Target Shooting

Gegeben: Zwei Mengen ein Subset S von einem Universum U

und eine Indikatorfunktion $I_S(u) = \begin{cases} 1 & \text{falls } u \in S \\ 0 & \text{sonst} \end{cases}$

Problemstellung: Schätze den relativen Anteil $\frac{|S|}{|U|}$

Algorithmus:

1. Wähle $u_1, \dots, u_N \in U$ zufällig
2. return $\frac{1}{N} \cdot \sum_{i=1}^{N} I_S(u_i)$ = Durchschnitt

- Sei $Y := \frac{1}{N} \sum_{i=1}^{N} I_S(u_i)$ (= Ausgabe des Algorithmus)
- $\mathbb{E}[Y] = \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}[I_S(u_i)] = \frac{1}{N} \cdot N \cdot \frac{|S|}{|U|} = \frac{|S|}{|U|}$

  Linearität; $\mathbb{E}[I_S(u_i)] = \frac{|S|}{|U|}$

Wie gross muss N sein?

- Sei $N \geq 3 \frac{|U|}{|S|} \cdot \varepsilon^{-2} \ln\left(\frac{2}{\delta}\right)$ für $\varepsilon, \delta > 0$. Dann ist die Ausgabe des Algorithmus mit Wahrscheinlichkeit $\geq 1-\delta$ im Intervall $\left[(1-\varepsilon)\frac{|S|}{|U|}, (1+\varepsilon)\frac{|S|}{|U|}\right]$

Beweis

Y nicht im Intervall: Y zu klein / Y zu gross

$$\Pr\left[Y < (1-\varepsilon)\tfrac{|S|}{|U|}\right] + \Pr\left[Y > (1+\varepsilon)\tfrac{|S|}{|U|}\right]$$

↳ $<$ wird zu $\leq$, und $>$ wird zu $\geq$

$$\leq \Pr\left[Y \leq (1-\varepsilon)\tfrac{|S|}{|U|}\right] + \Pr\left[Y \geq (1+\varepsilon)\tfrac{|S|}{|U|}\right]$$

↳ $\mathbb{E}[Y] = \frac{|S|}{|U|}$

$$= \Pr\left[Y \leq (1-\varepsilon)\mathbb{E}[Y]\right] + \Pr\left[Y \geq (1+\varepsilon)\mathbb{E}[Y]\right]$$

↳ beide Seiten der Ungleichungen mal N

$$= \Pr\left[NY \leq (1-\varepsilon)\mathbb{E}[NY]\right] + \Pr\left[NY \geq (1+\varepsilon)\mathbb{E}[NY]\right]$$

↳ Chernoff i) rechts und ii) links

$$\leq e^{-\frac{1}{2}\varepsilon^2 \mathbb{E}[NY]} + e^{-\frac{1}{3}\varepsilon^2 \mathbb{E}[NY]}$$

$\leq 2\, e^{-\frac{1}{3}\varepsilon^2 \mathbb{E}[NY]}$ ↳ weil $e^{-\frac{1}{3}} > e^{-\frac{1}{2}}$

$= 2\, e^{-\frac{1}{3}\varepsilon^2 N \frac{|S|}{|U|}}$ ↳ $\mathbb{E}[NY] = N\,\mathbb{E}[Y] = N \cdot \frac{|S|}{|U|}$

Wir setzen $2\, e^{-\frac{1}{3}\varepsilon^2 N \frac{|S|}{|U|}} \leq \delta$ und lösen nach N auf

$\Leftrightarrow e^{-\frac{1}{3}\varepsilon^2 N \frac{|S|}{|U|}} \leq \frac{\delta}{2}$ ↳ $\cdot \frac{1}{2}$ auf beiden Seiten

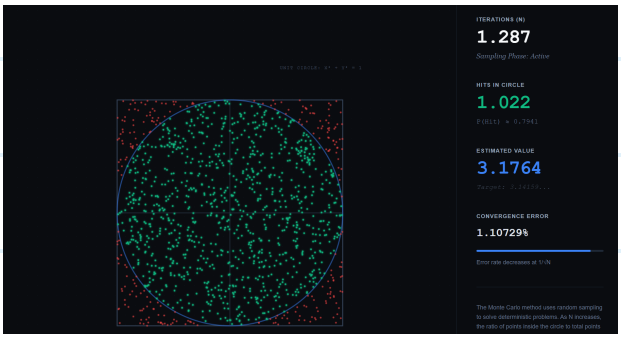
$\Leftrightarrow -\frac{1}{3}\varepsilon^2 N \cdot \frac{|S|}{|U|} \leq \ln\left(\frac{\delta}{2}\right)$ ↳ $\ln(\,)$ auf beiden Seiten

$\Leftrightarrow N \geq -3\frac{|U|}{|S|}\,\varepsilon^{-2} \ln\left(\frac{\delta}{2}\right)$ ↳ $\cdot -3\frac{|U|}{|S|}\varepsilon^{-2}$, $\leq$ dreht sich um bei Multiplikation mit negativem Wert

$\Leftrightarrow N \geq 3\frac{|U|}{|S|}\,\varepsilon^{-2} \ln\left(\frac{2}{\delta}\right)$ ↳ Logarithmusgesetz: $\ln\left(\frac{a}{b}\right) = \ln\left(\left(\frac{b}{a}\right)^{-1}\right) = -\ln\left(\frac{b}{a}\right)$

- solche Beweise verstehen, d.h schrittweise nachvollziehen ist wichtig, aber man muss sie nicht auswendig können

Bsp.



→ Randomisierte Algorithmen

Monte-Carlo vs Las-Vegas

- Korrektheit ist eine Zufallsvariable
- "meistens" korrekt
- immer schnell

- Laufzeit ist eine Zufallsvariable
- "meistens" schnell
- immer korrekt
- äquivalente Definition: Algorithmus, der immer schnell ist, aber manchmal ??? ausgibt, und sonst immer korrekt ist

"=>"
normalen Las-Vegas Algorithmus nach konstanter Zeit abbrechen und ??? ausgeben falls das Ergebnis noch nicht gefunden wurde

"<="
wiederholen, solange ??? ausgegeben wird

- kann in Monte-Carlo umgewandelt werden
(nach konstanter Zeit abbrechen und raten falls das Ergebnis noch nicht gefunden wurde)

Aufgabe:

Entscheide ob es sich um einen Monte-Carlo oder einen Las-Vegas Algorithmus handelt

- Maximum Suche: Wähle ein zufälliges Element und prüfe ob es grösser gleich alle anderen ist. Falls ja, dann gib das Element aus, falls nein, wiederhole.

Las-Vegas

- Target Shooting, wobei wir eine Ausgabe als korrekt werten, falls sie im Intervall ist.

Monte-Carlo

- Target Shooting, wobei wir eine Ausgabe als korrekt werten, falls die Ausgabe exakt dem echten Verhältnis $\frac{|S|}{|U|}$ entspricht.

keines von beiden · z.B π ist irrational, d.h. die Ausgabe wird nie exakt sein

- QuickSort mit zufälliger Pivot-Element Wahl

Las-Vegas

- Prüfe ob ein Graph ein Dreieck hat: Drei zufällige Knoten werden ausgewählt. Falls sie ein Dreieck bilden gib JA zurück, ansonsten gib NEIN zurück.

Monte-Carlo

- Suche einen Hamiltonkreis in einem Graphen in einem Hamiltonschen Graphen: Wähle eine zufällige Permutation der Knoten und prüfe ob zwischen allen Knoten eine Kante existiert. Falls ja, dann gib die Permutation aus, falls nein, wiederhole.

Las-Vegas

Fehlerreduktion

Für Las-Vegas:

- Sei $\mathcal{A}$ ein Las-Vegas Algorithmus der manchmal ??? ausgibt
- aktuelle Korrektheit $\cdot \Pr[\mathcal{A}(I) \text{ ist korrekt}] \geq \varepsilon$
- Wir wählen den Zielfehler δ
- $\mathcal{A}_\delta$ ruft $\mathcal{A}$ bis zu $N := \lceil \varepsilon^{-1} \ln(\delta^{-1}) \rceil$ mal auf
  - sobald $\mathcal{A}$ etwas anderes als ??? ausgibt, gibt $\mathcal{A}_\delta$ dasselbe aus
  - sonst gibt $\mathcal{A}_\delta$ am Ende ??? aus
- $\Pr[\mathcal{A}_\delta(I) \text{ ist korrekt}] \geq 1 - \delta$
- Beweis: $\Pr[\mathcal{A}_\delta(I) = ???] = \Pr[\mathcal{A}(I) = ???]^N$

  $\leq (1-\varepsilon)^N$

  $\leq (e^{-\varepsilon})^N$

  ↳ $\Pr[\mathcal{A}(I) = ???] \geq 1 - \varepsilon$

  ↳ $1 + x \leq e^x \quad \forall x \in \mathbb{R}$

  hier mit $x = -\varepsilon$

$$\leq e^{-\varepsilon \varepsilon^{-1} \ln(\delta^{-1})}$$ ↳ def $N = \lceil \varepsilon^{-1} \cdot \ln(\delta^{-1}) \rceil$

$$= e^{-\ln(\delta^{-1})}$$ ↳ $\varepsilon \varepsilon^{-1} = 1$

$$= e^{\ln(\delta)}$$ ↳ Logarithmusgesetz: $\ln(x^{-1}) = -\ln(x)$

$$= \delta$$ ↳ e^x und $\ln(x)$ sind Umkehrfunktionen

Für Monte-Carlo mit einseitigem Fehler:

- Sei $\mathcal{A}$ ein Monte-Carlo Algorithmus der immer korrekt ist für JA-Instanzen, aber nur mit Wahrscheinlichkeit $\geq \varepsilon$ korrekt ist für NEIN-Instanzen
- Wir wählen den Zielfehler δ
- $\mathcal{A}_\delta$ ruft $\mathcal{A}$ bis zu $N := \lceil \varepsilon^{-1} \ln(\delta^{-1}) \rceil$ mal auf
  - sobald $\mathcal{A}$ einmal NEIN ausgibt, gibt $\mathcal{A}_\delta$ auch NEIN aus
  - sonst gibt $\mathcal{A}_\delta$ am Ende JA aus
- $Pr[\mathcal{A}_\delta(I) \text{ ist korrekt}] \geq 1 - \delta$

Für Monte-Carlo mit Fehlerwahrscheinlichkeit $< \frac{1}{2}$:

- Sei $\mathcal{A}$ ein Monte-Carlo Algorithmus der immer entweder JA oder NEIN ausgibt
- aktuelle Korrektheit $\cdot\ Pr[\mathcal{A}(I) \text{ ist korrekt}] \geq \frac{1}{2} + \varepsilon$
- Wir wählen den Zielfehler δ
- $\mathcal{A}_\delta$ ruft $\mathcal{A}$ genau $N := \lceil 4 \cdot \varepsilon^{-2} \cdot \ln(\delta^{-1}) \rceil$ mal auf
  - $\mathcal{A}_\delta$ gibt am Ende die Mehrheit der JA's und NEIN's aus
- $Pr[\mathcal{A}_\delta(I) \text{ ist korrekt}] \geq 1 - \delta$

Aufgaben:

Ein Programm sucht nach einem Weg durch ein Labyrinth. In 20% der Fälle findet es blitzschnell den korrekten Weg zum Ziel. In den restlichen 80% der Fälle verläuft es sich in einer Endlosschleife, bricht irgendwann ab und gibt die Meldung "???" aus. Wir möchten den Weg mit einer Sicherheit von 99% finden.

Können wir Fehlerreduktion anwenden? Wenn ja, wie oft müssen wir ihn aufrufen?

Fehlerreduktion für Las-Vegas: Korrektheit: $\varepsilon = 0.2$

Zielfehler: $\delta = 0.01$

$$N = \lceil \varepsilon^{-1} \cdot \ln(\delta^{-1}) \rceil = \lceil \tfrac{1}{0.2} \ln(\tfrac{1}{0.01}) \rceil = \lceil 5 \cdot \ln(100) \rceil = \underline{\underline{24}}$$

Ein Algorithmus überprüft, ob ein Computernetzwerk sicher ist. Wenn das Netzwerk sicher ist (JA), sagt der Algorithmus immer die Wahrheit. Wenn es eine Sicherheitslücke gibt (NEIN), erkennt der Algorithmus diese Lücke leider nur in 25% der Fälle. Wir möchten das Netzwerk absichern und eine maximale Irrtumswahrscheinlichkeit von 5% zulassen.

Können wir Fehlerreduktion anwenden? Wenn ja, wie oft müssen wir ihn aufrufen?

Fehlerreduktion für Monte-Carlo mit einseitigem Fehler:

Korrektheit für NEIN-Instanzen. $\varepsilon = 0.25$

Zielfehler: $\delta = 0.05$

$$N = \lceil \varepsilon^{-1} \cdot \ln(\delta^{-1}) \rceil = \lceil \tfrac{1}{0.25} \ln(\tfrac{1}{0.05}) \rceil = \lceil 4 \ln(20) \rceil = \underline{\underline{12}}$$

Ein Algorithmus versucht vorherzusagen, ob eine Aktie morgen steigt (JA) oder fällt (NEIN). Er ist extrem schlecht und liegt nur in 40% der Fälle richtig. Wir wollen eine Sicherheit von 95% erreichen, um unser Geld sicher anzulegen.

Können wir Fehlerreduktion anwenden? Wenn ja, wie oft müssen wir ihn aufrufen?

Fehlerreduktion für Monte-Carlo mit Fehlerwahrscheinlichkeit $< \frac{1}{2}$:

Idee: wir negieren die Ausgabe des Algorithmus: $\Pr[\overline{A(I)} \text{ ist korrekt}] = 0.6 = \frac{1}{2} + \varepsilon$ wobei $\varepsilon = 0.1$

Zielfehler: $\delta = 0.05$

$$N = \lceil 4 \, \varepsilon^{-2} \ln(\delta^{-1}) \rceil = \lceil 4 \, \tfrac{1}{0.1^2} \ln(\tfrac{1}{0.05}) \rceil = \lceil 4 \cdot 100 \ln(20) \rceil = \underline{\underline{1199}}$$

Primzahl-Tests

Idee: n ist eine Primzahl $\Rightarrow$ $ggT(a,n) = 1 \quad \forall a \in \{1,2,..,n-1\}$

EUKLID-PRIMZAHLTEST(n)

1: Wähle $a \in [n-1]$, zufällig gleichverteilt
2: **if** $\text{ggT}(a,n) = 1$ **then return** 'Primzahl'
3: **else return** 'keine Primzahl'

- Immer korrekt für JA-Instanzen (= Primzahlen)
- Korrekt mit Wahrscheinlichkeit $\geq 1 - O(\frac{1}{n})$ für NEIN-Instanzen (= nicht-Primzahlen)
- Monte-Carlo Algorithmus mit einseitigem Fehler
- Laufzeit $O(\log(n)^3)$

DiskMat Recap.

a mod b = Rest, wenn wir b so oft wie möglich von a subtrahieren und ≥ 0 bleiben

Bsp 10 mod 3 = 1, $4 \equiv_2 0$, $7 \equiv_4 3$

$\mathbb{Z}_n = \{0,1,..,n-1\}$ ist eine additive Gruppe mit der Operation $\oplus_n$
↳ assoziativität, neutrales Element und Inverse bezüglich Addition

$\mathbb{Z}_n^* = \{a \in [n-1] \mid ggT(a,n) = 1\}$ ist eine multiplikative Gruppe mit der Operation $\odot_n$
(stellt sicher dass es multiplikative Inversen gibt)

Lagrange: $\forall a \in \mathbb{Z}_n^*: \quad a^{|\mathbb{Z}_n^*|} \equiv_n 1$

kleiner Fermatscher Satz: Falls p eine Primzahl ist, dann gilt für alle $a \in \{1, 2, .., p-1\}$:

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

FERMAT-PRIMZAHLTEST(n)

1: Wähle $a \in [n-1]$, zufällig gleichverteilt
2: **if** $a^{n-1} \bmod n = 1$ **then return** 'Primzahl'
3: **else return** 'keine Primzahl'

- Immer korrekt für JA-Instanzen (= Primzahlen)
- Korrekt mit Wahrscheinlichkeit $> \frac{1}{2}$ für NEIN-Instanzen (= nicht-Primzahlen) die keine Carmichael-Zahlen sind
- Ein Zeuge / Zertifikat ist ein $a \in \mathbb{Z}_n^*$, das bezeugt, dass n keine Primzahl ist (also hier falls $a^{n-1} \not\equiv_n 1$)
- Eine Carmichael-Zahl ist eine zusammengesetzte Zahl, für die es aber keine Zeugen gibt

  $\Leftrightarrow$ n ist nicht prim aber $\forall a \in \mathbb{Z}_n^*$ gilt $a^{n-1} \equiv_n 1$

- a ist eine Pseudoprimzahlbasis von n, falls $a^{n-1} \equiv_n 1$ aber n trotzdem keine Primzahl ist ("Lügner")
- Fehlerreduktion nicht möglich (wegen Carmichael-Zahlen)

Idee: $\mathbb{Z}_n^*$ ist ein Körper $\Leftrightarrow$ n ist prim

und ein Polynom vom Grad 2 hat höchstens 2 Nullstellen in einem Körper

$\Rightarrow P(x) = x^2 - 1$ hat höchstens zwei Nullstellen

$\Rightarrow$ $x^2 = 1$ hat höchstens zwei Lösungen, nämlich $x = 1$ und $x = -1 \equiv_n n-1$

$\Rightarrow$ Gegeben ein $a \in \{2, \dots, n-1\}$

falls $a^{n-1} \not\equiv_n 1$ dann ist n sicher keine Primzahl (Fermat)

falls $a^{n-1} \equiv_n 1$ dann machen wir weiter falls wir die Wurzel ziehen können.

falls $\sqrt{a^{n-1}} \notin \{1, n-1\}$ dann ist n sicher keine Primzahl

falls $\sqrt{a^{n-1}} \equiv_n n-1$ können wir nicht weiter machen

falls $\sqrt{a^{n-1}} \equiv_n 1$ dann machen wir weiter falls wir nocheinmal die Wurzel ziehen können.

falls $\sqrt{\sqrt{a^{n-1}}} \notin \{1, n-1\}$ dann ist n sicher keine Primzahl

usw..

Formeller: 1) Sei also $n > 2$ prim, dann ist $\mathbb{Z}_n^*$ ein Körper.

2) Finde k und d, sodass $n-1 = 2^k d$ und d ungerade ist

3.) $\forall a \in \{2, .., n-1\}\ \forall 1 \le i \le k$: $a^{2^i d} \equiv_n 1 \Rightarrow a^{2^{i-1} \cdot d} \in \{1, n-1\}$

$x^2 = 1 \Rightarrow \sqrt{x} \in \{1, n-1\}$

4) Also entweder $\forall 0 \le i \le k$: $a^{2^i d} \equiv_n 1$

oder $\exists i \in \{0, \dots, k-1\}$: $a^{2^i d} \equiv_n n-1$

falls keiner dieser Fälle eintritt ist p sicher keine Primzahl

---

MILLER-RABIN-PRIMZAHLTEST(n) ($n > 1$, ungerade!)

1: Schreibe $n - 1 = 2^k d$ mit $d, k \in \mathbb{N}$, d ungerade
2: Wähle $a \in [n-1]$, zufällig gleichverteilt
3: **if** $a^d \bmod n = 1$ oder $\exists i \in \{0, \dots, k-1\} : a^{2^i d} \bmod n = n-1$ **then return** 'Primzahl' ← educated guess weil die Zwischenschritte 1, 1, 1, 1, ... sind oder 1, 1, ., 1, n-1, ?, ?, .. sind
4: **else return** 'keine Primzahl' ← immer korrekt

Weil es noch nicht kompliziert genug ist, drehen wir die Reihenfolge um :)

Falls $x^2 = 1$, dann muss x vor dem Quadrieren 1 oder n-1 gewesen sein

MILLER-RABIN-PRIMZAHLTEST(n)

1: **if** $n = 2$ **then**
2: **return** 'Primzahl' ← immer korrekt :)
3: **else if** n gerade oder $n = 1$ **then**
4: **return** 'keine Primzahl' ← immer korrekt
5: Wähle $a \in \{2, 3, \ldots, n-1\}$ zufällig und
6: berechne $k, d \in \mathbb{Z}$ mit $n - 1 = d2^k$ und d ungerade.
7: $x \leftarrow a^d \pmod n$
8: **if** $x = 1$ or $x = n - 1$ **then**
9: **return** 'Primzahl' ← educated guess, weil die Zwischenschritte 1,1,1,1,... sind falls $x=1$ oder n-1,1,1,1,.. falls $x = n-1$
10: **repeat** $k - 1$ mal
11: $x \leftarrow x^2 \pmod n$
12: **if** $x = 1$ **then**
13: return 'keine Primzahl' ← immer korrekt, weil x vorher nicht 1 oder n-1 war
14: **if** $x = n - 1$ **then**
15: **return** 'Primzahl' ← educated guess, weil die Zwischenschritte ?, ?, .., n-1, 1, 1, .. sind
16: **return** 'keine Primzahl' ← immer korrekt, weil hier $a^{n-1} \not\equiv_n 1$ ist

- Immer korrekt für JA-Instanzen (= Primzahlen)
- Korrekt mit Wahrscheinlichkeit $\geq \frac{3}{4}$ für NEIN-Instanzen (= nicht-Primzahlen)
- Laufzeit: $O(\log n)$

Verteilung der Primzahlen: $\pi(x)$ zählt wie viele Primzahlen $\leq x$ es gibt Bsp: $\pi(10) = 4$

$\frac{\pi(x)}{x} \approx \frac{1}{\ln(x)}$ (Anteil Primzahlen der ersten x Zahlen) sogar: $\lim_{x \to \infty} \frac{\frac{\pi(x)}{x}}{\frac{1}{\ln(x)}} = 1$

wow!!! absolut wunderschön!!! unglaublich cool!!!

$\Pr[\text{zufällige Zahl aus } \{1, 2, \ldots, x\} \text{ ist prim}] = \frac{1}{\ln(x)}$ für $x \to \infty$

Aufgaben:

Ist die Fehlerwahrscheinlichkeit bei Euklids Primzahltest kleiner für $n = 15$ oder $n = 16$?

Fehler genau dann wenn $ggT(a,n) = 1$

$\mathbb{Z}_{15}^* = \{a \in \{1,..,14\} \mid ggT(a,15) = 1\} = \{1,2,4,7,8,11,13,14\}$ $\Rightarrow$ Fehlerwahrscheinlichkeit $= \frac{8}{14}$

$\mathbb{Z}_{16}^* = \{a \in \{1,..,15\} \mid ggT(a,16) = 1\} = \{1,3,5,7,9,11,13,15\}$ $\Rightarrow$ Fehlerwahrscheinlichkeit $= \frac{8}{15}$

$\Rightarrow$ kleinere Fehlerwahrscheinlichkeit bei $n = 16$.

Bestimme ob $a = 4$ und $a = 2$ Zertifikate oder Pseudoprimzahlbasen sind bei Fermats Primzahltest für $n = 15$.

(Formel: $a^{n-1} \equiv_n 1$)

Für $a = 4$: $4^{15-1} = 4^{14} = (4^2)^7 = 16^7 \equiv_{15} 1^7 = \underline{1}$

$\Rightarrow$ $a = 4$ ist eine Pseudoprimzahlbasis, Algorithmus gibt fälschlicherweise "Primzahl" aus

Für $a = 2$: $2^{15-1} = 2^{14} = 2^{12}\,2^2 = (2^4)^3\,4 = 16^3 \cdot 4 \equiv_{15} 1^3\,4 = \underline{4}$ $(\neq 1)$

$\Rightarrow$ $a = 2$ ist ein Zertifikat, Algorithmus gibt korrekt "keine Primzahl" aus

Was gibt der Miller-Rabin Primzahltest aus für $n = 13, a = 2$?

$n - 1 = 12 = 2^2\,3$ $\Rightarrow k = 2, d = 3$

Schritt $i = 0$: $2^3 \equiv_{13} 8$ (teste $a^{2^0 d} \bmod n$)

$8 \neq 1$ und $8 \neq 12$ $\Rightarrow$ weiter machen

Schritt $i = 1$: $2^{2\,3} \equiv_{13} 64 \equiv_{13} 12$ (teste $a^{2^1 d} \bmod n$)

$12 = 12$ ✓ $\Rightarrow$ return "Primzahl" (korrekt)

Was gibt der Miller-Rabin Primzahltest aus für $n = 21, a = 2$?

$n-1 = 20 = 2^2 \cdot 5 \quad \Rightarrow k=2, d=5$

Schritt $i=0$: $2^5 \equiv_{21} 32 \equiv_{21} 11$ (teste $a^{2^0 d} \bmod n$)

$11 \neq 1$ und $11 \neq 20 \quad \Rightarrow$ weiter machen

Schritt $i=1$: $2^{2^5} \equiv_{21} (2^5)^2 \equiv_{21} 11^2 \equiv_{21} 121 \equiv_{21} 16$ (teste $a^{2^1 d} \bmod n$)

$16 \neq 1$ und $16 \neq 20$.

Schleife endet $\Rightarrow$ return "keine Primzahl" (korrekt)

Was gibt der Miller-Rabin Primzahltest aus für $n = 17, a = 3$?

$n-1 = 16 = 2^4 \quad \Rightarrow k=4, d=1$

Schritt $i=0$: $3^1 \equiv_{17} 3$ (teste $a^{2^0 d} \bmod n$)

$3 \neq 1$ und $3 \neq 16 \quad \Rightarrow$ weiter machen

Schritt $i=1$: $3^2 \equiv_{17} 9$ (teste $a^{2^1 d} \bmod n$)

$9 \neq 1$ und $9 \neq 16$. $\Rightarrow$ weiter machen

Schritt $i=2$: $3^{2^2} \equiv_{17} 3^4 \equiv_{17} 81 \equiv_{17} 13$ (teste $a^{2^2 d} \bmod n$)

$13 \neq 1$ und $13 \neq 16$. $\Rightarrow$ weiter machen

Schritt $i=3$: $3^{2^3} \equiv_{17} (3^4)^2 \equiv_{17} 13^2 \equiv_{17} 169 \equiv_{17} 16$ (teste $a^{2^3 d} \bmod n$)

$16 \overset{\checkmark}{=} 16 \Rightarrow$ return "Primzahl" (korrekt)