Datenströme

Prof. Dr. Nicole Schweikardt

Arbeitsgruppe Logik in der Informatik
Humboldt-Universität zu Berlin

Vorlesung „Big Data Analytics in Theorie und Praxis“

Datenströme

Situation:

- riesige Mengen von Daten
- automatisch generiert
- ständig kommen neue Daten hinzu

Beispiele:

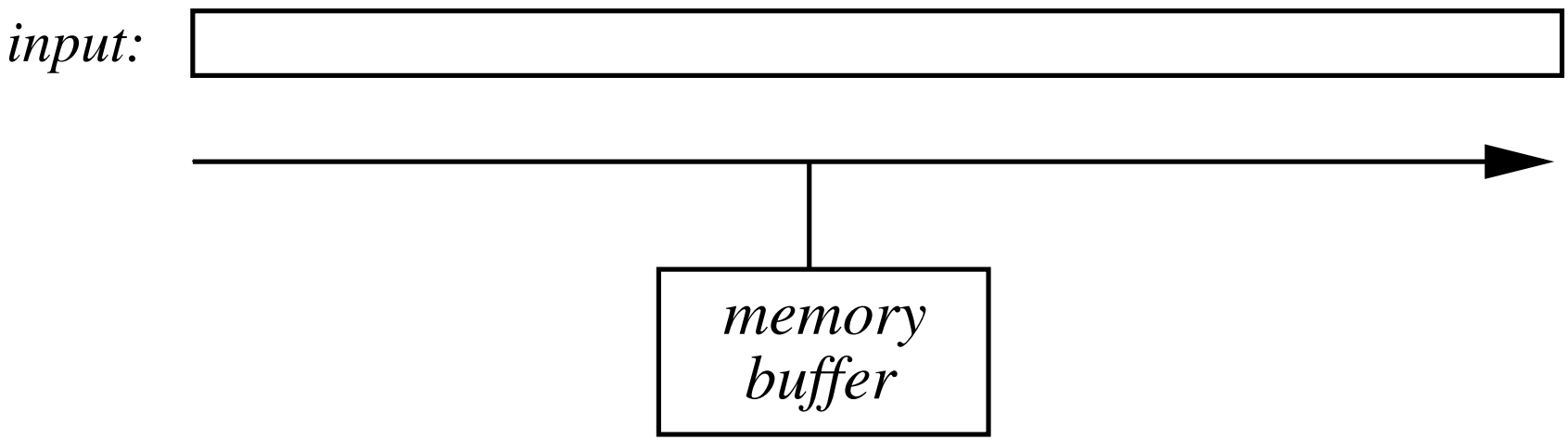
- Meteorologische Daten (Sensor-Netzwerke)
- Astronomische Daten
- Börsen-Transaktionen
- Netzwerk-Monitoring

Herausforderungen:

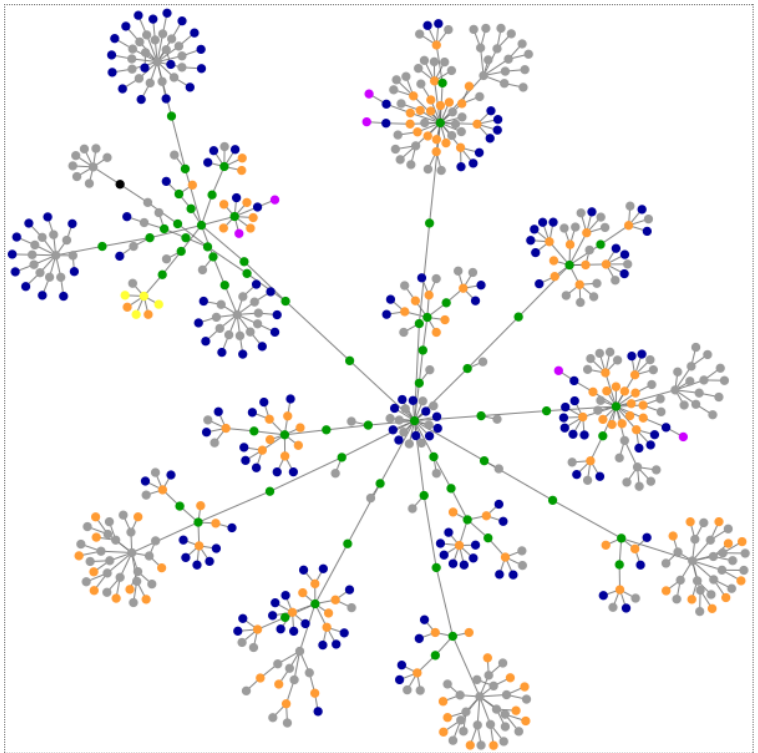
- Warten, bis alle Daten da sind, ist nicht möglich
  $\rightsquigarrow$ Daten müssen “on-the-fly” verarbeitet werden
- Speichern der gesamten Daten ist zu teuer $\rightsquigarrow$ speichere eine “Skizze”
- Daten können so schnell aufeinander folgen, dass man keine Gelegenheit hat, sich jeden einzelnen Datensatz anzusehen $\rightsquigarrow$ ziehe “Stichproben”

Einmaliges Lesen des Datenstroms

Szenario:



Ein Beispiel aus dem Bereich Netzwerk-Monitoring



A sei ein Knoten im Internet.
Als Eingabe erhält *A* einen Strom von “Paketen”

$$p_1, p_2, p_3, p_4, \ldots$$

Jedes Paket p_i enthält Informationen über

- die IP-Adresse, von der das Paket losgeschickt wurde,
- die IP-Adresse, an die das Paket geschickt werden soll,
- die eigentlichen Daten, die verschickt werden sollen

Frage: Wie viele verschiedene IP-Adressen haben seit heute morgen ein oder mehrere Pakete über Knoten *A* geleitet?

Problem: *A* will/kann nicht den gesamten Strom $p_1, p_2, p_3, \ldots$ zwischenspeichern.

Lösung: Nutze randomisierte Algorithmen, die eine gute Approximation für die Antwort auf obige Frage berechnen.

Beispiel: Suche nach der fehlenden Zahl

SUCHE NACH DER FEHLENDEN ZAHL

Eingabe: Strom $x_1, x_2, x_3, \ldots, x_{n-1}$ von $n-1$ verschiedenen Zahlen aus $\{1, \ldots, n\}$

Frage: Welche Zahl aus $\{1, \ldots, n\}$ fehlt?

Naive Lösung: $2\ 5\ 1\ 3\ 4\ 8\ 6\ \cdots\ n$ benötigt n Speicher-Bits

1	2	3	4	5	6	7	8	$\cdots$	n
✓	✓	✓	✓	✓	✓		✓	✓	✓

Bessere Lösung: Speichere Zwischensumme $O(\log n)$ Bits reichen aus

$$s := x_1 + x_2 + x_3 + x_4 + \cdots + x_{n-1}$$

$$\text{fehlende Zahl} = \frac{n \cdot (n+1)}{2} - s$$

Untere Schranke: mindestens $\log n$ Bits sind nötig

Das MULTISET-EQUALITY Problem

MULTISET-EQUALITY — *Eingabelänge:* $N = O(m \cdot \log n)$ *Bits*

Eingabe: Zwei Multimengen $\{\!\{x_1, \ldots, x_m\}\!\}$ und $\{\!\{y_1, \ldots, y_m\}\!\}$ von Zahlen x_i, y_j aus $\{1, \ldots, n\}$

Frage: Ist $\{\!\{x_1, \ldots, x_m\}\!\} = \{\!\{y_1, \ldots, y_m\}\!\}$?
D.h.: Kommen dieselben Zahlen mit denselben Vielfachheiten vor?

Satz:

Jeder deterministische Datenstrom-Algorithmus zur Lösung dieses Problems benötigt $\Omega(N)$ Speicher-Bits.

Beweismethode:

- Nutze Resultate aus der Kommunikationskomplexität:

  Zu entscheiden, ob zwei m-elementige Teilmengen von $\{1, \ldots, n\}$ gleich sind, benötigt man mindestens $\log \binom{n}{m}$ Kommunikations-Bits.

- Falls $n = m^2$, so sind $\log \binom{n}{m} \geqslant m \cdot \log m$ Kommuniktions-Bits nötig, und die Eingabelänge der MULTISET-EQUALITY Instanz ist $N = \Theta(m \cdot \log m)$.

Kommunikationskomplexität

Yaos 2-Parteien Kommunikations-Modell:

- 2 Spieler: Alice & Bob
- beide kennen eine Abbildung $f : A \times B \to \{0, 1\}$
- Alice sieht nur die Eingabe $a \in A$, Bob sieht nur die Eingabe $b \in B$
- beide wollen gemeinsam den Wert $f(a, b)$ berechnen
- Ziel: Tausche so wenig Kommunikations-Bits wie möglich aus

Satz: Zu entscheiden, ob zwei m-elementige Mengen

$$a = \{x_1, .., x_m\} \subseteq \{1, .., n\} \quad \text{und} \quad b = \{y_1, .., y_m\} \subseteq \{1, .., n\}$$

gleich sind, erfordert mindestens $\log\binom{n}{m}$ Kommunikations-Bits.

Das MULTISET-EQUALITY Problem

MULTISET-EQUALITY — *Eingabelänge:* $N = O(m \cdot \log n)$ *Bits*

Eingabe: Zwei Multimengen $\{\!\{x_1, \ldots, x_m\}\!\}$ und $\{\!\{y_1, \ldots, y_m\}\!\}$ von Zahlen x_i, y_j aus $\{1, \ldots, n\}$

Frage: Ist $\{\!\{x_1, \ldots, x_m\}\!\} = \{\!\{y_1, \ldots, y_m\}\!\}$?
D.h.: Kommen dieselben Zahlen mit denselben Vielfachheiten vor?

Satz:

Jeder deterministische Datenstrom-Algorithmus zur Lösung dieses Problems benötigt $\Omega(N)$ Speicher-Bits.

Beweismethode:

- Nutze Resultate aus der Kommunikationskomplexität:

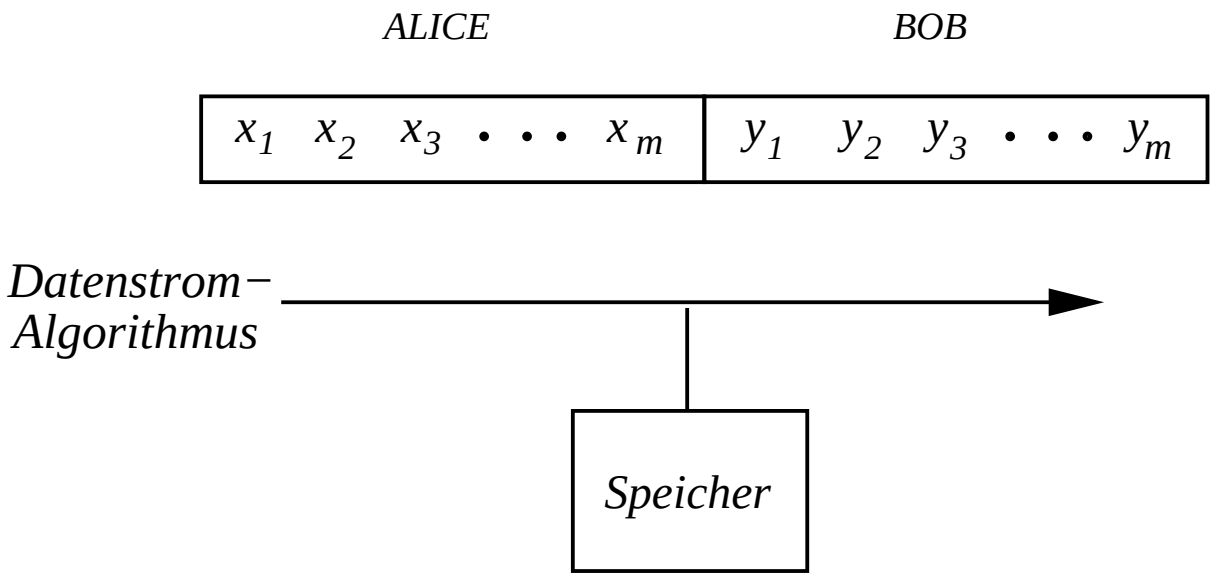
  Zu entscheiden, ob zwei m-elementige Teilmengen von $\{1, \ldots, n\}$ gleich sind, benötigt man mindestens $\log \binom{n}{m}$ Kommunikations-Bits.

- Falls $n = m^2$, so sind $\log \binom{n}{m} \geqslant m \cdot \log m$ Kommuniktions-Bits nötig, und die Eingabelänge der MULTISET-EQUALITY Instanz ist $N = \Theta(m \cdot \log m)$.

Eine untere Schranke für MULTISET-EQUALITY

Beweis (Fortsetzung):

- **Wir haben eben gesehen:** Eingabelänge: $N = \Theta(m \cdot \log m)$;
  zum Lösen von MULTISET-EQUALITY benötigte Kommunikations-Bits: $\geqslant m \cdot \log m$
- Ein deterministischer Datenstrom-Algorithmus, der MULTISET-EQUALITY löst und dabei B Speicher-Bits nutzt, führt zu einem Kommunikations-Protokoll, das B Kommunikations-Bits nutzt:



- Somit: Untere Schranke für Kommunikationskomplexität $\rightsquigarrow$ untere Schranke für Speicherplatz des Datenstrom-Algorithmus

Ein randomisierter Datenstrom-Algorithmus

Satz: *(Grohe, Hernich, Schweikardt, PODS'06)*

Das MULTISET-EQUALITY *Problem kann im folgenden Sinn durch einen randomisierten Datenstrom-Algorithmus, der $O(\log N)$ Speicher-Bits nutzt, gelöst werden:* *Sind m, n und ein Strom von Zahlen $a_1, \ldots, a_m, b_1, \ldots, b_m$ aus $\{1, \ldots, n\}$ gegeben, so*

- *akzeptiert der Algo mit Wk 1* *falls* $\{\!\{a_1, \ldots, a_m\}\!\} = \{\!\{b_1, \ldots, b_m\}\!\}$
- *verwirft der Algo mit Wk* $\geqslant 0.9$ *falls* $\{\!\{a_1, \ldots, a_m\}\!\} \neq \{\!\{b_1, \ldots, b_m\}\!\}$.

Grundprinzip des Algorithmus: Benutze "Fingerprinting"-Techniken:

- Repräsentiere $\{\!\{a_1, \ldots, a_m\}\!\}$ durch ein Polynom $f(x) := \sum_{i=1}^{m} x^{a_i}$.
- Repräsentiere $\{\!\{b_1, \ldots, b_m\}\!\}$ durch ein Polynom $g(x) := \sum_{i=1}^{m} x^{b_i}$.
- Wähle eine Zufallszahl z und überprüfe, ob $f(z) = g(z)$.
- Akzeptiere, falls $f(z) = g(z)$. Ansonsten verwerfe.

Falls $\{\!\{a_1, \ldots, a_m\}\!\} = \{\!\{b_1, \ldots, b_m\}\!\}$, so ist $f(x) = g(x)$, und daher akzeptiert der Algorithmus stets. Falls $\{\!\{a_1, \ldots, a_m\}\!\} \neq \{\!\{b_1, \ldots, b_m\}\!\}$, dann gibt es nur $Grad(f-g)$ viele Zahlen z mit $f(z) = g(z)$, und daher verwirft der Algorithmus mit hoher Wk.