

Algorithms and Probability

Week 9

G08 - mkilic

17.IV.2025

Overview

1. Peer Grading 3 - Ex. 1
2. Randomisierte Algorithmen & Fehlerreduktion
3. Target-Shooting
4. Primzahltest
5. Duplikate Finden
6. Exercises

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

Randomisierte Algorithmen

→ **Deterministisch:**

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

Randomisierte Algorithmen

→ **Deterministisch:**

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

→ **Randomisiert:**

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

Monte Carlo & Las Vegas

Monte Carlo Algorithmen

- immer schnell
- manchmal falsch
(Korrektheit ist Zufallsvariable)

Beispiele:

- ?

Las Vegas Algorithmen

- immer korrekt
- 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 Antwort 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

```
1 Monte_Carlo_Prime_Test(n, k):
2
3     repeat k times:
4
5         pick random a in [2, n-2]
6
7         if a^(n-1) mod n != 1:
8             return 'Composite'
9
10    return 'Probably Prime'
11
```

```
1 Las_Vegas_Prime_Test(n):
2
3     factors = shuffle([sqrt(n)])
4
5     for d in factors:
6
7         if n mod d == 0:
8             return 'Composite'
9
10    return 'Prime'
11
```

Reduktion der Fehlerwahrscheinlichkeit

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}_δ 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: Auf der Tafel (Skript s. 147)

Reduktion der Fehlerwahrscheinlichkeit

Reduktion der Fehlerw'keit - Monte Carlo - Einseitiger Fehler

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

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

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

Dann gilt für alle $\delta > 0$:

Sei \mathcal{A}_δ 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

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

$Pr[\mathcal{A}(\mathcal{I}) = \text{NEIN}] \geq \varepsilon$ 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

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

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

Monte Carlo mit einseitiger Fehler

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

$Pr[\mathcal{A}(\mathcal{I}) = \text{NEIN}] \geq \varepsilon$ 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 klassifiziert.
- Ein Beispiel ist der `Monte_Carlo_Prime_Test` aus Folie 7 und die Frage: "Ist mein Zahl zusammengesetzt?"

Reduktion der Fehlerwahrscheinlichkeit

Reduktion der Fehlerw'keit - Monte Carlo - Zweiseitiger Fehler

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

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

Dann gilt für alle $\delta > 0$: bezeichnet man mit \mathcal{A}_δ 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}_δ , 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 Algorithmus 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}_δ 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

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

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).

Target-Shooting

Target-Shooting

Target-Shooting

Gegeben: eine endliche Menge U und eine Untermenge $S \subseteq U$ unbekannter Grösse

Gesucht: $\frac{|S|}{|U|} = ?$

Ausserdem haben wir per Annahme die folgenden:

- eine Indikatorfunktion $I_S : U \rightarrow \{0, 1\}$, 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 Wähle  $u_1, \dots, u_N \in U$  i.i.d.
```

```
2
```

```
3 return  $\frac{1}{N} \cdot \sum_{i=1}^N I_S(u_i)$ 
```

```
4
```

TARGET-SHOOTING

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

Target-Shooting Analyse

- Definiere Indikatorvariablen $Y_i := I_S(u_i)$.
- $(Y_i)_{i \in \{1, \dots, N\}}$ sind unabhängige Bernoulli-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 Bernoulli-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

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

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 äquivalent zu

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

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

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

⁰Wir 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 kann 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 – Approximieren

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.

Primzahltest

Primzahltest

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

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

0 - Brute Force

```
1   for all  $a \leq \sqrt{n}$ :
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



Sensitive Content

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* $\varphi : \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$ 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 $\mathbb{Z}_p^* = \{1, \dots, p-1\} = \mathbb{Z}_p \setminus \{0\}$, and $\varphi(p) = p-1$.

Lemma 5.12. If the prime factorization of m is $m = \prod_{i=1}^r p_i^{e_i}$, then¹¹

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

1 - Euklid-Primzahltest

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

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

euklid_prim_test(n)

1 - Euklid-Primzahltest

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

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

```
1   choose  $a \in [n - 1]$  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

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

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 $0 < a < n$

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

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

Ferat - #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

Miller-Rabin: Grundidee

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

Besonders gilt:

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

Miller-Rabin: Zerlegung

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

Wenn n prim ist, gilt nach Fermat:

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

Daraus folgt:

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

Miller-Rabin: Was passiert bei Primzahlen

Aufgrund der Struktur von Primzahlen folgt aus:

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

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 $a^d, a^{2^d}, \dots, a^{2^i d}$ 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:

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

Wenn keine dieser Bedingungen erfüllt ist:

$\Rightarrow a$ ist ein Zertifikat dafür, dass n 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: Wähle  $a \in \{2, 3, \dots, n-1\}$  zufällig und
7 6: berechne  $k, d \in \mathbb{Z}$  mit  $n-1 = d \cdot 2^k$  und  $d$  ungerade.
8 7:  $x \leftarrow a^d \pmod n$ 
9 8: if  $x = 1$  or  $x = n-1$  then
10 9:     return 'Primzahl'
11 10: repeat  $k-1$  mal
12 11:      $x \leftarrow x^2 \pmod n$ 
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: $O(\ln n)^2$

W'keiten:

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

Wir können die Fehlerwahrscheinlichkeit mit Fehlerreduktionsmethoden beliebig klein machen.

²Wegen binäre Exponentiation & Successive squaring

Duplikate Finden

Duplikate Finden

Gegeben: Ein Datensatz $\mathcal{S} := \{s_1, \dots, s_n\}$

Gesucht: Duplikate, also alle Paare $1 \leq i < j \leq n$ mit $s_i = s_j$

Erste Lösung: Hashing

Wir brauchen eine Hashfunktion: $h : U \rightarrow [m]$ m ist die Anzahl verfügbaren Hashzellen³.

Wir haben:

$$\text{für jedes } u \in U, i \in [m] : Pr[h(u) = i] = \frac{1}{m}$$

Wir gehen wie folgt:

1. Berechne $h(u)$ für alle Elemente des Datensatzes
2. Erstelle HashMap mit Einträge $(h(s_i), i)$
3. Sortiere die Elemente bezüglich der ersten Koordinate
4. Vergleiche nur die Einträge auf Gleichheit, die dieselbe erste Koordinate haben

³Manchmal auch 'bucket' genannt.

Hashing: Erwartete Kollisionen

Kollisionen sind die unerwünschten Duplikate: "*false positives*"

Wir definieren die Bernoulli-Variablen:

$K_{i,j} = 1 \iff (i,j)$ ist eine Kollision

$$Pr[K_{i,j} = 1] = \begin{cases} \frac{1}{m}, & \text{falls } s_i \neq s_j \\ 0, & \text{otherwise} \end{cases}$$

also gilt: $\mathbb{E}[K_{i,j}] \leq \frac{1}{m}$

dann haben wir $\mathbb{E}[\#Kollisionen] = \sum_{1 \leq i < j \leq n} \mathbb{E}[K_{i,j}] \leq \binom{n}{2} \frac{1}{m}$

Hashing: Fazit

$$\mathbb{E}[\#\text{Kollisionen}] \leq \binom{n}{2} \cdot \frac{1}{m} < 1 \quad \text{für } m = n^2$$

Mit $m = n^2$ ist der erwartete Mehraufwand durch Kollisionen konstant.

Laufzeit:

- n Berechnungen der Hashfunktion
- $\mathcal{O}(n \log n)$ für Sortieren von $\lceil 2 \log n \rceil$ -Bit Zahlen
- $(|\text{Dupl}(\mathcal{D})| + \#\text{Kollisionen})$ Vergleiche zum Finden der Duplikate

Zweite Lösung: Bloom Filters

Idee

- Speicher- und Laufzeitoptimierung durch Bloomfilter
- Verwende k Hashfunktionen:

$$h_1, \dots, h_k : \mathcal{U} \rightarrow [m]$$

- Speicher: m Bits, initialisiere alle auf 0

Algorithmus

Für jedes s_i , $1 \leq i \leq n$:

1. Berechne:

$$(x_1, \dots, x_k) := (h_1(s_i), \dots, h_k(s_i))$$

2. Wenn $M[x_1] = \dots = M[x_k] = 1$, dann s_i zur Liste \mathcal{L} hinzufügen

3. Sonst setze $M[x_1], \dots, M[x_k] := 1$

Ziel und Fehler

- Ziel: \mathcal{L} enthält alle Wiederholungen mit wenigen False Positives
- False positives:

$$s_i \notin \{s_1, \dots, s_{i-1}\}, \quad M[h_j(s_i)] = 1 \text{ für alle } j$$

Wahrscheinlichkeit für False Positives

- Für n Datensätze, k Hashfunktionen:

$$\Pr[M[x] = 0] \leq \left(1 - \frac{1}{m}\right)^{k(n-1)}$$

- Wahrscheinlichkeit, dass $M[x] = 1$:
hoch \Rightarrow Wahrscheinlichkeit für False Positives steigt

$$\Pr[X_i = 1] = \Pr[M[h_j(s_i)] = 1 \text{ für alle } 1 \leq j \leq k] \leq \left(1 - \left(1 - \frac{1}{m}\right)^{k(n-1)}\right)^k.$$

Wahrscheinlichkeit für False Positives

$$\mathbb{E}[\#\text{falsche } \mathcal{L}\text{-Einträge}] \leq \sum_{i=1}^n \mathbb{E}[X_i] \leq n \cdot \left(1 - \left(1 - \frac{1}{m}\right)^{k(n-1)}\right)^k.$$

Wenn wir daher k und m so wählen, dass

$$\left(1 - \left(1 - \frac{1}{m}\right)^{k(n-1)}\right)^k = \mathcal{O}\left(\frac{1}{n}\right)$$

gilt, so erwarten wir nur konstant viele “false positives”.

Exercises

Quiz auf der Webseite!

The End

