Algorithmen und Wahrscheinlichkeit

Kapitel 2.8

Randomisierte Algorithmen

Randomisierte Algorithmen

Eingabe I → **Algorithmus $\mathcal{A}$** → Ausgabe $\mathcal{A}(I,\mathcal{R})$

$\updownarrow$

Zufallszahlengenerator

Anfragen der Form:
liefere einen Wert für X,
wobei X verteilt ist wie …

Annahme:
alle Werte, die der Zufallszahlengenerator erzeugt sind unabhängig

Wir beweisen:

(1) **Korrektheit:**

für alle Eingaben I gilt: $\Pr[\mathcal{A}(I,\mathcal{R})$ ist korrekt$] \geq \ \ldots$

Idealer Weise:
W'keit „praktisch“ Eins

(2) **Laufzeit:**

für alle Eingaben I mit Länge $|I|=n$:

$\mathbb{E}[\text{Laufzeit}] = O(f(n))$ und/oder $\Pr[\text{Laufzeit} \leq O(f(n))] \geq \ \ldots$

Las-Vegas Algorithmen:

- geben nie eine falsche Antwort, aber
- Laufzeit ist eine Zufallsvariable

Ziel: $\mathbb{E}[\text{Laufzeit}] = $ „polynomiell“ (in Eingabelänge)

Monte-Carlo Algorithmen:

- Laufzeit immer polynomiell, aber
- geben zuweilen eine falsche Antwort

Ziel: $\Pr[\text{Antwort falsch}] = $ „winzig“

Las-Vegas Algorithmen:

- geben nie eine falsche Antwort, aber
- Laufzeit ist eine Zufallsvariable T

mit: $\mathbb{E}[T]$ = „polynomiell (in Eingabelänge)"

$\Downarrow$ **stoppe Alg nach $2\mathbb{E}[T]$ Schritten**

… wdh 100 Mal

- Laufzeit immer polynomiell, aber
- zuweilen Antwort „???"

Pr[Antwort „???"] $\leq (1/2)^{100}$ (wg Markov Ungleichung)

Las-Vegas Algorithmen:

- geben nie eine falsche Antwort, aber
- Laufzeit ist eine Zufallsvariable T

$$\mathbb{E}[T] = \frac{1}{1-\delta} \cdot poly$$

$\Uparrow$ **while Antwort „???“: repeat**

(Anzahl Versuche: Geo(1-δ))

- Laufzeit immer polynomiell, aber
- zuweilen Antwort „???“

mit: Pr[Antwort „???“] $= \delta$

Monte-Carlo Algorithmen für Entscheidungsprobleme

Einseitiger Fehler:

Pr[Alg antwortet „nein“] = 0 $\forall$ Ja-Instanzen

Pr[Alg antwortet „ja“] $\leqq$ 1-ε $\forall$ Nein-Instanzen

$\Rightarrow$ $\varepsilon^{-1} \ln \delta^{-1}$ Wiederholungen reduzieren Fehler auf

Pr[Antwort falsch] $\leqq \delta$

(Antwort „nein“: wenn mind. ein Aufruf „nein“ ausgibt,
Antwort „ja“: wenn alle Wdh “ja“ ausgeben)

Zweiseitiger Fehler:

Pr[Antwort falsch] $\leqq$ **1/2-ε** $\forall$ Instanzen

$\Rightarrow$ $4\,\varepsilon^{-2} \ln \delta^{-1}$ Wiederholungen reduzieren Fehler

Pr[Antwort falsch] $\leqq \delta$ *(Antwort: Mehrheit der gesehenen Antworten)*

Target-Shooting

Gegeben: zwei Mengen $S \subseteq U$

Aufgabe: bestimme $|S| / |U|$

Annahmen: - *wir können ein Element aus U effizient zufällig gleichverteilt wählen*
- *es gibt eine effizient berechenbare Funktion*

$$\mathbb{I}_S(u) := \begin{cases} 1 & \text{falls } u \in S \\ 0 & \text{sonst} \end{cases}$$

Target-Shooting

1: Wähle $u_1, \ldots, u_N \in U$ zufällig, gleichverteilt und unabhängig

2: **return** $N^{-1} \cdot \sum_{i=1}^{N} \mathbb{I}_S(u_i)$

TARGET-SHOOTING

1: Wähle $u_1, \ldots, u_N \in U$ zufällig, gleichverteilt und unabhängig
2: **return** $N^{-1} \cdot \sum_{i=1}^{N} \mathbb{I}_S(u_i)$

Satz Seien $\delta, \varepsilon > 0$. Falls $N \geq 3\frac{|U|}{|S|} \cdot \varepsilon^{-2} \cdot \log(2/\delta)$, so ist die Ausgabe des Algorithmus TARGET-SHOOTING mit Wahrscheinlichkeit mindestens $1 - \delta$ im Intervall $\left[(1-\varepsilon)\frac{|S|}{|U|}, (1+\varepsilon)\frac{|S|}{|U|}\right]$.

Beweis: Chernoff-Schranke …

Primzahltest

MILLER-RABIN-PRIMZAHLTEST(n)

if $n = 2$ **then**
    return ‘Primzahl’
else if n gerade oder $n = 1$ **then**
    return ‘keine Primzahl’
Wähle $a \in \{2, 3, \ldots, n-1\}$ zufällig und
berechne $k, d \in \mathbb{Z}$ mit $n - 1 = d2^k$ und d ungerade.
$x \leftarrow a^d \pmod{n}$
if $x = 1$ or $x = n - 1$ **then**
    return ‘Primzahl’
repeat $k - 1$ mal
    $x \leftarrow x^2 \pmod{n}$
    if $x = 1$ **then**
        return ‘keine Primzahl’
    if $x = n - 1$ **then**
        return ‘Primzahl’
return ‘keine Primzahl’

Kapitel 2.85

Hashing und Zuordnungsverfahren

Idee: *eine Hashfunktion bildet*
(potentiell sehr grosse) Datenmenge
auf eine
(kleine) natürliche Zahl ab

$$f \quad : \quad \text{Daten} \rightarrow \{0,\ldots,m-1\}$$

Hashing - Beispiel

Idee: *eine Hashfunktion bildet*

(potentiell sehr grosse) Datenmenge

auf eine

(kleine) natürliche Zahl ab

Freunde

Adressbuch

Hashing - Beispiel

Idee: *eine Hashfunktion bildet*

(potentiell sehr grosse) Datenmenge

auf eine

(kleine) natürliche Zahl ab

Freunde

Adressbuch

Hashing - Beispiel

Idee: *eine Hashfunktion bildet*

(potentiell sehr grosse) Datenmenge

auf eine

(kleine) natürliche Zahl ab

f :  → 

Freunde

Adressbuch

Hashing - Beispiel

Idee: *eine Hashfunktion bildet*

(potentiell sehr grosse) Datenmenge

auf eine

(kleine) natürliche Zahl ab

f :

Freunde

$\rightarrow$

Adressbuch

Idee: *eine Hashfunktion bildet*
(potentiell sehr grosse) Datenmenge
auf eine
(kleine) natürliche Zahl ab

$$f \quad : \quad \text{Daten} \rightarrow \{0,\ldots,m-1\}$$

Gewünschte Eigenschaften:

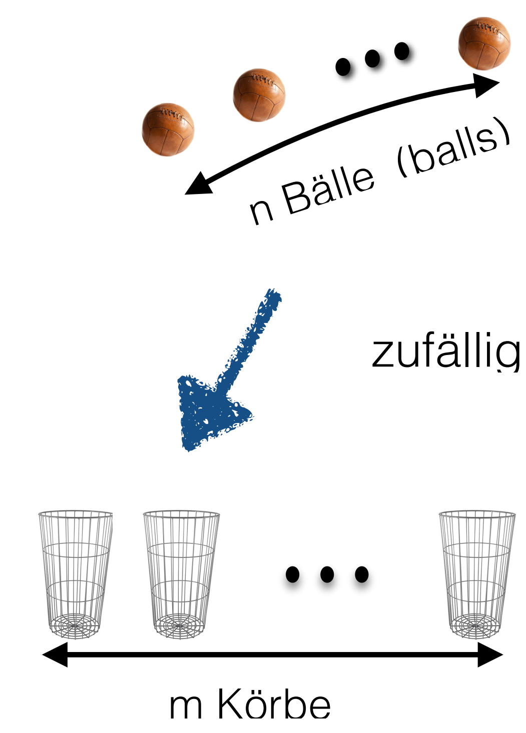
- alle Hashwerte sollen „gleich oft“ vorkommen
- „geringe“ Wahrscheinlichkeit von Kollisionen
- ähnliche Eingaben sollen zu verschieden Ergebnissen führen
- *f soll effizient berechenbar sein*
- [in der Kryptographie] f^{-1} soll nicht effizient berechenbar sein

Anwendungen:

- Datenbanken
- Prüfsummen
- Kryptographie
- Algorithmen für grosse Datenmengen

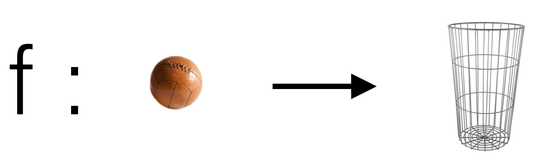
Theorie:

In der Analyse von Algorithmen geht man meist davon, dass Hashfunktionen die gegebenen Daten zufällig und unabhängig von einander auf {0,…,m-1} abbilden.



Praxis:

Man wählt eine Hashfunktion f aus einer vorgegeben Menge von Funktionen (universelle Hashklasse) zufällig.



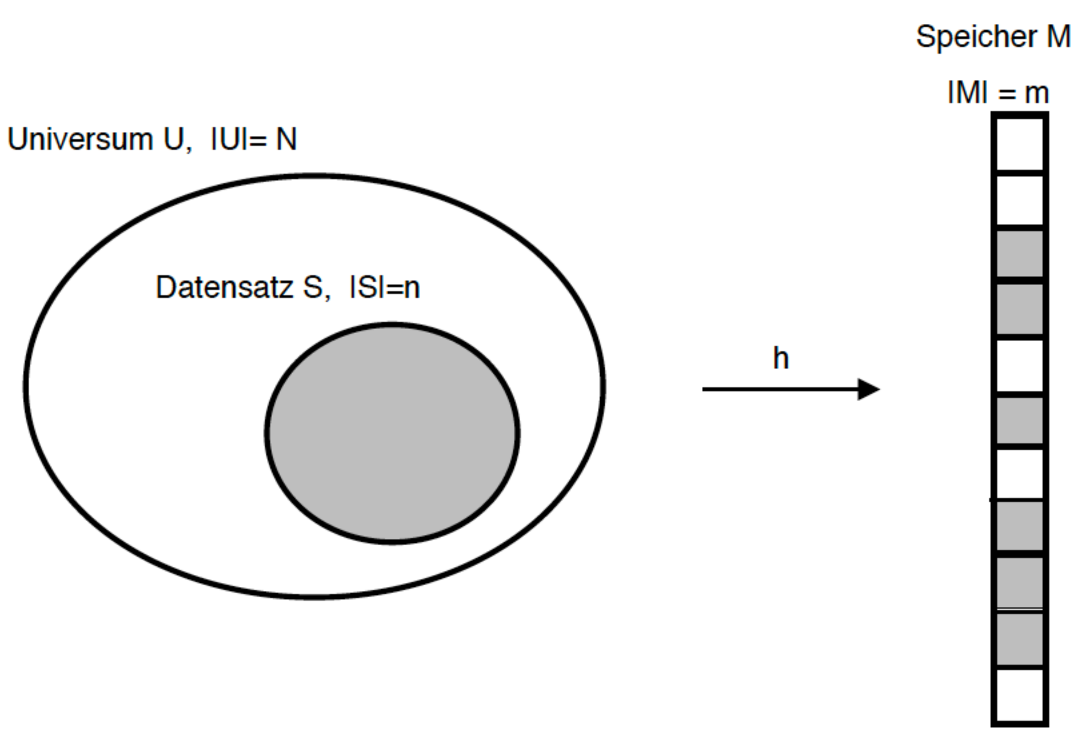
(deterministisch)

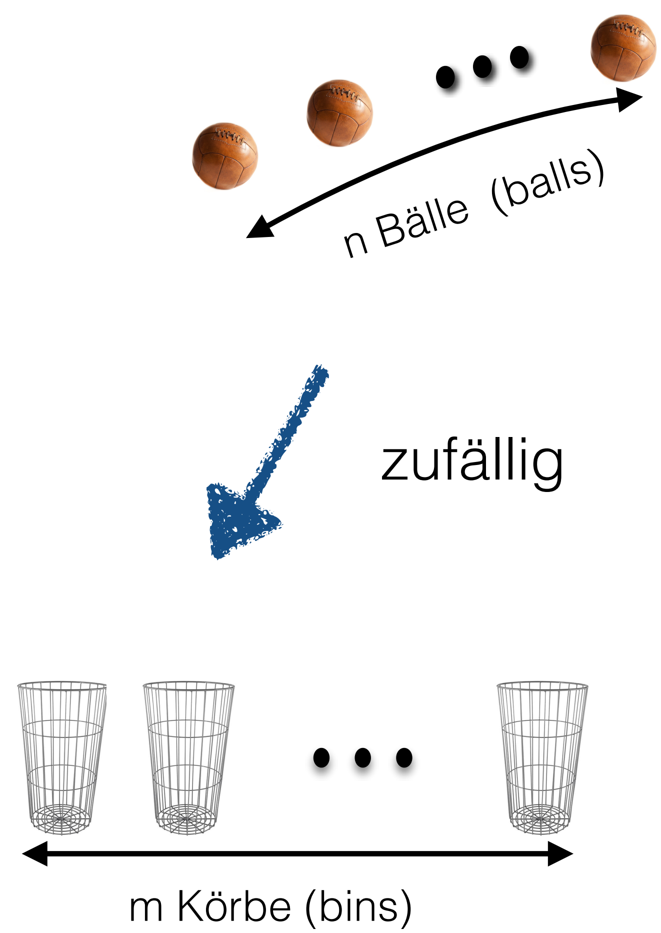
Man geht davon aus, dass die Funktion f die in der theoretischen Analyse gemachten Annahmen erfüllt.

Probleme die es zu lösen gilt:

- Konstruktion einer Hashfunktion  (mod m)
- Umgang mit Kollisionen 

notwendig: theoretisches Verständnis „was ist überhaupt möglich“





X_i := Anzahl Bälle im i-ten Korb

Frage: Wie gross ist $\max_{1 \leq i \leq n} X_i$?