterbuchproblem. Die zu verwaltende Schlüsselmenge S liegt also von vorne herein fest. Das wird ausgenutzt werden, um vor der Beantwortung von Suchanfragen eine gewisse „Vorverarbeitung“ vorzunehmen, in deren Verlauf die zu verwendende Hashfunktion bestimmt wird.

Wir benutzen wieder die universelle Familie H von Hashfunktionen $h_{a,b}(x) = ax + b$ aus Definition 13.11.

13.25 Aus der Universalität von H folgt, dass bei zufälliger Wahl einer Hashfunktion h der Erwartungswert für die Anzahl von Kollisionen für eine Schlüsselmenge $|S|$ gerade

$$\sum_{x \neq y \in S} \mathbf{P}(h(x) = h(y)) \leqslant \binom{|S|}{2} \cdot \frac{1}{n}$$

ist. Wählt man also z. B. eine Tabelle der Größe $n = |S|^2$, so ist der Erwartungswert kleiner oder gleich $1/2$. Das heißt aber auch, dass mit einer Wahrscheinlichkeit größer oder gleich $1/2$ jenes h injektiv ist.

Wählt man eine Tabelle der Größe $n = |S|$, so ist der Erwartungswert kleiner oder gleich $|S|/2$. Die Wahrscheinlichkeit, dass für ein zufällig gewähltes h die Anzahl der Kollisionen größer oder gleich $|S|$ ist, ist daher nach der Markov-Ungleichung (Satz 4.10) kleiner oder gleich $1/2$.

13.26 Kann man sich eine Hashtabelle der Größe $|S|^2$ leisten, dann kann man alle Anfragen schnell beantworten, indem man in der Vorverarbeitungsphase eine zufällig gewählte Hashfunktion h nach der anderen durch vollständige Überprüfung aller Funktionswerte daraufhin untersucht, ob h injektiv ist. Solche h müssen nach Punkt 13.25 existieren und man findet mit großer Wahrscheinlichkeit auch schnell eines. (Der Zeitbedarf für die Überprüfung jedes h ist in $\Theta(|S|)$.)

Im weiteren soll nun noch der Fall betrachtet werden, dass die Hashtabelle nur eine Größe proportional zu $|S|$ hat. Ziel ist auch in diesem Fall eine Datenstruktur, die jede FIND-Operation in konstanter Zeit erledigen kann, wiederum um den Preis einer Vorverarbeitung. Sie wird aber wie oben nur Zeitbedarf $\Theta(|S|)$ haben.

13.27 (Datenstrukturen für Algorithmus 13.28) Die im Folgenden dargestellte Idee, Hashtabellen zweistufig zu benutzen, stammt von Fredman, Komlós und Szemerédi (1984).

Es gibt eine primäre Hashtabelle N. Für jedes $i \in N$ gibt es eine sekundäre Hashtabelle N_i mit einer separat zu bestimmenden Hashfunktion h_i. In $N[i]$ werden die Parameter gespeichert, die h_i charakterisieren und jeweils einen Verweis auf eine sekundäre Hashtabelle N_i. Deren Größe wird im nachfolgenden Algorithmus so gewählt, dass der Gesamtspeicherbedarf in $\Theta(n)$, also proportional zur Anzahl Schlüssel, bleibt.

13.28 Algorithmus.

$N \leftarrow \langle$*Hashtabelle der Größe* $n = |S|\rangle$
$H \leftarrow \langle$*Hashfamilie für* $N\rangle$

$\langle$*1. Phase: suche primäre Hashfunktion*$\rangle$
repeat
$h \leftarrow \langle$*zufällig aus* H *gewählt*$\rangle$

 © Th. Worsch 2000-2020

```
        for i ← 0 to n − 1 do
            S_i ← ∅
            k_i ← 0
        od
        for each s ∈ S do
            ⟨add s to S_{h(s)}⟩
            k_{h(s)} ← k_{h(s)} + 1
        od
    until Σ_{i<n} (k_i choose 2) < n        ⟨Anzahl Kollisionen kleiner n⟩

⟨2. Phase: suche sekundäre injektive Hashfunktionen⟩
    for i ← 0 to n − 1 do
        N_i ← ⟨Hashtabelle der Größe k_i^2⟩
        H_i ← ⟨Hashfamilie für Tabelle mit k_i^2 Einträgen⟩
        repeat
            h_i ← ⟨zufällig aus H_i gewählt⟩
        until ⟨h_i ist auf S_i injektiv⟩
    od
```

13.29 (Analyse von Algorithmus 13.28)

1. Wegen der letzten Bemerkung in Punkt 13.25 ist der Erwartungswert für die Anzahl der Durchläufe durch die **repeat**-Schleife in der 1. Phase höchstens 2 und somit der Erwartungswert für deren Zeitbedarf in $O(n)$.

2. Analog ist in der 2. Phase wegen der vorletzten Bemerkung in Punkt 13.25 der Erwartungswert für die Anzahl der Durchläufe durch die **repeat**-Schleife jeweils konstant und daher der Zeitbedarf (der durch den für den Test nach **until** bestimmt wird) proportional zu k_i^2.

   Der Gesamtzeitbedarf für die 2. Phase daher größenordnungsmäßig durch $\sum \binom{k_i}{2}$ bestimmt, und diese Größe ist durch die Gesamtzahl Kollisionen, also durch n nach oben beschränkt.

13.6 Derandomisierung

Eine weitere schöne Anwendung von Hashfunktionen ist die *Derandomisierung* randomisierter Algorithmen; damit ist die Verringerung der Anzahl benötigter Zufallsbits gemeint, ohne die erwartete Laufzeit gravierend zu erhöhen. Die totale Elimination von Zufall ist auf diese Weise allerdings bislang nicht generell möglich (obwohl manche Forscher vermuten, dass **P** = **BPP** ist). Für eine vertiefende Darstellung dieses Themas sei auf die Mitschriften einer Vorlesung von Goldreich (2001) und auf den technischen Bericht von Luby und Wigderson (1995) verwiesen.

13.30 Die grundsätzliche Idee wird darin bestehen, aus einer Zahl echter Zufallsbits mit Hilfe von Hashfunktionen mehr Bits zu berechnen, die zwar nicht mehr echt unabhängig voneinander sind, „aber noch so aussehen“, d. h. man weiß über diese Bits noch so viel, dass man sie in Rechnungen/Abschätzungen vorteilhaft einsetzen kann.

 © Th. Worsch 2000-2020

13.31 Definition Zufallsvariablen $X_1, \ldots, X_k$ mit gleichen Definitionsbereich heißen *paarweise unabhängig*, wenn für alle $i \neq j$ und alle x und y gilt:

$$\mathbf{P}\big(X_i = x \wedge X_j = y\big) = \mathbf{P}(X_i = x) \cdot \mathbf{P}\big(X_j = y\big) \ .$$

In diesem Fall ist $\mathbf{E}\left[X_i X_j\right] = \mathbf{E}\left[X_i\right] \mathbf{E}\left[X_j\right]$. ◇

13.32 Bemerkung. Man beachte, dass paarweise Unabhängigkeit eine schwächere Forderung ist als (totale) Unabhängigkeit. Sind etwa X und Y unabhängige Zufallsvariablen mit Wertebereich $\{0,1\}$ und ist Z eine dritte mit $Z = X \oplus Y$, dann sind X, Y, Z paarweise unabhängig, aber (offensichtlich) nicht unabhängig.

In diesem Abschnitt werden wir uns (technisch gesehen) für die Summe paarweise unabhängiger Zufallsvariablen interessieren. Da wir die Ungleichung von Chebyshev anwenden wollen, benötigen wir Informationen über die Varianz solcher Summen.

13.33 Lemma. Es seien $X_1, \ldots, X_k$ paarweise unabhängige Zufallsvariablen. Dann ist

$$\mathbf{var}\left[\sum_i X_i\right] = \sum_i \mathbf{var}\left[X_i\right] \ .$$

13.34 Beweis. Da die Varianz einer Zufallsvariablen $\mathbf{var}[X] = \mathbf{E}\left[X^2\right] - (\mathbf{E}[X])^2$ ist, ergibt sich:

$$\begin{aligned}
\mathbf{var}\left[\sum_i X_i\right] &= \mathbf{E}\left[\left(\sum_i X_i\right)^2\right] - \left(\mathbf{E}\left[\sum_i X_i\right]\right)^2 \\
&= \mathbf{E}\left[\sum_i X_i^2 + \sum_{i\neq j} X_i X_j\right] - \left(\sum_i \mathbf{E}\left[X_i\right]\right)^2 \\
&= \sum_i \mathbf{E}\left[X_i^2\right] + \sum_{i\neq j} \mathbf{E}\left[X_i X_j\right] - \sum_i \left(\mathbf{E}\left[X_i\right]\right)^2 - \sum_{i\neq j} \mathbf{E}\left[X_i\right] \mathbf{E}\left[X_j\right] \\
&= \sum_i \mathbf{E}\left[X_i^2\right] - \sum_i \left(\mathbf{E}\left[X_i\right]\right)^2 + \sum_{i\neq j} \underbrace{\mathbf{E}\left[X_i X_j\right] - \mathbf{E}\left[X_i\right] \mathbf{E}\left[X_j\right]}_{=0} \\
&= \sum_i \left(\mathbf{E}\left[X_i^2\right] - \left(\mathbf{E}\left[X_i\right]\right)^2\right) \\
&= \sum_i \mathbf{var}\left[X_i\right]
\end{aligned}$$

■

13.35 Definition Es seien M und N wie oben mit $|M| = m \geqslant |N| = n$. Eine Familie H von Hashfunktionen $h : M \to N$ heißt *stark 2-universell*, wenn für alle $x, y \in M$ mit $x \neq y$ und für alle $x', y' \in N$ gilt: Für ein zufällig gleichverteilt aus H ausgewähltes h ist $\mathbf{P}(h(x) = x' \wedge h(y) = y') = 1/n^2$. ◇

13.36 Offensichtlich sind stark 2-universelle Familie von Hashfunktionen 2-universell im Sinne von Definition 13.3, denn $\mathbf{P}(h(x) = h(y)) = \sum_{x'\in N} \mathbf{P}(h(x) = x' \wedge h(y) = x') = |N|/n^2 = 1/n$.

Für jedes $x' \in N$ gilt außerdem:

$$\mathbf{P}\big(h(x) = x'\big) = \sum_{y'\in N} \mathbf{P}\big(h(x) = x' \wedge h(y) = y'\big) = 1/n$$

Also ist für eine zufällig gleichverteilt gewählte Hashfunktion aus einer stark 2-universellen Familie

$$\mathbf{P}\big(h(x) = x' \wedge h(y) = y'\big) = \mathbf{P}\big(h(x) = x'\big) \cdot \mathbf{P}\big(h(y) = y'\big)$$

 © Th. Worsch 2000-2020

13.37 Beispiel. Eine leichte Abwandlung der Familie aus Definition 13.11 liefert eine stark 2-universelle Familie von Hashfunktionen. Wir gehen im Folgenden davon aus, dass $m = n = 2^r$ ist. Wörter, die r Bits lang sind, interpretieren wir als Elemente des Körpers $F = GF[2^r]$. Für $a, b \in F$ sei $h_{a,b} : F \to F$ die Hashfunktion mit $h_{a,b}(i) = ai + b$. Die Menge der 2^{2r} solchen Hashfunktionen ist eine stark 2-universelle Familie.

13.38 Für den Rest dieses Abschnittes benutzen wir folgende Bezeichnungen. Wahrscheinlichkeitsraum Ω ist eine stark 2-universelle Familie von Hashfunktionen (die von M nach N abbbilden). Für $i \in M$ sei H_i die Zufallsvariable auf Ω mit $H_i(h) = h(i)$. Die Ausführungen in Punkt 13.36 zeigen, dass diese Zufallsvariablen H_i paarweise unabhängig sind.

Wir führen nun noch ein wenig Notation ein.

13.39 Es sei $L \subseteq A^*$ eine formale Sprache und T eine randomisierte **BPP**- oder **RP**-Turingmaschine für L. Wir fassen im Folgenden T so auf, dass es ein „echtes“ Wort $x \in A^n$ und eine passende Folge $y \in \{0,1\}^r$ von Zufallsbits als zusätzliche Eingabe erhält. Wir schreiben $T(x,y) \in \{0,1\}$ für das Ergebnis der Berechnung. Mit $W_x = \{y \mid T(x,y) = 1\}$ bezeichnen wir die Menge der y, die bezeugen, dass $x \in L$ ist. Außerdem sei $\mu(W_x) = |W_x| / 2^r$.

Im Fall von **RP** gilt also

$$\begin{aligned} x \in L &\Rightarrow \mu(W_x) > 1/2 \\ x \notin L &\Rightarrow \mu(W_x) = 0 \end{aligned}$$

und im Fall von **BPP** gilt

$$\begin{aligned} x \in L &\Rightarrow \mu(W_x) > 3/4 \\ x \notin L &\Rightarrow \mu(W_x) < 1/4 \end{aligned}$$

Außerdem ist in beiden Fällen r polynomiell in n.

13.40 Definition Eine formale Sprache $L \subseteq A^*$ ist in $\mathbf{P/poly}$, wenn es eine deterministische Polynomialzeit-Turingmaschine $T(x,y)$ und ein Polynom $r(n)$ gibt mit der Eigenschaft:

$$\forall n\ \exists y \in \{0,1\}^{r(n)}\ \forall x \in A^n\ : \begin{cases} x \in L \Rightarrow T(x,y) = 1 \\ x \notin L \Rightarrow T(x,y) = 0 \end{cases} \qquad \diamond$$

13.41 Satz.

$$\mathbf{RP} \subseteq \mathbf{P/poly} \qquad \textit{und} \qquad \mathbf{BPP} \subseteq \mathbf{P/poly}$$

Das bedeutet, dass es für jede Wortlänge n *ein einzelnes Wort* y gibt, das für alle Eingaben der Länge n so viel „Hilfestellung“ gibt, dass man deterministisch in Polynomialzeit das Wortproblem entscheiden kann. Könnte man diese „advice strings“ in Polynomialzeit berechnen, wäre $\mathbf{P/poly} = \mathbf{P}$. Das kann man aber eben im Allgemeinen nicht.

Wir beschränken uns im Folgenden auf den Beweis für den Fall **RP**, der von Adleman (1978) stammt.

13.42 Beweis. Sei etwa T eine **RP**-Turingmaschine, die eine Sprache L akzeptiert. Es sei n eine Wortlänge und $r = r(n)$ für ein Polynom $r(n)$. Man betrachte die Matrix M, von der man sich vorstelle, dass ihre Zeilen mit den $|A|^n$ Wörtern $x \in A^n$ indiziert werden und ihre Spalten mit allen 2^r

 © Th. Worsch 2000-2020

möglichen Folgen von r (Zufalls-)Bits. Der Eintrag M_{xy} sei 1 oder 0, je nach dem, ob $y \in W_x$ ist oder nicht.

Als erstes entferne man die Zeilen von M, die zu Wörtern gehören, die nicht in L sind. Die entstehende Matrix heiße M_0. In jeder ihrer Zeilen ist mindestens die Hälfte aller Einträge 1.

Wenn in einer Matrix in jeder Zeile mindestens die Hälfte aller Einträge 1 ist, dann ist insgesamt mindestens die Hälfte aller Einträge 1, und dann muss in mindestens einer Spalte mindestens die Hälfte aller Einträge 1 sein. Sei y_0 der Index dieser Spalte.

Zu gegebenen M_i und y_i sei allgemein M_{i+1} die Matrix, die aus M_i entsteht, wenn man Spalte y_i streicht und auch alle Zeilen, die in der gestrichenen Spalte eine 1 hatten. Falls in y_i mindestens die Hälfte aller Einträge 1 war, wird mindestens die Hälfte aller Zeilen von M_i gestrichen und in M_{i+1} hat wieder jede Zeile die Eigenschaft, dass mindestens die Hälfte aller Einträge 1 ist.

Also hat man nach spätestens $\log_2 |A|^n = cn \in O(n)$ Iterationen alle Zeilen gestrichen.

Eine „normale“ deterministische Turingmaschine D, die mit $y = y_0 \cdots y_{cn}$ als Hilfestellung jedes Wort der Länge n in Polynomialzeit auf Zugehörigkeit zu L überprüft, kann wie folgt arbeiten: Zu gegebener Eingabe x bestimmt D zunächst $n = |x|$ und zerlegt y in Teilwörter y_i passender Länge. Anschließend simuliert D alle $T(x, y_i)$ bis einmal akzeptiert wurde und akzeptiert dann auch, oder lehnt x ab, falls kein $T(x, y_i)$ akzeptiert.

Die Länge von y ist polynomiell in n und mit T arbeitet auch D in Polynomialzeit. ∎

Um die Fehlerwahrscheinlichkeit auf 2^{-k} zu senken, haben wir in vorangegangenen Kapiteln einen gegebenen randomisierten Algorithmus mehrfach (k mal) ausgeführt und jedes Mal unabhängig neue Zufallsbits verwendet, insgesamt also $r \cdot k$ Zufallsbits. Im Folgenden werden wir eine Möglichkeit kennenlernen, bei **BPP**-Algorithmen die Fehlerwahrscheinlichkeit mit Hilfe von $O(r)$ Zufallsbits auf $O(1/k)$ zu reduzieren (sofern $k \leqslant 2^r$ ist).

13.43 Algorithmus. (Chor und Goldreich (1989)) Es sei T eine **BPP**-Turingmaschine für eine formale Sprache L, die für Eingaben der Länge n eine (polynomielle) Zahl r von Zufallsbits benötigt. Es sei H eine stark 2-universelle Familie von Hashfunktionen $h : \{0,1\}^r \to \{0,1\}^r$, etwa die aus Beispiel 13.37. Dann genügen $2r$ Zufallsbits (a, b) um zufällig gleichverteilt ein $h \in H$ auszuwählen.

Der neue Algorithmus arbeitet dann so:

- Wähle zufällig ein $h_{a,b}$ aus H aus.
- Berechne für $i = 1, \ldots, k$ jeweils $y_i = h_{a,b}(i)$.
- Zähle, für wieviele i gilt: $y_i \in W_x$.
- Falls das mindestens $k/2$ mal der Fall ist, wird x akzeptiert,
- andernfalls wird x abgelehnt.

13.44 Satz. *Die Fehlerwahrscheinlichkeit von Algorithmus 13.43 ist* $O(1/k)$.

13.45 Beweis. Man betrachte die Zufallsvariablen

$$X_i = \begin{cases} 1 & \text{falls } y_i \in W_x \\ 0 & \text{sonst} \end{cases} .$$

 © Th. Worsch 2000-2020

Sie sind identisch verteilt und paarweise unabhängig, denn

$$\begin{aligned}
\mathbf{P}\big(X_i = a \wedge X_j = b\big) &= \mathbf{P}\big([y_i \in W_x] = a \wedge [y_j \in W_x] = b\big) \\
&= \mathbf{P}([h(i) \in W_x] = a \wedge [h(j) \in W_x] = b) \\
&= \sum_{z \in W_x} \sum_{z' \in W_x} \mathbf{P}\big([h(i) \in \{z\}] = a \wedge [h(j) \in \{z'\}] = b\big) \\
&= \sum_{z \in W_x} \sum_{z' \in W_x} \mathbf{P}(h[(i) \in \{z\}] = a) \cdot \mathbf{P}\big([h(j) \in \{z'\}] = b\big) \\
&= \sum_{z \in W_x} \mathbf{P}([h(i) \in \{z\}] = a) \cdot \sum_{z' \in W_x} \mathbf{P}\big([h(j) \in \{z'\}] = b\big) \\
&= \mathbf{P}([h(i) \in W_x] = a) \cdot \mathbf{P}([h(j) \in W_x] = b) \\
&= \mathbf{P}(X_i = a) \cdot \mathbf{P}\big(X_j = b\big)
\end{aligned}$$

Der Erwartungswert ist $\mu = \mathbf{E}[X_i] = \mu(W_x)$ und die Varianz $\mathbf{var}[X_i] = \mathbf{E}\left[X_i^2\right] - (\mathbf{E}[X_i])^2 = \mathbf{E}[X_i] - (\mathbf{E}[X_i])^2 = \mu(1-\mu)$. Da $0 \leqslant \mu \leqslant 1$, ist $\mathbf{var}[X_i] \leqslant 1/4$. Also ist $\mathbf{var}[\sum X_i] \leqslant k/4$.

Der Algorithmus liefert das falsche Ergebnis, falls

- $x \in L$ und $\sum X_i < k/2$ oder $x \notin L$ und $\sum X_i \geqslant k/2$, d. h.
- $\mu(W_x) > 3/4$ und $\sum X_i < k/2$ oder $\mu(W_x) < 1/4$ und $\sum X_i \geqslant k/2$, d. h.
- $k\mu(W_x) > 3k/4$ und $\sum X_i < k/2$ oder $k\mu(W_x) < k/4$ und $\sum X_i \geqslant k/2$.

Beide Fälle können zu der Form $|\sum X_i - \mathbf{E}[\sum X_i]| > k/4$ zusammen gefasst werden. Nach der Ungleichung von Chebyshev kann man die Fehlerwahrscheinlichkeit abschätzen durch

$$\mathbf{P}\left(\left|\sum X_i - \mathbf{E}\left[\sum X_i\right]\right| > k/4\right) \leqslant \frac{16\,\mathbf{var}[\sum X_i]}{k^2} \leqslant \frac{4}{k}\,.$$

■

Literatur

Adleman, Leonard (1978). „Two Theorems on Random Polynomial Time“. In: *19th Annual Symposium on Foundations of Computer Science (FOCS '78)*. IEEE Computer Society Press, S. 75–83 (siehe S. 119).

Chor, Benny und Oded Goldreich (1989). „On the Power of Two-Point Based Sampling“. In: *Journal of Complexity* 5.1, S. 96–106 (siehe S. 120).

Fredman, Michael L., János Komlós und Endre Szemerédi (1984). „Storing a Sparse Table with O(1) Worst Case Access Time“. In: *Journal of the ACM* 31.2, S. 538–544 (siehe S. 116).

Goldreich, Oded (2001). *Randomized Methods in Computation*. Lecture notes. URL: http://www.wisdom.weizmann.ac.il/~oded/rnd-sum.html (siehe S. 117).

Luby, Michael und Avi Wigderson (1995). *Pairwise Independence and Derandomization*. Technical report TR-95-035. International Computer Science Institute. URL: http://www.icsi.berkeley.edu/icsi/node/2707 (siehe S. 117).

 © Th. Worsch 2000-2020