

Probabilistische Algorithmen

Oft: - garantiert korrekte Lösung nicht erforderlich
- optimale Lösung nicht erforderlich (Näherungslösung)

Beispielproblem:

gegeben: n Zahlen x_1, x_2, \dots, x_n
gesucht: irgendeine Zahl z aus der oberen Hälfte

Mögliche Lösung: bestimme Maximum (mit $n-1$ Vergleichen)

Andere Lösung: bestimme Maximum m von $n/2+1$ Zahlen (mit $n/2$ Vergleichen)
=> garantiert, daß m in oberer Hälfte

Folie 7.1

Probabilistische Algorithmen

gegeben: n Zahlen x_1, x_2, \dots, x_n
gesucht: irgendeine Zahl z aus der oberen Hälfte

wenn korrekte Lösung nicht garantiert sein muß:

$p(\text{irgendeine Zahl } x_i \text{ ist in unterer Hälfte}) = 1/2$
 $p(\text{irgendwelche } x_i \text{ und } x_j \text{ in unterer Hälfte}) = 1/4$
 $p(\text{max}(x_i, x_j) \text{ ist in unterer Hälfte}) = 1/4$
 $p(\text{max}(x_{i_1}, x_{i_2}, \dots, x_{i_k}) \text{ in unterer Hälfte}) = 2^{-k}$
 $p(\text{max}(x_{i_1}, x_{i_2}, \dots, x_{i_k}) \text{ in oberer Hälfte}) = 1 - 2^{-k}$

also steigt die Wahrscheinlichkeit für große k asymptotisch gegen 1, z.B. $k=20 \Rightarrow p > 0.999999$.
Aufwand ist konstant $O(k)$ unabhängig von n .

Folie 7.2

Probabilistische Algorithmen

Einteilung probabilistische Algorithmen:

Monte Carlo

- gibt mit kleiner Wahrscheinlichkeit unkorrektes Ergebnis aus
- hat kürzere Laufzeit als der beste deterministische Algorithmus

Las Vegas

- gibt immer korrektes Ergebnis aus
- Laufzeit ist nicht garantiert kürzer als deterministisch
- durchschnittliche Laufzeit ist kürzer

Folie 7.3

Probabilistische Algorithmen

Für probabilistische Algorithmen werden Zufallszahlen benötigt.

Wenige Computer haben „echte“ physikalische Zufallsgeneratoren.

Jeder Programmablauf ist deterministisch
=> keine echten Zufallszahlen

Daher in der Praxis: **Pseudozufallszahlen**

- deterministisch bestimmt
- Zusammenhang zwischen Zufallszahlen nicht erkennbar
- erfüllen bestimmte Verteilungen (gleich/normal)

Folie 7.4

Probabilistische Algorithmen

Ein **Pseudozufallszahlengenerator**

Die erste Zufallszahl $r(1)$ wird **seed** genannt (engl. *seed* = Saat).
- wird echt zufällig gewählt (z.B. aktuelle Zeit)
- wird gleich gewählt (=> Programmablauf reproduzierbar)

$r(i)$ berechnet sich aus $r(i-1)$ gemäß

$$r(i) = (r(i-1) \cdot b + 1) \pmod{t} \quad (b, t \text{ Konstanten})$$

- b, t sollten sorgfältig gewählt werden
- $r(i)$ ist immer zwischen 0 und $t-1$

Folie 7.5

Probabilistische Algorithmen

Ein Beispiel:

gegeben: Menge S mit n Elementen
Untermengen s_1, s_2, \dots, s_k mit je r Elementen ($k \leq 2^{r-2}$)

gesucht: Eine von zwei Farben für jedes Element so daß jedes s_1, s_2, \dots, s_k von jeder Farbe mindestens 1 Elemente hat.

Probabilistischer Ansatz:

Ordne jedem Element irgend eine Farbe zu (zufällig).

Offensichtlich: Nicht immer korrekte Lösung.
Frage: Wahrscheinlichkeit für falsche Lösung?

Folie 7.6

Probabilistische Algorithmen

$$p(\text{alle Elemente von } s_i \text{ sind rot}) = 2^{-r}$$

$$p(\text{irgend ein } s_i \text{ ist nur rot}) \leq k \cdot 2^{-r}$$

Wenn nun (wie vorgegeben) $k \leq 2^{r-2}$,
dann ist

$$p(\text{irgend ein } s_i \text{ ist nur rot}) \leq k \cdot 2^{-r} \leq 2^{r-2} \cdot 2^{-r} = 1/4$$

$$p(\text{irgend ein } s_i \text{ ist nur rot oder nur blau}) \leq 2 \cdot 1/4 = 1/2 \quad (<1)$$

Da diese Wahrscheinlichkeit echt kleiner 1 ist existiert immer ein
korrekte Lösung.

Folie 7.7

Probabilistische Algorithmen

bisher: Monte Carlo Algorithmus findet mit Wahrscheinlichkeit $> 1/2$
eine korrekte Lösung.

Erweiterung zu Las Vegas Algorithmus (garantiert korrekte Lösung):

wiederhole zufällige Färbung prüfe ob Färbung korrekt ($O(n)$) bis Färbung korrekt

Erwartungswert für Anzahl Schleifendurchläufe ist 2.
(Es kann aber auch jede andere Laufzeit eintreten.)

Folie 7.8