Algorithms and Probability

Week 8

2025/04/10 — Georg Hasebe

Algorithmus

Klassischer Algorithmus:

Wir beweisen:

(1) **Korrektheit:**

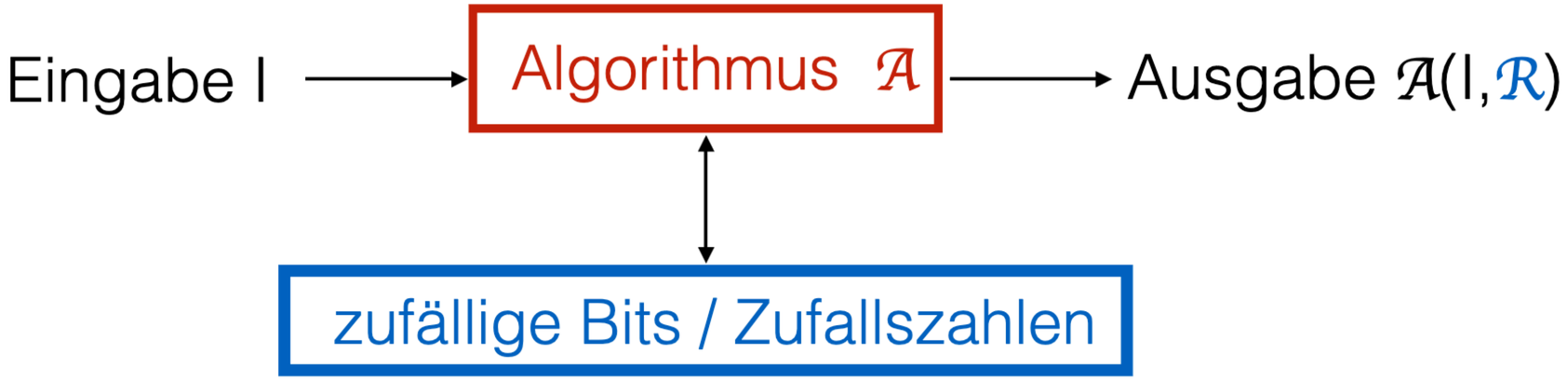
für alle Eingaben I gilt: $\mathcal{A}(I)$ ist korrekt (d.h. was es sein soll)

(2) **Laufzeit:**

für alle Eingaben I mit Länge $|I|=n$: Laufzeit = $O(f(n))$

As we have seen in AnD.

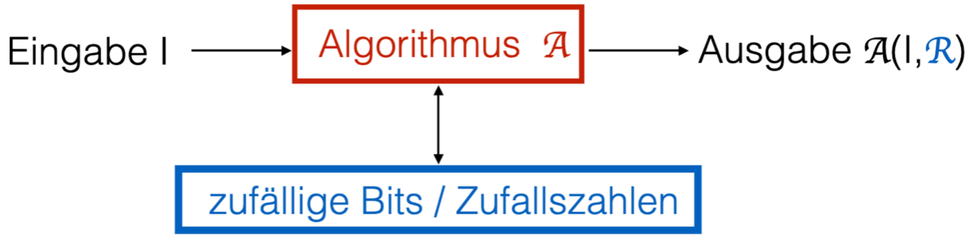
Randomisierte Algorithmen



Eigenschaften:

Ausgabe $\mathcal{A}(I,\mathcal{R})$ hängt von Eingabe I *und* Zufallszahlen $\mathcal{R}$ ab.

Insbesondere: Ergebnis lässt sich i.A. nicht reproduzieren .. !



Randomisierte Algorithmen

Die Hinzugabe von Zufallselementen $\mathscr{R}$ muss auch bei der Korrektheit und Laufzeit berücksichtigt werden.

(1) Korrektheit:

Wir wollen: praktisch 1…

$$\Pr[\mathscr{A}(I, \mathscr{R}) \text{ ist korrekt}] \geq \ldots$$

(2) Laufzeit:

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

Las-Vegas/ Monte-Carlo Algorithmen

Las-Vegas Algorithmen:

- geben nie falsche Antwort
- Laufzeit ist eine Zufallsvariable

Ziel: $\mathbb{E}[\text{Laufzeit}] = \text{"polynomiell"}$

Monte-Carlo Algorithmen:

- Laufzeit immer polynomiell
- geben manchmal eine falsche Antwort

Ziel: $\Pr[\text{Antwort falsch}] = \text{"sehr klein"}$

🐎's-brücken:

Las-**V**egas = “**L**aufzeit **V**ariabel”
Monte-**C**arlo = “**M**anchmal **C**orrect”

Benutzung auf eigene Gefahr.

Beispiele

Las-Vegas Algorithmus:

- QuickSort (Laufzeit hängt von Pivotwahl ab)

Monte-Carlo Algorithmus:

- Teste

- Testen einer Münze: fair (Kopf/Zahl) vs. fake (Kopf/Kopf)

Algorithmus: werfe Münze ~~ein~~ **100** Mal, falls $\geq$ **1 mal** Zahl: return „fair“
ansonsten: return „fake“

Fehlerw’lichkeit:

$$\begin{cases} 0, & \text{falls Münze fake} \\ (1/2)^{100}, & \text{falls Münze fair} \end{cases}$$

Wir können jeden Las Vegas Algorithmus mit erwarteter Laufzeit T, in einen randomisierten Algorithmus mit Laufzeit höchstens $10T$ und Erfolgswahrscheinlichkeit mindestens 0.9 umwandeln.

Der randomisierte Algorithmus A löst Problem P mit Wahrscheinlichkeit p, und gibt sonst aus: “Keine Antwort”. Wie oft müssen wir A im Erwartungswert laufen lassen, bis er Problem P löst?

Bitte stellen sie sicher, dass ihre Antwort keine Leerzeichen enthält.

Monte-Carlo bei JA/NEIN Ausgaben

Algorithmus mit einseitigem Fehler:

- Eine Antwort ist immer richtig, bei der anderen gibt es eine Fehlerwahrscheinlichkeit.
- Wiederhole um die Fehlerwahrscheinlichkeit zu reduzieren.

Algorithmus mit zweiseitigem Fehler:

- Beide Antworten besitzen eine Fehlerwahrscheinlichkeit.
- Reduktion der Fehlerwahrscheinlichkeiten durch Wiederholung nur möglich, wenn Fehlerwahrscheinlichkeit von Anfang an strikt kleiner als $1/2$.

Monte-Carlo bei JA/NEIN Ausgaben

Satz 2.74. Sei $\mathcal{A}$ ein randomisierter Algorithmus, der immer eine der beiden Antworten JA oder NEIN ausgibt, wobei

$$\Pr[\mathcal{A}(\mathrm{I}) = \textsc{Ja}] = 1 \quad \text{falls I eine JA-Instanz ist,}$$

und

$$\Pr[\mathcal{A}(\mathrm{I}) = \textsc{Nein}] \geq \varepsilon \quad \text{falls I eine NEIN-Instanz ist.}$$

Dann gilt für alle $\delta > 0$: bezeichnet man mit $\mathcal{A}_\delta$ den Algorithmus, der $\mathcal{A}$ solange aufruft, bis entweder der Wert NEIN ausgegeben wird (und $\mathcal{A}_\delta$ dann ebenfalls NEIN ausgibt) oder bis $\mathrm{N} = \varepsilon^{-1} \ln \delta^{-1}$-mal JA ausgegeben wurde (und $\mathcal{A}_\delta$ dann ebenfalls JA ausgibt), so gilt für alle Instanzen I

$$\Pr[\mathcal{A}_\delta(\mathrm{I}) \text{ korrekt}] \geq 1 - \delta.$$

“Einseitiger Fehler”

Beispiel “Einseitig”

Teste ob n eine Primzahl.

Antwortmöglichkeiten:

- **prime** (JA)
- **composite** (NEIN)

Algorithmus $\mathscr{A}$

Euklid-Primzahltest(n)

1: Wähle $a \in [n-1]$, zufällig gleichverteilt
2: **if** ggT$(a, n) > 1$ **then return** ‘keine Primzahl’
3: **else return** ‘Primzahl’

JA-Instanz: Primzahl p

NEIN-Instanz: zusammengesetzte Zahl m

Antwort NEIN ist **immer richtig**, Antwort JA **kann falsch sein**: Einseitig!

Target-Shooting

Gegeben Menge U und $S \subseteq U$, wie gross ist $|S|/|U|$?

Idee: wähle zufällige $u_i \in U$ für $i \in \{1,\ldots,N\}$ und teste ob $u \in S$ und gebe das Verhältnis der gefundenen Elemente in S zu N aus.

TARGET-SHOOTING

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

Hier ist $I_S \colon U \to \{0,1\}$ eine Indikatorvariable, sodass $I_S(u) = 1$ genau dann gilt, wenn $u \in S$.

Target-Shooting Analyse

Wir definieren für $i \in \{1,\ldots,N\}$

$$Y_i := I_S(u_i).$$

u_i werden uniform und unabhängig gewählt $\Rightarrow Y_1, \ldots, Y_N$ sind unabhängige Bernoulli-Variablen. $Y_i \sim \text{Ber}(p)$, was ist p?

$p = |S|/|U|$ und somit $\Pr[Y_i = 1] = |S|/|U|$ für alle $i \in \{1,\ldots,N\}$.

Weiters definieren wir:

$$Y := 1/N \cdot \sum_{i=1}^{N} Y_i = 1/N \cdot \sum_{i=1}^{N} I_S(u_i)$$

Target-Shooting Analyse

Wegen

$$Y := 1/N \cdot \sum_{i=1}^{N} Y_i = 1/N \cdot \sum_{i=1}^{N} I_S(u_i)$$

Sehen wir schnell $\mathbb{E}[Y] = |S|/|U|$ (unabhängig von N).

Wie präzise ist dieses Resultat für kleine N?

“die zu erwartende Abweichung vom Erwartungswert”

Definition 2.39. Für eine Zufallsvariable X mit $\mu = \mathbb{E}[X]$ definieren wir die *Varianz* Var[X] durch

$$\mathrm{Var}[X] := \mathbb{E}[(X-\mu)^2] = \sum_{x \in W_X} (x-\mu)^2 \cdot \Pr[X = x].$$

Die Grösse $\sigma := \sqrt{\mathrm{Var}[X]}$ heisst *Standardabweichung* von X.

Target-Shooting Analyse

$$\mathrm{Var}[Y] = 1/N \cdot (|S|/|U| - (|S|/|U|)^2)$$

Ist also von N abhängig.

Wie gross soll N sein, damit der Algorithmus eine sehr gute Antwort liefert?

Satz 2.79. Seien $\delta, \varepsilon > 0$. Falls $N \geq 3\frac{|U|}{|S|} \cdot \varepsilon^{-2} \cdot \ln(2/\delta)$, 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 im Skript.

Wir betrachten eine Variante des Target Shooting Algorithmus, den wir in der Vorlesung gesehen haben. Bitte beachten Sie, dass es sich nicht um genau den gleichen Algorithmus handelt. Wir gehen, wie in der Vorlesung davon aus, dass es eine unbekannte Menge S gibt, die Teilmenge einer bekannten Menge U ist. Ausserdem können wir Elemente $u \in U$ uniform zufällig auswählen und überprüfen, ob $u \in S$.

Angenommen, wir wählen so oft zufällige Elemente $u \in U$ aus, bis wir insgesamt 100 Elemente gesehen haben, die $u \in S$ erfüllen. Sei X die Anzahl an Elementen, die wir auswählen müssen, bis das zutrifft.

- X ist geometrisch verteilt. 
- Falls $|S| = |U|$, dann $X = 100$. 
- $\mathbb{E}[X] = 100\,|U|/|S|$. 
- Falls $X = 100$, dann $|S| = |U|$. 

Gegeben Mengen $S \subset U$ wobei die Grösse von U mit $|U| = 10^8$ bekannt ist. Wir wollen die Grösse von S mit Hilfe von Target Shooting abschätzen. Wir wählen N Elemente aus U unabhängig von einander und zählen, wie vieler dieser Elemente auch in S sind. Wir bezeichnen diese Grösse mit Z.

Wenn $|S| = 10^3$ und $N = 10^6$ dann erwarten wir...

- $\mathbb{E}[Z] = 10^0$ Elemente in S. 
- $\mathbb{E}[Z] = 10^3$ Elemente in S. 
- $\mathbb{E}[Z] = 10^1$ Elemente in S. 
- $\mathbb{E}[Z] = 10^2$ Elemente in S.