Die Methode der Fingerabdrücke

Seminar Algorithmik WS14/15

Ole Groth

Inhalt

1. Einleitung
    1. Randomisierte Algorithmen
    2. Grundlagen der Methode der Fingerabdrücke
2. Kommunikationsprotokoll: Datenbank-Abgleich
3. Das Teilstringproblem
4. Verifikation der Matrixmultiplikation
5. Fazit

Randomisierte Algorithmen

- Probleme können häufig durch deterministische oder randomisierte Algorithmen gelöst werden
- Randomisierte Algorithmen finden z.T. ein näherungsweise korrektes Ergebnis
- Randomisierte Algorithmen trotz Fehlerwahrscheinlichkeit ≠ 0?
  - Effizienz
  - Einfache Implementierung
  - Kein deterministischer Algorithmus bekannt

Grundlagen der Methode der Fingerabdrücke

- Äquivalenzproblem: Sind zwei Objekte äquivalent?
  - Deterministischer Vergleich komplexer Objekte ist ungünstig
- Grundidee: zwei vollständige zum effizienten Vergleich zu komplexe Darstellungen von Objekten durch deren Fingerabdrücke vergleichen
- Anforderungen an Fingerabdrücke:
  - Einfach und kurz, effizienten Vergleich ermöglichen
  - Komprimierte oder unvollständige Darstellung der Objekte
  - So viel wie möglich substanzielle Information trotz beschränkter Größe

Grundlagen der Methode der Fingerabdrücke

- Menge M von Funktionen: vollständige Darstellung → Fingerabdruck
- Für beliebige unterschiedliche Objekte O_1 und O_2 soll es hinreichend viele Fingerabdrücke in M geben, die bezeugen, dass $O_1 \neq O_2$
  - $h \in M$ bezeugt $O_1 \neq O_2$, wenn $h(O_1) \neq h(O_2)$
- Kompromiss zwischen
  - Fingerabdruck soll das Objekt genau repräsentieren
  - Fingerabdrücke klein halten, um effizienten Vergleich zu gewährleisten

Kommunikationsprotokoll: Datenbank-Abgleich

- Zwei Rechner R_1 und R_2 haben je eine Datenbank mit ursprünglich gleichem Inhalt
  - Änderungen sollen möglichst auf beiden Datenbanken ausgeführt werden
  - Nach einiger Zeit: Haben die Datenbanken von R_1 und R_2 den gleichen Inhalt?

- n: Größe der Datenbank in Bits
- Deterministischer Algorithmus
  - R_1 schickt seine Datenbank als Folge von Bits an R_2
  - R_2 prüft Bit für Bit, ob Datenbanken übereinstimmen
  - Zuverlässig, aber ineffizient; alle n Bits übertragen und prüfen

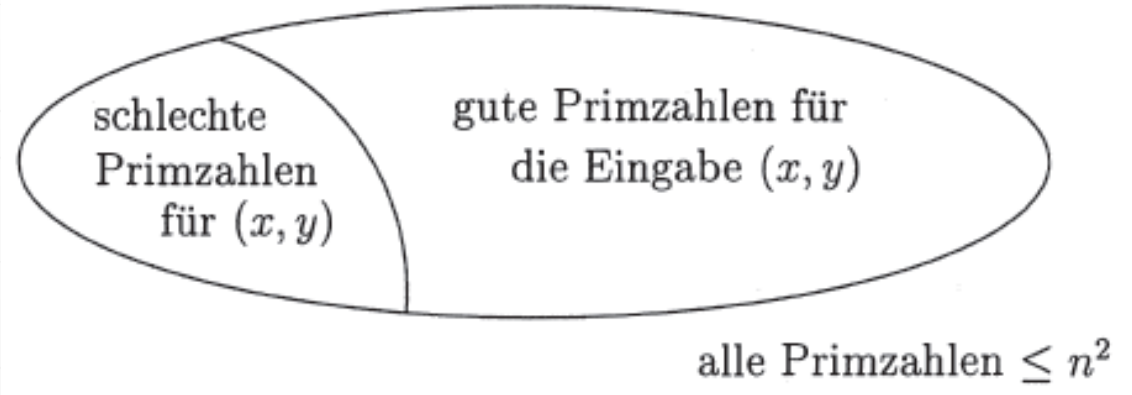
Kommunikationsprotokoll: Datenbank-Abgleich

- Randomisierter Algorithmus
  - R_1 hat n Bits $x = x_1 \dots x_n$, R_2 hat n Bits $y = y_1 \dots y_n$

1. R_1 wählt zufällig eine Primzahl $p \leq n^2$ (alle Primzahlen haben die gleiche Wahrscheinlichkeit gewählt zu werden)
2. R_1 berechnet $s = Num(x) \bmod p$,
   sendet s und p an R_2
3. R_2 berechnet $q = Num(y) \bmod p$
4. R_2 entscheidet: Wenn $s = q$ → die Datenbanken sind gleich
   Wenn $s \neq q$ → die Datenbanken sind ungleich

Kommunikationsprotokoll: Datenbank-Abgleich

- Komplexität (gemessen an Anzahl der Kommunikationsbits)
  - R_1 überträgt s und p an R_2
  - $s \leq p < n^2$
  - Länge der übertragenen binären Nachricht: $2 * \log_2 n^2$
  - Für $n=10^{16}$: ~213 Bits
- Fehlerwahrscheinlichkeit



Kommunikationsprotokoll: Datenbank-Abgleich

- Fehlerwahrscheinlichkeit
- 1. Fall: x = y
  - Dann gilt: Num(x) mod p = Num(y) mod p (für alle Primzahlen p)
  - Fehlerwahrscheinlichkeit = 0
- 2. Fall: x ≠ y
  - Es gibt „schlechte“ Primzahlen, wenn Num(x) mod p = Num(y) mod p
  - Fehlerwahrscheinlichkeit = $\frac{Anzahl\ der\ schlechten\ Primzahlen\ f\ddot{u}r\ (x,y)}{Anzahl\ aller\ Primzahlen < n^2}$

Kommunikationsprotokoll: Datenbank-Abgleich

- Fehlerwahrscheinlichkeit für x ≠ y:
- Abschätzen der Anzahl aller Primzahlen < n² (Prim(n²)) über Primzahlsatz:
  - $\lim_{n\to\infty}\left(\frac{Prim(m)}{m/\ln m}\right) = 1$
  - Für großes m verhält sich die Anzahl der Primzahlen ($Prim(m)$) annähernd wie $\frac{m}{\ln m}$
- Anzahl aller Primzahlen < n²: $Prim(n^2) \approx \frac{n^2}{\ln n^2}$

Kommunikationsprotokoll: Datenbank-Abgleich

- Wann ist eine Primzahl p schlecht für die Eingabe (x,y)?

- s = Num(x) mod p = Num(y) mod p
- Num(x) = x‘ * p + s und Num(y) = y‘ * p + s
- Num(x) – Num(y) = x‘ * p – y‘ * p = (x‘ – y‘) * p
- → p teilt Num(x) – Num(y)

Kommunikationsprotokoll: Datenbank-Abgleich

- Fehlerwahrscheinlichkeit für x ≠ y:
- Anzahl der „schlechten“ Primzahlen für (x,y)?
  - n-1

- Fehlerwahrscheinlichkeit = $\frac{Anzahl\ der\ schlechten\ Primzahlen\ f\ddot{u}r\ (x,y)}{Anzahl\ aller\ Primzahlen < n^2} = \frac{n-1}{n^2 / \ln n^2}$
- Für $n=10^{16}$: Fehlerwahrscheinlichkeit = $0{,}36892 * 10^{-14}$

Kommunikationsprotokoll: Datenbank-Abgleich

- Verringern der Fehlerwahrscheinlichkeit:

- 10 Fingerabdrücke mit verschiedenen, unabhängigen Primzahlen erstellen
  - Nur wenn zufällig 10 schlechte Primzahlen gewählt wurden, tritt ein Fehler auf

- Höheren Exponenten für n^d wählen
  - Fehlerwahrscheinlichkeit = $\frac{n-1}{n^d / \ln n^d}$ statt $\frac{n-1}{n^2 / \ln n^2}$

Das Teilstringproblem

- Mustererkennung (aus Text- oder Bildverarbeitung)
- Kommt ein Muster x in einem Text y vor? An welchem Index?

- Trivialer deterministischer Algorithmus: O(n*m)
- Algorithmus von Knuth-Morris-Pratt: O(n+m)

Das Teilstringproblem

- Randomisierter Algorithmus STRING(f):
  - y(r,n): Teilstring von y ab Position r mit Länge n
  - Gegeben: Muster $x = x_1x_2\ldots x_n$, Text $y = y_1y_2\ldots y_m$ über {0,1}, $n \leq m$
    Funktion f zur Auswahl einer Primzahl, z.B.: $f(n,m) = n^2m * \ln(n^2m)$

1. Wähle zufällig eine Primzahl p aus PRIM(f(n,m))
2. Berechne $\text{Finger}_p(x) := \text{Num}(x) \bmod p$
3. Berechne $\text{Finger}_p(y(r,n)) := \text{Num}(y(r,n)) \bmod p$
   für alle $r \in \{1, 2, \ldots m\text{-}n+1\}$
4. Wenn $\text{Finger}_p(y(r,n)) == \text{Finger}_p(x)$:
   Wenn y(r,n) == x: Gib Index r als Ergebnis aus
   Sonst fahre mit r+1 fort

Das Teilstringproblem

- Der randomisierte Algorithmus gibt immer das korrekte Ergebnis aus
- Laufzeit:
- Nächste Nummer lässt sich aus vorheriger Nummer berechnen (O(1)):
  - $Num(y(r+1,n)) = 2 * (Num(y(r,n)) - 2^{n-1} * y_r) + y_{r+n}$
- Berechnung von $Finger_p$(y(r+1,n)) aus $Finger_p$(y(r,n)):
  - $Finger_p(y(r+1),n)) = (2 * (Finger_p(\mathrm{y(r,n)}) - (2^{n-1} * y_r) \bmod p) + y_{r+n}) \bmod p$
- Laufzeit: O(m)

Verifikation der Matrixmultiplikation

- Gegeben: 3 (n x n)-Matrizen A,B und C
- Verifizieren, ob A * B = C

- Deterministische Matrixmultiplikation:
  - Schulalgorithmus: $O(n^3)$
  - Strassen-Algorithmus: $O(n^{log2(7)})$
  - Bester bekannter Algorithmus: $O(n^{2,376})$

- Freivalds Algorithmus: Randomisierter Algorithmus zur Verifikation

Verifikation der Matrixmultiplikation

- Freivalds Algorithmus
  - Gegeben: 3 $(n\ x\ n)$-Matrizen A, B und C

1. Wähle einen zufälligen Vektor $\alpha \in \{0,1\}^n$
2. Berechne die Vektoren $\beta := A * (B * \alpha)$
   $\gamma := C * \alpha$
3. Wenn $\beta = \gamma \rightarrow A * B = C$
   Wenn $\beta \neq \gamma \rightarrow A * B \neq C$

Verifikation der Matrixmultiplikation

- Laufzeitverbesserung: Multiplikation mit Vektor statt Matrix
  - Laufzeit: $O(n^2)$

- Fehlerwahrscheinlichkeit für A * B = C:
  - Es gilt: $A * B * \alpha = C * \alpha$ für alle $\alpha \in \{0,1\}^n$
  - → Der Algorithmus entscheidet immer richtig

Verifikation der Matrixmultiplikation

- Fehlerwahrscheinlichkeit für A * B ≠ C: $\leq \frac{1}{2}$

- Verbesserung durch k-faches Wiederholen des Algorithmus
  - Fehlerwahrscheinlichkeit sinkt auf $\leq \frac{1}{2^k}$
  - Aber: die Laufzeit erhöht sich auf $O(k*n^2)$

Fazit

- Methode der Fingerabdrücke: Äquivalenzproblem entscheiden
  - Fingerabdruck: vereinfachte Repräsentation eines Objekts
  - Vereinfachte Vergleichbarkeit
- Effizienter als deterministische Algorithmen
  - Datenaustausch geringer
  - Bessere Laufzeiten
- Fehlerwahrscheinlichkeit beachten
  - Möglichkeiten zur Verringerung der Fehlerwahrscheinlichkeit

Danke für die Aufmerksamkeit!