Der kontrollierte Zufall

in der Informatik

Andreas Schwill
Institut für Informatik
Universität Potsdam

Überblick

- **Motivation – Bürokrat – Spieler**
- **Das verirrte Kalb - das verlorene Auto**
- **Matrixmultiplikation**
- **Leader election in Rechnernetzen**
- **Abgleich von Datenbanken**
- **Ausblick**

1 Motivation

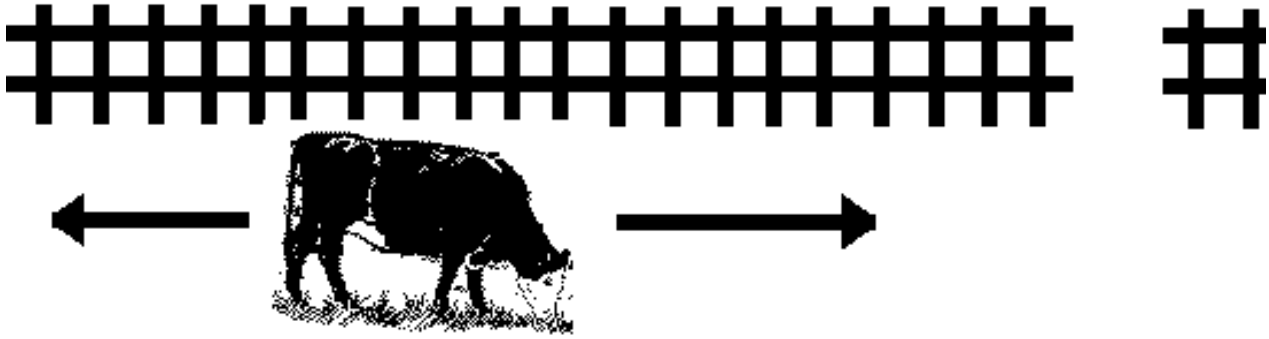
Der Bürokrat

- **Zufall kontrollieren - das widerspricht sich doch.**
- **Zufälle muß man ausschließen.**
- **Zufall ist gefährlich.**
- **Ich bin froh, daß mein Computer vorhersehbar und korrekt arbeitet.**
- **Zufall kann überhaupt nichts nutzen.**

Der Spieler

- **Ich kann mit Glück gewinnen.**
- **Ich kann mehr gewinnen, als ich in meiner Lebenszeit erarbeiten kann.**
- **Ein paar Fehler/Verluste kann ich schon verschmerzen.**
- **Mein Betriebssystem arbeitet sowieso scheinbar zufällig.**
- **Das Leben ist sowieso gefährlich.**
- **Zufall eröffnet mir neue Perspektiven.**

2 Das verirrte Kalb (the lost cow problem)



Anwendung:

- **Roboterprogrammierung**
- **Orientierung in unbekanntem Gelände**
- **systematische Suche nach Zielen – "wo habe ich mein Auto geparkt?"**

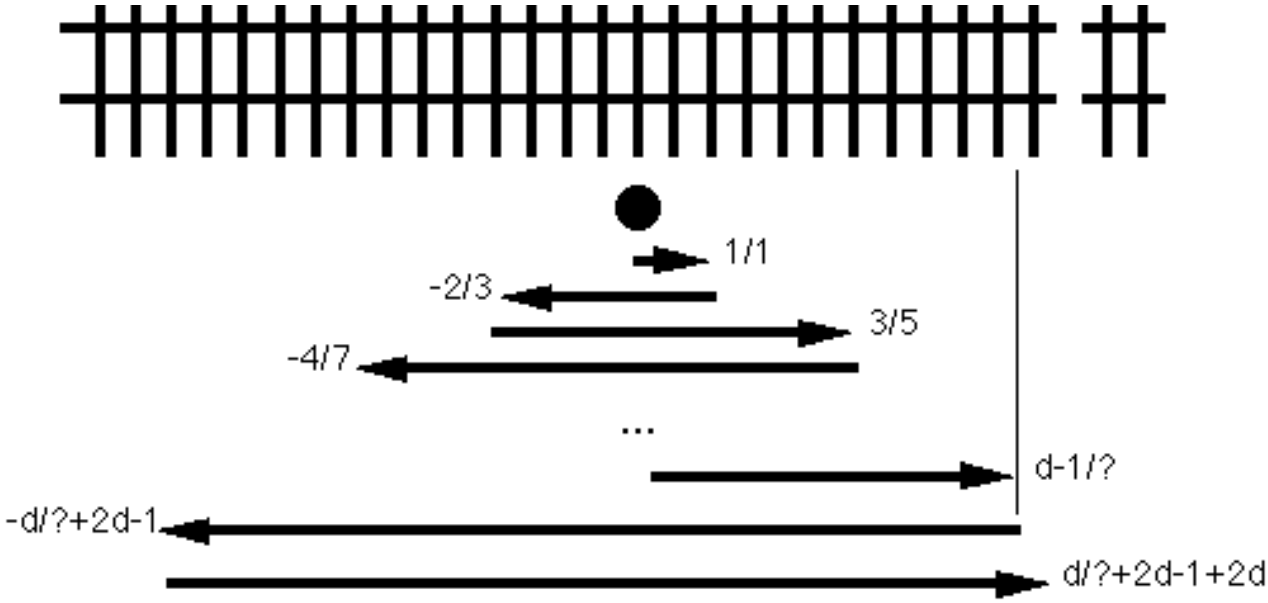
Allgemeiner Algorithmus:

Im i-ten Schritt suche in Richtung

i mod 2 (1=rechts, 0=links) bis zur Distanz f_i.

Das sorgfältige Kalb

$f_1=1, f_2=2, f_3=3, ..., f_i=i, ...$



Auswertung: worst case

Im schlimmsten Fall hat das Kalb im letzten Schritt das Tor gerade verpaßt und muß dann noch einmal die Strecke d-1+d in die falsche Richtung sowie d+d in die richtige Richtung traben:

Durchlaufene Strecke also

$$(1+(1+2)+(2+3)+(3+4)+...((d-1)+d)+2d$$
$$=2\Sigma k + 3d = d(d-1)+3d = d^2+2d=O(d^2)$$

Es geht noch viel schneller.

Das Turbo-Kalb

Verdoppelungsstrategie

$$f_0=0,\ f_1=4,\ f_2=8,\ \ldots,\ f_i=2^{i+1},\ \ldots$$

Auswertung: worst case

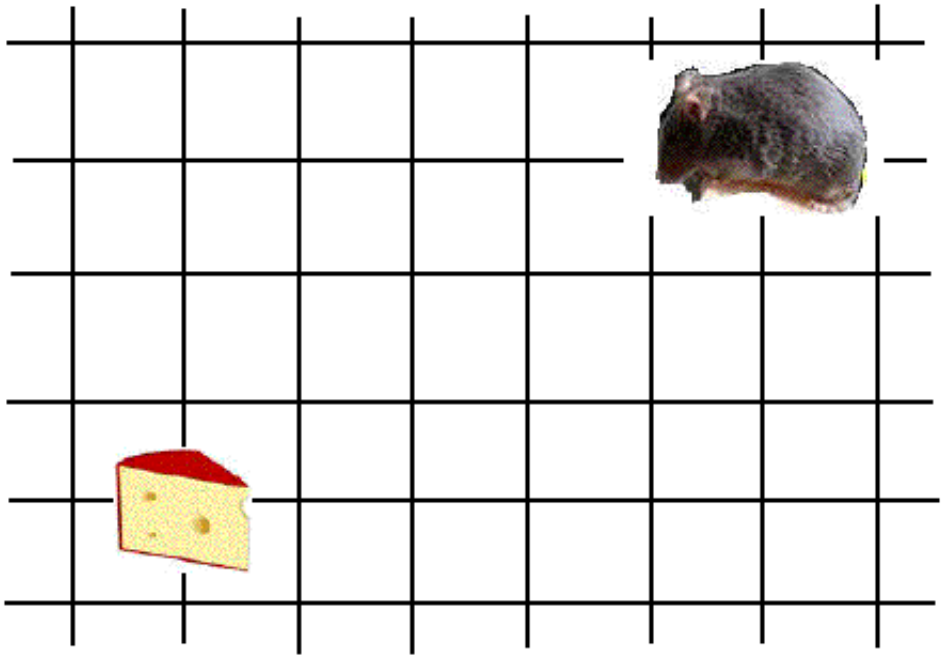
Wiederum hat das Kalb im schlimmsten Fall im i-ten Schritt das Tor gerade verpaßt und muß dann noch einmal die Strecke f_i+f_{i+1} in die falsche Richtung sowie $f_{i+1}+d$ in die richtige Richtung traben (findet das Tor also im (i+2)-ten Schritt):

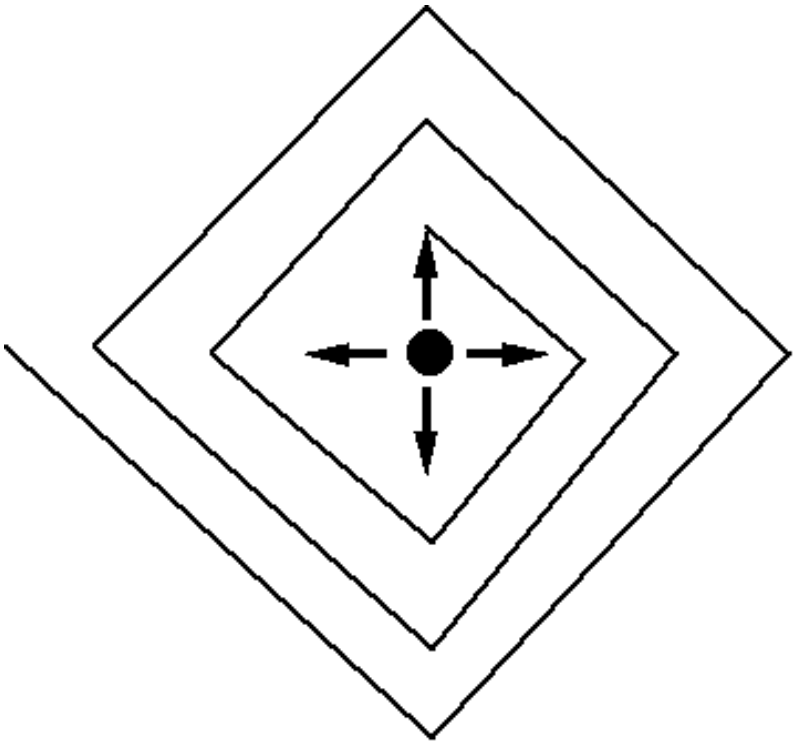
Durchlaufene Strecke also

$$2\cdot\Sigma\, f_j+d = 2\cdot\Sigma 2^j + d = 9\cdot 2^{i+1}-6+1 \leq 9f_i-5 \leq 9(f_i+1) = 9d.$$

9d ist die bestmögliche (!) Schranke für dieses Problem.

Erweiterung auf 2-dimensionale Ebene:





Spiralsuche ist mit geeigneten Schrittweiten optimal.

Das panische Kalb – der unkontrollierte Zufall

Ein einfacher Zufallsalgorithmus:

- **Solange Tor nicht gefunden tue folgendes**
  - **Wähle zufällig eine Richtung R (links oder rechts).**
  - **Gehe einen Schritt in Richtung R.**

Überraschend: Das Kalb findet das Tor mit Wahrscheinlichkeit 1.

Begründung: [Polya 1921: "Random Walks"]

Sei e_n eine Zufallsvariable mit

e_n=1, falls Kalb im n-ten Schritt an Ausgangsposition 0

und e_n=0, sonst.

Dann ist

$$T = \sum e_n$$

die Gesamtzahl aller "Nulldurchgänge" und

$$E(T) = \sum E(e_n)$$

der zugehörige Erwartungswert.

Sei u_n die Wahrscheinlichkeit, daß sich das Kalb im n-ten Schritt wieder an der Ausgangsposition befindet (u_0=1). Dann ist $E(e_n)=u_n$, also

$$E(T) = \sum u_n.$$

Zur Rückkehr an die Ausgangsposition wird eine geradzahlige Anzahl von Schritten benötigt: alle $u_{2n+1}=0$.

Wir berechnen u_{2n}:

$$u_{2n} = \binom{2n}{n} \frac{1}{2^{2n}}$$

Dann ist mit der Stirlingschen Formel $n! \sim \sqrt{2\pi n}\,(n/e)^n$:

$$u_{2n} \sim 1/\sqrt{\pi n}$$

und damit

$$E(T) = \sum u_n = \sum 1/\sqrt{\pi n} = \infty.$$

Folglich wird der Ausgangspunkt 0 unendlich oft erreicht.

Da jeder beliebige Punkt immer wieder eine positive Wahrscheinlichkeit besitzt, vom Ausgangspunkt erreicht zu werden, wird er schließlich auch erreicht.

Erweiterungen:

- **Der Algorithmus funktioniert auch in der 2-dimensionalen Ebene (-> Maus und Käse).**

- **Der Algorithmus funktioniert nicht im d-dimensionalen Raum für $d \geq 3$. Hier kehrt man nicht mit Wahrscheinlichkeit 1 an den Ausgangsort zurück und erreicht folglich auch nicht jeden Punkt.**

Der kontrollierte Zufall

Ein besserer Algorithmus:

- **Wähle r>1 fest.**
- **Wähle zufällig die Richtung R (links oder rechts).**
- **Wähle zufällig eine Zahl $\varepsilon \in [0,1)$.**
- **Sei $f_i = r^{i+\varepsilon}$ für i=1,2,3,...**
- **i:=1.**
- **Solange Tor nicht gefunden tue folgendes**
  - **falls R=links dann gehe bis zu Position $-f_i$**
    sonst gehe bis zu Position $+f_i$.
  - **Setze R auf die Gegenrichtung.**
  - **i:=i+1.**

Der Erwartungswert für die Zahl der Schritte beträgt

$$d \cdot \min_{r>1} (1+(1+r)/\ln r) \sim 4{,}59112 \cdot d$$

Erweiterung auf die Ebene ist möglich.

3 Matrixmultiplikation

Gegeben seien zwei Matrizen:

$$A = \begin{matrix} a_{11} \dots a_{1n} \\ \dots \\ a_{n1} \dots a_{nn} \end{matrix} \qquad B = \begin{matrix} b_{11} \dots b_{1n} \\ \dots \\ b_{n1} \dots b_{nn} \end{matrix}$$

Gesucht ist die Produktmatrix

$$C=(c_{ij})$$

Standardalgorithmus: traditionelles Multiplizieren

Aufwand:

n^3 **Multiplikationen und** $\sim n^3$ **Additionen.**

Strassen-Algorithmus [Duden Informatik]:

$O(n^{2,8074})$ **Multiplikationen.**

Bester Algorithmus:

$O(n^{2,376})$ **Multiplikationen.**

Der kontrollierte Zufall:

Gegeben drei Matrizen A, B, C.

Problem: Gilt $A \cdot B = C$?

Algorithmus [Freivalds 1979]:

- **Wähle $r \in \{0,1\}^n$ zufällig, d.h. $r_k=0$ oder 1 mit Wahrscheinlichkeit jeweils 1/2.**
- **Berechne**

  $x := B \cdot r$,

  $y := A \cdot x$,

  $z := C \cdot r$.
- **Falls y=z, dann Ausgabe "ja", sonst Ausgabe "nein".**

Beobachtung:

- **Falls $A \cdot B = C$ gilt, so ist Ausgabe immer korrekt;**
- **falls $A \cdot B \neq C$ gilt, so ist Ausgabe mit Wahrscheinlichkeit $\geq 1/2$ korrekt.**

  -> Monte-Carlo-Algorithmus (einseitige Fehlermöglichkeit).
- **k-malige Wiederholung des Algorithmus drückt die Fehlerwahrscheinlichkeit auf**

  $(1/2)^k$.
- **$O(n^2)$ Multiplikationen**

Begründung:

Seien A, B, C drei nxn Matrizen mit $A \cdot B \neq C$.

Sei $D := A \cdot B - C$. D ist nach Voraussetzung nicht die Nullmatrix.

Wir schätzen die Wahrscheinlichkeit für

$$D \cdot r = 0$$

ab.

Sei o.B.d.A. die erste Reihe von D nicht identisch 0 und seien alle von 0 verschiedenen Einträge dieser Reihe von D vor den Nulleinträgen, also

$$D = \begin{matrix} d_1\ d_2\ d_3\ \ldots\ d_k\ 0\ \ldots\ 0 \\ \ldots \\ \ldots \end{matrix} \quad =: d$$

d sei der so gewonnene Zeilenvektor von D, wobei die ersten $k \geq 1$ Einträge $\neq 0$ sind.

Das Skalarprodukt $d \cdot r$ liefert den ersten Eintrag von $D \cdot r$. Folglich ist die Wahrscheinlichkeit von $d \cdot r = 0$ eine obere Schranke für die Wahrscheinlichkeit von $D \cdot r = 0$.

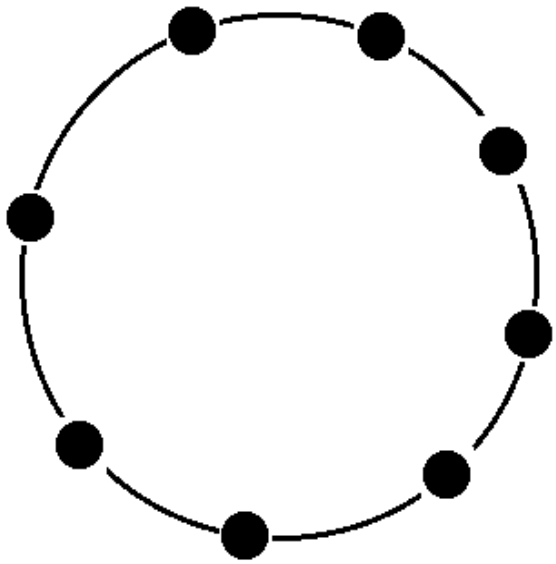
Dann gilt:

$$d \cdot r = 0 \iff \sum d_i \cdot r_i = 0 \iff r_1 = \left(-\sum d_i \cdot r_i\right)/d_1.$$

Wir können annehmen, daß $r_2, \ldots, r_n$ vor r_1 gewählt wurden. Dann ist der Bruch auf der rechten Seite eine Konstante w, und r_1 kann nur mit Wahrscheinlichkeit 1/2 gleich w sein.

Also ist die Wahrscheinlichkeit von $d \cdot r = 0$ und damit auch von $D \cdot r = 0$ höchstens 1/2.

4 Leader election in Rechnernetzen



Motivation:

- **automatisches Wiederanfahren eines Rechnernetzes**
- **Bestimmung eines ausgezeichneten Rechners für erste Verwaltungaufgaben**
- **Bestimmen eines neuen Leaders nach Ausfall des alten**
- **Auszeichnung einer Wurzel zur Erzeugung eines Spannbaumes**
- **Generierung eines Token in Tokennetzwerken**

Varianten:

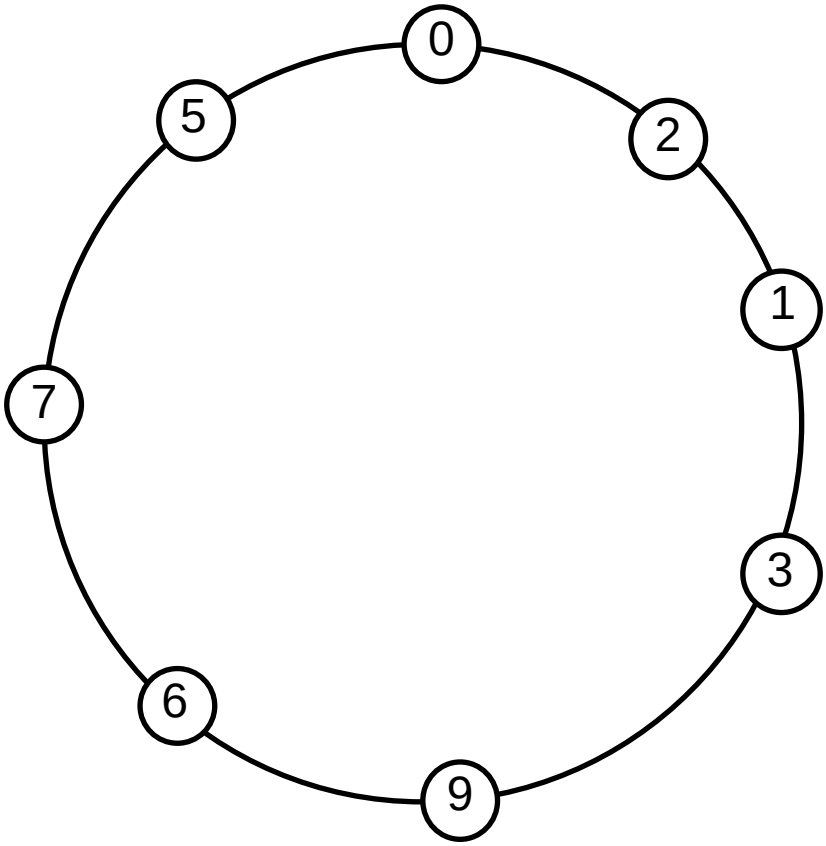
- **Uniformität – Größe des Netzwerkes bekannt**
- **Anonymität – Eindeutige Identifikation eines Rechners (z.B. IP-Nr.)**
- **Synchronität – Asynchronität**
- **Direktionalität von Kanten**
- **verschiedene Netztopologien (hier nur Ringe)**

Algorithmus [Franklin 1982]:

Zu Beginn seien alle Rechner weiß gefärbt (aktiv <-> passiv/schwarz).
Jeder Rechner besitzt eine eindeutige Identifikation (id).

Programm für jeden Rechner

- **sende beiden Nachbarn eine Nachricht mit Information id**
- **empfange Nachrichten mit Information id_1 und id_2 von beiden Nachbarn**
- **falls id_1>id oder id_2>id dann werde schwarz (schalte auf "Durchzug")**
- **falls id_1,id_2<id dann starte Programm erneut für die nächste Runde**
- **falls id_1=id_2=id dann erkläre dich zum Leader.**



Effizienz:

- **In einem Ring mit n Rechnern gibt es mind. n/2 Rechner, für die entweder der linke oder rechte eine größere id besitzt.**
- **In jeder Runde wird mind. die Hälfte der noch aktiven Rechner passiv.**
- **Nach max. log n Runden ist nur noch ein Rechner aktiv und damit der eindeutige Leader.**

- **Je Runde sendet/weiterleitet jeder der n Rechner 2 Nachrichten mit je log n Bits, insgesamt also $O(n (\log n)^2)$ Bits.**

Problem

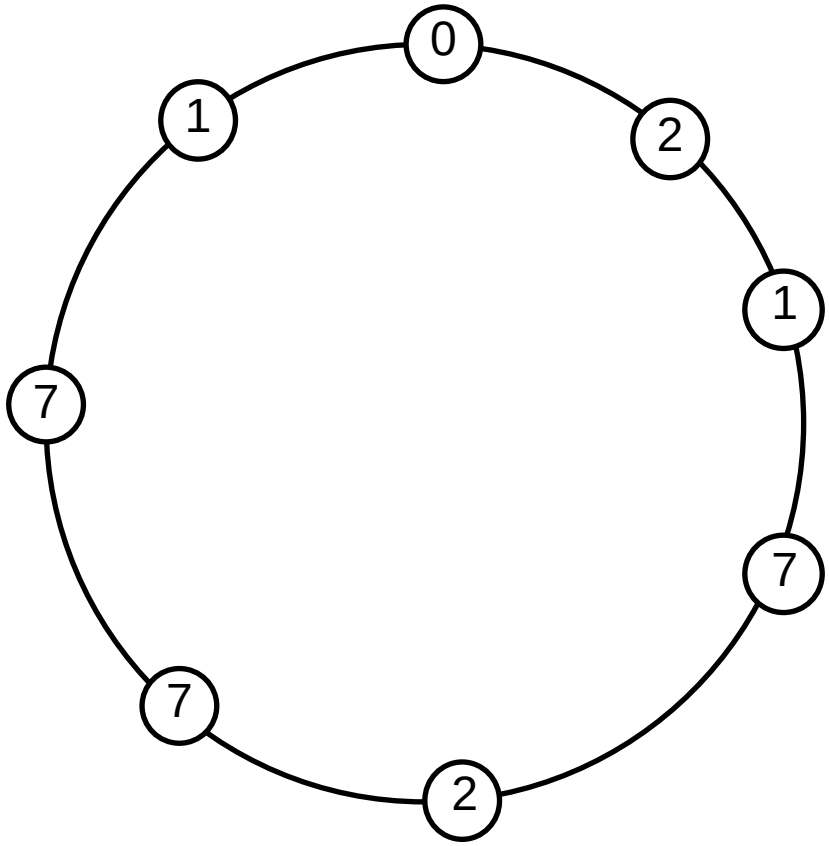
Es gibt keinen anonymen Leader-election Algorithmus für synchrone Rechnernetzringe.
(Grund: nach jeder Runde sind alle Prozessoren im selben Zustand – Folge der Symmetrie)

Der kontrollierte Zufall – Symmetry breaking

Programm für jeden Rechner

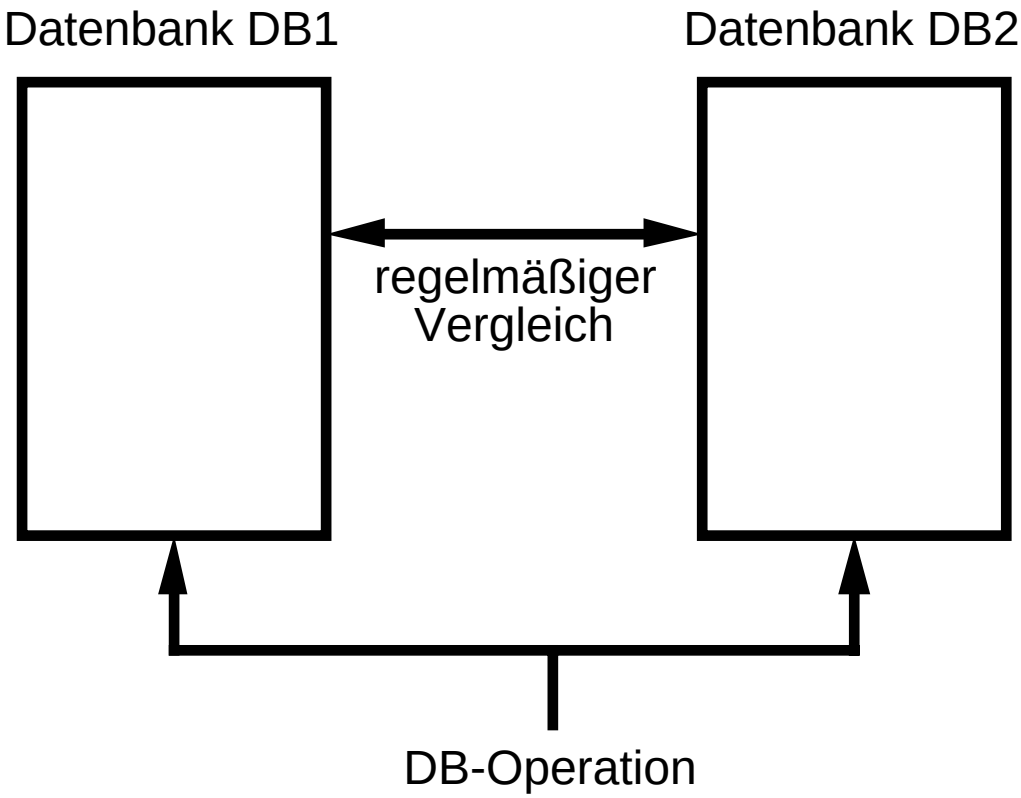
- **var s: list of {1,...,K}**
- **wiederhole**
    - **s:=∅**
    - **id sei eine zufällige Zahl aus {1,...K}**
    - **wiederhole n-mal**
        - **füge id zu s hinzu**
        - **sende id an linken Nachbarn**
        - **empfange id vom rechten Nachbarn**

  solange bis eine id in s eindeutig ist
- **Der Rechner mit der größten eindeutigen id wird der Leader.**



4 Abgleich von Datenbanken

Motivation:



Herkömmlicher Abgleich:

Übertragung von Terabits von DB1 an DB2.

Idee: **Vergleiche nur die Zertifikate der beiden Datenbanken.**

Hintergrund und Annahmen:

- **Unter den Zahlen 1, 2, 3, ..., n gibt es etwa n/ln n Primzahlen.**
- **Für eine Bitfolge $b=b_1b_2...b_n$ sei $N(b)=\sum b_i \cdot 2^{n-i}$ die Zahl mit Binärdarstellung b.**
- **Auf DB1 liegen die n Bits $x=x_1...x_n$, auf DB2 die n Bits $y=y_1...y_n$.**

Der kontrollierte Zufall

Algorithmus für DB1:

- **DB1 wählt zufällig gleichverteilt eine der etwa $n^2/\ln n^2$ Primzahlen p zwischen 1 und n^2.**
- **s:=N(x) mod p.**
- **Sende (s,p) an DB2.**

Algorithmus für DB2:

- **Empfange (s,p) von DB1.**
- **t:=N(y) mod p.**
- **Falls s≠t dann Ausgabe "DB1≠DB2" sonst Ausgabe "DB1=DB2".**

Kommunikationseffizienz:

- **Es wird nur das Paar (s,p) übertragen.**
- **Wegen $s \leq p \leq n^2$ ist die Länge der Nachricht**
  $\leq 2 \cdot \log_2 n^2 + 2 \leq 4 \cdot \log_2 n + 2$.
- **Bei 10^{12} Bits auf DB1 und DB2 sind also nur $4 \cdot 40 + 2 = 162$ Bits zu übertragen.**

Analyse:

- **Annahme: $x=y$.**

  Dann ist $N(x) \bmod p = N(y) \bmod p$. DB2 liefert die korrekte Antwort "DB1=DB2". Wenn also die Antwort "DB1≠DB2" geliefert wird, kann man sicher sein, daß diese Antwort korrekt ist.

- **Annahme: $x \neq y$.**

  DB2 liefert eine falsche Antwort, wenn zufällig

  $z = N(x) \bmod p = N(y) \bmod p$

  oder $N(x) = a*p+z$ und $N(y) = b*p+z$ für gewisse $a,b \in \mathbb{N}$

  bzw. $N(x)-N(y) = (a-b)*p$

  oder p teilt $N(x)-N(y)$.

  Wie oft kommt es vor, daß unter den $n^2/\ln n^2$ Primzahlen Teiler von $N(x)-N(y)$ sind?

  Weil x und y jeweils n Bits haben, gilt $w=|N(x)-N(y)|<2^n$.
  Sei $w=p_1^{i_1} p_2^{i_2}...p_k^{i_k}$ eine Primfaktorzerlegung von w mit $p_1<p_2<...<p_k$. Wir zeigen $k \leq n-1$.
  Das aus $w=p_1^{i_1} p_2^{i_2}...p_k^{i_k} > p_1 p_2...p_k > 1 \cdot 2 \cdot 3 \cdot ... \cdot n = n! > 2^n$ ein Widerspruch folgt, hat w höchstens n-1 Primfaktoren.

Da jede Primzahl aus {2,...,n^2} mit gleicher Wahrscheinlichkeit gewählt wird, ist die Wahrscheinlichkeit, ein p zu wählen, daß w teilt, höchstens

$$(n-1)/(n^2/\ln n^2) < (\ln n^2)/n.$$

Damit ist die Fehlerwahrscheinlichkeit des Programms für unterschiedliche Datenbanken DB1 und DB2 höchstens

$$(\ln n^2)/n.$$

Fehlerwahrscheinlichkeit für $n=10^{12}$:

$$0{,}553 \cdot 10^{-11}$$

5 Ausblick

- **Zufall steckt apriori in jedem Rechnersystem**
- **Zufall ist unvermeidbar**
- **Zufall vereinfacht Probleme**
- **Zufall ist nützlich**
- **Zufall löst Probleme**
- **Zufall ist notwendig**
- **Zufall muß beherrschbar bleiben**

DER KONTROLLIERTE ZUFALL

Literatur

- **Hromkovic: Zufall und zufallsgesteuerte Algorithmen LOGIN 21 (2001)**

- **Motwani, Raghavan: Randomized Algorithms Cambridge University Press 1995**

- **Niedermeier: Randomisierte Algorithmen, Skript 1997 <http://www-fs.informatik.uni-tuebingen.de/~niedermr/teaching>**

- **Vöcking: Randomized Algorithms, Skript 2001 <http://www.mpi-sb.mpg.de/~voecking>**