Quiz

Wenn ein probabilistischer Algorithmus mit einer Wahrscheinlichkeit von mindestens $2/3$ eine korrekte JA/NEIN-Antwort liefert und wir ihn unabhängig n-mal ausführen, ist die Wahrscheinlichkeit, dass er mehr als $n/2$ Mal eine falsche Antwort gibt, kleiner als $\exp(-0.001n)$.

(viel zu schwierige Frage)

[X] Wahr
[ ] Falsch

Sei X die Anzahl falsche Antworten

$\Rightarrow X \sim Bin(n, p)$ wobei $p \leq \frac{1}{3}$

$\Rightarrow \mathbb{E}[X] = np \leq \frac{n}{3}$

$$\Pr[X > \tfrac{n}{2}] = \Pr[X > \tfrac{3}{2}\tfrac{n}{3}]$$

mit 3 erweitern

$$\leq \Pr[X > \tfrac{3}{2}\mathbb{E}[X]]$$

↳ $\mathbb{E}[X] \leq \frac{n}{3}$

$$= \Pr[X > (1+\tfrac{1}{2})\mathbb{E}[X]]$$

↳ $\frac{3}{2} = 1 + \frac{1}{2}$

$$\leq \Pr[X \geq (1+\tfrac{1}{2})\mathbb{E}[X]]$$

↳ "$X > (1+\frac{1}{2})\mathbb{E}[X]$" $\leq$ "$X \geq (1+\frac{1}{2})\mathbb{E}[X]$"

$$\leq e^{-\frac{1}{3}\cdot(\frac{1}{2})^2 \mathbb{E}[X]}$$

↳ Chernoff (i)

$$= e^{-\frac{1}{12}\cdot\frac{n}{3}}$$

↳ $\mathbb{E}[X] \leq \frac{n}{3}$

$$\leq e^{-\frac{1}{36}n}$$

$$< e^{-0.001n}$$

Es gibt einen probabilistischen Algorithmus, der testet, ob n eine Primzahl ist, und zwar in einer Zeit, die polynomiell in $\log n$ ist.

[X] Wahr
[ ] Falsch

Miller-Rabin

Sei A ein probabilistischer Algorithmus, dessen Ausgabe immer entweder 0 oder 1 ist, und gelte $\mathbb{E}[A] = s > 0$. Seien außerdem $0 < \epsilon < 1$ und $0 < \delta < \frac{1}{2}$. Wenn wir A unabhängig $m = \lceil 100 \log(1/\delta)/(s\epsilon^2) \rceil$ Mal ausführen und $\hat{s}$ als Durchschnitt der Ausgaben definieren, dann gilt mit Wahrscheinlichkeit mindestens $1 - \delta$: $\hat{s} \in [(1-\epsilon)s, (1+\epsilon)s]$.

(auch sehr schwierig.)

[X] Wahr
[ ] Falsch

Target-Shooting mit Indikatorfunktion A: $A_i = 1 \rightarrow A_i \in S$, $A_i = 0 \rightarrow A_i \notin S$

und $s = \frac{|S|}{|U|}$

und $m \geq N \geq 3 \frac{|U|}{|S|} \varepsilon^{-2} \ln(\frac{2}{\delta})$

↑ weil 100 viel grösser als 3 ist, können wir $\ln(\frac{2}{\delta})$ statt $\ln(\frac{1}{\delta})$ nehmen

Wir können jeden Las-Vegas-Algorithmus mit bekannter erwarteter Laufzeit höchstens T in einen randomisierten Algorithmus umwandeln, dessen Laufzeit höchstens $10T$ ist und der mit Wahrscheinlichkeit mindestens 0.9 erfolgreich ist.

[X] Wahr → einfach nach $10T$ Zeit abbrechen
[ ] Falsch

Sei X die Laufzeit (nicht-negativ). $\mathbb{E}[X] \leq T$

$Pr[X \geq 10T] \leq Pr[X \geq 10\,\mathbb{E}[X]]$ ← Markov

$\leq \frac{1}{10} = 0.9$

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

[ ] Wahr
[X] Falsch

anders herum

Sarah besitzt zwei spezielle Münzen: eine zeigt immer 'Kopf', die andere immer 'Zahl'. Sie wählt eine der beiden Münzen zufällig und wirft sie $n = 1000$ Mal. Sie bezeichnet mit X die Anzahl 'Kopf', die sie sieht. Dann gilt für $\delta := 1/4$: $Pr[X \geq (1+\delta)n/2] \leq e^{-\delta^2 n/6}$.

[ ] Wahr
[X] Falsch

Chernoff darf nicht angewendet werden, weil die X_i nicht unabhängig sind

Ein deterministischer Algorithmus kann immer als randomisierter Algorithmus gesehen werden.

[x] Wahr
[ ] Falsch

Pr[korrekt] = 1

Seien X, Y, Z drei unabhängige Zufallsvariablen. Dann gilt immer $\mathbb{E}[X + Y \cdot Z] = \mathbb{E}[X] + \mathbb{E}[Y] \cdot \mathbb{E}[Z]$.

[x] Wahr
[ ] Falsch

Multiplizität des Erwartungswertes gilt bei Unabhängigkeit

Die erwartete Laufzeit von Quickselect ist $O(n)$.

[x] Wahr
[ ] Falsch

(Lemma Skript)

Quicksort ist ein Monte-Carlo-Algorithmus.

[ ] Wahr
[x] Falsch

immer korrekt
Laufzeit vom Zufall abhängig
=> Las-Vegas

QuickSort mit zufälligem Pivot Element

- Las-Vegas Algorithmus mit erwarteter Laufzeit $O(n \ln n)$

Beweis: Sei $(a_1, a_2, \dots, a_n)$ das Array nach dem Sortieren

Seien a_i und a_j zwei beliebige Elemente, wobei $i < j$

Sei $I_{i,j} = \begin{cases} 1 & \text{falls } a_i \text{ mit } a_j \text{ verglichen wird} \\ 0 & \text{sonst} \end{cases}$ eine Indikatorvariable

Irgendwann wird das erste mal ein Pivot $p \in \{a_i, \dots, a_j\}$ ausgewählt.

Case $p \in \{a_i, a_j\}$: a_i und a_j werden einmal verglichen

Case $p \in \{a_{i+1}, \dots, a_{j-1}\}$: a_i und a_j werden nie verglichen, weil sie a_i im linken rekursiven Aufruf ist und a_j im rechten

$$(a_1, \dots, a_{i-1}, \boxed{a_i, a_{i+1}, \dots, a_{j-1}, a_j}, a_{j+1}, \dots, a_n)$$

$j-i+1$ viele Elemente

$$\mathbb{E}[I_{i,j}] = \Pr[a_i \text{ wird mit } a_j \text{ verglichen}] = \frac{2}{j-i+1}$$

$$\mathbb{E}[\text{totale Anzahl Vergleiche}] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathbb{E}[I_{i,j}]$$

Summe über alle möglichen Paare $1 \le i < j \le n$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

$\mathbb{E}[I_{i,j}] = \frac{2}{j-i+1}$

$$= \sum_{i=1}^{n-1} \sum_{k=2}^{n-i+1} \frac{2}{k}$$

Index Substitution $k = j-i+1$

$j = i+1 \iff j-i+1 = 2 \iff k = 2$

$j = n \iff j-i+1 = n-i+1 \iff k = n-i+1$

$$\leq \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{2}{k}$$ Summen gehen bis n

$$= 2 \cdot \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{1}{k}$$ Distributivgesetz

$$= 2n \cdot H_n$$ Harmonische Reihe $H_n \overset{\text{def}}{=} \sum_{k=1}^{n} \frac{1}{k}$

$$\leq O(n \log n)$$ weil $H_n \leq \ln(n) + O(1)$

QuickSelect

Ziel: finde das k.-kleinste Element in einem Array

QuickSelect(A, ℓ, r, k)

1: $p \leftarrow \text{Uniform}(\{\ell, \ell+1, \ldots, r\})$ ▷ wähle Pivotelement zufällig
2: $t \leftarrow \text{Partition}(A, \ell, r, p)$
3: **if** $t = \ell + k - 1$ **then** (k-1 Elemente sind links von t)
4: **return** $A[t]$ ▷ gesuchtes Element ist gefunden
5: **else if** $t > \ell + k - 1$ **then**
6: **return** QuickSelect$(A, \ell, t-1, k)$ ▷ gesuchtes Element ist links
7: **else**
8: **return** QuickSelect$(A, t+1, r, k-t)$ ▷ gesuchtes Element ist rechts

Erwartete Laufzeit: $O(n)$

(kam meines Wissens nach noch nie bei einer Prüfung)

Duplikate finden mit direktem Sortieren

Universum: U ist die Menge aller möglichen Elementen

Datensatz: $D = (s_1, s_2, \dots, s_n)$ ist eine Folge von n Elementen aus einem Universum U

Duplikat: Ein Paar (i, j) sodass $s_i = s_j$, wobei $i < j$

1.) Sortieren

2) Liste durchlaufen und dabei alle Duplikate ausgeben

Bsp $(4, 19, 1, 2, 4, 5, 2, 4)$ ⟩ Sortieren

Index: 1 2 3 4 5 6 7 8

$(1, 2, 2, 4, 4, 4, 5, 19)$

Duplikate: $(2,3), (4,5), (4,6), (5,6)$

Laufzeit: $O(n \log n + |\text{Duplikate}|)$

$n \log n$: Sortieren; $|\text{Duplikate}|$: Duplikate ausgeben

Wie viele Duplikate gibt es höchstens bei n Elementen? $\rightarrow \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$

$= \binom{n}{2}$

Duplikate finden mit Hashfunktion

Motivation: Vergleiche von "schwierigen" Elementen (z.B. Graphen, DNA-Strings) können teuer sein

Hashfunktion: $h: U \to \{1, \dots, m\}$ berechnet für ein Element $u \in U$ den Hashwert $h(u)$

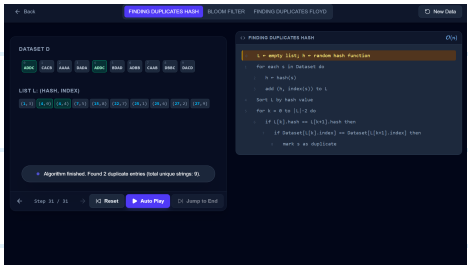
- m soll viel kleiner als $|U|$ sein
- h soll effizient berechenbar sein
- h soll zufällig gleichverteilt sein. $\forall u \in U \; \forall i \in [m]: \Pr[h(u) = i] = \frac{1}{m}$
- h hat die Eigenschaft: $s_i = s_j \Rightarrow h(s_i) = h(s_j)$

Kollisionen: sind Paare (i, j) mit $s_i \neq s_j$ aber $h(s_i) = h(s_j)$

1.) Hashen und Index merken

2) Sortieren nach Hashwert aufsteigend

3) Liste durchlaufen und dabei alle Hashwert-Duplikate überprüfen und ausgeben, wenn es keine Kollisionen sind



Laufzeit: $O(n \log n + |\text{Duplikate}| + |\text{Kollisionen}|)$

- $n \log n$: Hashen und Sortieren
- $|\text{Duplikate}|$: Duplikate ausgeben
- $|\text{Kollisionen}|$: Kollisionen überprüfen

Für $m = n^2$: ist die erwartete Anzahl Kollisionen konstant

Beweis: Sei $K_{i,j}$ die Indikatorvariable für das Ereignis "(i,j) ist eine Kollision"

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

(weil $s_i = s_j$ ein Duplikat ist, und daher keine Kollision sein kann)

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

Linearität des Erwartungswertes

$$\mathbb{E}[\text{Anzahl Kollisionen}] = \sum_{1 \le i < j \le n} \mathbb{E}[K_{i,j}]$$

↳ $\mathbb{E}[K_{i,j}] \le \frac{1}{m}$

$$\le \sum_{1 \le i < j \le n} \frac{1}{m}$$

↳ $\binom{n}{2}$ = Anzahl Paare (i,j) mit $i < j$

$$= \binom{n}{2} \cdot \frac{1}{m}$$

↳ def. Binomialkoeffizient $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

$$= \frac{n(n-1)}{2} \cdot \frac{1}{m}$$

↳ ausmultiplizieren

$$= \frac{n^2 - n}{2} \cdot \frac{1}{m}$$

↳ Annahme: $m = n^2$

$$= \frac{n^2 - n}{2} \cdot \frac{1}{n^2}$$

↳ dividieren

$$= \frac{1 - \frac{1}{n}}{2}$$

$$= \frac{1}{2} - \frac{1}{2n}$$

$< \frac{1}{2} \Rightarrow$ Konstant

Duplikate finden mit Hase-Igel-Algorithmus (Floyd)

Gegeben: ein Array $a[1..n]$ wobei alle Elemente zwischen 1 und $n-1$ liegen

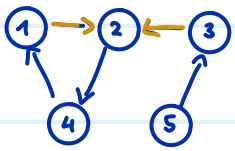
Gesucht: ein Duplikat, also zwei verschiedene Indices i, j sodass $a[i] = a[j]$

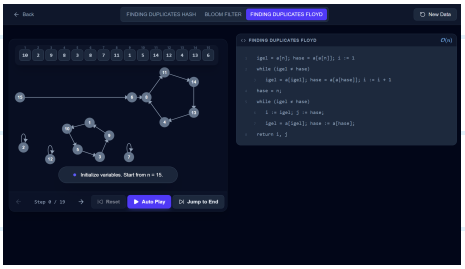
(existiert immer wegen Pigeonhole-Prinzip)

Umformulierung als Graph: Knoten $V = \{1, .., n\}$ und Kanten $\{(i, a[i]) \mid 1 \leq i \leq n\}$

neues Ziel: finde einen Knoten mit zwei eingehenden Kanten

Bsp. $A = (2, 4, 2, 1, 3)$



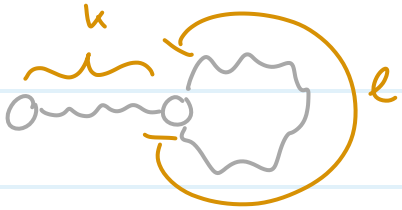


Subgraph: Wir betrachten nur noch alle von n aus erreichbaren Knoten.

Jeder Knoten hat genau eine ausgehende Kante (nämlich von i zu $a[i]$)

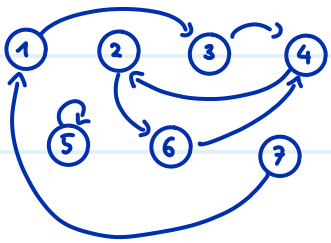
Der Subgraph muss ein Pfad der Länge $k \geq 1$ sein,

gefolgt von einem Kreis der Länge $\ell \geq 1$



Knotenfolge: $x_0 = n$, und rekursiv $x_{t+1} = a[x_t]$ für alle $t \geq 0$

Bsp $A = (3, 6, 4, 2, 5, 4, 1)$



$x_0 = 7, x_1 = 1, x_2 = 3, x_3 = 4, x_4 = 2$

$x_5 = 6, x_6 = 4, x_7 = 2, x_8 = 6, ..$

1 Treffen: Es gibt ein $t \in \{1, \dots, n\}$ sodass $x_t = x_{2t}$

(also wenn Hase und Igel mit Geschwindigkeit 2 (Hase) und 1 (Igel) bei Knoten n loslaufen, treffen sie sich spätestens nach n Schritten)

```
igel = a[n]; hase := a[a[n]]; i := 1
while (igel ≠ hase)
    igel = a[igel]; hase := a[a[hase]]; i := i + 1
```

(a[igel]: 1 Schritt; a[a[hase]]: 2 Schritte)

Beweis: Sei r der "Rest" den wir auffüllen müssen, sodass k+r das kleinstmögliche Vielfache der Kreislänge ℓ ist: $k + r = \lceil \frac{k}{\ell} \rceil \cdot \ell \iff r = \lceil \frac{k}{\ell} \rceil \cdot \ell - k$

($\lceil \frac{k}{\ell} \rceil$: Anzahl angefangene male, wie oft ℓ in k hinein passt)

$\Rightarrow$ Dann liegt x_{k+r} auf dem Kreis (weil der Kreis bei k beginnt und $k+r \geq k$)

$\Rightarrow$ $x_{k+r} = x_{2(k+r)}$ weil k+r ein Vielfaches der Kreislänge ist. Also $x_{2(k+r)}$ macht einfach einige Kreisumrundungen mehr)

$\Rightarrow$ wir wählen also $t = k+r$

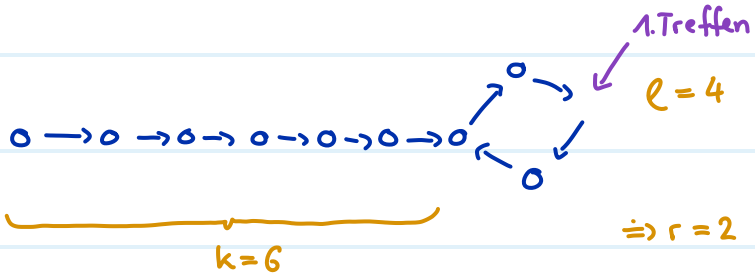
Und es gilt: $t = k + r$

$= \lceil \frac{k}{\ell} \rceil \ell$ (per Definition)

$< \frac{k}{\ell} \ell + \ell$ (aufrunden)

$= k + \ell$

$\leq n$ (Pfadlänge + Kreislänge $\leq n$ weil es nur n Kanten gibt)



2 Treffen: es gilt: $x_k = x_{k+t}$

Hase Igel

(der Hase startet noch einmal bei Knoten n und hat nun Geschwindigkeit 1 Er wird den Igel genau nach k Schritten treffen, also beim Anfang des Kreises)

```
hase = n;    Hase wird zurück an den Starthnoten n gestellt
while (igel ≠ hase)
    i := igel; j := hase;
    igel = a[igel]; hase := a[hase];
return i, j
```

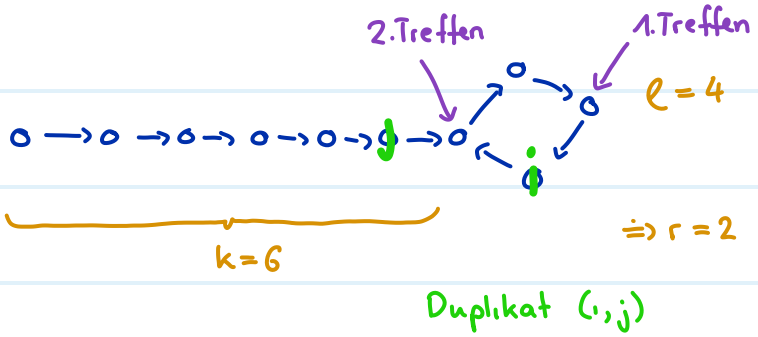
1 Schritt 1 Schritt

Beweis: $x_{k+t} = x_k$ weil x_k auf dem Kreis liegt (x_k ist per Definition der erste Knoten des Kreises)

und t ein Vielfaches der Kreislänge ℓ ist (nach jeder Kreisumdrehung sind wir wieder bei x_k)

Laufzeit: $O(n)$

Zusätzlicher Speicher: 4 Variablen



Wahrscheinlichkeits - DP

Du befindest dich in einem Casino an einem seltsamen Spieltisch. Du darfst eine gezinkte Münze werfen. Diese Münze fällt zu **70 % auf Kopf** und zu **30 % auf Zahl**.
Der Croupier macht dir folgendes Angebot: Du darfst die Münze genau N-mal werfen. Du gewinnst den Hauptpreis, sobald du **dreimal direkt hintereinander "Kopf"** wirfst (ein sogenannter 3er-Streak). Sobald du das schaffst, ist das Spiel sofort gewonnen, egal wie viele Würfe du noch übrig hättest.

Dimension: $DP[0,\dots,N][0,\ ,3]$

Teilproblem: $DP[i,s]$ = Wahrscheinlichkeit zu gewinnen mit i übrigen Würfen und einem aktuellen Streak s (angenommen wir haben zuvor noch nicht gewonnen)

Base Case: $DP[0,s] = 0$ für alle $s \in \{0,1,2\}$

$DP[i,3] = 1$ für alle $0 \le i \le N$

Satz der totalen Wahrscheinlichkeit

falls Kopf: dann Streak +1 und übrige Versuche −1

falls Zahl: dann Streak verloren (=0) und übrige Versuche −1

Rekursion: $DP[i,s] = Pr[\text{Kopf}] \cdot DP[i-1, s+1] + Pr[\text{Zahl}]\ DP[i-1, 0] \quad \forall 1 \le i \le N\ \forall 0 \le s \le 2$

$= 0.7\ DP[i-1, s+1] + 0.3\ DP[i-1, 0]$

Reihenfolge: IMMER TOP DOWN!

Lösung: $DP[N,0]$

Laufzeit: $O(n)$