Typen randomisierter Algorithmen

Ein randomisierter Algorithmus A besitzt viele Berechnungen auf der selben Eingabe. Und wenn A unterschiedliche Ausgaben berechnet?

Angenommen, A gibt nur die Aussagen 0 oder 1.

- Dann ist

$$p_w = \text{prob}[A \text{ berechnet Ausgabe 1 für Eingabe } w]$$

die Akzeptanzwahrscheinlichkeit für Eingabe w.

- Wir sagen, dass A die Sprache

$$L(A) = \{ w \mid p_w > 1/2 \}$$

akzeptiert.

Und wenn p_w sehr nahe bei $1/2$ liegt?

Nichtdeterminismus

Jeder nichtdeterministische Algorithmus N lässt sich als ein randomisierter Algorithmus auffassen! Warum?

- Wir machen aus N einen randomisierten Algorithmus A, der eine Eingabe w genau dann akzeptiert, wenn N akzeptiert und fast genau so schnell wie N ist.
  - A simuliert N, indem A jede nichtdeterministische Entscheidung durch Zufallsbits ausführt.
  - Wenn N akzeptiert, dann akzeptiert A ebenfalls.
  - Wenn N verwift, dann wirft A eine faire Münze und akzeptiert, bzw. verwirft mit Wahrscheinlichkeit genau $1/2$.
- Wir können also $\mathcal{NP}$-vollständige Probleme effizient mit randomisierten Algorithmen lösen!

Wenn die Akzeptanzwahrscheinlichkeit p_w sehr nahe bei $1/2$ liegt, dann benötigen wir aber EXTREM VIELE Berechnungen, um die Mehrheitsausgabe mit kleinem Fehler voraussagen zu können!

Las Vegas Algorithmen

Ein randomisierter Algorithmus A heißt genau dann ein Las Vegas Algorithmus, wenn A **nie** einen Fehler macht.

- Warum der Name „Las Vegas“?

  Das Casino gewinnt immer!
- Um einen Las Vegas Algorithmus zu evaluieren, bestimmen wir die erwartete Laufzeit.

  Ein schneller Las Vegas Algorithmus muss also nicht stets halten!
- Warum betrachten wir die erwartete Laufzeit und warum lassen wir unendlich lange Berechnungen zu?

$\mathcal{ZPP}$ (zero-error probabilistic polynomial time) besteht aus allen Sprachen, die mit Las Vegas Algorithmen in polynomieller erwarteter Laufzeit berechenbar sind.

Einseitiger Fehler

Ein randomisierter Algorithmus A akzptiert eine Sprache L mit **einseitig** beschränktem Fehler, falls für alle Eingaben w gilt

$$\begin{array}{lcll} w \in L & \Rightarrow & \text{prob}[\, A \text{ akzeptiert } w \,] & > \frac{1}{2}, \\ w \notin L & \Rightarrow & \text{prob}[\, A \text{ verwirft } w \,] & = 1. \end{array}$$

- Warum erlauben wir keinen Fehler für Worte w der Sprache?
  - Jeder Algorithmus mit einseitigem Fehler für L lässt sich auch als ein nichtdeterministischer Algorithmus für L auffassen.
  - Algorithmen mit einseitigem Fehler raten zufällig!
- $\mathcal{RP}$ (random polynomial time) besteht aus allen Sprachen, die durch Algorithmen mit einseitigem Fehler in polynomieller worst-case Laufzeit berechenbar sind.

$$\mathcal{P} \subseteq \mathcal{ZPP} \subseteq \mathcal{RP} \subseteq \mathcal{NP}.$$

Fehlerreduktion

Sei A ein Algorithmus mit einseitigem Fehler.

- Wenn die Eingabe w zur Sprache gehört, dann muss A nur mit Wahrscheinlichkeit größer als $1/2$ akzeptieren.
- Können wir die Akzeptanzwahrscheinlichkeit erhöhen?

- Wiederhole Algorithmus A k-mal auf Eingabe w.
  - A verwirft in einem Versuch mit Wahrscheinlichkeit höchstens $1/2$.
  - Die k Wiederholungen entsprechen unabhängigen Experimenten.
  - A verwirft in k Versuchen mit Wahrscheinlichkeit höchstens 2^{-k}.
- Nach $k = 250$ Wiederholungen ist die inverse Fehlerwahrscheinlichkeit größer als die Anzahl der Atome im Universum.

Zweiseitiger Fehler

Ein randomisierter Algorithmus A akzeptiert eine Sprache L mit **zweiseitigem** Fehler, falls für alle Eingaben $w \in L$ gilt

$$\begin{aligned} w \in L \quad &\Rightarrow \quad \text{prob}[\ A \text{ verwirft } w\ ] \quad &< \tfrac{1}{3}, \\ w \notin L \quad &\Rightarrow \quad \text{prob}[\ A \text{ akzeptiert } x\ ] \quad &< \tfrac{1}{3}. \end{aligned}$$

- Die Fehlerwahrscheinlichkeit $1/3$ ist willkürlich gewählt.
- Können wir die Fehlerwahrscheinlichkeit drastisch senken?
  - Wende den Algorithmus k Mal auf Eingabe w an und übernimm das Mehrheitsergebnis.
  - Wir führen k unabhängige Experimente mit Fehlerwahrscheinlichkeit höchstens $1/3$ aus: Das Mehrheitsergebnis ist mit Wahrscheinlichkeit höchsten $e^{-k/48}$ falsch.
    - ★ Bei $k = 11232$ Wiederholungen ist die inverse Fehlerwahrscheinlichkeit größer als die Anzahl der Atome im Universum.

$\mathcal{BPP}$

$\mathcal{BPP}$ (bounded-error probabilistic polynomial time) besteht aus allen Sprachen, die durch Algorithmen mit zweiseitigem Fehler in polynomieller worst-case Laufzeit berechenbar sind.

- $\mathcal{P} \subseteq \mathcal{ZPP} \subseteq \mathcal{RP} \subseteq \mathcal{BPP}$.
- Wie wir später sehen werden, gibt es starke Indizien, dass tatsächlich $\mathcal{P} = \mathcal{BPP}$ gilt.

Wenn die Eingabe vollständig bekannt ist, liefern randomisierte Algorithmen also u. U. signifikante Leistungsverbesserungen, aber keine „Leistungsexplosion“.

Vermeidung des Wort-Case

Ein randomisierter Algorithmus A repräsentiert eine Klasse $\mathcal{A}$ deterministischer Algorithmen:

> Für jede Setzung der Zufallsbits erhalten wir einen deterministischen Algorithmus aus $\mathcal{A}$.

Die umgekehrte Sichtweise:

- Angenommen, die meisten Algorithmen einer Klasse $\mathcal{A}$ deterministischer Algorithmen „funktionieren“ für jede Eingabe.
- Dann ist die zufällige Wahl eines Algorithmus aus $\mathcal{A}$ ein guter randomisierter Algorithmus:

  > Der zufällig gewählte Algorithmus wird hochwahrscheinlich die vorgegebene Eingabe nicht als worst-case Eingabe besitzen.

Beispiel Quicksort:

Für jedes Array sind die meisten Pivots gut.

Zwei-Personen Spiele

- Spieler A und B treten gegen einander an.
- Spieler A beginnt.
- Die Spieler ziehen alternierend bis eine Endsituation erreicht ist, die für einen der beiden Spieler gewinnenbringend ist.

Wir modellieren das Spiel durch einen Spielbaum T:

- Die Blätter von T sind entweder mit Null oder Eins markiert, wobei eine Eins (bzw. Null) andeutet, dass Spieler A gewonnen (bzw. verloren) hat.
- Knoten mit geradem (bzw. ungeradem) Abstand von der Wurzel heißen MAX-Knoten (bzw. MIN-Knoten) und sind mit A (bzw. B) beschriftet.

Wie wertet man den Spielbaum aus?

Auswertung von Spielbäumen

- Der Spielwert eines Blatts ist die Beschriftung des Blatts,
- der Spielwert eines MAX-Knotens v ist das Maximum der Spielwerte der Kinder von v und
- der Spielwert eines MIN-Knotens v ist das Minimum der Spielwerte der Kinder von v.

- Spieler A besitzt genau dann einen gewinnenden Anfangszug, wenn die Wurzel den Spielwert Eins erhält.
- Lassen sich Spielbäume effizient auswerten?
  - Insbesondere, wieviele Blätter muss ein deterministischer Algorithmus im worst-case inspizieren?
  - Und wieviele Blätter muss ein randomisierter Algorithmus inspizieren?

Deterministische Strategien

Wir konzentrieren uns auf die Spielbäume T_k^t.

- T_k^t ist ein vollständig k-ärer Baum der Tiefe $2t$.
- T_k^t modelliert also Spiele der Spieldauer $2t$, wobei in jedem Zug genau k Zugmöglichkeiten bestehen.

Wieviele Blätter von T_k^1 muss ein deterministischer Algorithmus A im worst-case inspizieren?

- T_k^1 besteht aus der Wurzel w und den Kindern $w_1, \ldots, w_k$.
- Wir beantworten alle Anfragen von A an Blätter eines Kinds w_i mit Eins, solange bis A das letzte Blatt von w_i nachfragt.
  - Bisher haben wir das Schicksal von w_i offen gehalten.
  - Jetzt müssen wir das Schicksal von w offenhalten: Wir antworten mit dem Spielwert 0 für das letzte Blatt von w_i.

Der Induktionsschritt

- Wir haben einen beliebigen Algorithmus A gezwungen, alle Blätter von T_k^1 nachzufragen!
- Angenommen, wir können A auch zwingen, alle Blätter von T_k^t nachzufragen. Schaffen wir dies auch für T_k^{t+1}?

- Die MAX-Wurzel sei w mit ihren MIN-Kindern $w_1, \ldots, w_k$ und den MAX-Enkeln $w_{i,1}, \ldots, w_{i,k}$.

  Jeder MAX-Enkel $w_{i,j}$ ist die Wurzel eines Baums T_k^t.
- Für Nachfragen nach Blättern im Baum von $w_{i,j}$ wenden wir das Antwortschema für T_k^t an und halten das Schicksal von $w_{i,j}$ solange wie möglich offen.
  - Wenn das letzte Blatt von $w_{i,j}$ nachgefragt wird, zwingen wir den MAX-Knoten $w_{i,j}$ auf Eins, es sei denn $w_{i,j}$ ist der letzte offene MAX-Knoten von w_i: Unsere Antwort zwingt w_i auf Null.

Wir haben A gezwungen, alle Blätter abzufragen!

Und die Konsequenz?

- Sei T ein Spielbaum. T habe den Max-Knoten w als Wurzel.
- Was ist das Problem, wenn wir T rekursiv mit einem deterministischen Algorithmus auswerten?
  - Wir finden erst bei der Evaluation des letzten Min-Kindes von w heraus, dass A einen gewinnenden Zug hat!
  - Wenn wir die Kinder aber in zufälliger Reihenfolge evaluieren, dann finden wir den Gewinnzug früher:

    Bei k Kindern in erwarteter „Zeit“ $k/2$.
  - Aber dann halbieren wir doch nur die Anzahl abgefragter Blätter?!?

Und wo, bitte schön, ist der große Vorteil?
In der rekursiven Anwendung!

Eine Las Vegas Auswertung

Der Spielbaum wird rekursiv,

beginnend mit der Wurzel $v = w$,

ausgewertet.

(1) Bestimme ein Kind v_i von v zufällig und werte den Teilbaum mit Wurzel v_i rekursiv aus.

Falls v ein Max-Knoten ist:

- Wenn v_i den Spielwert 1 hat, dann brich ab: Wir haben gerade einen Gewinnzug gefunden.
- Wenn v_i den Spielwert 0 hat, dann wiederhole die zufällige Kinderwahl solange, bis

  ein Gewinnzug gefunden ist, bzw. bis festgestellt werden muss, dass es keinen Gewinnzug gibt.

Falls v ein Min-Knoten ist, gehe analog vor.

(2) Der Spielwert der Wurzel wird ausgegeben.

Wieviele Blätter in T_2^1 werden abgefragt?

- Fall 1: Der Spielwert der Max-Wurzel ist Null.
  - Beide MIN-Kinder besitzen den Spielwert Null.
  - Die erwartete Anzahl nachgefragter Blätter eines MIN-Kinds ist höchstens
$$\frac{1}{2} \cdot 1 + \frac{1}{2} \cdot 2 = \frac{3}{2}.$$
  - Die erwartete Anzahl nachgefragter Blätter ist $\frac{3}{2} + \frac{3}{2} = 3$.
- Fall 2: Der Spielwert der Max-Wurzel ist Eins.
  - Mindestens ein Min-Kind v besitzt den Spielwert 1.
  - Mit Wahrscheinlichkeit $1/2$ wird v zuerst gewählt und die erwartete Anzahl nachgefragter Blätter ist höchstens
$$\frac{1}{2} \cdot 2 + \frac{1}{2} \cdot 4 = 3.$$

Wieviele Blätter werden in T_2^t nachgefragt?

Wir zeigen durch Induktion, dass die erwartete Anzahl nachgefragter Blätter in T_2^t höchstens 3^t beträgt.
(Deterministische Strategien fragen im worst-case 4^t Blätter nach).

- Fall 1: Die Max-Wurzel von T_2^{t+1} hat den Spielwert 0.
  - Beide Min-Kinder besitzen den Spielwert Null.
  - Nach Induktionsannahme genügen

$$2 \cdot \left[\frac{1}{2} \cdot 3^t + \frac{1}{2} \cdot (3^t + 3^t) \right] = 3^{t+1}$$

  nachgefragte Blätter.
- Fall 2: Die Max-Wurzel von T_2^{t+1} hat den Spielwert 1.
  - Ein Min-Kind v hat den Spielwert 1.
  - v wird mit Wahrscheinichkeit $1/2$ zuerst gezogen und deshalb ist

$$\frac{1}{2} \cdot (3^t + 3^t) + \frac{1}{2} \cdot 4 \cdot 3^t = 3^{t+1}$$

  die erwartete Anzahl nachgefragter Blätter.

Skip-Listen

Wir entwerfen ein besonders einfaches Wörterbuch, also eine Datenstruktur, die die Operationen

$$\text{Insert}, \text{Delete} \quad \text{und} \quad \text{Lookup}$$

unterstützt.

- Die erwartete Laufzeit ist logarithmisch.
- Keinerlei Wartungsarbeiten, wie die Rebalancierung von Knoten, sind auszuführen.

- Skip-Listen speichern ihre Daten in einem (nicht-binären) Baum ab.
- Wie wird ein Schlüssel x eingefügt?
  - Würfle eine Tiefe k aus und füge x in maximaler Tiefe „bis Tiefe k“ einschließlich ein.
  - Hoffentlich ist die Tiefe des Baums logarithmisch und der durchschnittliche Knotengrad nicht zu groß.

Abspeicherung der Schlüssel in Schichten

Angenommen, eine Skip-Liste S speichert die Schlüssel $x_1 < x_2 < \cdots < x_{n-1} < x_n$.

- Die Schlüsselmenge wird in Schichten

$$L_0 = \{-\infty, \infty\} \subseteq L_1 \subseteq \cdots L_{t-1} \subseteq L_t = \{-\infty, x_1, \ldots, x_n, \infty\}$$

  zerlegt.
- Jede Schicht $L_k = \{-\infty, x_{i_1}, \ldots, x_{i_s}, \infty\}$ besteht aus den Suchintervallen

$$[-\infty, x_{i_1}], [x_{i_1}, x_{i_2}], \ldots, [x_{i_{s-1}}, x_{i_s}], [x_{i_s}, \infty]$$

  - Der geordnete Baum besitzt genau $s + 1$ Knoten der Tiefe k, die der Reihe nach mit den Suchintervallen von L_k markiert sind.
  - Knoten gleicher Tiefe werden „gefädelt“: Der Knoten mit Intervall $[x_{i_{m-1}}, x_{i_m}]$ zeigt auf den Knoten mit Intervall $[x_{i_m}, x_{i_{m+1}}]$.

Abspeicherung der Schlüssel in Schichten II

Wie werden Kanten eingesetzt?

Setze die Kante (v, w) genau dann ein, wenn

- $\text{Tiefe}(w) = \text{Tiefe}(v) + 1$ und
- Knoten v mit dem Intervall I, Knoten w mit dem Intervall J markiert ist, wobei $J \subseteq I$ gilt.

Achtung:

Wir erzeugen unter Umständen Knoten mit großem Grad!

Wenn ein Knoten viele Kinder hat, dann muss die Suche möglicherweise alle Kinder inspizieren!

Lookup für Schlüssel x

- Wir beginnen an der Wurzel.
- Traversiere die Suchintervalle der Wurzel-Kinder mit Hilfe der Fädelungszeiger.
  - Wenn x Randpunkt eines Intervalls ist, dann haben wir den Schlüssel gefunden!
  - Ansonsten setzten wir unsere Suche mit dem eindeutig bestimmten Intervall fort, das den Schlüssel x enthält.

Die von Lookup(x) benötigte Zeit stimmt mit der Länge des Suchpfads, inklusive Fädelungszeiger, überein.

Delete für Schlüssel x

- Führe zuerst die Operation Lookup(x) aus.
- Wenn x in Tiefe k zum ersten Mal auftaucht, dann werden wir dort zwei Intervalle $[y, x]$ und $[x, z]$ finden.
- Wir verschmelzen die beiden Intervalle in das Intervall $[y, z]$.
  - Der Schlüssel x ist jetzt entfernt.
  - Wir müssen den Verschmelzungsprozess für Tiefe $k + 1, \ldots, t$ fortsetzen:

    Verschmelze die Suchintervalle $[y', x]$ und $[x, z']$ gleicher Tiefe.

Die Laufzeit ist proportional zur Summe aus der Länge des Suchpfads, inklusive Fädelungszeiger, und der Tiefe des Baums.

Insert für Schlüssel x

1. Die höchste Schicht, in die x einzufügen ist, wird zufällig bestimmt:

   Wir würfeln mit Erfolgswahrscheinlichkeit $\frac{1}{2}$ solange, bis wir im $k$ten Versuch den ersten Erfolg erhalten.

2. Wenn $k \leq t$,
   - dann fügen wir x mit Hilfe von Lookup(x) in die Schicht L_{t-k+1} ein.
   - Wenn wir im Intervall $[y, z]$ „landen“, dann zerlege $[y, z]$ in die Intervalle $[y, x]$ und $[x, z]$. Dieser Zerlegungsprozess ist jetzt für die entsprechenden Knoten der Tiefen $k+1, \ldots, t$ fortsetzen.

2'. Wenn $k > t$,
   - dann erhöhen wir die Tiefe von t auf k: Ersetze die Wurzel durch eine Kette der Länge $k - t$.
   - Füge x in das Kind der Wurzel mit Hilfe der beiden Intervalle $[-\infty, x]$ und $[x, \infty]$ ein und
   - setze den Zerlegungsprozess für alle nachfolgenden Schichten fort.

Wie schnell sind Skip-Listen?

Die Laufzeit einer Insert-Operation ist proportional zur Summe aus

- Länge des Suchpfads, inklusive Fädelungszeiger, und
- Tiefe des Baums.

- Wie stark wächst die Tiefe des Baums mit der Anzahl eingefügter Schlüssel an?
- Wie verhält sich die Länge des Suchpfads, wenn wir auch Fädelungszeiger mitzählen?

Wir beginnen mit einer Analyse der Tiefe des Baums.

Die Tiefe des Baums

Nur Insert-Operationen verändern die Tiefe des Baums.

- Angenommen, wir führen n Insert-Operationen aus.
- Sei X_i die Zufallsvariable, die die zufällig bestimmte Schicht bei der $i$ten Insert-Operation angibt.
- Wir erhalten

$$\text{prob}[\ X_i > T\ ] \leq 2^{-T}$$

und als Konsequenz

$$\text{prob}[\ \max_i X_i > T\ ] \leq \frac{n}{2^T}.$$

- Für $T = \alpha \cdot \log_2 n$ folgt

$$\text{prob}[\ \text{Der Baum hat mehr als } \alpha \cdot \log_2 n \text{ Schichten}\ ] \leq \frac{1}{n^{\alpha-1}}.$$

Wie groß ist der Knotengrad?

Sei v ein beliebiger Knoten im Baum.
Wieviele rechte Geschwisterknoten besitzt v?

- v besitze das Intervall $[y_0, y_1]$ und gehöre zur Schicht L_k.
  $[y_1, y_2], \ldots, [y_r, y_{r+1}]$ seien die Intervalle der rechten Geschwister.
  - $y_1 y_2, \ldots, y_r$ landen alle in Schicht L_k
  - und der rechteste Schlüssel y_{r+1} gehört zur Schicht L_{k-1} oder einer höheren Schicht.
- Angenommen, v gehört zur Schicht L_t.

  Dann hat v genau r rechte Geschwisterknoten mit Wahrscheinlichkeit $2^{-(r+1)}$ und

$$E[r] = \sum_{i=1}^{\infty} 2^{-(i+1)} \cdot i = 1.$$

- Die erwartete Anzahl rechter Geschwisterknoten nimmt ab, wenn v zu einer höheren Schicht gehört: Es ist stets $E[r] \leq 1$.

Die erwartete Länge des Suchpfads

Ein Knoten hat im Erwartungsfall höchstens einen rechten Geschwisterknoten. Also ist die erwartete Kinderzahl höchstens zwei.

- Wir müssen die erwartete Länge $E[\ L\ ]$ eines Suchpfads bestimmen.
  - Es ist $E[\ L\ ] \leq 2 \cdot E[\ \text{Tiefe}\ ]$.
  - Wir wissen $\text{prob}[\ \text{Tiefe} \geq \alpha \cdot \log_2\ ] \leq n^{1-\alpha}$.
  - Mit Wahrscheinlichkeit höchstens $1/n$ ist die Tiefe größer als $2 \cdot \log_2 n$.
- Für n Knoten gilt also

$$E[L] \leq \frac{1}{n} \cdot n + (1 - \frac{1}{n}) \cdot 2 \cdot \log_2 n.$$

Auch die erwartete Länge des Suchpfads ist logarithmisch.

Zusammenfassung

Angenommen, n Insert-, Delete- und Lookup-Operationen sind auszuführen.

Dann schaffen wir dies in erwarteter Zeit $O(n \cdot \log_2 n)$.

Routing in Würfeln

Ein Prozessor-Netzwerk $G = (V, E)$ ist gegeben.

- Für eine Permutation π möchte jeder Prozessor $v \in V$ ein Paket P_v zu einem Prozessor $\pi(v)$ schicken.
- Das Paket P_v ist über einen Weg in G von v nach $\pi(v)$ zu verschicken.
- Unser Ziel ist der Versand aller Pakete in möglichst kurzer Zeit, wobei eine Kante ein Paket in einem Schritt transportieren kann.

Häufig benutzt man konservative Routing-Algorithmen:

- Ein Routing-Algorithmus heißt *konservativ*, wenn der Weg eines jeden Pakets P_v nur vom Empfänger $\pi(v)$ abhängt.
- Die anderen Pakete spielen also keine Rolle in der Berechnung des Wegs: Ist das nicht blau-äugig?

Katastrophe pur

Sei G ein Netzwerk von n Prozessoren, wobei jeder Prozessor in G höchstens d Nachbarn habe.

Dann gibt es für jeden konservativen, deterministischen Routing-Algorithmus A eine Permutation π, für die A mindestens $\Omega(\sqrt{\frac{n}{d}})$ Schritte benötigt.

- Ist das schlimm?
  - Ein gutes Prozessor-Netzwerk mit n Knoten hat Wege der Länge $O(\log_2 n)$ zwischen je zwei Knoten.
    - ⋆ Prinzipiell sollte ein paralleler Transport aller Pakete in Zeit $O(\log_2 n)$ gelingen.
    - ⋆ Tatsächlich werden aber $\Omega(\sqrt{\frac{n}{d}})$ Schritte benötigt!
- Ja, das ist schlimm, denn der effiziente parallele Nachrichtenaustausch zwischen Prozessoren ist eine Vorbedingung für effiziente parallele Programme.

Der d-dimensionale Würfel $W_d = (\{0,1\}^d, E_d)$

Eine Kante $\{u, v\}$ liegt genau dann in E_d, wenn sich u und v in genau einem Bit unterscheiden.

W_d hat 2^d Knoten und $\frac{d}{2} \cdot 2^d$ Kanten.

Bit-Fixing ist eine beliebte konservative Routing-Strategie, um ein Paket P_v vom Start v zum Ziel w zu transportieren:

- P_v wird zuerst über die Kante bewegt, der das linkeste, in v und w unterschiedliche Bit entspricht.
- Wiederhole solange, bis der Empfänger w erreicht wird.

Für $v = 1111$ und $w = 1000$ ist $(1111, 1011, 1001, 1000)$ der Bit-Fixing Weg von v nach w.

Und wo ist das Problem?

Eine schwierige Permutation π für Bit-Fixing

Wir setzen

$$\pi(x, y) = (y, x),$$

wobei x der Präfix der ersten $d/2$ Bits des Senders und y der Suffix der letzten $d/2$ Bits ist.

- Was passiert?
  - Die $2^{d/2}$ Pakete der Sender $(x, 0)$ überschwemmen das Nadelöhr $(0, 0)$.
  - Der Knoten $(0, 0)$ kann aber höchstens d Pakete in einem Schritt weiter transportieren und mindestens $2^{d/2}/d$ Schritte benötigt das Nadelöhr allein, um die Paketschwemme aufzulösen.
- Leider ist $\pi(x, y) = (y, x)$ eine in praktischen Anwendungen häufig auftretende Permutation.

Für eine passable Leistung müssen wir entweder nicht-konservative Routing-Algorithmen betrachten oder Determinismus aufgeben.

Von Frankfurt nach Köln (über Accra)

Wir arbeiten weiter mit Bit-Fixing, einer konservativen Routing Strategie, aber wählen eine zufällige Variante.

- Experimentelle Beobachtungen zeigen, dass Bit-Fixing für die meisten Permutationen gut ist.
- Unsere Zufallsstrategie für W_d:

Phase 1: Versand an einen zufälligen Empfänger.

Jedes Packet P_v wählt einen Empfänger $z_v \in \{0,1\}^d$ zufällig. Das Paket P_v folgt sodann dem Bit-Fixing Weg von v nach z_v.

Phase 2: Versand an den ursprünglichen Empfänger.

Das Paket P_v folgt dem Bit-Fixing Weg von z_v nach $\pi(v)$.

Analyse

Wir benötigen Queues, um Warteschlangen in einem Knoten aufzulösen. Wie sollen die Queues arbeiten?

Völlig egal, solange ein wartendes Paket transportiert wird, wenn mindestens ein Paket wartet.

- Wenn zwei Pakete P_v und P_w sich zu irgendeinem Zeitpunkt der ersten Phase trennen, dann werden sie sich in der ersten Phase nicht wieder treffen.
- Das Paket P_v laufe über den Weg $(e_1, \ldots, e_k)$.

  Dann ist die Wartezeit von P_v höchstens so groß wie die Anzahl der Pakete, die irgendwann während der ersten Phase über eine Kante in $\{e_1, \ldots, e_k\}$ laufen.

  Ist das wirklich offensichtlich?

Analyse II

Wir betrachten die Zufallsvariablen

$$H_{v,w} = \begin{cases} 1 & P_v \text{ und } P_w \text{ laufen über eine gemeinsame Kante,} \\ 0 & \text{sonst.} \end{cases}$$

- Nach unserer Vorüberlegung ist die Gesamtwartezeit von P_v während der ersten Phase durch $\sum_w H_{v,w}$ beschränkt.
- Wir müssen das erwartete Verhalten von $\sum_w H_{v,w}$ verstehen.
  - W_e sei die Anzahl der Bit-Fixing Wege, die über Kante e laufen.
  - Wenn Paket P_v den Weg $(e_1, \ldots, e_k)$ einschlägt, dann ist

$$E[\sum_w H_{v,w}] \leq E[\sum_{i=1}^{k} W_{e_i}] = k \cdot E[W_{e_1}].$$

Die erwartete Belastung einer Kante

Was ist $E[\ W_e\ ]$, also die erwartete Anzahl von Paketen, die über Kante e laufen?

- Die erwartete Weglänge eines jeden Pakets ist $\frac{d}{2}$, denn Start und Ziel werden sich im Erwartungsfall in genau der Hälfte aller Bitpositionen unterscheiden.
- Die $d \cdot 2^{d-1}$ Kanten des Würfels tragen alle dieselbe Belastung und deshalb ist

$$E[\ W_e\ ] = \frac{\frac{d}{2} \cdot 2^d}{d \cdot 2^{d-1}} = 1.$$

- Bei einer Weglänge von k ist die (zusätzliche) Wartezeit also höchstens k.

Die erwartete Wartezeit ist höchstens die erwartete Weglänge, also höchstens $d/2$.

Analyse III

Die erwartete Dauer der ersten Phase für ein Paket ist also höchstens d. Aber sind auch alle Pakete hochwahrscheinlich schnell fertig?

Wir wissen:

- Die erwartete Wartezeit für Paket P_v ist höchstens $E[\sum_w H_{v,w}] \leq d/2$.
- Sind starke Abweichungen vom Erwartungswert zu befürchten?
- Die (binären) Zufallsvariablen $H_{v,w}$ sind unabhängig!

Wir können die Ungleichungen von Chernoff anwenden!

Die Chernoff-Schranken

$$\text{prob}[\sum_{w} H_{v,w} \geq (1+\beta) \cdot \frac{d}{2}] \leq e^{-\beta^2 \cdot \frac{d}{2}/3}.$$

- $\beta = 3$: Die Wahrscheinlichkeit einer Gesamtwartezeit für P_v von mindestens $2d$ in der ersten Phase ist höchstens $e^{-3 \cdot d/2}$.
- Wir besitzen insgesamt 2^d Pakete:

  Die Wahrscheinlichkeit, dass *irgendein* Paket in der ersten Phase länger als $2d$ Schritte wartet, ist höchstens $2^d \cdot e^{-3 \cdot d/2} \leq e^{-d/2}$.

Und die zweite Phase?

Fazit

Wenn wir in der ersten Phase eine zufällige Permutation ρ ausgewählt haben, dann müssen wir in der zweiten Phase die Permutation $\pi \circ \rho^{-1}$ „ausführen“.

- Aber wenn ρ zufällig ist, dann auch $\pi \circ \rho^{-1}$.
- Die Analyse der zweiten Phase ist identisch mit der Analyse der ersten Phase.

Mit Wahrscheinlichkeit mindestens $1 - 2 \cdot e^{-d/2}$:

(a) Jedes Paket wird in jeder der beiden Phasen nach einer Wartezeit von höchstens $2d$ Schritten sein Ziel erreichen.

(b) Nach einer Wartezeit von höchstens $4d$ Schritten ist jedes Paket am Ziel.

Der kleine Satz von Fermat

Sei G eine endliche Gruppe.

(a) Ist H eine Untergruppe von G, dann ist $|H|$ ein Teiler von $|G|$.

(b) Der kleine Satz von Fermat: Für jedes Element $g \in G$ ist

$$g^{|G|} = 1.$$

- $H = \{g^k \mid k \in \mathbb{N}\}$ ist eine Untergruppe von G.
- Nach Teil (a): Die Ordnung $|H|$ der Untergruppe ist ein Teiler der Gruppenordnung $|G|$: Es gilt $|G| = |H| \cdot T$.
- Also folgt

$$g^{|G|} = g^{|H| \cdot T} = (g^{|H|})^T = 1,$$

denn $g^{|H|} = 1$ gilt.

Der kleine Satz von Fermat folgt also aus Teil (a).

Das RSA Kryptosystem

p, q seien zwei große Primzahlen, die nur Alice bekannt seien.

- Alice gibt das Produkt $N = p \cdot q$ ebenso bekannt wie einen Kodierungsexponenten e.
- Bob verschlüsselt seine Nachricht x gemäß $y = x^e \bmod N$.
- Sei $\phi(N) = |\{1 \leq x \leq N-1 \| \ (x, N) = 1\}|$ die Anzahl der primen Restklassen modulo N.

  Die Berechnung von $\phi(N)$ ist für Außenstehende sehr schwer, während Alice natürlich weiß, dass $\phi(N) = (p-1) \cdot (q-1)$ gilt.

  Alice berechnet das multiplikative Inverse d von e modulo $\phi(N)$ und dekodiert durch

$$y^d \equiv (x^e)^d \equiv x^{e \cdot d \bmod \phi(N)} \equiv x \mod N.$$

Große Primzahlen

- Wie kann sich Alice „ihre geheimen“ großen Primzahlen p, q beschaffen?
- Was heißt groß? Primzahlen mit mehreren zehntausend Bits!

  In 2007 wurde eine „schwierige“ 1039-Bit lange Zahl faktorisiert.

- Der Primzahlsatz:
  - Betrachte die Primzahlfunktion

$$\pi(x) = |\{p \leq x \mid p \text{ ist eine Primzahl } \}.$$

  - Der Primzahlsatz zeigt

$$\frac{x}{\pi(x)} = \ln(x) + o(\ln(x)).$$

- Um eine Primzahl mit B Bits zufällig auszuwürfeln, müssen im Mittel $O(B)$ Zufallszahlen mit B Bits gezogen werden.

Wir brauchen sehr schnelle Primzahltests!

Der Fermat Test für eine Zahl N

- Wenn N prim ist, dann besitzt die Gruppe $\mathbb{Z}_N^*$ der prímen Restklassen, also

$$\mathbb{Z}_N^* = \{x \mod N \mid x \in \mathbb{N},\ \text{ggT}(x, N) = 1\ \}$$

genau $\phi(N) = N - 1$ Elemente.

  - Ziehe eine zufällige Restklasse a modulo N.
  - Wenn $a^{N-1} \not\equiv 1 \mod N$, dann ist N nicht prim.

- Die Hoffnung:
  - Wenn N keine Primzahl ist, dann gibt es sehr viele Restklassen a mit $a^{N-1} \not\equiv 1 \mod p$.
- Die Enttäuschung:
  - Es gibt unendlich viele zusammengesetzte Zahlen N, die Carmichael Zahlen, mit $a^{N-1} \equiv 1 \mod p$ für **alle** Restklassen a.
  - Beispiele von Carmichael Zahlen sind $561, 1105, 1729, 2465, 2821$.

Der erweiterte Fermat Test

Wenn N eine Primzahl ist, dann ist $\mathbb{Z}_N$ ein Körper.

- Aber die quadratische Gleichung $x^2 = 1$ besitzt über jedem Körper nur die Lösungen $x \in \{-1, 1\}$.
- Wenn N eine Primzahl ist, dann ist $N - 1$ durch zwei teilbar und es muss

$$a^{(N-1)/2} \equiv -1 \bmod N \quad \text{oder} \quad a^{(N-1)/2} \equiv 1 \bmod N$$

für jede Restklasse a gelten.

- **Wenn** $N - 1$ durch vier teilbar ist **und** $a^{(N-1)/2} \equiv 1 \bmod N$ gilt? Dann folgt

$$a^{(N-1)/4} \equiv -1 \bmod N \quad \text{oder} \quad a^{(N-1)/4} \equiv 1 \bmod N.$$

Der erweiterte Fermat Test II

Es gelte $N - 1 = 2^k \cdot m$ und k sei maximal.

Überprüfe, ob $a^{N-1} \equiv 1 \mod N$ und für jedes r, $0 \leq r < k$, ob

$$a^{(N-1)/2^r} \equiv 1 \bmod N \quad \Rightarrow \quad a^{(N-1)/2^{r+1}} \equiv -1, 1 \bmod N$$

gilt.

- Alle Primzahlen bestehen den erweiterten Fermat Test.
- Der Miller-Rabin Primzahltest nutzt aus, dass zusammengesetzte Zahlen N den erweiterten Fermat Test für viele Restklassen a nicht bestehen.

Der Miller-Rabin Primzahltest

(1) N sei die Eingabezahl und k sei maximal mit $N - 1 = 2^k \cdot m$.

(2) Wähle eine Restklasse $a \in \mathbb{Z}_N$ mit $a \not\equiv 0 \bmod N$ zufällig. Klassifiziere N als zusammengesetzt, wenn

- N gerade ist oder
- $\mathrm{ggT}(a, N) \neq 1$ oder
- $N = a^b$ für natürliche Zahlen $a, b \geq 2$ oder
- N den erweiterten Fermat Test für Restklasse a nicht besteht.

(4) Wenn N nicht als zusammengesetzt klassifiziert wurde, dann klassifiziere N als Primzahl.

- Jede Primzahl wird als Primzahl klassifiziert.
- Mit welcher Wahrscheinlichkeit wird eine zusammengesetzte Zahl als „nicht prim“ erkannt?

Analyse: Die Zahl h

Wähle h maximal, so dass 2^{h+1} die Ordnung mindestens einer Restklasse b modulo N teilt.

Angenommen, $2^j \cdot m'$ ist die Ordnung einer Restklasse a **und** es gilt $a^{N-1} \equiv 1 \mod N$.

- Da $a^{N-1} \equiv 1 \mod N$, ist die Ordnung von a ein Teiler von $N - 1 = 2^k \cdot m$. Also ist m' ein Teiler von m.
- Nach Wahl von h ist
  - $j \leq h + 1$
  - und die Ordnung der Restklasse a ist ein Teiler von $2^{h+1} \cdot m$.

Wenn $a^{N-1} \equiv 1 \mod N$, dann ist $a^{2^{h+1} \cdot m} \equiv 1 \mod N$.

Gute und schlechte Restklassen a

Die Menge

$$G = \{a \in \mathbb{Z}_N^* \mid a^{2^h \cdot m} \equiv -1, 1 \bmod N\}$$

ist eine Gruppe. (Warum?)

Angenommen, a liegt nicht in G:

- Dann ist $a^{2^h \cdot m} \not\equiv -1, 1 \bmod N$.
- Wenn $a^{N-1} \not\equiv 1 \bmod N$, dann wird N als zusammengesetzt klassifiziert. Sonst gilt $a^{N-1} \equiv 1 \bmod N$.

  Dann ist aber $a^{2^{h+1} \cdot m} \equiv 1 \bmod N$ und N wird ebenfalls als zusammengesetzt klassifiziert, denn $a \notin G$.

Wenn Miller-Rabin eine Restklasse $a \notin G$ auswürfelt, dann wird N als zusammengesetzt klassifiziert!

Wieviele schlechte Restklassen gibt es?

Die Gruppe $G = \{a \in \mathbb{Z}_N^* \mid a^{2^h \cdot m} \equiv -1, 1 \bmod N\}$ ist die Menge der schlechten Restklassen, da N für jedes $a \notin G$ enttarnt wird.

- Wenn G eine echte Untergruppe von $\mathbb{Z}_N^*$ ist, dann ist $|G|$ ein echter Teiler von $|\mathbb{Z}_N^*|$:

  Mindestens die Hälfte aller Restklassen ist dann gut und die Fehlerwahrscheinlichkeit von Miller-Rabin ist höchstens $1/2$.
- Sei b eine Restklasse, so dass 2^{h+1} die Ordnung von b teilt.

Wir konstruieren eine Restklasse $a \notin G$ mit Hilfe von b.

Eine Restklasse a, die nicht in G liegt.

Es gelte $N = x \cdot y$ mit $\mathrm{ggT}(x, y) = 1$.

- Die Ordnung von b (modulo N) ist das kleinste gemeinsame Vielfache der Ordnung von b modulo x und b modulo y.

  O.B.d.A ist 2^{h+1} also ein Teiler der Ordnung von b modulo x.
- Sei a die Restklasse modulo N mit

$$a \equiv b \bmod x \quad \text{und} \quad a \equiv 1 \bmod y.$$

- Es ist

$$a^{2^h \cdot m} \equiv b^{2^h \cdot m} \not\equiv 1 \bmod x.$$

  - Nach Konstruktion $a \equiv b \bmod x$ und deshalb $a^{2^h \cdot m} \equiv b^{2^h \cdot m} \bmod x$.
  - 2^{h+1} teilt die Ordnung von b modulo x. Wenn $b^{2^h \cdot m} \equiv 1 \bmod x$, dann wird $2^h \cdot m$ von der Ordnung von b modulo x geteilt.

Eine Restklasse a, die nicht in G liegt. II

Was wissen wir?

1. $a \equiv b \mod x$ und $a \equiv 1 \mod y$.
2. $a^{2^h \cdot m} \equiv b^{2^h \cdot m} \not\equiv 1 \mod x$.

- Da $a \equiv 1 \mod y$ folgt

$$a^{2^h \cdot m} \equiv 1 \mod y.$$

- Also ist $a^{2^h \cdot m} \not\equiv 1 \mod x$ und $a^{2^h \cdot m} \equiv 1 \mod y$.
- Dann kann aber nicht $a^{2^h \cdot m} \equiv -1, 1 \mod N$ gelten.

Die Restklasse a liegt nicht in G.

Fazit

- Auswertung von Spielbäumen:

  Evaluiere ein zufällig gewähltes Kind.
- Skip-Listen:

  Füge einen Schlüssel „irgendwo“ ein, wobei tendenziell unten im Baum eingefügt wird.
- Routing im Würfel:

  Bit-Fixing funktioniert fast immer.
  Also würfle Zwischenziele zufällig aus.
- Der Miller-Rabin Primzahltest:

  Der erweiterte Fermat Test funktioniert für mindestens 50% aller Restklassen.

Die ersten drei Algorithmen sind Las Vegas Algorithmen, der Miller-Rabin Primzahltest ist ein Algorithmus mit einseitigem Fehler

(kein Fehler, wenn die Ausgabe “nicht prim” ist).

Fingerprinting

Sei U ein Universum von Schlüsseln.

Für eine beliebige Funktion $h : U \to \{0, \ldots, N\}$ ist $h(x)$ der Fingerabdruck von x.

- Im Regelfall ist $N << |U|$
  - und zwangsläufig muss $h(x) = h(y)$ für verschiedene Schlüssel x und y gelten.
  - Natürlich gilt $h(x) = h(y)$ wann immer x und y identisch sind.
- Ein Fingerabdruck bewahrt einige Eigenschaft des Originals, verliert andere.

Anwendungen von Fingerabdrücken

- Hashing,
- Datenstromalgorithmen,
- Gleichheitstests ...

Jobinterview bei IBM

? Welche Datenstruktur verwenden Sie für die Implementierung eines Wörterbuchs?

! Wenn die Antwort „Hashing“ nicht gegeben wurde, war das Interview bald vorbei.

- Hashing hat aber trotzdem ein großes Problem:

  Es gibt für jede Hashfunktion eine Folge von Schlüsseln, so dass alle Schlüssel auf dieselbe Zelle gehasht werden!
- Hashing sollte für jede Folge von Schlüsseln effizient durchführbar sein. Aber wie schaffen wir das?
  - Wähle nicht eine Hashfunktion h, sondern eine Klasse $\mathcal{H}$ von Hashfunktionen.
  - Zu Begin der Berechnung wähle eine Hashfunktion $h \in \mathcal{H}$ zufällig aus **und** verwende h für den Rest der Berechnung.

Universelles Hashing

- Eine einzelne Hashfunktion wird an einer Folge von Schlüsseln scheitern.
+ Eine gute Klasse $\mathcal{H}$ von Hashfunktion sollte für **jede** Schlüsselfolge hochwahrscheinlich funktionieren.
? Aber was sind gute Klassen $\mathcal{H}$?

Sei U die Menge aller möglichen Schlüssel.

Eine Menge $\mathcal{H} \subseteq \{h \mid h : U \to \{0, ..., m-1\}\}$ ist c-universell, falls für alle $x, y \in U$ mit $x \neq y$

$$|\{h \in \mathcal{H} \mid h(x) = h(y)\}| \leq c \cdot \frac{|\mathcal{H}|}{m}$$

gilt.

c-Universalität

Wenn $\mathcal{H}$ c-universell ist, dann gilt

$$\frac{|\{h \in \mathcal{H} \mid h(x) = h(y)\}|}{|\mathcal{H}|} \leq \frac{c}{m}.$$

- Es gibt somit keine zwei Schlüssel x, y, die mit Wahrscheinlichkeit größer als $\frac{c}{m}$ auf die gleiche Zelle abgebildet werden.
- Gibt es c-universelle Klassen von Hashfunktionen für kleine Werte von c?
  - Die Klasse $\mathcal{H}$ aller Funktionen von U auf $\{0, \ldots, m-1\}$ liefert sogar $c = 1$.
  - Und wie, bitte schön, sollen wir eine zufällig gewählte, **beliebige** Hashfunktion auswerten?

Eine gute Klasse von Hashfunktionen

Es sei $U = \{0, 1, 2, ..., p-1\}$ für eine Primzahl p.

Setze

$$\mathcal{H} = \{h_{a,b} \mid 0 \leq a, b < p, \quad h_{a,b}(x) = ((ax + b) \bmod p) \bmod m\}.$$

- Eine Hashfunktion $h_{a,b}(x) = ((ax + b) \mod p) \mod m$
  - permutiert das Universum mit Hilfe von $(ax + b) \mod p$ zufällig
  - und hasht dann das Ergebnis in eine Restklasse modulo m.
- Die Annahme, dass das Universum aus genau „Primzahl vielen“ Schlüsseln besteht ist doch sehr fragwürdig!
  - Keine große Geschichte: Vergrößere einfach das tatsächliche Universum auf Primzahlgröße p!

Für welches c ist $\mathcal{H}$ c-universell?

Wie klein ist c?

Wir fixieren $x, y \in U$. Wann gilt $h_{a,b}(x) = h_{a,b}(y)$?

- Dann gibt es $q \in \{0, ..., m-1\}$ und $r, s \in \{0, ..., \lceil \frac{p}{m} \rceil - 1\}$ mit

$$\begin{aligned} ax &+ b = q + r \cdot m \mod p \\ ay &+ b = q + s \cdot m \mod p. \end{aligned}$$

- Dieses Gleichungssystem mit den Unbekannten a und b ist eindeutig lösbar. Warum?
- Zu jedem Vektor (q, r, s) gibt es also genau ein Paar (a, b) mit $h_{a,b}(x) = h_{a,b}(y)$.
- Wieviele Vektoren (q, r, s) gibt es? $m \cdot \left(\lceil \frac{p}{m} \rceil\right)^2$ viele!

Wie klein ist c? II

Unsere Klasse $\mathcal{H}$ ist c-universell mit $c = (\lceil \frac{p}{m} \rceil \ / \ \frac{p}{m})^2$.

Es ist $|\mathcal{H}| = p^2$ und deshalb

$$\begin{aligned} |\{h_{a,b} \in \mathcal{H} \ | \ h_{a,b}(x) = h_{a,b}(y)\}| &\leq m \cdot \left(\lceil \frac{p}{m} \rceil\right)^2 \\ &= \left(\lceil \frac{p}{m} \rceil / \frac{p}{m}\right)^2 \frac{p^2}{m} = c \cdot \frac{|\mathcal{H}|}{m}. \end{aligned}$$

Und wie klein ist c?

$$c = (\lceil \frac{p}{m} \rceil \ / \ \frac{p}{m})^2 \leq ((\frac{p}{m} + 1)/\frac{p}{m})^2 = (1 + \frac{m}{p})^2.$$

Wenn das Universum sehr viel größer als die Größe der Hashtabelle ist, dann ist unsere Hashklasse fast perfekt!

Die erwartete Laufzeit für universelles Hashing

Sei $\mathcal{H}$ eine beliebige c-universelle Klasse von Hashfunktionen

Für eine beliebige Folge von n Insert-, Delete- und Lookup-Operationen: Bestimme die erwartete Laufzeit E_n der n-ten Operation.

- Sei S die Menge der vor Ausführung der n-ten Operation präsenten Elemente aus U.
- Es ist $E_n = 1 + K_n$, wobei K_n die erwartete Anzahl der Kollisionen für das Argument x der $n$ten Operation bezeichnet. Also

$$\begin{aligned} E_n &= 1 + \frac{1}{|\mathcal{H}|} \cdot \sum_{h \in \mathcal{H}} |\{y \in S \mid h(x) = h(y)\}| \\ &= 1 + \frac{1}{|\mathcal{H}|} \cdot \sum_{y \in S} |\{h \in \mathcal{H} \mid h(x) = h(y)\}| \\ &\leq 1 + \frac{1}{|\mathcal{H}|} \cdot \sum_{y \in S} c \cdot \frac{|\mathcal{H}|}{m}. \end{aligned}$$

Die erwartete Laufzeit für universelles Hashing II

Es ist

$$E_n \leq 1 + \frac{1}{|\mathcal{H}|} \cdot \sum_{y \in S} c \cdot \frac{|\mathcal{H}|}{m}.$$

- Also ist $E_n \leq 1 + c \cdot \frac{n-1}{m}$.
- Wie groß ist die erwartete Laufzeit E aller n Operationen?

$$\begin{aligned} E &= E_1 + \ldots + E_n \leq \sum_{i=1}^{n} \left(1 + c \cdot \frac{i-1}{m}\right) \\ &= n + \frac{c}{m} \cdot \sum_{i=1}^{n} (i-1) = n + \frac{c}{m} \cdot \frac{n \cdot (n-1)}{2} \\ &\leq n \cdot (1 + \frac{c}{2} \cdot \frac{n}{m}). \end{aligned}$$

Universelles Hashing funktioniert!

Sei $\mathcal{H}$ eine c-universelle Klasse von Hashfunktionen.

Die erwartete Laufzeit für eine beliebige Folge von n Operationen ist höchstens

$$n\left(1 + \frac{c}{2} \cdot \frac{n}{m}\right)$$

- Was können wir bestenfalls erwarten?
  - Die bis zu n eingefügten Schlüssel werden geichmäßig über die m Zellen der Hashtabelle verteilt.
  - Da idealerweise jede Zelle aus m/n Schlüsseln besteht, wäre die Laufzeit

    $$\Theta(n \cdot (1 + \frac{n}{m}))$$

    optimal.
- Und das schafft universelles Hashing fast!

Ein Gleichheitstest

Gegeben sind zwei Polynome p und q als „**Black-Box**“:

Die Black-Box für p (bzw. q) bestimmt für Eingabe x die Ausgabe $p(x)$ (bzw. $q(x)$).

Wir sollen entscheiden, ob $p \neq q$ ist.

- p und q sind Polynome in n Veränderlichen.
- Wir nehmen an, dass wir den maximalen Grad d von p und q kennen, wobei der Grad eines Polynoms

$$p(x_1, \ldots, x_n) = \sum_{(i_1,\ldots,i_n) \in \mathbb{N}_0^n} c_{(i_1,\ldots,i_n)} \cdot x_1^{i_1} \cdots x_n^{i_n}$$

in n Variablen $x_1, \ldots, x_n$ durch

$$\text{grad}(p) := \max\{i_1 + \cdots + i_n \mid c_{(i_1,\ldots,i_n)} \neq 0\}$$

definiert ist.

Der Algorithmus

0. Die Polynome p und q in n Veränderlichen sind durch jeweils eine Black-Box gegeben. Der Grad beider Polynome sei durch d nach oben beschränkt.

   Der Algorithmus soll genau dann akzeptieren, wenn $p \not\equiv q$.

1. Sei S eine beliebige Menge von $2d$ Elementen des Grundkörpers.
2. Wähle k Vektoren $x^{(1)}, \ldots, x^{(k)} \in S^n$ zufällig gemäß der Gleichverteilung.

   $p(x^{(1)}), \ldots, p(x^{(k)})$ ist der Fingerabdruck von p,
   $q(x^{(1)}), \ldots, q(x^{(k)})$ ist der Fingerabdruck von q.
3. Akzeptiere, wenn $p\left(x^{(i)}\right) \neq q\left(x^{(i)}\right)$ für mindestens ein i und verwirf sonst.

\+ Wenn $p = q$, dann wird stets verworfen.

? Mit welcher Wahrscheinlichkeit wird verworfen, wenn $p \neq q$?

Sei K ein Körper und sei $S \subseteq K$ eine endliche Menge.

Wenn $x \in S^n$ gemäß der Gleichverteilung gewählt wird, dann gilt

$$\text{prob}[\, r(x) = 0 \,] \leq \frac{d}{|S|}$$

für jedes vom Nullpolynom **verschiedene** Polynom r vom Grad d.

- Beweis durch Induktion über die Anzahl n der Variablen.
- Wenn $n = 1$, dann ist r ein einstelliges Polynom.
  - Das Polynom r ist vom Grad d und verschwindet deshalb auf höchstens d der $|S|$ Elemente von S.
- Wir müssen den Induktionsschritt von n auf $n+1$ durchführen.
  - Ziehe die Variable x_1 heraus.
  - Dann ist

$$r(x) = \sum_{i=0}^{d^*} x_1^i \cdot r_i(x_2, \ldots, x_{n+1}).$$

für Polynome r_i vom Grad höchstens $d - i$.

Der Induktionsschritt

Es ist $r(x) = \sum_{i=0}^{d^*} x_1^i \cdot r_i(x_2, \ldots, x_{n+1})$.

- Nach Induktionsannahme ist $r_{d^*}(x_2, \ldots, x_{n+1}) = 0$ mit Wahrscheinlichkeit höchstens $(d - d^*)/|S|$.
- Wenn andererseits $r_{d^*}(x_2, \ldots, x_{n+1}) \neq 0$ ist, dann betrachte das einstellige Polynom

$$R(x_1) = \sum_{i=0}^{d^*} x_1^i \cdot r_i(x_2, \ldots, x_{n+1})$$

vom Grad d^*.

  - Da $r_{d^*}(x_2, \ldots, x_{n+1}) \neq 0$, stimmt R nicht mit dem Nullpolynom überein.
  - Deshalb ist $R(x_1) = 0$ mit Wahrscheinlichkeit höchstens $d^*/|S|$.

Der Induktionsschritt II

Es ist

$$\begin{aligned} \text{prob}[\, r(x) = 0 \,] &\leq \text{prob}[\, R_1(x) = 0 \mid r_{d^*} \neq 0 \,] + \text{prob}[\, r_{d^*} = 0 \,] \\ &\leq \frac{d^*}{|S|} + \frac{d - d^*}{|S|} = \frac{d}{|S|}. \end{aligned}$$

Wenn $S \subseteq K$ und wenn wir einen Vektor $x \in S^n$ zufällig auswürfeln, dann ist

$$\text{prob}[\, r(x) = 0 \,] \leq d/|S|,$$

falls das Polynom $r \not\equiv 0$ vom Grad d ist.

Wie gut ist der Algorithmus?

Setze $r = p - q$. Dann gilt

$$\text{prob}[r(x) = 0] \leq \frac{d}{|S|}.$$

- Wähle eine beliebige Teilmenge S von $2 \cdot d$ Körperelementen.
- Wenn fälscherlicherweise verworfen wird, dann
  - ist $r = p - q$ vom Nullpolynom verschieden,
  - aber k-maliges zufälliges Ziehen von Vektoren führt stets auf Nullstellen.
- Wir erhalten höchstens eine Wahrscheinlichkeit von $(d/|S|)^k = 2^{-k}$ für das k-malige Ziehen einer Nullstelle.

Unser Algorithmus besitzt einen einseitigen Fehler von höchstens 2^{-k}.

Stichproben

1. Ziehe eine Stichprobe,
2. werte die Stichprobe aus und
3. extrapoliere.

- Ein sehr erfolgreiches Paradigma, das wir im Kapitel über Markoff-Ketten im Detail untersuchen.
- Hier untersuchen wir zuerst das Closest Pair Problem:
  - n Punkte $p_1, \ldots, p_n \in \mathbb{R}^2$ sind gegeben,
  - bestimme ein Paar (p_i, p_j) nächstliegender Punkte.

Eine Lösung in quadratischer Zeit ist trivial: Bestimme alle paarweisen Abstände.

Geht es schneller?

Der Gitter-Ansatz

Der Minimalabstand, also

der kleinste Abstand zwischen zwei Punkten,

sei δ.

- Angenommen, wir haben ein Gitter Γ_μ mit quadratischen Zellen der Seitenlänge $\mu \geq \delta$ über den $\mathbb{R}^2$ gelegt.
  - Die Punkte eines nächstliegenden Paares liegen in derselben oder benachbarten Zellen des Gitters Γ_μ.
  - Eine Zelle von Γ_μ enthält höchstens $(2 \cdot \frac{\mu}{\delta})^2$ Punkte.
- Die Bestimmung eines nächstliegenden Paars gelingt schnell, wenn wir δ scharf approximiert haben.
  - Wir müssen nur Zellen (und ihre Nachbarn) mit mindestens einem Punkt betrachten.
  - Nur wenige Punkte in einer Zelle.

Eine erste Approximation von δ

Sei P_0 die Menge der n gegebenen Punkte.

- Wähle einen Punkt $p \in P_0$ zufällig und bestimme

$$\delta_0 = \min_{q \in P_0,\, p \neq q} ||p - q||$$

  in Zeit proportional zu $|P_0| = n$.
- δ_0 wird im Allgemeinen eine schlechte Approximation von δ sein.
  - Wir bestimmen das Gitter $\Gamma_{\delta_0/3}$ und
  - entfernen alle Punkte aus P_0, die keinen weiteren Punkt aus P_0 in ihrer oder einer benachbarten Zelle besitzen.
  - Bleiben keine Punkte übrig, dann ist $\delta_0/3 \leq \delta \leq \delta_0$, und wir haben eine gute Approximation von δ gefunden.
  - Und wenn Punkte übrig bleiben?

Der Algorithmus

Wiederhole, solange bis $P_i = \emptyset$:

1. Wähle einen Punkt $p \in P_i$ zufällig. Bestimme

$$\delta_i = \min_{q \in P_i,\, p \neq q} ||p - q||.$$

   Die Laufzeit ist proportional zur Größe von P_i.
2. Setze $Q = P_i$ und entferne alle „isolierten“ Punkte aus Q, also alle Punkte in Q, die keinen weiteren Punkt aus P_i in ihrer oder in einer benachbarten Zelle von $\Gamma_{\delta_i/3}$ besitzen.
   Achtung: Wie bestimmt man die Menge der isolierten Punkte in Linearzeit?
3. Setze $P_{i+1} = Q$ und $i = i + 1$.

Und wenn $P_i = \emptyset$?

Dann ist $\delta_i/3 \leq \delta \leq \delta_i$.

- Also gilt insbesondere $\delta \leq \delta_i \leq 3 \cdot \delta$.
- Also befinden sich höchstens $(2 \cdot \frac{\delta_i}{\delta})^2 = (2 \cdot 3)^2 = 36$ Punkte in einer Zelle von Γ_{δ_i}.
- Der Abstand zwischen je zwei Punkten $p, q \in P_{i-1}$, die sich in derselben oder benachbarten Zellen von Γ_{δ_i} befinden, kann also in Zeit $O(|P_{i-1}|)$ bestimmt werden.

Aber auch: P_{j+1} kann schnell bestimmt werden, wenn P_j bekannt ist!

Wie schnell ist unser Algorithmus?

+ Die Berechnung von P_{j+1} gelingt in Zeit $O(|P_j|)$.
+ Wenn $P_j = \emptyset$, dann kann das nächstliegende Punktepaar in Zeit proportional zu $|P_{j-1}| = O(|P_0|)$ bestimmt werden.
? Wie groß ist $\sum_j |P_j|$?

- $\delta(q) = \min_{r \in P_j,\, r \neq q} ||r - q||$ ist der Abstand von $q \in P_j$ zu einem nächstliegenden Punkt in P_j.
- Wenn $\mu \leq \delta(q)$, dann ist q im Gitter $\Gamma_{\mu/3}$ isoliert:

  Der Abstand zwischen Punkten aus benachbarten Zellen von $\Gamma_{\mu/3}$ ist höchstens $\sqrt{8} \cdot \mu/3 < \mu$.

Wenn wir einen Punkt $p \in P_j$ zufällig ziehen:

? Wieviele Punkte $q \in P_j$ werden zu isolierten Punkten?
? Für wieviele Punkte $q \in P_j$ gilt also $\delta(p) \leq \delta(q)$?

Wieviele Punkte werden zu isolierten Punkten?

Für $P_j = \{q_1, \ldots q_m\}$ gelte o.B.d.A.

$$\delta(q_1) \leq \cdots \leq \delta(q_m).$$

- Wieviele Punkte aus P_j sind im Gitter $\Gamma_{\delta(q_k)/3}$ isoliert?
  Mindestens die $m - k + 1$ Punkte $q_k, \ldots, q_m$.
- Wählen wir einen Punkt $p \in P_j$ zufällig, dann ist die erwartetet Anzahl isolierter Punkte mindestens

$$\sum_{k=1}^{m} \frac{m-k+1}{m} = \frac{m \cdot (m+1)}{2 \cdot m} = \frac{m+1}{2}.$$

Die erwartete Anzahl isolierter Punkte in P_j ist $\geq |P_j|/2$.

Fazit

Die Laufzeit ist proportional zu $\sum_j |P_j|$.

Wir zeigen induktiv, dass

$$E[\,|P_j|\,] \leq \frac{n}{2^j}$$

gilt, wobei $n = |P_0|$. Beachte

$$\begin{aligned} E[\,|P_{j+1}|\,] &= \sum_{m=0}^{\infty} \text{prob}[\,|P_j| = m\,] \cdot E[\,|P_{j+1}| \mid |P_j| = m\,] \\ &\leq \sum_{m=0}^{\infty} \text{prob}[\,|P_j| = m\,] \cdot \frac{m-1}{2} \leq \frac{1}{2} E[\,|P_j|\,] \leq \frac{n}{2^{i+1}}. \end{aligned}$$

$\sum_j |P_j| = O(n)$: Unser Algorithmus läuft in erwartet linearer Zeit!

Datenströme

Eine Datenflut ist in Echtzeit zu bewältingen. Beispiele sind

- die Protokollierung von Telefonverbindungen und die damit verbundenen Reaktionen auf überlastete Leitungen,
- die Protokollierung eines Paketstroms durch einen Router im Internet
- oder die Datenanalyse in der Abwehr einer Denial-of-Service Attacke.

- Typischerweise ist die Länge des Datenstroms weitaus größer als der verfügbare Plattenplatz.

  Eine Abspeicherung des Datenstroms ist genau so unmöglich wie ein zweiter Datendurchlauf.
- Und dann, bitte schön, rechne zusätzlich noch in Echtzeit!

Die Anzahl verschiedener Schlüssel

Für einen Datenstrom $(x_n \mid n \in \mathbb{N})$ und für jeden Zeitpunkt n:
Bestimme die Anzahl verschiedener Schlüssel.

- Kein Problem, wenn die tatsächliche Anzahl verschiedener Schlüssel klein ist:

  Verwende Hashing!
- Was aber tun, wenn die tatsächliche Anzahl verschiedener Schlüssel extrem groß ist?

  Die Anzahl m verschiender Schlüssel kann nur festgestellt werden, wenn Speicher $\Omega(m)$ zur Verfügung steht!

Also, bestimme die Anzahl verschiedener Schlüssel approximativ.

Stichproben verschiedener Schlüssel: Unser Ansatz

- Berechne eine verlässliche Stichprobe verschiedener Schlüssel.
  - Weise jedem gesehenen Schlüssel eine zufällig berechnete Priorität zu.
  - Halte alle Schlüssel ab einer Mindestpriorität als Stichprobe fest
  - und beachte Schlüssel mit zu niedriger Priorität nicht mehr.
- Und wie sollen wir die Anzahl verschiedener Schlüssel verlässlich abschätzen, wenn wir einige Schlüssel nicht beachten?
  - Schließe von der Anzahl der verbleibenden verschiedenen Schlüssel auf die Gesamtzahl zurück!

Und warum kann das funktionieren?

Weil wir zufällige Prioritäten zuweisen!

Der Algorithmus

1. Wähle Parameter $A, B \in \{0, \ldots, p-1\}$ zufällig und arbeite mit der Hashfunktion $h_{A,B}(u) = ((A \cdot u + B) \mod p) \mod m$.
2. Die Priorität eines Schlüssels u ist

$$p(u) = \text{die Anzahl der führenden Nullen in der Binärdarstellung von } A \cdot u + B \mod p \,.$$

   Wir fordern $\text{prob}[\, p(u) \geq k \,] = 2^{-k}$.
3. Setze PRIORITÄT$=0$ und STICHPROBE$=\emptyset$. S sei die erlaubte Maximalgröße einer Stichprobe. Wiederhole:
   (a) Füge x_i zur Menge STICHPROBE hinzu, falls $p(x_i) \geq$ PRIORITÄT.
   (b) Solange STICHPROBE mehr als S Schlüssel besitzt, entferne alle Schlüssel x mit $p(x) =$ PRIORITÄT und erhöhe PRIORITÄT um 1.
4. Setze $p =$PRIORITÄT und gib die Schätzung $2^p \cdot |\text{STICHPROBE}|$ aus.

Das erwartete Verhalten

Es ist $\text{prob}[\ p(x) \geq k\ ] = 2^{-k}$. Also folgt für $k = \text{PRIORITÄT}$:

$$\begin{aligned} E[\ |\text{STICHPROBE}|\ ] &= \sum_{x \text{ ein Schlüssel}} \text{prob}[\ p(x) \geq k\ ] \\ &= 2^{-k} \cdot \text{Anzahl verschiedener Schlüssel.} \end{aligned}$$

- Unsere Schätzung ist gut, wenn große Abweichungen vom Erwartungswert $E[\ |\text{STICHPROBE}|\ ]$ nicht wahrscheinlich sind.
- Übungsaufgabe: Bestimme die Varianz $V[\ |\text{STICHPROBE}|\ ]$ und zeige, dass $V[\ |\text{STICHPROBE}|\ ] = O(E[\ |\text{STICHPROBE}|\ ])$.
- Wie wahrscheinlich sind große Abweichungen vom Erwartungswert?

  Wir wenden die Ungleichung von Tschebyscheff an.

Die Ungleichung von Tschebyscheff

Setze $E = E[\ |\text{STICHPROBE}|\ ]$. Die Tschebyscheff Ungleichung liefert

$$\text{prob}[\ |\ |\text{STICHPROBE}| - E| > \varepsilon \cdot E\ ] = O(\frac{E}{\varepsilon^2 \cdot E^2}) = O(\frac{1}{\varepsilon^2 \cdot E}).$$

- Sei V die Anzahl der verschiedenen Schlüssel. Dann ist $V = 2^p \cdot E$.
- Und deshalb

$$\begin{aligned} & \text{prob}[\ |\ 2^p \cdot |\text{STICHPROBE}| - V| > \varepsilon \cdot V\ ] \\ = \quad & \text{prob}[\ |\ 2^p \cdot |\text{STICHPROBE}| - 2^p \cdot E| > \varepsilon \cdot 2^p \cdot E\ ] = O(\frac{1}{\varepsilon^2 \cdot E}). \end{aligned}$$

Wie können wir die Verläßlichkeit boosten?

Führe mehrere unabhängige Zufallsexperimente durch und gib den Median aller Resultate aus.

Byzantine Agreement

Die Armee von Byzanz ist in mehrere Divisionen aufgeteilt. Jede Division wird von einem loyalen oder illoyalen General befehligt.

- Die Generäle kommunizieren über Boten, um sich entweder auf „Angriff“ oder auf „Rückzug“ einheitlich festzulegen.

  Die illoyalen Generäle versuchen allerdings dieses Ziel zu sabotieren.
- Haben die loyalen Generäle eine Chance?
  - Wenn alle loyalen Generäle dieselbe Vorgehensweise vorschlagen, kann diese dann beschlossen werden?
  - Besteht kein Einvernehmen, kann dann mindestens irgendeine gemeinsame Vorgehensweise beschlossen werden?

Die Anwendung

In einem Netzwerk von n Prozessoren sind t Prozessoren fehlerhaft und verhalten sich möglicherweise bösartig.

- Zu Anfang starten die Prozessoren mit jeweils einem Eingabebit.
- Jeder Prozessor kann sogar unterschiedliche Nachrichten an andere Prozessoren schicken.
- Unser Ziel ist ein Protokoll mit den folgenden Eigenschaften:
  (a) Die fehlerfreien Prozessoren haben sich am Ende der Berechnung auf ein Bit b geeinigt.
  (b) Wenn alle fehlerfreien Prozessoren dasselbe Eingabebit c besitzen, dann ist eine Einigung auf Bit c erforderlich.

Und wenn alle intakten Prozessoren ihr Bit kommunizieren und dann einen Mehrheitsentscheid durchführen?

Vorbereitung

Wir erlauben bis zu $t = n/8 - 1$ fehlerhafte Prozessoren.

- Wir führen mehrere Abstimmungsrunden mit unterschiedlich hohen Mehrheitsforderungen durch:

  Für $T = \frac{n}{8}$ benutzen wir die Schwellenwerte

  - ★ NIEDRIG$= \frac{n}{2} + T + 1$,
  - ★ Hoch$= \frac{n}{2} + 2T + 1$ und
  - ★ ENTSCHEIDUNG$= \frac{n}{2} + 3T + 1$.
- In jeder Abstimmungsrunde nehmen wir an, dass alle intakten Prozessoren auf dasselbe Zufallsbits zugreifen können.

  Wir fordern, dass Zufallsbits für verschiedene Runden unabhängig sind.

Anfänglich votiert jeder Prozessor für sein Eingabebit.

Der Algorithmus

Wiederhole, solange bis eine Entscheidung getroffen wird:

(1) Jeder Prozessor sendet sein Votum an alle anderen Prozessoren.

Bösartige Prozessoren können verschiedene oder gar keine Voten senden. Interpretiere Nichterhalt eines Votums als Votum für Bit 0.

(2) Jeder Prozessor empfängt die Voten seiner Kollegen.

- Erhält Prozessor i eine Mehrheit von Voten für Bit 1, dann wird $\text{Bit}_i = 1$ und ansonsten $\text{Bit}_i = 0$ gesetzt.
- Stimmen_i ist die Stimmenanzahl für das gewinnende Bit.

(3) Abhängig vom Wert des globalen Zufallsbits wählt jeder Prozessor den Schwellenwert $S \in \{ \text{NIEDRIG}, \text{HOCH} \}$.

(4) Wenn $\text{Stimmen}_i \geq S$, dann behält Prozessor i seine bisherige Wahl bei und setzt ansonsten $\text{Bit}_i = 0$.

(5) Wenn $\text{Stimmen}_i \geq \text{ENTSCHEIDUNG}$, dann legt sich Prozessor i **endgültig** auf seine bisherige Wahl fest.

Eine zentrale Beobachtung

Für je zwei intakte Prozessoren und für jede Runde: Die erhaltenen Stimmen für Bit 0 bzw. Bit 1 unterschieden sich um höchstens t.

Nur bösartige Prozessoren geben unterschiedliche Voten ab.

- Und wenn irgendwann alle intakten Prozessoren dasselbe Bit besitzen?
  - Die Stimmenzahl übertrifft den Schwellenwert ENTSCHEIDUNG.
  - Alle intakten Prozessoren legen sich sofort auf dasselbe Bit fest.
  - Die erste Anforderung für das Byzantine Agreement ist erfüllt.
- Wenn $\text{Stimmen}_i \geq \text{HOCH} = \frac{n}{2} + 2T + 1$ für *irgendeinen* intakten Prozessor,
  - dann ist $\text{Stimmen}_i \geq \text{NIEDRIG}$ für jeden intakten Prozessor i.
  - Mit Wahrscheinlichkeit $\frac{1}{2}$ wird NIEDRIG als Schwellenwert gewählt und alle fehlerfreien Prozessoren beginnen in diesem Fall die nächste Runde mit demselben Bit.
  - In diesem Fall erfolgt also Einigung in der nächsten Runde.

Erfolgt stets Einigung auf dasselbe Bit?

- Wenn $\text{Stimmen}_i < \text{HOCH} = \frac{n}{2} + 2T + 1$ für **alle** intakten Prozessor gilt, dann setzen die intakten Prozessoren mit Wahrscheinlichkeit $\frac{1}{2}$ ihr Bit auf Null.
  - Warum?
  - Der Schwellenwert HOCH wird mit Wahrscheinlichkeit $\frac{1}{2}$ gewählt.
- Und die Konsequenz:
  - Entweder übertrifft irgendein Prozessor HOCH und
    - ⋆ Einigung erfolgt in der nächsten Runde mit Wahrscheinlichkeit $1/2$
  - oder kein Prozessor erreicht HOCH
    - ⋆ und wiederum erfolgt Einigung in der nächsten Runde mit Wahrscheinlichkeit $1/2$.

Die intakten Prozessoren einigen sich nach einer erwarteten Zahl von höchstens zwei Runden auf ein gemeinsames Bit.

Warum waren wir erfolgreich?

Wir haben mit dem globalen Zufallsbit

gegen bösartige Prozessoren randomisiert.

- Die Stimmzahlen für die einzelnen Prozessoren unterscheiden sich um höchstens T.
- Wenn die bösartigen Prozessoren irgendwo eine große Mehrheit zulassen, dann droht Schwellenwert NIEDRIG in der nächsten Runde.
- Wird nirgendwo eine große Mehrheit erlaubt, dann droht Schwellenwert HOCH in der nächsten Runde.

Shared Memory

Parallel arbeitende Prozessoren greifen auf die Register eines gemeinsamen Speichers zu.

- Jedes Register ist von einer unbeschränkten Anzahl von Prozessoren lesbar.
- Sollten mehrere Prozessoren dasselbe Register beschreiben wollen, so “gewinnt” irgendeiner der beteiligten Prozessoren.

- In effizienten parallelen Algorithmen werden Prozessoren die überwiegende Zeit auf den ihnen zugewiesenen Aufgaben arbeiten.

  Die teure Kommunikation mit anderen Prozessoren muss auf ein absolutes Minimum reduziert werden.

- Wie erreicht man **ohne Kommunikation**, dass sich die Prozessoren in verschiedene Typen mit unterschiedlichen Zielen aufspalten?

Zusammenhangskomponenten

Sei $G = (V, E)$ ein ungerichteter Graph.

- $W \subseteq V$ ist genau dann zusammenhängend, wenn je zwei Knoten durch einen Weg verbunden sind.
- Eine zusammenhängende Knotenmenge $W \subseteq V$ ist genau dann eine Zusammenhangskomponente, wenn keine echte Obermenge von W zusammenhängend ist.

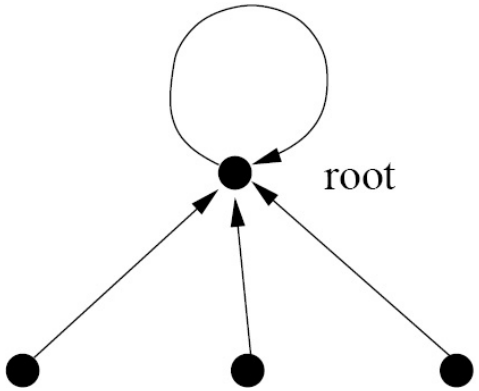
Bestimme alle Zusammenhangskomponenten von G blitzschnell.

- Je zwei Zusammenhangskomponenten sind knoten-disjunkt.
- Zusammenhangskomponenten können sequentiell in linearer Zeit $O(|V| + |E|)$ bestimmt werden:
  - Die Menge aller Knoten, die von irgendeinem Knoten v aus erreichbar sind, ist eine Zusammenhangskomponente.
  - Bestimme die Menge der erreichhbaren Knoten mit Tiefensuche.

Weder Tiefen- noch Breitensuche ist hochgradig parallelisierbar.

Sterne

- Wir repräsentieren Teilmengen einer Zusammenhangs-komponente durch Sterne:
  - genau ein Knoten ist als Wurzel ausgezeichnet,
  - alle Knoten, inklusive die Wurzel selbst, zeigen auf die Wurzel.



- Anfänglich müssen wir annehmen, dass alle Knoten ihre eigene Zusammenhangskomponente bilden.
- Warum Sterne? Verschmelze zwei Sterne durch Anhängen des eines Sterns unter den anderen:

  Alle Knoten des angehängten Sterns werden in einem parallelen Schritt auf die Wurzel des neuen Sterns gerichtet.

Symmetry Breaking

Wir können zwei Sterne S_1 und S_2 verschmelzen, wenn es eine Kante von einem Knoten in S_1 zu einem Knoten in S_2 gibt.

- Viele Kanten verbinden einen Stern S mit anderen Sternen: Welche Verschmelzungsschritte sollten wir ausführen?
  - Wenn wir S_1 und S_2, S_2 und S_3, S_3 und S_4 ... verschmelzen, dann ist unser Verfahren inhärent sequentiell.
  - Wir müssen lange Ketten von Verschmelzungsschritten vermeiden.
- Kettenlänge zwei nach Brechen der Symmetrie:
  - Jeder Stern wählt ein Geschlecht zufällig.
  - Jeder männliche Stern wählt einen weiblichen Stern aus, unter den er sich hängt.

Der Algorithmus

(0) Wir haben einen Prozessor pro Kante und pro Knoten.

(1) for $i = 1$ to n pardo Vater$[i] = i$. // Ein Wald aus Einzelbäumchen.

(2) while (mindestens eine Kante vebindet verschiedene Sterne) do

(a) Jede Wurzel wählt zufällig ein Geschlecht und vererbt das Geschlecht auf die Kinderknoten.

(b) Wenn eine Kante e einen weiblichen Stern w und einen männlichen Stern m verbindet, dann versucht der Prozessor von e, den Namen von w in das Register von m zu schreiben.

(c) Bei Schreibkonflikten wird irgendein weiblicher Stern $w(m)$ ausgewählt:

Die Wurzel von m wird auf die Wurzel von $w(m)$ gerichtet.

(d) for $i = 1$ to n pardo

Vater$[i] =$ Vater[Vater$[i]$];

// Die Stern-Eigenschaft ist wieder hergestellt.

Die Analyse I

- Korrektheit:
  - Sterne werden nur verschmolzen, wenn sie zu derselben Zusammenhangskomponente gehören.
  - Nenne eine Kante genau dann lebendig, wenn die Endpunkte zu verschiedenen Sternen gehören.

    Wenn es keine lebendigen Kanten gibt, dann haben wir alle Komponenten gefunden.
- Die Laufzeit:
  - Die Zeit pro Iteration ist konstant.
  - Wie viele Iterationen gibt es?

Die Analyse II

Wir nennen ein Stern S genau dann lebendig, wenn es eine Kante gibt, die ihn mit einem anderen Stern verbindet. Ansonsten verschwindet S.

- Ein Stern S verschwindet mit Wahrscheinlichkeit mindestens $\frac{1}{4}$. Warum?
  - Die Kante $\{u, v\}$ verbinde $u \in S$ mit einem anderen Stern T.
  - Mit Wahrscheinlichkeit $\frac{1}{4}$: S wird männlich, T weiblich und der Stern S verschwindet.
- Bestimme die Wahrscheinlichkeit p_w, dass der Stern von Knoten w nach $5 \cdot \log_2 n$ Iterationen nicht verschwindet. Dann

$$\begin{aligned} p_w &\leq \left(1 - \frac{1}{4}\right)^{5 \cdot \log_2 n} = \left(\frac{3}{4}\right)^{5 \cdot \log_2 n} = \left(\frac{243}{1024}\right)^{\log_2 n} \\ &\leq \left(\frac{1}{4}\right)^{\log_2 n} = 2^{-2 \cdot \log_2 n} = n^{-2}. \end{aligned}$$

Zusammenfassung

- Ein spezieller Knoten überlebt $5 \cdot \log_2 n$ Iterationen mit Wahrscheinlichkeit höchstens n^{-2}.
- Irgendein Knoten überlebt $5 \cdot \log_2 n$ Iterationen mit Wahrscheinlichkeit höchstens $n \cdot n^{-2} = n^{-1}$.

- $5 \cdot \log_2 n$ Iterationen der While-Schleife genügen mit Wahrscheinlichkeit mindestens $1 - 1/n$.
- Die erwartete Laufzeit ist durch $O(\log_2 n)$ beschränkt. Warum?
- $|V| + |E|$ Prozessoren werden benutzt.

Maximale unabhängige Mengen

- $I \subseteq V$ ist eine unabhängige Menge eines ungerichteten Graphen $G = (V, E)$, wenn keine zwei Knoten aus I durch eine Kante verbunden sind.
- Unabhängige Mengen sind „kollisions-frei“.
- I ist eine maximale unabhängige Menge, wenn I unabhängig ist und in in keiner größeren unabhängigen Menge enthalten ist.

Bestimme eine maximale unabhängige Menge.

- Die Bestimmung einer größten unabhängigen Menge ist schwierig, denn $\{(G, k) \mid G \text{ hat eine unabhängige Menge } I \text{ mit } |I| \geq k\}$ ist $\mathcal{NP}$-vollständig.
- Größte unabhängige Mengen entsprechen globalen Maxima, maximale unabhängige Mengen lokalen Maxima.

Die Parallelisierung einer einfachen For-Schleife

$I := \emptyset$.

for $v = 1$ to n do

  If (v ist nicht mit einem Knoten aus I verbunden) then

    $I = I \cup \{v\}$.

- I ist eine maximale unabhängige Menge.
- Können wir die For-Schleife parallelisieren?

  Nein! Man kann zeigen, dass die von der For-Schleife berechnete unabhängige Menge in aller Wahrscheinlichkeit nicht von superschnellen parallelen Algorithmen berechnet werden kann.
- Können wir zumindest irgendeine maximale unabhängige Menge wirklich schnell berechnen?

Der Ansatz

- Wir möchten in einem Schritt großen Fortschritt in der Berechnung einer maximalen unabhängigen Menge erreichen.

  Symmetry Breaking: Jeder Knoten v entscheidet zufällig, ob er in dem Vergrößerungschritt der unabhängigen Menge teilnehmen will.
- Und wenn zwei teilnahmeberechtigte Knoten durch eine Kante verbunden sind?

  Im schlimmsten Fall müssen wir beide Knoten disqualifizieren!
- Hoffentlich disqualifizieren wir nur wenige Knoten.

  Die Wahrscheinlichkeit, dass v teilnimmt, sollte mit wachsender Nachbarzahl $d(v)$ fallen.

Wie arbeiten wieder mit einem Prozessor pro Knoten und einem Prozessor pro Kante.

Der parallele Algorithmus

(1) Setze $I = \emptyset$.

(2) while $V \neq \emptyset$ do

(2a) for $v \in V$ pardo

if (v ist isoliert) then
$I = I \cup \{v\}$. $V = V \setminus \{v\}$.
else v wird mit Wahrscheinlichkeit $\frac{1}{2 \cdot d(v)}$ markiert.

(2b) for $\{u, v\} \in E$ pardo

if (u und v sind beide markiert) then
disqualifiziere den Knoten vom kleineren Grad, bzw.,
beide Knoten bei übereinstimmender Nachbarzahl.

(2c) // X sei die Menge aller markierten, nicht disqualifizierten Knoten.
Setze $I = I \cup X$; $V = V \setminus \big(X \cup \text{Nachbarn}(X)\big)$.
// I ist unabhängig, aber mgl. nicht maximal.
// Die Berechnung der $d(v)$ ist der teuerste Schritt pro Iteration.

Wieviel Fortschritt pro Iteration?

- Wir disqualifizieren den Knoten mit der geringeren Nachbarzahl.
  - Warum?
  - Die größere Anzahl der Nachbarn kann entfernt werden!
- Entfernen wir einen konstanten Prozentsatz aller Knoten in einer Iteration?
  - Betrachte den vollständigen bipartiten Graph mit t „linken“ und $n - t$ „rechten“ Knoten.
  - Für $t = n^{1/4}$ wird **irgendein** linker Knoten mit Wahrscheinlichkeit höchstens $n^{1/4} \cdot \frac{1}{2 \cdot (n - n^{1/4})} \approx n^{-3/4}/2$ gewählt.

    Linke Knoten nehmen also wahrscheinlich nicht teil.
  - Ein rechter Knoten wird mit Wahrscheinlichkeit $\frac{1}{2n^{1/4}}$ gewählt und ungefähr $n^{3/4}/2$ rechte Knoten werden deshalb teilnehmen.
  - Nur die $n^{1/4}$ linken Knoten sind Nachbarn teilnehmender Knoten.
- Nur die linken Knoten werden entfernt, aber

  wieviele Kanten überleben?

Fazit

- In jeder Iteration wird im Erwartungsfall ein konstanter Prozentsatz aller Kanten entfernt!

  Der Beweis ist im Skript.
- Die erwartete Anzahl der Iterationen ist also höchstens logarithmisch.
- Unser Algorithmus ist pfeilschnell, da die Bestimmung der Nachbarzahlen die teuerste Operation pro Iteration ist.

Die probabilistische Methode

Das Führen von Existenzbeweisen: Gibt es Objekte mit bestimmten Eigenschaften?

- Die explizite Konstruktion solcher Objekten ist häufig sehr schwierig.
- Ein häufiger Ansatz ist deshalb die Untersuchung, ob ein zufällig ausgewürfeltes Objekt die geforderten Eigenschaft hat.

  Wir geben einen Existenzbeweis, indem wir sogar zeigen, dass die meisten Objekte die geforderten Eigenschaften haben!

Die meisten Objekte sind „unstrukturiert“, weil zufällig, und haben deshalb manchmal überraschende Eigenschaften.

Eine Anwendung

Gibt es ungerichtete Graphen G mit n Knoten

sehr vielen Kanten, nämlich ungefähr $\binom{n}{n/2}/2$ Kanten,

so dass G nur kleine Cliquen und unabhängige Mengen besitzt?

- Wir betrachten Zufallsgraphen:
  Setze eine Kante $\{u, v\}$ mit Wahrscheinlichkeit $1/2$ ein.
- Für $X \subseteq \{1, \ldots, n\}$ mit $|X| = k$: Bestimme die Wahrscheinlichkeit p_X, dass X eine Clique oder eine unabhängige Menge ist.

$$p_X = 2 \cdot 2^{-\binom{k}{2}}.$$

- p_k sei die Wahrscheinlichkeit, dass ein Zufallsgraph eine Clique der Größe k oder eine unabhängige Menge der Größe k besitzt:

$$p_k \leq \binom{n}{k} \cdot 2 \cdot 2^{-\binom{k}{2}} < \frac{n^k}{k!} \cdot \frac{2 \cdot 2^{k/2}}{2^{k^2/2}}.$$

Eine Anwendung II

$$p_k < \frac{n^k}{k!} \cdot \frac{2 \cdot 2^{k/2}}{2^{k^2/2}}.$$

- Wenn $p_k < 1$, dann gibt es Graphen mit sehr vielen Kanten, aber nur sehr kleinen Cliquen und unabhängigen Mengen.
- Setze $k = 2 \cdot \log_2 n$:
  - Dann $n^k = 2^{k^2/2}$.
  - Es ist $2 \cdot 2^{k/2} < k!$ für $k \geq 3$.
  - Also ist $p_k < 1$ für $k \geq 3$.

Es gibt Graphen, die nur Cliquen oder unabhängige Mengen der Größe höchstens $2 \log_2 n - 1$ besitzen.