Randomisierte Algorithmen

2. Erste Beispiele

Thomas Worsch

Fakultät für Informatik
Karlsruher Institut für Technologie

Wintersemester 2016/2017

Überblick

Ein randomisierter Identitätstest

Vergleich von Wörtern

Ein randomisierter Quicksortalgorithmus

2.1 Aufgabe

- **Gegeben:** drei $n \times n$ Matrizen $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{F}^{n \times n}$.
- **Gesucht:** Antwort auf die Frage, ob $\mathbf{AB} = \mathbf{C}$ ist.

- $\mathbb{F}$: Körper
- Bits: neutrale Elemente 0 bzw. 1

2.2 Deterministische Lösungen

- naiv: $\Theta(n^3)$ Schritte
- schneller:
  - Strassen (1969): $\Theta(n^{2.808\cdots})$ Schritte
  - Coppersmith/Winograd (1987): $\Theta(n^{2.376\cdots})$ Schritte

2.2 Deterministische Lösungen

- naiv: $\Theta(n^3)$ Schritte
- schneller:
  - Strassen (1969): $\Theta(n^{2.808\cdots})$ Schritte
  - Coppersmith/Winograd (1987): $\Theta(n^{2.376\cdots})$ Schritte
  - Vassilevska Williams (2011): $\Theta(n^{2.373\cdots})$ Schritte

2.3 Randomisierter Algorithmus (Freivalds, 1977)

2.3 Randomisierter Algorithmus (Freivalds, 1977)

$\mathbf{r} \leftarrow \langle \textit{Vektor von } n \textit{ unabhängigen Zufallsbits} \rangle$

2.3 Randomisierter Algorithmus (Freivalds, 1977)

$\mathbf{r} \leftarrow \langle \textit{Vektor von } n \textit{ unabhängigen Zufallsbits} \rangle$
$\langle \textit{Berechne und vergleiche } \mathbf{ABr} \textit{ und } \mathbf{Cr} \rangle$

2.3 Randomisierter Algorithmus (Freivalds, 1977)

$\mathbf{r} \leftarrow \langle \textit{Vektor von } n \textit{ unabhängigen Zufallsbits} \rangle$
$\langle \textit{Berechne und vergleiche } \mathbf{ABr} \textit{ und } \mathbf{Cr} \rangle$
$\mathbf{x} \leftarrow \mathbf{Br}$
$\mathbf{y} \leftarrow \mathbf{Ax}$
$\mathbf{z} \leftarrow \mathbf{Cr}$
if $(\mathbf{y} \neq \mathbf{z})$ **then**
    return NO
else
    return YES
fi

Offensichtlicher Zeitbedarf $\Theta(n^2)$.

2.4 Fingerabdrücke

- $\mathbf{y} = \mathbf{A}(\mathbf{Br})$ „Fingerabdruck“ von $\mathbf{AB}$.
- $\mathbf{z} = \mathbf{Cr}$ „Fingerabdruck“ von $\mathbf{C}$

2.5 Lemma

Ist $\mathbf{AB} \neq \mathbf{C}$ und $\mathbf{r}$ ein Vektor unabhängiger Zufallsbits, dann ist die Fehlerwahrscheinlichkeit des Algorithmus

$$\mathbf{Pr}\left[\mathbf{ABr} = \mathbf{Cr}\right] \leq 1/2 \ .$$

Beachte: Im Fall $\mathbf{AB} = \mathbf{C}$ kann kein Fehler passieren.

2.6 Beweis

- Sei $\mathbf{D} = \mathbf{AB} - \mathbf{C}$, also $\mathbf{D} \neq \mathbf{0}$.
  Sei $\mathbf{y} = \mathbf{ABr}$ und $\mathbf{z} = \mathbf{Cr}$.
- $\mathbf{Dr} = \mathbf{0}$ genau dann, wenn $\mathbf{y} = \mathbf{z}$ (Fehler)
- $\mathbf{d}$ sei eine Zeile von $\mathbf{D}$, die nicht der Nullvektor ist.
- $\mathbf{dr}$ ist Eintrag im Produktvektor $\mathbf{Dr}$
- obere Schranke für die Wahrscheinlichkeit, dass $\mathbf{dr} = \mathbf{0}$?

2.6 Beweis (Fortsetzung)

- Sei $\mathbf{d} = (d_1, \ldots, d_n)$ und o. B. d. A. seien Nichtnulleinträge von $\mathbf{d}$ gerade $d_1, \ldots, d_k$.
- $\mathbf{dr} = \mathbf{0}$ genau dann, wenn $\sum_{i=1}^{k} d_i r_i = 0$, also

$$r_k = (-\sum_{i=1}^{k-1} d_i r_i)/d_k$$

- Man stelle sich vor, $r_1, \ldots, r_{k-1}$ seien bereits gewählt.
- Gleichung offensichtlich für höchstens *einen* der beiden möglichen Werte für r_k richtig.
- Also ist Wahrscheinlichkeit für $\mathbf{dr} = \mathbf{0}$ höchstens $1/2$.
- Also ist Wahrscheinlichkeit für $\mathbf{Dr} = \mathbf{0}$ höchstens $1/2$.

□

2.7 Korollar

- Wenn $\mathbf{AB} = \mathbf{C}$ ist, liefert Algorithmus 2.3 stets die richtige Antwort.
- Wenn $\mathbf{AB} \neq \mathbf{C}$ ist, liefert Algorithmus 2.3 mit einer Wahrscheinlichkeit größer gleich $1/2$ die richtige Antwort.

2.8 Kleinere Fehlerwahrscheinlichkeit: wie?

2.8 Kleinere Fehlerwahrscheinlichkeit: wie?

- Ziehe – wenn möglich – die Komponenten des Zufallsvektors aus einer größeren Menge.
- Führe k unabhängige Wiederholungen des Algorithmus durch und produziere nur dann am Ende die Antwort YES, wenn jeder Einzelversuch diese Antwort geliefert hat.

$$\begin{aligned}&\mathbf{Pr}\left[Y_1 = \text{YES} \wedge Y_2 = \text{YES} \wedge \cdots \wedge Y_k = \text{YES}\right]\\ &= \prod_{i=1}^{k} \mathbf{Pr}\left[Y_i = \text{YES}\right]\end{aligned}$$

Überblick

2.9 Aufgabe

- **Gegeben:** Zwei „Datenbestände“ in Form von Bitfolgen $a_1 \cdots a_n$ und $b_1 \cdots b_n$.
- **Gesucht:** Antwort auf die Frage, ob die beiden Bitfolgen gleich sind.

- **Anwendung**:
  Vergleich räumlich weit entfernter großer Datenmengen
- Sind $a = \sum_{i=1}^{n} a_i 2^{i-1}$ und $b = \sum_{i=1}^{n} b_i 2^{i-1}$ gleich?
- Wieviele Bits muss man wohl übertragen, um
  mit W.keit $1 - 1/n$ sicher sein zu können, dass $a = b$ ist?

2.11 Algorithmus

⟨*Überprüfung, ob Bitfolgen* $a_1 \cdots a_n$ *und* $b_1 \cdots b_n$ *gleich sind*⟩

$p \leftarrow$ ⟨*Primzahl kleiner oder gleich* $n^2 \ln n^2$⟩
⟨*zufällig gleichverteilt aus diesen ausgewählt*⟩
$a \leftarrow \sum_{i=1}^{n} a_i 2^{i-1}$
$b \leftarrow \sum_{i=1}^{n} b_i 2^{i-1}$
if $(a \bmod p = b \bmod p)$ **then**
    return YES
else
    return NO
fi

2.12 Satz

Bei zufälliger gleichverteilter Wahl einer Primzahl p kleiner oder gleich $n^2 \log n^2$ ist

$$\mathbf{Pr}\left[a \bmod p = b \bmod p \mid a \neq b\right] \in O(\frac{1}{n}) \ .$$

Für den Beweis werden zwei Ergebnisse benötigt …

2.13 Satz (Chebyshev)

Für die Anzahl $\pi(n)$ der Primzahlen kleiner oder gleich n gilt:

$$\frac{7}{8}\frac{n}{\ln n} \leq \pi(n) \leq \frac{9}{8}\frac{n}{\ln n}$$

Es ist also $\pi(n) \in \Theta(n/\ln n)$.

2.14 Lemma

Die Anzahl k verschiedener Primteiler einer Zahl kleiner oder gleich 2^n ist höchstens n.

2.15 Beweis (von Satz 2.12)

- Sei $c = |a - b|$.
- falsche Antwort, wenn $c \neq 0$ und von p geteilt
- da $c \leq 2^n$, hat es höchstens n verschiedene Primteiler
- sei die gewählte Primzahl aus dem Intervall von 2 bis t
- dort gibt es $\pi(t) \in \Theta(t/\ln t)$ Primzahlen
- Die Wahrscheinlichkeit $\mathbf{Pr}\left[a \bmod p = b \bmod p \mid a \neq b\right]$, ein p zu wählen, das zu einer falschen Antwort führt, ist also höchstens $O(\frac{n}{t/\ln t})$.
- Für $t = n^2 \ln n^2$ ergibt sich eine obere Schranke in $O(1/n)$.

□

2.16 Pattern Matching

- **Gegeben:** *Text* $x = x_1 \cdots x_n$ und kürzeres *Suchmuster* $y = y_1 \cdots y_m$.
- **Gesucht:** Antwort auf die Frage, ob y in x vorkommt.
- Bezeichne $x(j)$ das Teilwort $x_j \cdots x_{j+m-1}$ (Länge m).
  Frage: Ist für ein $1 \leq j \leq n - m + 1$ das zugehörige $x(j) = y$?

- bezeichne $\hat{x}(j) = \sum_{i=1}^{m} x_{j+1+i-1} 2^{i-1}$

2.17 Idee

$$
\begin{aligned}
\hat{x}(j+1) &= \sum_{i=1}^{m} x_{j+1+i-1} 2^{i-1} \\
&= \frac{1}{2}(x_j - x_j) + \frac{1}{2} \sum_{i=1}^{m-1} x_{j+1+i-1} 2^{i} + x_{j+1+m-1} 2^{m-1} \\
&= \frac{1}{2} \left(\sum_{i=0}^{m-1} x_{j+i} 2^{i} - x_j \right) + x_{j+m} 2^{m-1} \\
&= \frac{1}{2} \left(\sum_{i=1}^{m} x_{j+i-1} 2^{i-1} - x_j \right) + x_{j+m} 2^{m-1} \\
&= \frac{1}{2} \left(\hat{x}(j) - x_j \right) + x_{j+m} 2^{m-1} \ .
\end{aligned}
$$

2.18 Algorithmus

⟨*Überprüfung, ob Bitfolge* $y_1 \cdots y_m$ *in* $x_1 \cdots x_n$ *vorkommt*⟩
⟨*Ausgabe: erstes* j*, wo das der Fall ist, oder* -1 *sonst.*⟩

$p \leftarrow$ ⟨*Primzahl kleiner oder gleich* $n^2 m \ln n^2 m$⟩
⟨*zufällig gleichverteilt aus diesen ausgewählt.*⟩
$y \leftarrow \sum_{i=1}^{m} y_i 2^{i-1} \bmod p$
$z \leftarrow \sum_{i=1}^{m} x_i 2^{i-1} \bmod p$
for $j \leftarrow 1$ **to** $n - m$ **do**
  if $(y = z)$ **then**
    return j ⟨*erste Stelle, an der „Übereinstimmung“*⟩
  fi
  $z \leftarrow (z - x_j)/2 + x_{j+m} 2^{m-1} \bmod p$
od
return -1 ⟨y *kommt sicher nicht in* x *vor*⟩

2.19 Satz

Algorithmus 2.18 liefert höchstens mit Wahrscheinlichkeit $O(1/n)$ eine falsche Antwort.

2.20 Beweis

- Sei wieder t die obere Schranke des Intervalls, aus dem Primzahlen gewählt werden.
- Nach Satz 2.13 gibt es dort $\pi(t) \in \Theta(t/\ln t)$ Primzahlen.
- Wahrscheinlichkeit $\mathbf{Pr}\left[y \bmod p = x(j) \bmod p \mid y \neq x(j)\right]$, ist höchstens $O(\frac{m}{t/\ln t})$,
- da $|y - x(j)|$ höchstens m verschiedene Primteiler besitzt.
- Die Wahrscheinlichkeit für falsche Antwort, weil an irgendeiner der $O(n)$ Stellen der Test versagt, ist höchstens $O(\frac{nm}{t/\ln t})$.
- Wählt man $t = n^2 m \ln n^2 m$, ergibt sich eine obere Schranke in $O(1/n)$.

□

Überblick

Ein randomisierter Identitätstest

Vergleich von Wörtern

Ein randomisierter Quicksortalgorithmus

2.21 Annahme

Alle Eingaben seien paarweise verschieden.

2.22 Algorithmus

proc $R[1 \ldots n] \leftarrow \text{RandQuickSort}(S[1:n])$
$\langle$*Eingabe: ein Feld* $S[1:n]$ *paarweise verschiedener Zahlen*$\rangle$
$\langle$*Ausgabe: ein Feld* $R[1:n]$ *die Zahlen aus* S *sortiert*$\rangle$
$\langle$*Zwischenablage in Feldern* S_1 *und* $S_2\rangle$
$i \leftarrow$ **random**$(1,n)$ $\langle$*gleichverteilt Zahl aus* $[1 \ldots n]\rangle$
$y \leftarrow S[i]$
$j_1 \leftarrow 1; j_2 \leftarrow 1;$
for $i \leftarrow 1$ **to** n **do**
    if $S[i] < y$ **then** $S_1[j_1] \leftarrow S[i]; j_1 \leftarrow j_1 + 1$ **fi**
    if $S[i] > y$ **then** $S_2[j_2] \leftarrow S[i]; j_2 \leftarrow j_2 + 1$ **fi**
od
return
$\text{RandQuickSort}(S_1[1:j_1-1]) \cdot y \cdot \text{RandQuickSort}(S_2[1:j_2-1])$

2.23 Eigenschaften

- Algorithmus 2.22 liefert *immer die korrekte Ausgabe*
- Wahl der y beeinflusst Laufzeit (bei gleicher Eingabe):
  - wenn z. B. y immer das Minimum: Laufzeit $\Theta(n^2)$.
  - wenn z. B. y immer der Median: Laufzeit $\Theta(n \log n)$.
- Die *Laufzeit* ist hier also eine *Zufallsvariable*.
- Was ist der *Erwartungswert der Laufzeit*?

2.24 Notation

- X_{ij}: Zufallsvariable
  $X_{ij} = 1$, falls „zwei Zahlen“ miteinander verglichen werden
  $X_{ij} = 0$, falls nicht

- Gesuchter Erwartungswert ist gleich

$$\mathbf{E}\left[\sum_{i=1}^{n-1}\sum_{j>i} X_{ij}\right] = \sum_{i=1}^{n-1}\sum_{j>i} \mathbf{E}\left[X_{ij}\right]$$

2.24 Notation

- X_{ij}: Zufallsvariable
  $X_{ij} = 1$, falls $R[i]$ und $R[j]$ miteinander verglichen werden
  $X_{ij} = 0$, falls nicht
- Beachte: Indizes in der *Resultatliste* (sortierte Reihenfolge)
- Gesuchter Erwartungswert ist gleich

$$\mathbf{E}\left[\sum_{i=1}^{n-1}\sum_{j>i} X_{ij}\right] = \sum_{i=1}^{n-1}\sum_{j>i} \mathbf{E}\left[X_{ij}\right]$$

- p_{ij}: Wahrscheinlichkeit für Vergleich von $R[i]$ und $R[j]$,
  also $\mathbf{E}\left[X_{ij}\right] = 1 \cdot p_{ij} + 0 \cdot (1 - p_{ij}) = p_{ij}$.

2.24 Notation

- X_{ij}: Zufallsvariable
  $X_{ij} = 1$, falls $R[i]$ und $R[j]$ miteinander verglichen werden
  $X_{ij} = 0$, falls nicht
- Beachte: Indizes in der *Resultatliste* (sortierte Reihenfolge)
- Gesuchter Erwartungswert ist gleich

$$\mathbf{E}\left[\sum_{i=1}^{n-1}\sum_{j>i} X_{ij}\right] = \sum_{i=1}^{n-1}\sum_{j>i} \mathbf{E}\left[X_{ij}\right]$$

- p_{ij}: Wahrscheinlichkeit für Vergleich von $R[i]$ und $R[j]$,
  also $\mathbf{E}\left[X_{ij}\right] = 1 \cdot p_{ij} + 0 \cdot (1 - p_{ij}) = p_{ij}$.
- Die p_{ij} sind *nicht* alle gleich! (Indizierung in R!)

2.26 Satz

Der Erwartungswert der Laufzeit von RandQuickSort für Eingaben der Länge n ist in $O(n \log n)$.

2.27 Beweis (1)

- Seien i und j $(1 \leq i < j \leq n)$ beliebig aber fest.
- Betrachte binäre Bäume mit den zu sortierenden Zahlen als Knoten, die durch je eine Ausführung von RandQuickSort wie folgt rekursiv festgelegt sind:
  - Wurzel des Baumes: zufällig gewähltes Pivotelement y.
  - linker Teilbaum: rekursiv nach gleicher Regel aus $\text{RandQuickSort}(S_1[1 : j_1 - 1])$ und
  - rechter Teilbaum: rekursiv nach gleicher Regel aus $\text{RandQuickSort}(S_2[1 : j_2 - 1])$.

2.27 Beweis (2)

- durchlaufe Baum
  - beginnend bei der Wurzel,
  - nacheinander absteigend jedes Niveau
  - jeweils von links nach rechts alle Knoten
- irgendwann erstmals Element $R[k]$ mit $i \leq k \leq j$
- hier Entscheidung, $R[i]$ *vor* $R[j]$ einzusortieren

2.27 Beweis (2)

- durchlaufe Baum
  - beginnend bei der Wurzel,
  - nacheinander absteigend jedes Niveau
  - jeweils von links nach rechts alle Knoten
- irgendwann erstmals Element $R[k]$ mit $i \leq k \leq j$
- hier Entscheidung, $R[i]$ *vor* $R[j]$ einzusortieren
- zwei Fälle:
  1. $k = i$ oder $k = j$, d. h. $R[i]$ oder $R[j]$ ist Pivotelement und die beiden werden miteinander verglichen.
  2. $i < k < j$, d. h.
     - ein anderes Element ist Pivot
     - $R[i]$ und $R[j]$ werden nicht miteinander verglichen
     - kommen in verschiedene Teilbäume
     - werden folglich auch später nie miteinander verglichen

2.27 Beweis (3)

- jedes (noch) zur Verfügung stehende Element gleichwahrscheinlich als Pivotelement ausgewählt
- offensichtlich eines der $j - i + 1$ Elemente $R[i], \ldots, R[j]$ ausgewählt
- Im betrachteten Schritt wird also gleichwahrscheinlich
  - eines von $j - i + 1$ Elementen ausgewählt und
  - in zwei Fällen ($k = i$, $k = j$) Vergleich von $R[i]$ und $R[j]$
- Wahrscheinlichkeit für Vergleich von $R[i]$ und $R[j]$: $p_{ij} = 2/(j - i + 1)$.

2.27 Beweis (4)

Also:

$$\begin{aligned}
\mathbf{E}\left[\sum_{i=1}^{n-1}\sum_{j>i} X_{ij}\right] &= \sum_{i=1}^{n-1}\sum_{j>i} \mathbf{E}\left[X_{ij}\right] \\
&= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \frac{2}{j-i+1} = \sum_{i=1}^{n-1}\sum_{k=2}^{n-i+1} \frac{2}{k} \\
&\leq 2\sum_{i=1}^{n}\sum_{k=1}^{n} \frac{1}{k} = 2nH_n \ .
\end{aligned}$$

n-te harmonische Zahl: $H_n = \sum_{k=1}^{n} \frac{1}{k} = \ln n + \Theta(1)$.

□

2.28

- Manche randomisierten Algorithmen liefern immer die richtige Antwort.
  $\Longrightarrow$ **Las Vegas Algorithmen** (sofern …)
- Manche randomisierten Algorithmen liefern manchmal eine falsche Antwort.
  $\Longrightarrow$ **Monte Carlo Algorithmen**

Zusammenfassung

1. Randomisierte Algorithmen enthalten eine Zufallskomponente.
2. Das führt im Allgemeinen dazu, dass — *bei festgehaltener Eingabe* — z. B. die Laufzeit eine randomisierten Algorithmus eine Zufallsvariable ist.
3. Manche randomisierten Algorithmen liefern immer das richtige Ergebnis.
4. Manche randomisierten Algorithmen liefern unter Umständen ein falsches Ergebnis. Dann ist man im Allgemeinen an kleinen Fehlerwahrscheinlichkeiten interessiert.