Algorithmen und Wahrscheinlichkeit

Woche 8

Ilya Maier

Minitest

Minitest

Password: capital

Nachbesprechung Serie & CodeExpert

- $\Pr[e \in E(S, V \backslash S)] = \frac{1}{2}$ muss man begründen
  - nur weil die Elementarereignisse die gleiche Wahrscheinlichkeit haben, haben nicht alle Ergebnisse die gleiche Wahrscheinlichkeit
- $\Longrightarrow$ für statements, $=$ für expressions
- Code Expert
  - Was ist X^2 und wieso ist $\mathbb{E}[X^2] \neq \mathbb{E}[X]^2$
  - Wie berechnet man $\mathbb{E}[X^2]$

Randomisierte Algorithmen

Randomisierter Algorithmus : Eingabe $I \rightarrow$ Algorithmus A mit Zufallszahlen $R \rightarrow$ Ausgabe $A(I, R)$

- deterministisch: selbe Eingabe, selber Output
- nicht-deterministisch: selbe Eingabe, nicht unbedingt selber Output

Monte Carlo Algorithmus: Primzahltest, Target-shooting
→ Korrektheit ist die Zufallsvariable

- immer gleiche Laufzeit
- manchmal falsches Ergebnis

Las Vegas Algorithmus: Quicksort, Duplikate finden
→ Laufzeit ist die Zufallsvariable
→ Geometrisch verteilt mit $p = \Pr[A(I) \neq \text{"???"}]$

- immer korrekte Antwort
- manchmal dauert zu lange /
  gibt nach einer bestimmter Zeit “???” aus

Quicksort/Quickselect

Quicksort: sortiert den Array
erwartete Laufzeit: $\mathcal{O}(n \log n)$

Quickselect: gibt das k-kleinste Element aus
erwartete Laufzeit: $\mathcal{O}(n)$

```
QuickSort(A, ℓ, r)
1: if ℓ < r then
2:     p ← Uniform({ℓ, ℓ + 1, ..., r})        ▷ wähle Pivotelement zufällig
3:     t ← Partition(A, ℓ, r, p)
4:     QuickSort(A, ℓ, t − 1)
5:     QuickSort(A, t + 1, r)
```

```
QuickSelect(A, ℓ, r, k)
1: p ← Uniform({ℓ, ℓ + 1, ..., r})            ▷ wähle Pivotelement zufällig
2: t ← Partition(A, ℓ, r, p)
3: if t = ℓ + k − 1 then
4:     return A[t]                            ▷ gesuchtes Element ist gefunden
5: else if t > ℓ + k − 1 then
6:     return QuickSelect(A, ℓ, t − 1, k)     ▷ gesuchtes Element ist links
7: else
8:     return QuickSelect(A, t + 1, r, k − t) ▷ gesuchtes Element ist rechts
```

Fehlerreduktion

Las-Vegas:

Sei A ein Las-Vegas-Algorithmus mit $\Pr[A(I) \neq \text{"???"}] \geq \epsilon$

Sei A_δ für $\delta > 0$ ein Algorithmus, der entweder die erste Ausgabe verschieden von $???$ ausgibt

oder der nach $N = \lceil \epsilon^{-1} \cdot \ln(\delta^{-1}) \rceil$ Versuchen $???$ ausgibt

dann gilt $\Pr[A_\delta(I) = \text{"???"}] \leq \delta$

ϵ	δ	N
0.1	0.01	47
0.5	0.01	10
0.5	10^{-80}	369
0.9	10^{-30}	77

Fehlerreduktion

Monte-Carlo - Einseitiger Fehler:

$\Pr[A(I) = \text{Ja}] = 1$ für alle Ja-Instanzen I
$\Pr[A(I) = \text{Nein}] \geq \epsilon$ für alle Nein-Instanzen I

$\Longrightarrow$ Wenn $A(I) = \text{Ja}$, dann könnte die Ausgabe falsch sein
Wenn $A(I) = \text{Nein}$, dann ist die Ausgabe immer korrekt

Sei A_δ für $\delta > 0$ ein Algorithmus, der entweder Nein ausgibt, sobald das erste Mal Nein vorkommt,

oder der nach $N = \lceil \epsilon^{-1} \cdot \ln(\delta^{-1}) \rceil$ Versuchen Ja ausgibt

dann gilt:

$\Pr[A_\delta(I) = \text{Ja}] = 1$ für alle Ja-Instanzen I

$\Pr[A_\delta(I) = \text{Nein}] \geq 1 - \delta$ für alle Nein-Instanzen I

Monte-Carlo - Zweiseitiger Fehler:

$\Pr[A(I)$ ist korrekt$] \geq 0.5 + \epsilon$ für alle Instanzen I

A_δ gibt die meiste Antwort aus nach N Wiederholungen

$\Longrightarrow$

Für $\delta > 0$, wenn $N = \lceil 4 \cdot \epsilon^{-2} \cdot \ln(\delta^{-1}) \rceil$

$\Pr[A_\delta(I)$ ist falsch$] \leq \delta$ für alle Instanzen I

Aufgaben

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 Antwort 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.

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 Antwort 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.

(b) 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.