18.09.24
Nicolas Wehrli

AnW - Woche 8

Heute:

I. Minitest
II. Kurze Kommentare zur Theorieaufgabe 3
III. Theory Recap
- Randomisierte Algorithmen
- Target Shooting
- Duplikate finden ?
- Primzahltest
- ~~Lange Pfade~~
IV. Aufgabe

I. Minitest

Passwort: capital

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.

Richtig	Falsch		
☑	☐	$\mathbb{E}[X] = 100\|U\|/\|S\|$	✔
☐	☑	Falls $X = 100$, dann $\|S\| = \|U\|$.	✔
☑	☐	Falls $\|S\| = \|U\|$, dann $X = 100$.	✔
☐	☑	X ist geometrisch verteilt.	✔

- X ist negativ binomialverteilt mit $n=100$ und $p = \frac{|S|}{|U|}$.

1. $\mathbb{E}[X] = \frac{n}{p} = 100 \cdot \frac{|U|}{|S|}$
2. Falsch, kann Zufall sein.
3. $|S|=|U| \Rightarrow p = \frac{|S|}{|U|} = 1 \Rightarrow Pr[X=100] = 1$
4. Nein, es ist die Summe von 100 geometrisch verteilten Zufallsvariablen.

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...

- ☐ ✖ ...$\mathbb{E}[Z] = 10^3$ Elemente in S.

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

$$Z \sim \text{Bin}\left(N, \frac{|S|}{|U|}\right)$$

$$\mathbb{E}[Z] = 10^6 \cdot \frac{10^3}{10^8} = \underline{\underline{10^1}}$$

Seien X und Y unabhängige Zufallsvariablen. Dann gilt $\mathbb{E}[\max(X, Y)] = \max(\mathbb{E}[X], \mathbb{E}[Y])$.

Bitte wählen Sie eine Antwort:

- Wahr ○
- Falsch ⊗

Sei $X \sim \text{Ber}\left(\frac{1}{2}\right)$, $Y \sim \text{Ber}\left(\frac{1}{2}\right)$ unabhängig.

$$\mathbb{E}[X] = \frac{1}{2}, \quad \mathbb{E}[Y] = \frac{1}{2}$$

$$\max\left(\mathbb{E}[X], \mathbb{E}[Y]\right) = \frac{1}{2}$$

$$\max(X, Y) = \begin{cases} 1 & \text{in } 3/4 \text{ Fällen} \\ 0 & \text{sonst} \end{cases} \sim \text{Ber}\left(\frac{3}{4}\right)$$

$$\mathbb{E}[\max(X, Y)] = \frac{3}{4}$$

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

Bitte wählen Sie eine Antwort:

- Wahr ✔
- Falsch

Sei X die Laufzeit vom Algorithmus.

$\mathbb{E}[X] = T$. Da $X \geq 0$ können wir Markov verwenden.

Misserfolg: $\Pr[X > 10T] \leq \Pr[X \geq 10T] \leq \frac{\mathbb{E}[X]}{10T} = \frac{1}{10}$

$\Rightarrow$ Wenn wir den Algo nach $10T$ abbrechen, erhalten wir einen Las Vegas Algo mit Erfolgsw'keit von mind. 0.9.

Seien X und Y zwei Zufallsvariabeln mit $\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y]$. Dann sind X und Y unabhängig.

Bitte wählen Sie eine Antwort:

- Wahr
- Falsch ✔

$\mathbb{E}[XY] = \mathbb{E}[X] \cdot \mathbb{E}[Y]$ $\not\Leftrightarrow$ X, Y unabhängig

X nimmt $-1, 0, 1$ mit W'keit $\frac{1}{3}$ an.

$$Y = \begin{cases} 1, & \text{falls } X = 0 \\ 0, & \text{sonst.} \end{cases}$$

$\mathbb{E}[X \cdot Y] = 0 = \underbrace{\mathbb{E}[X]}_{0} \cdot \mathbb{E}[Y]$, aber X, Y nicht unabhängig

Weiteres Gegenbeispiel:

Sei $\Omega = \{(0,1), (1,0), (-1,0), (0,-1)\}$

$X((\omega_1,\omega_2)) = \omega_1$, $Y((\omega_1,\omega_2)) = \omega_2$

$$Pr[X=x] = \begin{cases} \frac{1}{4}, & x=-1 \\ \frac{1}{2}, & x=0 \\ \frac{1}{4}, & x=1 \end{cases} \qquad Pr[Y=y] = \begin{cases} \frac{1}{4}, & y=-1 \\ \frac{1}{2}, & y=0 \\ \frac{1}{4}, & y=1 \end{cases}$$

$\mathbb{E}[X] = \mathbb{E}[Y] = 0$, $\mathbb{E}[XY] = 0$

aber $Pr[X=1] \cdot Pr[Y=1] = \frac{1}{4} \cdot \frac{1}{4} \neq 0 = Pr[X=1, Y=1]$

Seien X, Y, Z drei Zufallsvariabeln wobei X, Y unabhängig sind, dann gilt immer $E[X + Y \cdot Z] = E[X] + E[Y] \cdot E[Z]$

Bitte wählen Sie eine Antwort:

- ○ Wahr
- ◉ Falsch ✔

$$\mathbb{E}(X + Y \cdot Z) = \mathbb{E}(X) + \mathbb{E}(Y \cdot Z)$$

$\neq \mathbb{E}(Y) \cdot \mathbb{E}(Z)$ (im allg.)

nur X, Y unabh.
nicht Y, Z !

Sarah besitzt zwei sehr spezielle Münzen: eine zeigt immer 'Kopf' (weil auf beiden Seiten 'Kopf' abgebildet ist ...), die andere immer 'Zahl'. Sie wählt eine der beiden Münzen zufällig und wirft sie $n = 1000$ mal und bezeichnet mit X bezeichne die Anzahl 'Kopf' die sie sieht.

Dann gilt für $\delta := 1/4$:

$\Pr[X \geq (1+\delta)n/2] \leq e^{-\delta^2 n/6}$.

Bitte wählen Sie eine Antwort:

- Wahr
- Falsch ✔

Die einzelnen Münzwürfe sind nicht unabhängig.

Deshalb können wir Chernoff nicht verwenden.

Seien X, Y unabhängige Zufallsvariablen mit $Var(X) = 1$ und $Var(Y) = 4$.
Dann ist $Var(X - Y) = -3$.

Bitte wählen Sie eine Antwort:

- Wahr
- Falsch ✔

$$Var(X-Y) = Var(X) + Var(-Y) = Var(X) + (-1)^2 \cdot Var(Y)$$

$$= 1 + 4 = \underline{\underline{5}}$$

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?

Antwort: 1/p ✔

$X \sim$ Geometrisch verteilt mit p.

$$\mathbb{E}[X] = \frac{1}{p}$$

Jeder Monte Carlo Algorithmus kann in einen Las Vegas Algorithmus umgewandelt werden.

Bitte wählen Sie eine Antwort:

- Wahr
- Falsch ✔

Wir können die Erfolgswahrscheinlichkeit des Monte-Carlo Algo. M beliebig nah an 1 bringen, indem wir M wiederholt ausführen und am Ende die häufigste Antwort ausgeben.

Sei n die Anzahl Wiederholungen.

Die W'keit, das M nie die richtige Lösung ausgibt:

$$Pr[M(I) \text{ falsch}]^n > 0$$

Eine kleine Fehlern'keit bleibt immer.

Ein deterministischer Algorithmus kann immer als randomisierter Algorithmus gesehen werden.

Bitte wählen Sie eine Antwort:

- Wahr ✔
- Falsch

Jep.

I. Kurze Kommentare zur Theorieaufgabe 3

1. b und 2. wurden meist gut gelöst.

Bei 2. gewisse Schritte besser begründen:

Bsp. $Pr[B \cap A^c] = Pr[B] - Pr[B \cap A]$ (Satz der totalen W'keit oder Additionsatz)

oder $Pr[\bar{A} \cap \bar{B}] = 1 - Pr[A \cup B]$ ($\bar{A} \cap \bar{B} = \overline{A \cup B}$ Ide d'Morgan)

Nun zu 1a.):

Ich habe bei Abgaben mit den 2 folgenden Fehlern immer noch volle Punktzahl gegeben, aber nur wenn alles andere richtig war!

1. Fehler

$\Omega = \{S \mid S \subseteq V\}$ ist ein Laplace-Raum.

D.h. $\forall S \in \Omega$ gilt: $Pr[S] = \frac{1}{|\Omega|}$

Daraus folgt nicht automatisch, dass $Pr[u \in S] = Pr[u \notin S] = \frac{1}{2}$.

Dazu müsst ihr euch überlegen, in wievielen Teilmengen $S \subseteq V$ u enthalten ist.

(Bitstring Approach: $2^{|V|}$ viele Teilmengen. Wenn wir das bit für u fixieren, sind noch $2^{|V|-1}$ möglich ....)

2. Fehler

$$\Pr[\underbrace{u \in S}_{E_u} \wedge \underbrace{v \in V \setminus S}_{E_v}] = \underbrace{\Pr[E_u] \cdot \Pr[E_v]}_{\frac{1}{2} \cdot \frac{1}{2}}$$

Nur wenn E_u und E_v unabhängig!?

Ihr dürft dies nicht einfach so annehmen!

Noch 2 kleine Dinge:

Wenn ihr die Anzahl Teilmengen ausrechnen wollt $\Longrightarrow$ Bitstring 2^n

$\sum_{i=1}^{n} \binom{n}{i}$ ist auch richtig aber umständlicher. Approach

Verwendet $\mathbb{E}[|S|]$ nicht in der Berechnung für $\Pr[u \in S]$ etc. Das kommt sehr selten gut. In diesem Fall stimmt das Resultat da die Knoten gleichverteilt sind.

III. Theory Recap

Randomisierte Algorithmen

Klassisch (wie in AnD):

Für eine Eingabe I, ist $A(I)$ die Ausgabe des deterministischen Algorithmus.

Dann beweisen wir:

1. Korrektheit: für alle Eingaben I: $A(I)$ korrekt

2. Laufzeit: für alle Eingaben I mit $|I|=n$: Laufzeit in $O(f(n))$

Neu:

Für eine Eingabe I und Zufallszahlen R, ist $A(I,R)$ die Ausgabe des nichtdeterministischen Algorithmus.

Nun beweisen wir:

1. Korrektheit: für alle Eingaben I gilt: $Pr[A(I,R) \text{ korrekt}] \geq \ldots$

(Menge)

2. Laufzeit:

für alle Eingaben I mit $|I|=n$: $\mathbb{E}[\text{Laufzeit}] = O(f(n))$

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

Wir klassifizieren 2 Arten von randomisierten Algorithmen.

Las-Vegas-Algorithmen (Bsp. Quicksort)

- Geben nie eine falsche Antwort, aber die Laufzeit ist eine Zufallsvariable T.

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

Alternative Definition:

Wir können den Algorithmus abbrechen und '???' ausgeben lassen, falls eine gewisse Laufzeit überschreitet.
(Um dies zur vorigen Def. äquivalent zu machen, können wir den Algo solange wiederholen bis er nicht mehr '???' ausgibt)

Ziel: Pr[Antwort „???“] = „winzig“

Monte-Carlo-Algorithmus (Testen ob eine Münze fair ist)

- Laufzeit immer polynomiell, aber liefert manchmal falsche Antwort.

Ziel: Pr[Antwort falsch] = „winzig“

Satz:
- QuickSort bestimmt *immer* das richtige Ergebnis
- **E**[Laufzeit] = O(n ln n)

Las-Vegas

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

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

Fehlerw'lichkeit: 0, falls Münze fake
$(1/2)^{100}$, falls Münze fair

Monte-Carlo

Einseitiger Fehler

Analyse – Las Vegas

Satz 2.72. Sei $\mathcal{A}$ ein randomisierter Algorithmus, der nie eine falsche Antwort gibt, aber zuweilen '???' ausgibt, wobei

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

Dann gilt für alle $\delta > 0$: bezeichnet man mit $\mathcal{A}_\delta$ den Algorithmus, der $\mathcal{A}$ solange aufruft, bis entweder ein Wert verschieden von '???' ausgegeben wird (und $\mathcal{A}_\delta$ diesen Wert dann ebenfalls ausgibt) oder bis $N = \varepsilon^{-1} \ln \delta^{-1}$-mal '???' ausgegeben wurde (und $\mathcal{A}_\delta$ dann ebenfalls '???' ausgibt), so gilt für den Algorithmus $\mathcal{A}_\delta$, dass

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

Beweis: $\Pr[\underbrace{\mathcal{A} \text{ gibt } N\text{-mal '???'}}_{B}] \leq (1-\varepsilon)^N$

Da $1-x \leq e^{-x}$, $\forall x \in \mathbb{R}$ (Beweis per Analysis $f(x) = e^{-x} + x \geq 1 \; \forall x \in \mathbb{R}$)

folgt $(1-\varepsilon)^N \leq (e^{-\varepsilon})^N$

$$\Rightarrow \Pr[B] \leq e^{-\varepsilon N} = \delta \quad | \ln(\,)$$

$$-\varepsilon \cdot N \cdot \ln(e) = \ln(\delta) \quad | : (-\varepsilon)$$

$$N = -\varepsilon^{-1} \cdot \ln(\delta)$$

$$\underline{N = \varepsilon^{-1} \cdot \ln(\delta^{-1})}$$

Analyse - Monte-Carlo

Einseitiger Fehler

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

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

und

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

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 $N = \varepsilon^{-1} \ln \delta^{-1}$-mal JA ausgegeben wurde (und $\mathcal{A}_\delta$ dann ebenfalls JA ausgibt), so gilt für alle Instanzen I

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

Beweis: Falls I eine Ja-Instanz ist gilt $\Pr[A_\delta(I) \text{ korrekt}] = 1$

Falls I eine Nein-Instanz ist, gilt für

jeden Aufruf von A: $\Pr[A(I)=\text{Nein}] \geq \varepsilon$.

Die W'keit, dass dann in $N = \varepsilon^{-1} \ln(\delta^{-1})$ unabhängigen Aufruten kein einziges 'Nein' ausgegeben wird ist:

$$(1-\varepsilon)^N \leq e^{-\varepsilon N} = e^{\ln \delta} = \delta \qquad \square$$

Zweiseitiger Fehler

Satz 2.75. Sei $\varepsilon > 0$ und $\mathcal{A}$ ein randomisierter Algorithmus, der immer eine der beiden Antworten JA oder NEIN ausgibt, wobei

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

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

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

Beweis: lassen wir aus 😊

Optimierungsprobleme ausgelassen (siehe Skript)

Target Shooting

Problemstellung: Gegeben $S \subseteq U$ endlich.

Bestimme $\approx \frac{|S|}{|U|}$

Annahmen: Wir können Elemente von U gleichverteilt ziehen.

Wir haben eine Indikatorfunktion $I_S : U \to \{0,1\}$

$$I_S(u) = 1 \iff u \in S$$

Algo:

Target-Shooting

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

Wie gross sollte N sein?

Wir definieren $Y_i = I_S(u_i)$ für $i = 1, \ldots, N$

wobei Y_i unabhängig und $Y_i \sim \text{Ber}(p)$ $p := \frac{|S|}{|U|}$

Sei $Y := \frac{1}{N} \sum_{i=1}^{N} Y_i$ die Ausgabe des Algorithmus.

$$\mathbb{E}[Y] = \frac{1}{N} \mathbb{E}\left[\sum_{i=1}^{N} Y_i\right] = \frac{1}{N} \cdot \sum_{i=1}^{N} \mathbb{E}[Y_i] = \frac{1}{N}(N \cdot p) = p$$

$$\text{Var}[Y] = \frac{1}{N^2} \text{Var}\left[\sum_{i=1}^{N} Y_i\right] = \frac{1}{N^2} \sum_{i=1}^{N} \text{Var}[Y_i] = \frac{1}{N^2} \sum_{i=1}^{N} p(1-p) = \frac{1}{N}(p - p^2)$$

(Y_i unabhängig)

Wir wollen nun, dass für beliebig kleine, gegebene $\varepsilon, \delta > 0$,

$$(*) \quad \Pr\left[\left|Y - \frac{|S|}{|U|}\right| \leq \varepsilon \frac{|S|}{|U|}\right] \geq 1 - \delta \quad \text{gilt.}$$

Nun können wir N in Abhängigkeit von ε, δ und $\frac{|S|}{|U|}$ bestimmen, so dass $(*)$ erfüllt wird.

Satz

Seien $\delta, \varepsilon > 0$. Falls $N \geq 3\frac{|U|}{|S|} \cdot \varepsilon^{-2} \cdot \ln(2/\delta)$, ist die Ausgabe Y von Target-Shooting *mit Wahrscheinlichkeit mindestens $1 - \delta$ im Intervall $\left[\frac{|S|}{|U|} \pm \varepsilon \frac{|S|}{|U|}\right]$ (multiplikativer Fehler von $1 \pm \varepsilon$).*

Beweis:
äquivalent formuliert

$$\Pr\left[\left|Y - \frac{|S|}{|U|}\right| > \varepsilon \frac{|S|}{|U|}\right] = \Pr\left[|Y - \mathbb{E}[Y]| > \varepsilon\, \mathbb{E}[Y]\right]$$

$$= \Pr\left[|Z_N - \mathbb{E}[Z_N]| > \varepsilon\, \mathbb{E}[Z_N]\right] \leq \delta$$

Beachte: $Z_N := N \cdot Y$ (zeigt die ~~Abhängigkeit von N~~ (Y_N wird schon verwendet))

Da $Z_N = Y_1 + \ldots + Y_N$ eine Summe von N unabhängigen Zufallsvariablen ist $\Rightarrow$ Chernoff.

$$\Pr\left[|Z_N - \mathbb{E}[Z_N]| > \varepsilon\, \mathbb{E}[Z_N]\right] = \Pr\left[Z_N > (1+\varepsilon)\,\mathbb{E}[Z_N]\right] + \Pr\left[Z_N < (1-\varepsilon)\,\mathbb{E}[Z_N]\right]$$

A

!

Satz 2.70 (Chernoff-Schranken). Seien $X_1, \ldots, X_n$ unabhängige Bernoulli-verteilte Zufallsvariablen mit $\Pr[X_i = 1] = p_i$ and $\Pr[X_i = 0] = 1 - p_i$. Dann gilt für $X := \sum_{i=1}^{n} X_i$:

(i) $\Pr[X \geq (1+\delta)\mathbb{E}[X]] \leq e^{-\frac{1}{3}\delta^2 \mathbb{E}[X]}$ für alle $0 < \delta \leq 1$,

(ii) $\Pr[X \leq (1-\delta)\mathbb{E}[X]] \leq e^{-\frac{1}{2}\delta^2 \mathbb{E}[X]}$ für alle $0 < \delta \leq 1$,

(iii) $\Pr[X \geq t] \leq 2^{-t}$ für $t \geq 2e\mathbb{E}[X]$.

Per (i) und (ii)

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

$$\leq 2 \cdot e^{-\frac{1}{3}\varepsilon^2 \mathbb{E}[Z_N]} = 2 \cdot e^{-\frac{\varepsilon^2 N |S|}{3|U|}} \overset{\text{soll}}{\leq} \delta \quad | : 2$$

$$e^{-\frac{\varepsilon^2 N |S|}{3|U|}} \leq \frac{\delta}{2} \quad | \ln$$

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

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

Duplikate finden

$\mathcal{S} = (s_1, s_2, \ldots, s_n)$, Folge von n Elementen (**Datensatz**).
(i, j), $1 \leq i < j \leq n$, heisst **Duplikat** in $\mathcal{S}$, falls $s_i = s_j$.

Gegeben ein Datensatz $\mathcal{S}$, finde alle Duplikate.

$$\mathcal{S} = (\underset{1}{A}\ \underset{2}{C}\ \underset{3}{B}\ \underset{4}{Z}\ \underset{5}{C}\ \underset{6}{B}\ \underset{7}{C})$$

Duplikate $\mathrm{Dupl}(\mathcal{S}) = \{(2,5), (2,7), (5,7), (3,6)\}$ (4 Duplikate)

Elemente in S sind sehr gross.
Speicherzugriffe und Vergleiche sind recht teuer.

Wir verwenden Hashfunktionen. Seien die Elemente von S aus einem Universum U.

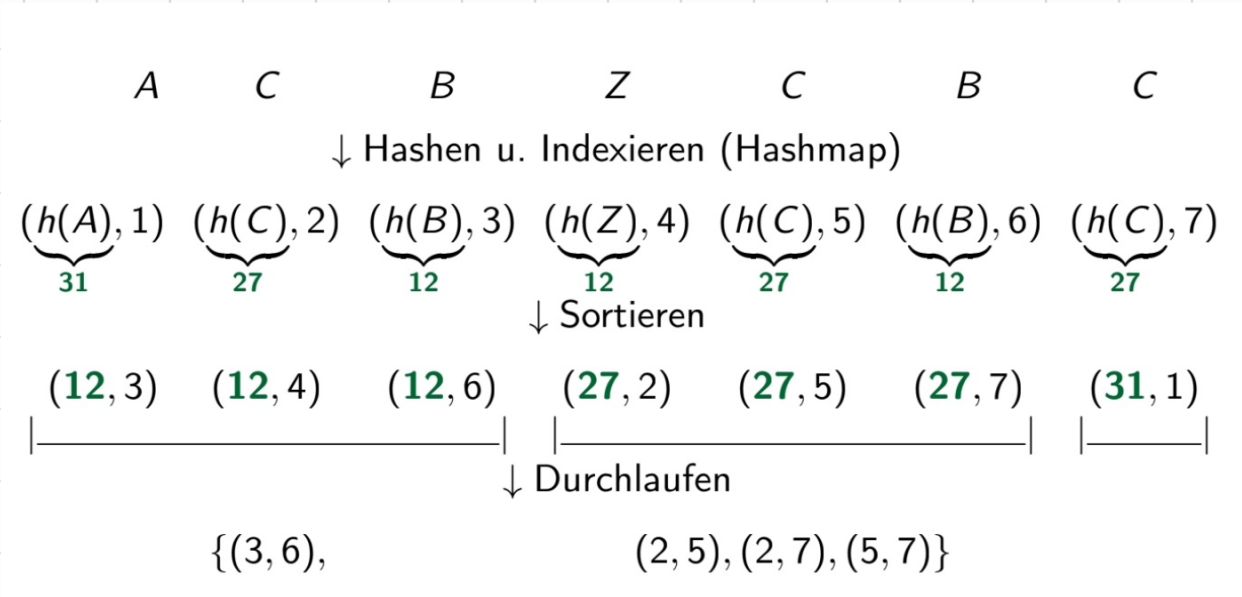
Dann für $h: U \to [m]$ nehmen wir folgendes an:

- h ist effizient berechenbar.
- h verhält sich wie eine Zufallsfunktion, d.h.

$$\forall u \in U\ \forall i \in [m] : \Pr[h(u) = i] = \frac{1}{m} \quad \text{(unabhängig)}$$

Naives Verfahren wäre: Sortieren & vergleichen in $O(n \log n)$

Nun machen wir



in $O(n \log n)$.

Auf den ersten Blick scheint sich nichts verändert zu haben.

Was sich aber verändert hat, ist die Grösse der Elemente die wir vergleichen und speichern.

Kollisionen sind die neuen (unerwünschten) Duplikate.
I.e. (i,j), $1 \leq i < j \leq n$ mit $s_i \neq s_j$
aber $h(s_i) = h(s_j)$.

Sei $K_{i,j} = 1 \Leftrightarrow (i,j)$ ist eine Kollision (bernoulli-verteilt)

$$\Pr[K_{i,j} = 1] = \begin{cases} 1/m & \text{falls } s_i \neq s_j, \\ 0 & \text{sonst,} \end{cases} \quad \text{und daher} \quad \mathbb{E}[K_{i,j}] \leq \frac{1}{m}.$$

Daraus folgt:

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

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

Alternativer Ansatz: Bloom Filter

Wir wählen $m, k \in \mathbb{N}$ und k Hashfunktionen (zufällig)

$$h_i : U \to [m], \quad i = 1, \ldots, k \ .$$

So assoziieren wir mit jedem s_i in $\mathcal{S}$ einen Hash-Vektor

$$(x_1, \ldots, x_k) = (x_1^{(i)}, \ldots, x_k^{(i)}) := (h_1(s_i), \ldots, h_k(s_i))$$

Dann bereiten wir noch ein boolsches Array $M[1,\ldots m]$ vor.

Beispiel:

$$\mathcal{S} = (\underset{1}{A}\ \underset{2}{C}\ \underset{3}{B}\ \underset{4}{Z}\ \underset{5}{F}\ \underset{6}{H}\ \underline{\underset{7}{C}}\ )$$

$m = 9,\ k = 3.$

$$A \mapsto (5,4,1), B \mapsto (4,6,1), C \mapsto (5,1,2),$$
$$F \mapsto (1,9,4), H \mapsto (4,7,5), Z \mapsto (4,2,5)$$

$$M = (\underset{1}{1}\ \underset{2}{1}\ \underset{3}{0}\ \underset{4}{1}\ \underset{5}{1}\ \underset{6}{1}\ \underset{7}{1}\ \underset{8}{0}\ \underset{9}{1}\ )$$

$$\mathcal{L} = \{4, 7\}$$

Analyse lassen wir.

Resultate:

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

Dieser Wert ist $O(1)$ für $k = \ln n$ und $m = n \ln n$.

k und m gross $\Rightarrow$ #Falsche Einträge klein.

Laufzeit: kn Hashfunktionsberechnungen. k gross $\Rightarrow$ langsamer.

Zusätzlicher Speicher: m. m gross $\Rightarrow$ mehr Speicher.

Primzahltest

Primzahlfunktion $\pi(x) := |\{n \in \mathbb{N} \mid n \leq x, n \text{ prim}\}| \sim \frac{x}{\ln x}$

↳ Beweis in 'Theoretische Informatik' nächstes Semester

Aber wenn wir einen Primzahltest hätten, könnten wir $\pi(x)$ mit Target-Shooting approximieren.

1. Euklidischer Primzahltest

Sei $m, n \in \mathbb{Z}$ beliebig.

Dann berechnet $ggT(m,n)$ den grössten gemeinsamen Teiler in $O((\log_2 nm)^3)$ nach dem Euklid'schen Algorithmus.

Es gilt:

$ggT(a,n) > 1$ für $a \in [n-1] \Rightarrow$ n nicht prim

↳ die Zahl $ggT(a,n) > 1$ ist ein Zertifikat.

Euklid-Primzahltest(n) ($\triangleq$ E-P(n))

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

↳ Monte-Carlo Algorithmus (schnell aber fehlerhaft) mit einseitigen Fehler.

Denn die Rückgabe 'keine Primzahl' erfolgt immer durch das Finden eines Zertifikats.
(Einem Teiler $b = ggT(a,n)$ von n mit $1 < b < n$)

Für n nicht prim:

$$Pr\left[\text{E-P}(n) = \text{Primzahl}\right] = \frac{|\mathbb{Z}_n^*|}{n-1}$$

Zur Erinnerung:

Von 'Diskrete Mathematik':

Multiplikative Gruppe $\mathbb{Z}_n^* = \{a \in [n-1] \mid ggT(a,n) = 1\}$

Eulersche Phi-Funktion: $\varphi(n) := |\mathbb{Z}_n^*|$

Es gilt:

$$n \text{ prim} \Leftrightarrow \mathbb{Z}_n^* = [n-1] \Leftrightarrow \varphi(n) = n-1$$

Die Fehlerw'keit kann beliebig nahe an 1 sein.

Sei p prim.

$n = p^2$ (nicht prim): $\varphi(n) = p(p-1) = n - \sqrt{n}$

$$\Rightarrow \frac{|\mathbb{Z}_n^*|}{n-1} = \frac{n-\sqrt{n}}{n-1} > \frac{n-\sqrt{n}}{n} = 1 - \frac{1}{\sqrt{n}}$$

Deshalb suchen wir etwas besseres.

Von DM:

Definition 5.7. A *group* is an algebra $\langle G; *, \widehat{\ }, e\rangle$ satisfying the following axioms:

G1 $*$ is associative.
G2 e is a neutral element: $a * e = e * a = a$ for all $a \in G$.
G3 Every $a \in G$ has an inverse element $\widehat{a}$, i.e., $a * \widehat{a} = \widehat{a} * a = e$.

$\langle \mathbb{Z}_n^*; \odot_n, ^{-1}, 1\rangle$ ist eine Gruppe.

G_1 folgt von Assoziativität der Multiplikation.

G_2 folgt mit $e=1$ per Def. von $\odot_n$

G_3 mühsam siehe DM Skript (Lemma 4.18)

Sei $a, b \in \mathbb{Z}_n^*$. $\Leftrightarrow$ $ggT(a,n)=1$ und $ggT(b,n)=1$

$\Rightarrow ggT(ab,n)=1 \Rightarrow ab \in \mathbb{Z}_n^*$

(abgeschlossen unter Multiplikation mod n)

Definition 5.11. A subset $H \subseteq G$ of a group $\langle G; *, \widehat{\ }, e\rangle$ is called a *subgroup* of G if $\langle H; *, \widehat{\ }, e\rangle$ is a group, i.e., if H is closed with respect to all operations:

(1) $a * b \in H$ for all $a, b \in H$,

(2) $e \in H$, and

(3) $\widehat{a} \in H$ for all $a \in H$.

Für eine Untergruppe $H \subseteq G$ von G gilt:

$|H|$ ist ein Teiler von $|G|$ (wobei: G endlich)

Sei $a \in \mathbb{Z}_n^*$ Dann ist $\langle a \rangle := \{a^0 = 1, a^1, a^2, \dots\}$ eine Untergruppe von $\mathbb{Z}_n^*$.

Sei $\text{ord}(a) := \min\{i \in \mathbb{N},\ a^i = 1\}$.

$\Rightarrow \text{ord}(a) = |\langle a \rangle|$

$\Rightarrow \text{ord}(a)$ teilt $|\mathbb{Z}_n^*|$

$\Rightarrow a^{|\mathbb{Z}_n^*|} = a^{\text{ord}(a)\frac{|\mathbb{Z}_n^*|}{\text{ord}(a)}} = 1^{\frac{|\mathbb{Z}_n^*|}{\text{ord}(a)}} = 1 \quad \forall a \in \mathbb{Z}_n^*$

Da $\varphi(n) = |\mathbb{Z}_n^*|$ folgt $a^{\varphi(n)} = 1 \quad \forall a \in \mathbb{Z}_n^*$

Satz (Kleiner fermatscher Satz)

Ist $n \in \mathbb{N}$ prim, so gilt für alle Zahlen $a \in [n-1]$

$$a^{n-1} \equiv 1 \pmod{n} \qquad (\text{bzw. } a^{n-1} = 1 \text{ in } \mathbb{Z}_n^*).$$

da $\mathbb{Z}_n^* = [n-1]$ und $|\mathbb{Z}_n^*| = n-1$.

2. Primzahltest nach Satz von kleinem Fermat

Fermat-Primzahltest(n)

1: Wähle $a \in [n-1]$, zufällig gleichverteilt
2: **if** $\text{ggT}(a,n) > 1$ oder $a^{n-1} \not\equiv 1 \pmod{n}$ **then**
3: **return** 'keine Primzahl'
4: **else**
5: **return** 'Primzahl'

Einseitiger Fehler.

Def. Pseudoprimzahlbasen von n

$$PB_n := \{ a \in [n-1] \mid \text{ggT}(a,n) = 1 \text{ und } a^{n-1} \equiv_n 1 \}$$
$$= \{ a \in \mathbb{Z}_n^* \mid a^{n-1} \equiv_n 1 \}$$

n heisst Carmichael-Zahl, falls n nicht prim und $PB_n = \mathbb{Z}_n^*$.

PB_n ist eine Untergruppe von $\mathbb{Z}_n^*$. (Beweis lassen wir aus)

für n nicht prim (*) $\Rightarrow$ Falls $PB_n \neq \mathbb{Z}_n^*$, ist $|PB_n|$ ein echter Teiler von $|\mathbb{Z}_n^*|$. Somit gilt $|PB_n| \leq \frac{|\mathbb{Z}_n^*|}{2} < \frac{|[n-1]|}{2} \leq \frac{n-1}{2}$

$\Rightarrow$ Fehlerwahrscheinlichkeit der Ausgabe 'Primzahl':

$$\frac{|PB_n|}{|[n-1]|} < \frac{\frac{n-1}{2}}{n-1} = \frac{1}{2}, \text{ falls } n \text{ keine Carmichael-Zahl ist.}$$

(+)

Wir können den Algorithmus k-mal wiederholen, um die Fehlerw'keit auf $\left(\frac{1}{2}\right)^k$ zu reduzieren.

Zertifikat für 'n nicht prim'

Ein Zertifikat für 'n nicht prim', ist etwas, was die Aussage 'n nicht prim' nachweisen kann.

Bis jetzt:
triviales Zertifikat, Euklid Zertifikat, Fermat-Zertifikat

3. Miller-Rabin-Zertifikat

Definition 5.18. A *ring* $\langle R; +, -, 0, \cdot, 1\rangle$ (inverse) is an algebra for which
(i) $\langle R; +, -, 0\rangle$ is a commutative group.
(ii) $\langle R; \cdot, 1\rangle$ is a monoid.
(iii) $a(b+c) = (ab) + (ac)$ and $(b+c)a = (ba) + (ca)$ for all $a, b, c \in R$ (left and right distributive laws).
A ring is called *commutative* if multiplication is commutative ($ab = ba$).[20]

Definition 5.20. An element u of a ring R is called a *unit*[22] if u is invertible, i.e., $uv = vu = 1$ for some $v \in R$. (We write $v = u^{-1}$.[23]) The set of units of R is denoted by R^*.

Definition 5.26. A *field*[34] is a nontrivial commutative ring F in which every nonzero element is a unit, i.e., $F^* = F \setminus \{0\}$.

Für n prim, ist $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$

ein Körper. (Das könnt ihr selber überprüfen, müsst ihr nicht wirklich können.)

Für die Gleichung $x^2 = 1$ gibt es in $\mathbb{Z}_n$ zwei Lösungen: $x = 1$ und $x = n-1$ $('=-1')$ $(*)$

Ist $n > 2$ prim und $a \in [n-1]$, dann gilt

$a^{n-1} \equiv_n 1$ (nach Fermat)

$a^{\frac{n-1}{2}} \in \{1, n-1\}$ $(*)$

falls $a^{\frac{n-1}{2}} = 1$ und $\frac{n-1}{2}$ gerade

folgt aus $(*)$ $a^{\frac{n-1}{4}} \in \{1, n-1\}$

usw. bis $a^{\frac{n-1}{2^i}} = n-1$ oder $\frac{n-1}{2^i}$ ungerade

Für $2 < n \in \mathbb{N}$, sei $n - 1 = 2^k d$, mit $d \in \mathbb{N}$ ungerade und $k \in \mathbb{N}_0$.
$a \in [n-1]$ is Miller-Rabin Zertifikat für "n nicht prim", falls

$$(a^d, a^{2d}, \ldots, a^{2^k d}) \neq \begin{cases} (1, 1, 1, \ldots, 1) & (\Leftrightarrow a^d \bmod n = 1) \\ (*, \ldots, *, n-1, 1, \ldots, 1) & (\Leftrightarrow \exists i : a^{2^i d} \bmod n = n-1) \end{cases}$$

Für $n < 1\,373\,653$, ist immer ein $a \in \{2, 3\}$ Miller-Rabin Zertifikat.

Miller-Rabin-Primzahltest(n) ($n > 1$, ungerade!)

1: $d, k \in \mathbb{N}$, mit $n - 1 = 2^k d$, d ungerade
2: Wähle $a \in [n-1]$, zufällig gleichverteilt
3: **if** $a^d \bmod n \neq 1$ und $\nexists i < k : a^{2^i d} \bmod n = n-1$ **then**
4: **return** 'keine Primzahl'
5: **else**
6: **return** 'Primzahl'

Einseitiger Fehler. 'keine Primzahl' immer richtig.

- Die Ausgabe 'Primzahl' ist falsch mit W'keit $\leq \frac{1}{4}$.

Lange Pfade

LONG-PATH Problem. Gegeben (G, B), G ein Graph und $B \in \mathbb{N}_0$, stelle fest ob es einen Pfad der Länge B in G gibt.

Ein **Pfad der Länge ℓ** in einem Graph $G = (V, E)$ ist eine Folge von paarweise verschiedenen Knoten

$$\langle v_0, v_1, \ldots, v_\ell \rangle, \text{ mit } \{v_{i-1}, v_i\} \in E \text{ für } i = 1, \ldots \ell.$$

Es liegen $\ell + 1$ Knoten auf einem Pfad der Länge ℓ.

Dieses Problem ist relativ schwer.

Generelle Version vom Hamiltonpfadproblem.

(Wir kennen (noch) keinen Polynomialzeit-Algorithmus)

In den meisten Fällen ist B relativ kurz ($B = O(\log n)$).

Für diesen Fall präsentieren wir eine Lösung.

Notation und Eigenschaften

- $[n] := \{1, \ldots, n\}$

- $[n]^k = \{(a_1, a_2, \ldots, a_k) \mid \forall i \in [k] : a_i \in [n]\}$ "alle Folgen der Länge k"
- $|[n]^k| = n^k$
- $\binom{[n]}{k} = \{s \subseteq [n] \mid |s| = k\}$, $\left|\binom{[n]}{k}\right| = \binom{n}{k}$
- Nachbarschaft von v in G: $N(v) = \{u \in V \mid \{u, v\} \in E\}$
- Handschlag-Lemma: Graph $G = (V, E)$: $\sum_{v \in V} \deg(v) = 2|E|$
- Für $c, n \in \mathbb{R}^+$: $c^{\log n} = n^{\log c}$ etc.
  (Auch $2^{O(\log n)} = n^{O(1)}$ ist polynomiell in n)
- Für $n \in \mathbb{N}_0$: $\sum_{i=0}^{n} \binom{n}{i} = 2^n$
  $\frac{n!}{n^n} \geq e^{-n}$ (folgt aus $e^n = \sum_{i=0}^{\infty} \frac{n^i}{n!} \geq \frac{n^n}{n!}$)

Zuerst ein anderes Problem:

Bunte Pfade

$k \in \mathbb{N}$. Graph $G = (V, E)$ mit Färbung $\gamma : V \to [k]$ (nicht notwendigerweise gültige Färbung!).

Ein Pfad heisst **bunt**, falls alle seine Knoten verschiedene Farben haben.

COLORFUL-PATH Problem. Gegeben (G, γ), $G = (V, E)$ ein Graph und $\gamma : V \to [k]$, stelle fest ob es einen bunten Pfad der Länge $k - 1$ (d.h. mit k Knoten) in G gibt.

Nun DP:

Für $v \in V$ und $i \in \mathbb{N}_0$: $DP[v][i] = P_i(v) =$

$$\left\{ S \in \binom{[k]}{i+1} \mid \exists \text{ ein mit } S \text{ gefärbter bunter Pfad der in } v \text{ endet} \right\}$$

Die Farbe von v $\gamma(v)$ ist immer in $S \in P_i(v)$ enthalten.

$\Rightarrow \forall S \in P_i(v): \gamma(v) \in S$

$P_0(v) = \{\{\gamma(v)\}\}$,

$P_1(v) = \{\{\gamma(x), \gamma(v)\} \mid x \in N(v), \gamma(x) \neq \gamma(v)\}$

$\exists$ bunter Pfad mit k Knoten $\iff \bigcup_{v \in V} P_{k-1}(v) \neq \emptyset$

DP Rekursion (Rückbezug auf Lösung kleinerer Fälle)

$$P_i(v) = \bigcup_{x \in N(v)} \{R \cup \{\gamma(v)\} \mid R \in P_{i-1}(x) \text{ und } \gamma(v) \notin R\}$$

Ein Iterationsschritt der äusseren Schleife.

(quasi die i-te Spalte von DP[v][i] berechnen)

Berechnung aller $P_i(v)$, $v \in V$, mit den $P_{i-1}(v)$, $v \in V$, gegeben:

Bunt(G, i) — G ein γ-gefärbter Graph

1: **for all** $v \in V$ **do** — $\forall v \in V$
2: $\quad P_i(v) \leftarrow \emptyset$
3: $\quad$ **for all** $x \in N(v)$ **do** — $\deg(v)$
4: $\quad\quad$ **for all** $R \in P_{i-1}(x)$ mit $\gamma(v) \notin R$ **do** — $|P_{i-1}(v)| \cdot i$
5: $\quad\quad\quad P_i(v) \leftarrow P_i(v) \cup \{R \cup \{\gamma(v)\}\}$

Analyse:

Da $P_{i-1}(v) \subseteq \binom{[n]}{i}$ folgt $|P_{i-1}(v)| \leq \binom{n}{i}$

$$\Rightarrow O\Big(\sum_{v \in V} \deg(v) \cdot \binom{k}{i} \cdot i\Big) = O\left(\binom{n}{i} \cdot i \cdot m\right)$$

Handschlag: $2m$

Regenbogen(G, γ) — G Graph, γ k-Färbung

1: **for all** $v \in V$ **do** $P_0(v) \leftarrow \{\{\gamma(v)\}\}$
2: **for** $i = 1..k-1$ **do** Bunt(G, i)
3: **return** $\bigcup_{v \in V} P_{k-1}(v) \neq \emptyset$

$$O\left(|V| + \sum_{i=1}^{k-1} \left(\binom{k}{i} \cdot i \cdot m\right) + |V|\right) = O(2^k k m)$$

$$\left(\sum_{i=1}^{k-1} \left(\binom{k}{i} \cdot i \cdot m\right) \leq \sum_{i=1}^{k-1} \binom{k}{i} \cdot k \cdot m \leq 2^k k m\right)$$

D.h. für $k \leq \log n$: Laufzeit $O(mn \log n)$,
$k = O(\log n)$: Laufzeit $O(\text{poly}(n))$.

Zurück zum Long-Path Problem.
Gibt es einen Pfad der Länge B in $G=(V,E)$.

Setze $k := B + 1$, färbe G zufällig mit k Farben, und suche einen bunten Pfad mit k Knoten.

Nehmen wir an es existiert ein Pfad P mit k Knoten.

Diese k Knoten des Pfades können auf k^k verschiedene Arten mit k Farben gefärbt werden.

Bei $k!$ dieser Färbungen, hat jeder Knoten eine andere Farbe.

$$p_{\text{Erfolg}} \geq \frac{k!}{k^k} \geq e^{-k}$$

Per Geometrischer Verteilung mit $\text{Geo}(e^{-k})$, braucht man im Erwartungswert e^k Versuche.

Angenommen G hat einen Pfad mit k Knoten.

Ein Versuch:

- Laufzeit $O(2^k km)$. $\quad p_{\text{Erfolg}} \geq e^{-k}$.

$\lceil \lambda e^k \rceil$ Versuche:

- Laufzeit $O(\lambda(2e)^k km)$.
- W'keit, dass der Algorithmus den Pfad nicht findet ist

$$\leq \left(1 - e^{-k}\right)^{\lceil \lambda e^k \rceil} \leq \left(e^{-e^{-k}}\right)^{\lceil \lambda e^k \rceil} \leq e^{-\lambda}.$$

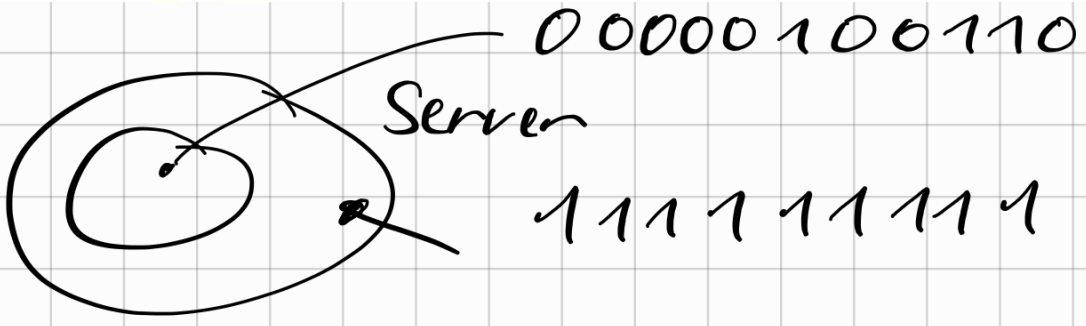
- Will man einen Pfad der Länge B tatsächlich finden (statt nur die Existenz festzustellen), kann man den Algorithmus leicht adaptieren. Man merkt sich dazu einfach zu jedem $S \in P_i(v)$ einen genau mit S gefärbten bunten Pfad $\langle u_0, u_1, \ldots, u_i \rangle$, $u_i = v$.
- Das Problem wird einfach in gerichteten azyklischen Graphen: Man gewichtet alle Kanten einfach mit -1 und berechnet kürzeste Pfade (siehe *Algorithmen und Datenstruktur*-Vorlesung im Herbst).

IV Aufgabe

Aufgabe 1 – *Broken servers*

Sie sind mit einem Netzwerk verbunden, das aus n Servern besteht, die von 1 bis n nummeriert sind. Sie können jeden Server i in Zeit $\mathcal{O}(1)$ kontaktieren und erhalten als Anwort entweder eine '0' oder eine '1'. Leider sind einige der Server kaputt uns Sie sollen herausfinden welche Server betroffen sind.

- Falls Server i kaputt ist, sendet er bei jeder Anfrage ein unabhängig gleichverteiltes Bit.
- Falls Server i intakt ist, dann antwortet er auf jede Anfrage mit dem gleichen Bit $a_i \in \{0, 1\}$. Allerdings ist der Wert a_i unbekannt und kann von Server zu Server variieren.



(a) Seien $\delta > 0$ und $i \in [n]$ gegeben. Beschreiben Sie einen Monte-Carlo Algorithmus, der herausfindet, ob Server i kaputt ist. Berechnen Sie die Fehlerwahrscheinlichkeiten (abhängig davon ob der Server kaputt/intakt ist) Ihres Algorithmus und stellen Sie sicher, dass Ihr Algorithmus Fehlerwahrscheinlichkeit höchstens $\frac{\delta}{n}$ hat.

Server Algo(i, t)

```
a_i ← Kontakt(Server i)
for j = 1, ..., t
    if (a_i ≠ Kontakt(Server i)) do
        return 'kaputt'
return 'intakt'
```

Laufzeit: $O(t)$

Nur einseitiger Fehler. Ausgabe 'kaputt' immer richtig.

'intakt' ist falsch mit W'keit: $1 - 2^{1-t}$

Fehlerw'keit höchstens 2^{1-t}

$$\frac{\delta}{n} \geq 2^{1-t} \quad \Big| \cdot \frac{n}{\delta}, 2^t$$

$$2^t \geq 2\frac{n}{\delta} \quad \Big| \log_2(\dots)$$

$$t \geq \log_2\left(2\frac{n}{\delta}\right)$$

$$t \geq 1 + \log_2\left(\frac{n}{\delta}\right)$$

(b) Falls ein Server i kaputt (bzw. intakt) ist, ist es dann sicher, dass Ihr Algorithmus aus (a) dies herausfindet? Umgekehrt, wenn Ihr Algorithmus ausgibt, dass ein Server i kaputt (bzw. intakt) ist, können Sie sich sicher sein, dass diese Ausgabe korrekt ist?

$$Pr[\text{Ausgabe "Server intakt"} \mid \text{Server intakt}] = 1$$

$$Pr[\text{Ausgabe "Server kaputt"} \mid \text{Server kaputt}] = 1 - 2^t$$

(c) Sei $\delta > 0$ gegeben. Beschreiben Sie einen Monte-Carlo Algorithmus, der eine Liste aller kaputten Server erstellt und Fehlerwahrscheinlichkeit höchstens δ hat. Hierbei sagen wir, dass der Algorithmus erfolgreich ist, wenn die Liste alle kaputten Server und keinen intakten Server enthält.

ServerList(n)

List ← ∅
for i = 1, ..., n do
  if (Server(i, $\lceil 1 + \log_2(\frac{n}{\delta}) \rceil$) = 'kaputt') then
    List ← List ∪ {i}
return List

Sei Z_i die Indikatorvariable dass die Ausgabe von Server(i, $\lceil 1 + \log_2(\frac{n}{\delta}) \rceil$) falsch ist.

Sei $Z = \sum_{i=1}^{n} Z_i$ die Anzahl falscher Einträge auf der Liste.

(a) $\Rightarrow \quad \mathbb{E}[Z_i] \leq \frac{\delta}{n} \quad \Rightarrow \mathbb{E}[Z] = \sum_{i=1}^{n} \mathbb{E}[Z_i] \leq \delta$

Z nimmt nicht-negative Werte an:

$$\Pr[Z \geq 1] \leq \frac{\mathbb{E}[Z]}{1} \leq \frac{\delta}{1} = \delta \qquad \text{(Markov)}$$

$$\Pr[\text{Liste falsch}] \leq \Pr[Z \geq 1] \leq \delta$$