Algorithms and Probability

Week 9

G09 - mkilic

23.IV.2026

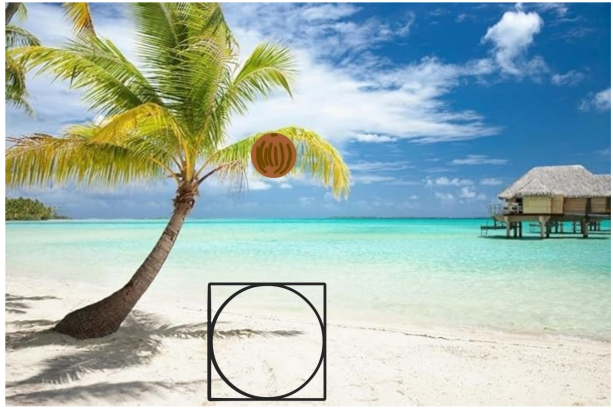
Overview

1. **Target-Shooting**
2. **Randomisierte Algorithmen & Fehlerreduktion**
3. **Exercise: Reducing the Error of Monte Carlo Algorithms**
4. **Primzahltest**

Roadmap

1. Graphentheorie
   - Zusammenhang
   - Kreise
   - Matchings
   - Färbungen
2. W'keitstheorie
   - Bedingte W'keit
   - Unabhängigkeit
   - (mehrere) Zufallsvariablen
   - Diskrete Verteilungen
   - Abschätzen von W'keiten
   - Randomisierte Algorithmen
3. Algorithmen
   - Lange-Bunte Pfade
   - MaxFlow
   - MinCut
   - Kleinster umschliessender Kreis
   - Konvexe Hülle

Target-Shooting



Target-Shooting

Target-Shooting

Gegeben: eine endliche Menge U und eine Untermenge $S \subseteq U$ unbekannter Grösse
Gesuct: $\frac{|S|}{|U|} = ?$
Ausserdem haben wir per Annahme die folgenden:

- eine Indikatorfunktion $I_S : U \to \{0, 1\}$, sodass $I_S(u) = 1$ g.d.w. $u \in S$
- die Möglichkeit Elemente $u \in U$ gleichverteilt zu generieren

Target-Shooting

$|S|/|U|$ approximieren: Für eine wahl von $N > 0$, wähle N Elemente aus U i.i.d. Dann gebe Verhältnis der gefundenen Elemente in S zu N aus.

```
1 Waehle u_1, ..., u_N ∈ U i.i.d.
2
3 return 1/N · Σ_{i=1}^{N} I_S(u_i)
4
```

TARGET-SHOOTING

[0] *i.i.d. = independent and identically distributed : unabhängig und gleichverteilt*

Target-Shooting Analyse

- Definiere Indikatorvariablen $Y_i := I_S(u_i)$.
- $(Y_i)_{i \in \{1,\ldots,N\}}$ sind unabhängige Bernouilli-Variablen mit $Pr[Y_i = 1] = \frac{|S|}{|U|}$
- Jetzt können wir die Ausgabe von unserem Algorithmus als eine Zufallsvariable Y schreiben:

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

Target-Shooting Analyse

- Definiere Indikatorvariablen $Y_i := I_S(u_i)$.
- $(Y_i)_{i \in \{1,\dots,N\}}$ sind unabhängige Bernouilli-Variablen mit $Pr[Y_i = 1] = \frac{|S|}{|U|}$
- Jetzt können wir die Ausgabe von unserem Algorithmus als eine Zufallsvariable Y schreiben:

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

$$\mathbb{E}[Y] = \frac{|S|}{|U|}$$

unabhängig von der Wahl von N

$$\begin{aligned} Var[Y] &= Var[\frac{1}{N} Y'] && (Y' \sim Bin(N, p)) \\ &= \frac{1}{N^2} Np(1-p) \\ &= \frac{1}{N} \left(\frac{|S|}{|U|} - \left(\frac{|S|}{|U|} \right)^2 \right) \end{aligned}$$

Target-Shooting Analyse

Satz 2.79.

Seien $\delta, \varepsilon > 0$. Falls

$$N \geq 3 \cdot \frac{|U|}{|S|} \cdot \varepsilon^{-2} \cdot \ln\left(\frac{2}{\delta}\right),$$

ist die Ausgabe Y von TARGET-SHOOTING mit Wahrscheinlichkeit mindestens $1-\delta$ im Intervall

$$\left[(1-\varepsilon)\frac{|S|}{|U|}, (1+\varepsilon)\frac{|S|}{|U|}\right].$$

Target-Shooting Analyse

Wir haben $\mathbb{E}[Y] = \frac{|S|}{|U|}$. Wir zeigen also:

$$Pr[|Y - \mathbb{E}[Y]| \geq \varepsilon \mathbb{E}[Y]] \leq \delta$$

Schreiben wir $Z := \sum_{i=1}^{N} Y_i = NY$, ist dies äquvialent zu

$$Pr[|Z - \mathbb{E}[Z]| \geq \varepsilon \cdot \mathbb{E}[Z]] \leq \delta$$

Da Y_i unabhängige Bernouilli-Variablen sind, können wir die Chernoff-Schranken benutzen.

$$Pr[|Z - \mathbb{E}[Z]| \geq \varepsilon \cdot \mathbb{E}[Z]] \leq 2e^{-\varepsilon^2 \frac{\mathbb{E}[Z]}{3}} = 2e^{-\varepsilon^2 N \frac{|S|}{3|U|}}$$

[0] *Wir machen kleine Modifizierung für Chernoff-Schranken damit wir die Schranken aus Satz 2.70 (i) und (ii) leichter addieren können. Die Ungleichung ist immer noch gültig.*

Target-Shooting Fazit

$$Pr[|Z - \mathbb{E}[Z]| \geq \varepsilon \cdot \mathbb{E}[Z]] \leq 2e^{-\varepsilon^2 N \frac{|S|}{3|U|}}$$

Wenn man $N \geq 3\frac{|U|}{|S|} \cdot \varepsilon^{-2} \cdot \ln(\frac{2}{\delta})$ wählt dann ist diese Wahrscheinlichkeit höchstens δ. $\square$

Unsere Schranke für N enthält $\frac{|U|}{|S|}$, was wir bestimmen wollen. Man kan dafür eine Schranke einsetzen.

Target-Shooting Beispiel

π − Aproximieren

Sei $U = [-1, 1]^2$ und sei $S = \{(x, y) \in \mathbb{R}^2 \mid x^2 + y^2 \leq 1\}$. Dann gilt:

$$\pi = 4\frac{|S|}{|U|}$$

Wie viele Male müssen wir *"schiessen"* damit wir π mit 99%-iger Wahrscheinlichkeit bis auf die 10te Nachkommastelle genau bestimmen?

Target-Shooting Beispiel

π – *Aproximieren*

Sei $U = [-1, 1]^2$ und sei $S = \{(x, y) \in \mathbb{R}^2 \mid x^2 + y^2 \leq 1\}$. Dann gilt:

$$\pi = 4\frac{|S|}{|U|}$$

Wie viele Male müssen wir *"schiessen"* damit wir π mit 99%-iger Wahrscheinlichkeit bis auf die 10te Nachkommastelle genau bestimmen?

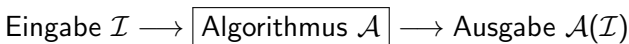
Wähle $\delta = 0.01$ und $\varepsilon = 10^{-10}$. Das heisst man muss

$$N = 3\pi^{-1} \cdot 10^{20} \cdot \ln 200$$

viele Versuche machen.

Randomisierte Algorithmen

→ **Deterministisch:**



Randomisierte Algorithmen

→ **Deterministisch:**

$$\text{Eingabe } \mathcal{I} \longrightarrow \boxed{\text{Algorithmus } \mathcal{A}} \longrightarrow \text{Ausgabe } \mathcal{A}(\mathcal{I})$$

→ **Randomisiert:**

$$\text{Eingabe } \mathcal{I} + \boxed{\text{Zufallsquelle: } \mathcal{R}} \longrightarrow \boxed{\text{Algorithmus } \mathcal{A}} \longrightarrow \text{Ausgabe } \mathcal{A}(\mathcal{I}, \mathcal{R})$$

Monte Carlo & Las Vegas

Monte Carlo Algorithmen

$\rightarrow$ immer schnell

$\rightarrow$ manchmal falsch
(Korrektheit ist Zufallsvariable)

Beispiele:

- ?

Las Vegas Algorithmen

$\rightarrow$ immer korrekt

$\rightarrow$ manchmal langsam
(Laufzeit ist Zufallsvariable)

Beispiele:

- ?

Monte Carlo & Las Vegas

Monte Carlo Algorithmen

→ immer schnell

→ manchmal falsch
(Korrektheit ist Zufallsvariable)

Beispiele:

Immer dieselbe Antwort geben

Las Vegas Algorithmen

→ immer korrekt

→ manchmal langsam
(Laufzeit ist Zufallsvariable)

Beispiele:

Randomisiertes QuickSort

Monte Carlo & Las Vegas

Monte Carlo Algorithmen

→ immer schnell

→ manchmal falsch
(Korrektheit ist Zufallsvariable)

Beispiele:

Immer dieselbe Antwort geben

Primzahltest durch 10 Teiler ausprobieren

Las Vegas Algorithmen

→ immer korrekt

→ manchmal langsam
(Laufzeit ist Zufallsvariable)

Beispiele:

Randomisiertes QuickSort

Primzahltest durch Teiler ausprobieren mit obere Schranke

Las Vegas mit oberer Schranke

Ein Las Vegas Algorithmus läuft bis er die richtige Antwort findet. Wir können die Laufzeit eines Las Vegas Algorithmus beschränken sodass der Algorithmus nur für eine beschränkte Zeit läuft, und falls am Ende die Antowrt noch nicht bekannt ist, ('???') zurückgibt.

```
1        Las_Vegas_Algo(params):
2
3            while(runtime < LIMIT):
4                do_work(params)
5
6            return ('???')
7
```

Beachte, dass diese Version **keine** Instanz von Monte-Carlo-Algorithmus ist. Bei einer Monte Carlo Algorithmus wird immer eine Antwort zurückgegeben, während hier der Algorithmus manchmal "Keine Ahnung!" sagt.

Beispiel: Primzahltest

```
Monte_Carlo_Prime_Test(n, k):

  repeat k times:

    pick random a in [2, n-2]

    if a^(n-1) mod n != 1:
      return 'Composite'

  return 'Probably Prime'

```

```
Las_Vegas_Prime_Test(n):

  factors = shuffle([sqrt(n)])

  for d in factors:

    if n mod d == 0:
      return 'Composite'

  return 'Prime'

```

Reduktion der Fehlerwahrscheinlichkeit

Satz 2.72: Reduktion der Fehlerw'keit - Las Vegas

Sei $\mathcal{A}$ ein randomisierter Algorithmus, der nie eine falsche Antwort gibt, aber zuweilen '???' ausgibt, wobei

$$\Pr[\mathcal{A}(\mathcal{I}) \text{ korrekt}] \geq \varepsilon.$$

Dann gilt für alle $\delta > 0$:
Sei $\mathcal{A}_\delta$ der Algorithmus, der $\mathcal{A}$ solange aufruft, bis entweder

- ein Wert verschieden von '???' ausgegeben wird (und diesen Wert dann ebenfalls ausgibt) oder bis
- $N = \lceil \varepsilon^{-1} \ln \delta^{-1} \rceil$ mal '???' ausgegeben wurde (und dann '???' ausgibt),

so gilt

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

Beweis: (Skript s. 147)

Reduktion der Fehlerwahrscheinlichkeit

Satz 2.74: Reduktion der Fehlerw'keit - Monte Carlo - Einseitiger Feher

Sei $\mathcal{A}$ ein randomisierter Algorithmus, der immer entweder `JA` oder `NEIN` ausgibt, wobei

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

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

Dann gilt für alle $\delta > 0$:
Sei $\mathcal{A}_\delta$ der Algorithmus, der $\mathcal{A}$ solange aufruft, bis entweder

- der Wert `NEIN` ausgegeben wird (und dann selbst `NEIN` ausgibt) oder bis
- $N = \lceil \varepsilon^{-1} \ln \delta^{-1} \rceil$ mal `JA` ausgegeben wurde (und dann selbst '`JA`' ausgibt),

so gilt

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

Monte Carlo mit einseitiger Fehler

$Pr[\mathcal{A}(\mathcal{I}) = \texttt{JA}] = 1$ falls $\mathcal{I}$ eine JA-instanz ist
$Pr[\mathcal{A}(\mathcal{I}) = \texttt{NEIN}] \geq \varepsilon$ falls $\mathcal{I}$ eine NEIN-instanz ist

- Können wir den Algorithmus vertrauen, wenn er für eine Eingabe $\mathcal{I}$ als Resultat `'JA'` ausspuckt?

Monte Carlo mit einseitiger Fehler

$Pr[\mathcal{A}(\mathcal{I}) = \texttt{JA}] = 1$ falls $\mathcal{I}$ eine JA-instanz ist
$Pr[\mathcal{A}(\mathcal{I}) = \texttt{NEIN}] \geq \varepsilon$ falls $\mathcal{I}$ eine NEIN-instanz ist

- Können wir den Algorithmus vertrauen, wenn er für eine Eingabe $\mathcal{I}$ als Resultat `'JA'` ausspuckt?
- Nein! Es kann auch sein, dass $\mathcal{I}$ ein NEIN-Instanz ist und mit $\leq (1 - \varepsilon)$ W'keit unser Algorithmus sie falsch klassifiziert!
- Wir können unser Algorithmus immer vertrauen, wenn er eine Eingabe als eine NEIN-Instanz klassifizert.

Monte Carlo mit einseitiger Fehler

$Pr[\mathcal{A}(\mathcal{I}) = \texttt{JA}] = 1$ falls $\mathcal{I}$ eine `JA`-instanz ist
$Pr[\mathcal{A}(\mathcal{I}) = \texttt{NEIN}] \geq \varepsilon$ falls $\mathcal{I}$ eine `NEIN`-instanz ist

- Können wir den Algorithmus vertrauen, wenn er für eine Eingabe $\mathcal{I}$ als Resultat `'JA'` ausspuckt?
- Nein! Es kann auch sein, dass $\mathcal{I}$ ein NEIN-Instanz ist und mit $\leq (1 - \varepsilon)$ W'keit unser Algorithmus sie falsch klassifiziert!
- Wir können unser Algorithmus immer vertrauen, wenn er eine Eingabe als eine `NEIN`-Instanz klassifizert.
- Ein Beispiel ist der `Monte_Carlo_Prime_Test` aus Folie 7 und die Frage: "Ist mein Zahl zusammengesetzt?"

Reduktion der Fehlerwahrscheinlichkeit

Satz 2.75: Reduktion der Fehlerw'keit - Monte Carlo - Zweiseitiger Feher

Sei $\varepsilon > 0$ und $\mathcal{A}$ ein randomisierter Algorithmus, der immer eine der beiden Antworten `JA` oder `NEIN` ausgibt, wobei

$$\Pr[\mathcal{A}(\mathcal{I}) \text{ korrekt}] \geq \tfrac{1}{2} + \varepsilon.$$

Dann gilt für alle $\delta > 0$: bezeichnet man mit $\mathcal{A}_\delta$ den Algorithmus, der $N = 4\varepsilon^{-2} \ln \delta^{-1}$ unabhängige Aufrufe von $\mathcal{A}$ macht und dann die Mehrheit der erhaltenen Antworten ausgibt, so gilt für den Algorithmus $\mathcal{A}_\delta$, dass

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

Beweis: Auf der Tafel (Skript s. 149)

Reduktion der Fehlerwahrscheinlichkeit

Reduktion der Fehlerw'keit - Optimierungsprobleme

Sei $\varepsilon > 0$ und $\mathcal{A}$ ein randomisierter Algrithmus für ein Optimierungsproblem, wobei gelte:

$$Pr[\mathcal{A}(\mathcal{I}) \geq f(\mathcal{I})] \geq \varepsilon$$

Dann gilt für alle $\delta > 0$: Sei $\mathcal{A}_\delta$ der Algorithmus, der $N = \lceil \varepsilon^{-1} \ln \delta^{-1} \rceil$ Aufrufe von $\mathcal{A}$ macht und die beste der erhaltenen Antworten ausgibt, so gilt

$$Pr[A_\delta(\mathcal{I}) \geq f(\mathcal{I})] \geq 1 - \delta$$

Man spielt mit der Ungleichheitszeichen je nachdem, ob das Problem ein Minimierungs- oder Maximierungsproblem ist (oben ist $\mathcal{A}(\mathcal{I}) \geq f(\mathcal{I})$: Maximierungsproblem - $\mathcal{A}(\mathcal{I}) \leq f(\mathcal{I})$: Minimierungsproblem).

Exercise: Reducing the Error of Monte Carlo Algorithms

Alice, Bob and Claire have learned that Monta Carlo algorithms can often be improved by running them multiple times. They are each given a blackbox [1] Monte Carlo algorithm $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ with a certain bound on the error. The task is to construct new algorithms $\mathcal{A}\_improved$, $\mathcal{B}\_improved$ resp. $\mathcal{C}\_improved$ (using $\mathcal{A}$, $\mathcal{B}$ resp. $\mathcal{C}$ as subroutine) with a success probability of at least 99%. Try to keep the number of calls to the subroutines small.

Exercise: Reducing the Error of Monte Carlo Algorithms

Alice, Bob and Claire have learned that Monta Carlo algorithms can often be improved by running them multiple times. They are each given a blackbox [1] Monte Carlo algorithm $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ with a certain bound on the error. The task is to construct new algorithms $\mathcal{A}\_improved$, $\mathcal{B}\_improved$ resp. $\mathcal{C}\_improved$ (using $\mathcal{A}$, $\mathcal{B}$ resp. $\mathcal{C}$ as subroutine) with a success probability of at least 99%. Try to keep the number of calls to the subroutines small.

(a) Given a graph (V, E), Alice tries to find a vertex set $\emptyset \neq S \subsetneq V$ that minimizes $|\delta(S)|$. She is given a Monte Carlo Algorithm $\mathcal{A}$ that, given a graph, returns some vertex set $\emptyset \neq S \subsetneq V$. With probability at least $p_{\mathcal{A}} \geq 1/\binom{n}{2}$ this set minimizes $|\delta(S)|$. Use this algorithm $\mathcal{A}$, to construct another algorithm $\mathcal{A}\_improved$ for the same problem, that has a success probability of at least 99%. *Hint:* $1 + x \leq e^x$

Exercise: Reducing the Error of Monte Carlo Algorithms

Alice, Bob and Claire have learned that Monta Carlo algorithms can often be improved by running them multiple times. They are each given a blackbox [1] Monte Carlo algorithm $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ with a certain bound on the error. The task is to construct new algorithms $\mathcal{A}\_improved$, $\mathcal{B}\_improved$ resp. $\mathcal{C}\_improved$ (using $\mathcal{A}$, $\mathcal{B}$ resp. $\mathcal{C}$ as subroutine) with a success probability of at least 99%. Try to keep the number of calls to the subroutines small.

(a) Given a graph (V, E), Alice tries to find a vertex set $\emptyset \neq S \subsetneq V$ that minimizes $|\delta(S)|$. She is given a Monte Carlo Algorithm $\mathcal{A}$ that, given a graph, returns some vertex set $\emptyset \neq S \subsetneq V$. With probability at least $p_{\mathcal{A}} \geq 1/\binom{n}{2}$ this set minimizes $|\delta(S)|$. Use this algorithm $\mathcal{A}$, to construct another algorithm $\mathcal{A}\_improved$ for the same problem, that has a success probability of at least 99%. *Hint:* $1 + x \leq e^x$

(b) Bob wants to check if a number is prime. He already knows a Monte Carlo algorithm $\mathcal{B}$ which takes as input a natural number N. If N is prime, the algorithm always returns 'prime'. If N is not prime, the algorithm returns 'not a prime' with probability $p_B \geq 1/2$. Use Bobs algorithm $\mathcal{B}$, to construct another algorithm $\mathcal{A}\_improved$ for the same problem, that has a success probability of at least 99%.

Exercise: Reducing the Error of Monte Carlo Algorithms

(c) Claire chose the most difficult problem. She wants to determine if a given graph has ans Hamiltonian cycle. She thought of an algorithm $\mathcal{C}$ that, given a graph, out puts 'YES' or 'NO'. If a graoh G has (resp. does not have) a Hamiltonian cycle, the algorithms $\mathcal{C}(G)$ returns 'YES' (resp. 'No') with probability $p_{\mathcal{C}} \geq 3/4$. Help Claire to find an algorithm C_*improved* thatis correct with a probability of at least 99%.

Exercise: Reducing the Error of Monte Carlo Algorithms

(c) Claire chose the most difficult problem. She wants to determine if a given graph has ans Hamiltonian cycle. She thought of an algorithm $\mathcal{C}$ that, given a graph, out puts 'YES' or 'NO'. If a graoh G has (resp. does not have) a Hamiltonian cycle, the algorithms $\mathcal{C}(G)$ returns 'YES' (resp. 'No') with probability $p_{\mathcal{C}} \geq 3/4$. Help Claire to find an algorithm C_*improved* thatis correct with a probability of at least 99%.

(d) Unfortunately, you forgot what Clairs initial algorithm $\mathcal{C}$ was. Thus, you have to find an initial algorithm yourself. Describe a fast Monte Carlo algorithm that, given a graph, outputs the correct answer 'YES' or 'NO' with probability 1/2. Can you boost this algorithm to a success probability of 99%?

Exercise: Reducing the Error of Monte Carlo Algorithms

(c) Claire chose the most difficult problem. She wants to determine if a given graph has ans Hamiltonian cycle. She thought of an algorithm $\mathcal{C}$ that, given a graph, out puts 'YES' or 'NO'. If a graoh G has (resp. does not have) a Hamiltonian cycle, the algorithms $\mathcal{C}(G)$ returns 'YES' (resp. 'No') with probability $p_{\mathcal{C}} \geq 3/4$. Help Claire to find an algorithm C_*improved* thatis correct with a probability of at least 99%.

(d) Unfortunately, you forgot what Clairs initial algorithm $\mathcal{C}$ was. Thus, you have to find an initial algorithm yourself. Describe a fast Monte Carlo algorithm that, given a graph, outputs the correct answer 'YES' or 'NO' with probability 1/2. Can you boost this algorithm to a success probability of 99%?

(e) **Difficult**: Assume you are given the Monte Carlo algorithm $\mathcal{C}$ from part (c). Can you construct an algorithm that not only determines whether there is a Hamiltonian cycle, but even computes one if it exists. Of course, again only with a success probability of 99%.

Primzahltest

Primzahltest

Problem: Gegeben $n \in \mathbb{N}$, ist n prim?

Viele Anwendungsbereiche z.B. Kyrpographie, Hashing, PRNGs usw.
Wir werden 3 Herangehensweisen sehen um Primalität zu testen.

0 - Brute Force

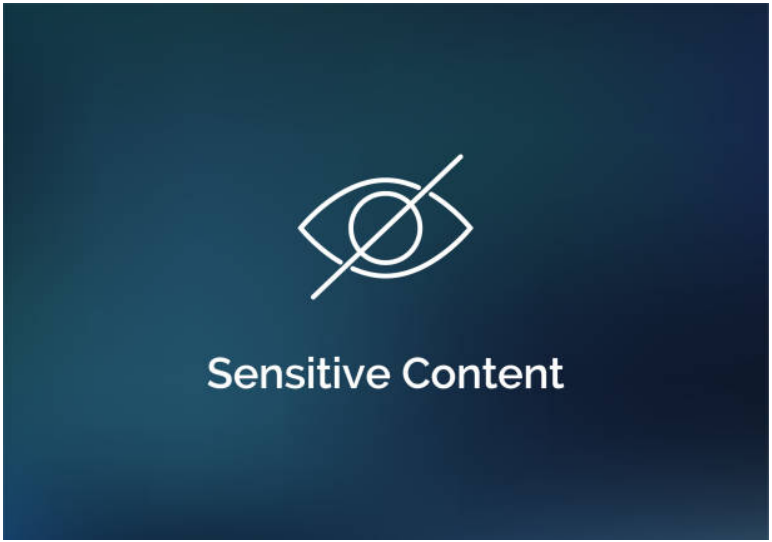
```
1     for all a ≤ √n:
2         if (n % a == 0):
3             return false
4     return true
5
```

naiv_prim_test(n)

Falls der Algorithmus einen Teiler a entdeckt, dann wissen wir, dass n prim ist. Wir sagen a ist ein **Zertifikat** [1] für Zusammengesetztheit von n. Sonst sucht unser Algorithmus weiter. Das ist noch nicht randomisiert. Ausgabe ist immer korrekt aber Algorithmus ist **zu langsam**.

[1] (Manchmal auch: *Zeuge/Witness*)

Some Discrete Math



Some Discrete Math

Definition 5.16. $\mathbb{Z}_m^* \stackrel{\text{def}}{=} \{a \in \mathbb{Z}_m \mid \gcd(a, m) = 1\}$.

Definition 5.17. The *Euler function* $\varphi : \mathbb{Z}^+ \to \mathbb{Z}^+$ is defined as the cardinality of $\mathbb{Z}_m^*$:

$$\varphi(m) = |\mathbb{Z}_m^*|.$$

Example 5.29. $\mathbb{Z}_{18}^* = \{1, 5, 7, 11, 13, 17\}$. Hence $\varphi(18) = 6$.

If p is a prime, then $\mathbb{Z}_p^* = \{1, \ldots, p-1\} = \mathbb{Z}_p \setminus \{0\}$, and $\varphi(p) = p - 1$.

Lemma 5.12. *If the prime factorization of m is $m = \prod_{i=1}^{r} p_i^{e_i}$, then*[11]

$$\varphi(m) = \prod_{i=1}^{r} (p_i - 1) p_i^{e_i - 1}.$$

Discrete Maths Script Page 102-103

1 - Euklid-Primzahltest

n prim $\Rightarrow ggT(a, n) = 1 \ \ \forall a \in [n-1]$

```
    choose a ∈ [n − 1] i.i.d.:
    if ggT(a,n) > 1:
        return false
    else:
        return true

```

euklid_prim_test(n)

1 - Euklid-Primzahltest

n prim $\Rightarrow ggT(a,n) = 1 \ \ \forall a \in [n-1]$

```
1    choose a ∈ [n − 1] i.i.d.:
2    if ggT(a,n) > 1:
3        return false
4    else:
5        return true
6
```

euklid_prim_test(n)

Zu viele 'false positives'. Falls n zusammengesetzt ist dann ist nur korrekt mit W'keit

$$\frac{|a \in [n-1] : ggT(a,n) = 1|}{n-1} = \frac{|\mathbb{Z}_n^*|}{n-1} = \frac{\varphi(n)}{n-1}$$

1 - Euklid-Primzahltest

n prim $\Rightarrow ggT(a,n) = 1 \ \ \forall a \in [n-1]$

```
choose a ∈ [n − 1] i.i.d.:
if ggT(a,n) > 1:
    return false
else:
    return true
```

euklid_prim_test(n)

Zu viele 'false positives'. Falls n zusammengesetzt ist dann ist nur korrekt mit W'keit

$$\frac{|a \in [n-1] : ggT(a,n) = 1|}{n-1} = \frac{|\mathbb{Z}_n^*|}{n-1} = \frac{\varphi(n)}{n-1}$$

Falls $n = pq$ für p und q prim dann ist diese W'keit $\frac{2}{\sqrt{n}}$ und das würde $O(\sqrt{n})$ Wiederholungen brauchen um Fehlerw'keit "genügend" zu reduzieren. Wie brute-force!

Kleiner Fermatscher Satz

Satz 2.77 (Kleiner fermatscher Satz). Ist $n \in \mathbb{N}$ prim, so gilt für alle Zahlen $0 < a < n$

$$a^{n-1} \equiv 1 \mod n.$$

A&W script page 155

Corollary 5.14 (Fermat, Euler). *For all $m \geq 2$ and all a with $\gcd(a, m) = 1$,*

$$a^{\varphi(m)} \equiv_m 1.$$

In particular, for every prime p and every a not divisible by p,

$$a^{p-1} \equiv_p 1.$$

discrete maths script page 104

a	$a^{14} \mod 15$	Result	Is Witness?
1	1	✅ Fermat holds	No
2	$16384 \mod 15 = 4$	❌ Violates Fermat	Yes
3	$4782969 \mod 15 = 9$	❌ Violates Fermat	Yes
4	1	✅ Fermat holds	No
5	$6103515625 \mod 15 = 10$	❌ Violates Fermat	Yes
6	$78364164096 \mod 15 = 6$	❌ Violates Fermat	Yes
7	$678223072849 \mod 15 = 13$	❌ Violates Fermat	Yes
8	$4398046511104 \mod 15 = 4$	❌ Violates Fermat	Yes
9	$22876792454961 \mod 15 = 6$	❌ Violates Fermat	Yes
10	$100000000000000 \mod 15 = 10$	❌ Violates Fermat	Yes
11	$289254654976 \mod 15 = 1$	✅ Fermat holds	No
12	$1283918464548864 \mod 15 = 9$	❌ Violates Fermat	Yes
13	$302875106592253 \mod 15 = 4$	❌ Violates Fermat	Yes
14	$11112006825558016 \mod 15 = 1$	✅ Fermat holds	No

Fermat - #witnesses: 10

a	$\gcd(a, 15)$	Witness?
1	1	❌
2	1	❌
3	3	✅
4	1	❌
5	5	✅
6	3	✅
7	1	❌
8	1	❌
9	3	✅
10	5	✅
11	1	❌
12	3	✅
13	1	❌
14	1	❌

GCD - #witnesses: 6

The witnesses are numbers that verify the compositeness of 15. Fermat can help us find more witnesses in this case, increasing the probability of us finding a witness randomly.

Miller-Rabin: Grundidee

n prim $\Rightarrow \mathbb{Z}_n^* = \{1, \ldots, n-1\}$ *Körper unter addition und Multiplikation modulo n*

Besonders gilt:

$$x^2 \equiv 1 \pmod{n} \quad \Rightarrow \quad x = 1 \text{ oder } x = n-1 \quad (\star)$$

Miller-Rabin: Zerlegung

Wir schreiben $n - 1 = d \cdot 2^k$ mit d ungerade.

Wenn n prim ist, gilt nach Fermat:

$$a^{n-1} \equiv 1 \pmod{n} \quad \text{für alle } a \in \{1, \ldots, n-1\}$$

Daraus folgt:

$$(a^d)^{2^k} \equiv 1 \pmod{n}$$

Miller-Rabin: Was passiert bei Primzahlen

Aufgrund $(\star)$ folgt aus folgender Aussage:

$$(a^d)^{2^k} \equiv 1 \pmod{n}$$

dass entweder

$$(a^d)^{2^{k-1}} \equiv 1 \pmod{n} \quad \text{oder} \quad (a^d)^{2^{k-1}} \equiv -1 \pmod{n}$$

Diese Idee wird rekursiv genutzt, um $a^d, a^{2d}, \ldots, a^{2^i d}$ zu prüfen.

Miller-Rabin: Iterativer Test

Man prüft iterativ:

$$(a^d)^{2^i} \equiv 1 \pmod{n} \quad \text{für alle } 0 \leq i \leq k$$

Oder: Für ein $0 \leq i < k$ gilt:

$$(a^d)^{2^i} \equiv n - 1 \pmod{n}$$

Wenn keine dieser Bedingungen erfüllt ist:

$$\Rightarrow a \text{ ist ein Zertifikat dafür, dass } n \text{ nicht prim ist.}$$

Miller-Rabin: Algorithmus

```
1  Miller-Rabin-Primzahltest(n)
2  1: if n = 2 then
3  2:     return 'Primzahl'
4  3: else if n gerade oder n = 1 then
5  4:     return 'keine Primzahl'
6  5: Wahle a ∈ {2,3,...,n-1} zufallig und
7  6: berechne k,d ∈ ℤ mit n-1 = d·2^k und d ungerade.
8  7: x ← a^d mod n
9  8: if x = 1 or x = n-1 then
10 9:     return 'Primzahl'
11 10: repeat k-1 mal
12 11:     x ← x^2 mod n
13 12:     if x = 1 then
14 13:         return 'keine Primzahl'
15 14:     if x = n-1 then
16 15:         return 'Primzahl'
17 16: return 'keine Primzahl'
```

Miller-Rabin-Primzahltest(n)

Miller-Rabin: Fazit

Laufzeit[2]**:** $O(\ln n)$
W'keiten:

Eingabe: n	**Ausgabe**
prim	immer prim
zusammengesetzt	mit 3/4 W'keit: "nicht prim"

Wir können die Fehlerwahrscheinlichkeit mit Felherreduktionsmethoden beliebig klein machen.

[2] Wegen binäre Exponentiation & Successive squaring

The End

♠