Hashing für Webanwendungen

- Konsistentes Hashing.

  CHORD: Ein Peer-to-Peer System für das File Sharing.
- **Min-Hashing**.

  Stelle Ähnlichkeit von Dokumenten schnell fest.
- **Bloom Filter**.

  Platz-effizientes Hashing.

CHORD:
Ein Peer-to-Peer System mit effizienter Suche

Peer-to-Peer Netze

In einem Peer-to-Peer-Netz sind alle beteiligten Rechner gleichberechtigt:

- Dienste können in Anspruch genommen
- oder zur Verfügung gestellt werden.

Wir nehmen hier an, dass das Peer-to-Peer Netz einen **verteilten Speicher** implementiert:

Dateien, verteilt über die Rechner des Netzes, sind zu speichern.

Welche Eigenschaften sollte ein solches Peer-to-Peer Netz haben?

- Rechner sollten mit geringem Aufwand dem Netz **beitreten** oder das Netz **verlassen** können.
- Unabhängig von dem sich ändernden Netz sollten die Dateien
  - verlässlich gespeichert werden und
  - für alle Rechner des Netzes schnell auffindbar sein.

- Rechner werden durch ihre IP-Adresse,
- Dateien durch ihre Datei-ID identifiziert.

$\mathcal{R}$ sei die Menge aller IP-Adressen und $\mathcal{D}$ die Menge aller Datei-ID's.

- CHORD arbeitet mit einer **konsistenten Hashfunktion** $h : \mathcal{R} \cup \mathcal{D} \rightarrow [0, 1]$, die sowohl auf IP-Adressen wie auch auf Datei-IDs anwendbar ist.
  - Konsistentes Hashing!
- Der Hashwert definiert eine Ordnung auf Dateien wie auch auf den Rechnern.
  - Die Ordnung definiert Nachfolger und Vorgänger für jeden Rechner
  - und weist Rechnern Dateien zu.

Konsistente Hashfunktionen

(a) Die Klasse konsistenter Hashfunktionen besteht aus allen Funktionen der Form $h : \mathcal{R} \cup \mathcal{D} \to [0, 1]$.

(b) Sei R eine Menge von Rechnern, repräsentiert durch ihre IP-Adressen.

- Wenn $h(r_1) < h(r_2)$ für zwei Rechner $r_1, r_2 \in R$ gilt und wenn es keinen Rechner $r_3 \in R$ mit $h(r_1) < h(r_3) < h(r_2)$ gibt, dann heisst r_2 der Nachfolger von r_1 und r_1 der Vorgänger von r_2 bezüglich h.
- Wir rechnen modulo 1, d.h. der Rechner mit größtem Hashwert ist Vorgänger des Rechners mit kleinstem Hashwert.

  Demgemäß sollte man sich das Intervall $[0, 1]$ als Kreis vorstellen.

(c) Es gelte $R \subseteq \mathcal{R}$ und $D \subseteq \mathcal{D}$.
Rechner $r \in R$ ist für die Dateien in der Menge

$$D' = \{d \in D \mid h(r') < h(d) < h(r)\}$$

zuständig, falls r der Nachfolger von $r' \in R$ ist.

Wir suchen Dateien mit Routing-Tabellen

Routing Tabellen

Die Dateienmenge D ist von den Rechnern in R abzuspeichern.

- Die Dateienmenge D ist von den Rechnern in R abzuspeichern. Ein Rechner speichert genau die Dateien, für die er zuständig ist.
- Wenn Rechner $r \in R$ auf Datei $d \in D$ zugreifen möchte, dann berechnet r den Hashwert $h(d)$ und konsultiert seine **Routing Tabelle**. Die Routing Tabelle von r enthält
  - die IP-Adresse seines Vorgängers und Nachfolgers.
  - Für jedes k die IP-Adresse $\mathbf{R_r(k)}$ des ersten Rechners mit Abstand **mindestens** 2^{-k}:
    - ⋆ Es ist $\mathbf{h(R_r(k)) \geqslant h(r) + 2^{-k}}$ und
    - ⋆ für jeden Rechner $t \in R$ zwischen r und $R_r(k)$ ist $\mathbf{h(t) < h(r) + 2^{-k}}$.

Auf der Suche nach Datei d

Rechner r

- bestimmt k_1, so dass $\mathbf{h(R_r(k_1)) \leqslant h(d) \leqslant h(R_r(k_1 - 1))}$.
- und sendet seine Anfrage nach d an den Rechner $\mathbf{s = R_r(k_1)}$.
  - Rechner s bestimmt seinerseits k_2 mit
    $\mathbf{h(R_s(k_2)) \leqslant h(d) \leqslant h(R_s(k_2 - 1))}$ und kontaktiert $\mathbf{R_s(k_2)}$.
  - Das Verfahren wird solange wiederholt bis der Vorgänger $\mathbf{v}$ des für die Datei d zuständigen Rechners erreicht ist.
  - Rechner v kontaktiert seinen Nachfolger, der d dann direkt an den ursprünglich anfragenden Rechner r verschickt.

Wie schnell ist dieses Verfahren?

Die Distanz zum Ziel v wird stets mindestens halbiert

r und alle auf der Suche nach d beteiligten Rechner kontaktieren jeweils Rechner r' mit $h(r') \leqslant h(v)$.

Wenn r' auf der Suche nach Datei d Rechner r'' kontaktiert, dann ist

$$\mathbf{h(v) - h(r'') \leqslant \frac{h(v) - h(r')}{2}}.$$

- Es gelte $h(r') + 2^{-k} \leqslant h(v) < h(r') + 2^{-(k-1)}$ und r' wird den Rechner r'' mit $h(r') + 2^{-k} \leqslant h(r'') \leqslant h(v)$ kontaktieren.
- Es ist $2^{-k} \leqslant h(r'') - h(r')$ und $h(v) - h(r') < 2^{-(k-1)}$. Also ist $h(v) - h(r'') = h(v) - h(r') + h(r') - h(r'') < 2^{-(k-1)} - 2^{-k} = 2^{-k}$.
- Als Konsequenz

$$\begin{aligned} h(v) - h(r') &= h(v) - h(r'') + h(r'') - h(r') \\ &\geqslant h(v) - h(r'') + 2^{-k} \geqslant 2(h(v) - h(r'')). \end{aligned}$$

Lastverteilung

- Sind die Rechner gleichmäßig auf dem Kreis verteilt?
- Wieviele Dateien erhält ein Rechner ungefähr?

Übungsaufgabe

Sei $|R| = n$ und $|D| = m$. Nach zufälliger Wahl einer konsistenten Hashfunktion gilt mit Wahrscheinlichkeit mindestens $1 - O(1/n)$:

(a) Es gibt eine Konstante α, so dass jeder Rechner für höchstens $\alpha \cdot \frac{m}{n} \cdot \log_2(n)$ Dateien verantwortlich ist.
   - Wenn $m >> n$, dann entfällt der Log-Faktor.

(b) Es gibt eine Konstante β, so dass jedes Interval der Länge $\frac{\beta \ln n}{n}$ mindestens einen Rechner und jedes Interval der Länge $\frac{\beta \ln m}{m}$ mindestens eine Datei enthält.

(c) Die Suche nach der Datei s ist nach höchstens $\alpha \cdot \log_2 n$ Schritten erfolgreich.

Beitritt von Rechnern (1/2)

Rechner r möchte dem System beitreten.

- r bittet einen beliebigen Rechner s des Systems, die Suche nach $h(r)$ durchzuführen.
- s teilt die IP-Adresse des zukünftigen Nachfolgers r' von r mit.
  - r teilt r' mit, dass er sein neuer Vorgänger ist und bittet um die Adresse des alten Vorgängers von r'.
  - Danach baut r seine Routing-Tabelle mit Hilfe seines Nachfolgers r' auf: Wenn r' auf die Anfrage nach dem fiktiven Hashwert $h(r) + 2^{-k}$ die IP-Adresse r_k erhält, dann ist $R_r(k) = r_k$.

Der Beitritt von r wird nicht global bekannt gemacht!

Beitritt von Rechnern (2/2)

Übungsaufgabe

- Zu einem gegebenen Zeitpunkt bestehe das Netz aus n Rechnern und alle Zeiger-Informationen (also Vorgänger, Nachfolger und Distanz 2^{-k} Rechner) seien korrekt.
- Wenn n Rechner beitreten und selbst wenn nur Vorgänger und Nachfolger richtig aktualisiert werden, dann gelingt eine Suche mit Wahrscheinlichkeit mindestens $1 - O(1/n)$ in Zeit $O(\log_2 n)$.

- In **periodischen Abständen** sollte aber jeder Rechner seine Routing Tabelle überprüfen:

  Sonst wird die Suche nach neuen Rechnern zu teuer.

Austritt von Rechnern

Der Rechner r meldet sich ab.

- r meldet sich bei seinem Vorgänger und Nachfolger ab.
- Der Nachfolger von r ist ab jetzt für die Dateien von r zuständig.

Auch der Austritt von r wird nicht global bekannt gemacht.

Ist das gefährlich?

- Wenn r während der Suche nach einer Datei herausfindet, dass Rechner $R_r(k)$ verschwunden ist:
  - r kontaktiert $R_r(k-1)$ mit dem imaginären Schlüsselwert $h(r) + 2^{-k}$ und
  - kann nach erhaltener Antwort die Dateisuche fortsetzen.
- Der teuren Reparatur während der Suche wird teilweise durch **periodische Aktualisierungen** der Routing Tabellen vorgebeugt.

Zusammenfassung

Die **zufällige Wahl der Hashfunktion** ist verantwortlich dafür, dass

- die Suche nach Dateien hochwahrscheinlich schnell ist,
- alle Rechner ungefähr für die selbe Anzahl von Dateien verantwortlich sind
- und periodische Aktualisierungen der Routing Tabellen für die Aufrechterhaltung einer schnellen Suche genügen.

Rechner, die austreten ohne sich ordnungsgemäß abzumelden:

- **Dateiverlust droht!**
- **Schaffe Abhilfe durch Datenreplikation.**

Min-Hashing
Auf der Suche nach ähnlichen Webseiten

Ähnliche Webseiten

- Ungefähr **20%** aller Web-Dokumente im Datenbestand einer Suchmaschine scheinen Duplikate oder **Fast-Duplikate** zu sein.
  - lokale Kopien beliebter Webseiten,
  - Mirroring
  - falsch arbeitende Crawler.
- Ein für Suchmaschinen äußerst ärgerliches Phänomen:
  - Anwender werden sich nicht durch sehr ähnliche Dokumente wühlen wollen.
  - Kostbare Ressourcen werden verschwendet.

Unser Ziel ist deshalb die schnelle Bestimmung aller Paare ähnlicher Dokumente in einer großen Dokumentenmenge.

Ähnlichkeitsmaße

Ähnlichkeitsmaße

Sei X eine Menge von Objekten.

- Eine Funktion $\mathbf{e} : \mathbf{X^2} \rightarrow [\mathbf{0}, \mathbf{1}]$ ist ein **Ähnlichkeitsmaß**, falls $e(a, b) = 1$ genau dann gilt, wenn $a = b$.
- Bestimmte Ähnlichkeitsmaße erlauben eine sehr schnelle Auswertung.
  - Sei Y eine Menge und $\mathcal{F}$ eine Menge von Funktionen $h : X \rightarrow Y$.
  - $\mathcal{F}$ ist eine Klasse **ähnlichkeitsbewahrender Hashfunktionen** für X und das Ähnlichkeitsmaß e, wenn

$$\text{prob}_{\mathbf{h} \in \mathcal{F}}[\, \mathbf{h}(\mathbf{x}) = \mathbf{h}(\mathbf{y}) \,] = \mathbf{e}(\mathbf{x}, \mathbf{y}).$$

  - Werte **wenige zufällige Hashfunktionen** aus, um die Ähnlichkeit zweier Elemente x und y approximativ zu bestimmen.

Ähnlichkeitsbewahrendes Hashing: Beispiele (1/4)

- **Hamming Ähnlichkeit**:
  - Für $X = \{0,1\}^n$ sei $H(x,y)$ der Hamming-Abstand von x und y.
  - Für das Ähnlichkeitsmaß $e(x,y) = 1 - H(x,y)/n$ wähle $\mathcal{F} = \{p_i \mid p_i(x) = x_i\}$ als Klasse von Hashfunktionen:

$$\mathbf{e(x,y) = 1 - H(x,y)/n = (n - H(x,y))/n = |\{i \mid x_i = y_i\}|/n.}$$

    $e(x,y)$ ist die Wahrscheinlichkeit, dass x und y in einer zufällig gezogenen Bitposition übereinstimmen.
- Der **Jaccard-Koeffizient** und **Min-Hashing**.
  - $X = \mathcal{P}(U)$ ist die Klasse aller Teilmengen von U.
  - Für zwei Teilmengen A und B ist

$$J(A,B) = \frac{|\,A \cap B\,|}{|\,A \cup B\,|}.$$

    der Jaccard-Koeffizient der Teilmengen A und B.
  - Welche Hashfunktionen sollte man wählen?

Ähnlichkeitsbewahrendes Hashing: Beispiele (2/4)

Min-Hashing:

- Für eine Permutation π von U setze $\min_\pi(A) = \min\{\pi(j) \mid j \in A\}$.
- Wähle $\mathcal{F} = \{\min_\pi \mid \pi\}$.

- $\mathcal{F}$ ist eine Klasse ähnlichkeitsbewahrender Hashfunktionen für den Jaccard-Koeffizienten. Warum?
- Es gelte $\min_\pi(A \cup B) = r$.
  - Es gibt $|A \cup B|$-viele Möglichkeiten für die Wahl von r, von denen
  - genau $|A \cap B|$-viele zur Gleichheit $\min_\pi(A) = \min_\pi(B)$ führen.

Ähnlichkeitsbewahrendes Hashing: Beispiele (3/4)

Ähnlichkeit von Vektoren:

- $\mathbf{X} = \{\mathbf{x} \in \mathbb{R}^{\mathbf{n}} \mid ||\mathbf{x}|| = \mathbf{1}\}$ ist der n-dimensionale Ball vom Radius 1.
- Für zwei Vektoren $x, y \in X$ sei $\phi(x, y)$ der Winkel zwischen x und y. Dann ist $e(x, y) = 1 - \phi(x, y)/\pi$ ein Ähnlichkeitsmaß.
- Für einen Vektor $r \in X$ definiere $\mathbf{h_r} : \mathbf{X} \to \{\mathbf{0}, \mathbf{1}\}$ durch

$$h_r(z) = \begin{cases} 1 & \langle z, r \rangle \geqslant 0 \\ 0 & \text{sonst.} \end{cases}$$

- Übungsaufgabe: $\mathcal{F} = \{h_r \mid r \in X\}$ ist eine Klasse ähnlichkeitsbewahrender Hashfunktionen für e.

Ähnlichkeitsbewahrendes Hashing: Beispiele (4/4)

Repräsentiere eine endliche Menge Z durch ihren Inzidenzvektor e_Z.

- Wir können die Ähnlichkeit zweier Mengen A, B über die Ähnlichkeit ihrer Vektoren e_A und e_B definieren:
  - Der Ähnlichkeitswert von A und B ist dann $1 - \phi(e_A, e_B)/\pi$.
- Beachte

$$\cos(\phi(e_A, e_B)) = \frac{\langle e_A, e_B \rangle}{||e_A|| \cdot ||e_B||} = \frac{|A \cap B|}{\sqrt{|A| \cdot |B|}}.$$

- Also ist $\mathbf{e(A, B)} = \frac{|\mathbf{A} \cap \mathbf{B}|}{\sqrt{|\mathbf{A}| \cdot |\mathbf{B}|}}$ ebenfalls ein Ähnlichkeitsmaß:

  Bestimme $e(A, B)$ durch Approximation des Winkels $\phi(e_A, e_B)$ mit Hilfe des ähnlichkeitsbewahrenden Hashings für Vektoren.

Ähnliche Dokumente

Wie finden Suchmaschinen heraus, dass Webseiten ähnlich sind?

- Repräsentiere ein Dokument **C** durch die Menge $\mathbf{M_C}$ seiner Shingles (Deutsch: Dachziegel):

  Für $C \equiv x_1 \cdots x_n$ und Shingle-Größe zehn ist $M_C = \{x_i \cdots x_{i+9} \mid i \leqslant n - 9\}$.

- Berechne $J(M_A, M_B)$ durch ähnlichkeitsbewahrendes Hashing.
  - Wähle k Permutationen $\pi_1, \ldots, \pi_k$ aller möglichen Shingles aus.

    In Anwendungen werden $k = 100$ lineare Permutationen $\pi_i(x) \equiv a_i \cdot x + b_i \mod N$ ausgewürfelt.

  - Repräsentiere den **Sketch** des Dokuments A durch

    $$\text{sketch}(A) = (\min_{\pi_i}(M_A) \mid 1 \leqslant i \leqslant k)$$

    Die erwartete Anzahl identischer Komponenten von sketch(A) und sketch(B) stimmt mit $k \cdot J(M_A, M_B)$ überein.

Das Verfahren

1. **Berechne Sketches** für alle Web-Dokumente A:

   Bestimme $\min_\pi(A)$ in Zeit proportional zur Länge von A.

2. **Vergleiche Sketches** mit konventionellem Hashing:
   - Füge Dokument A mit Argument $w = \min_{\pi_i}(M_A)$ in eine Hashtabelle ein und
   - vermerke einen Treffer für alle Dokumentenpaare mit Hashwert w.
     - ⋆ Die Laufzeit ist quadratisch in der Anzahl der Dokumente mit gleichem Hashwert.
     - ⋆ Die Anzahl dieser Dokumente ist meist gering: Ein einziger gemeinsamer Treffer für nicht-ähnliche Dokumente ist unwahrscheinlich.

3. **Baue einen Dokumentengraphen**.
   - Die Knoten sind die Dokumente,
     die Kanten (beschriftet mit der Trefferzahl) verbinden Dokumente mit einer Mindestanzahl gemeinsamer Treffer.
   - Die Zusammenhangskomponenten werden als Cluster ähnlicher Webseiten aufgefasst.

Das Market Basket Modell

Market Basket Modell

Evaluiere das Einkaufsverhalten in Warenhäusern:

Bestimme **korrelierte** Waren u und v selbst dann, wenn beide Waren nur selten gekauft werden.

- Für Ware u sei S_u die Menge der Warenkörbe mit Ware u.
  Bestimme alle Paarmengen $\{u, v\}$ von Waren u und v mit hinreichend großem Jaccard-Koeffizienten $J(S_u, S_v)$.
- Für zufällige Permutationen $\pi_1, \ldots, \pi_k$ der Warenkörbe definiere

$$\text{sketch}(u) = (\min_{\pi_i}(S_u) \mid 1 \leqslant i \leqslant k)$$

  als den Sketch der Ware u.
  - Man verwendet wieder lineare Permutationen.
  - Berechne Sketch-Vektoren in einem Datendurchlauf:
    Beginne mit Wert ∞ und aktualisiere den Sketch-Vektor der in einem neuen Warenkorb enthaltenen Waren.

Wie bestimmen wir korrelierte Waren?

Bestimmung korrelierter Waren

Bestimme alle Paare mit genügend großem Jaccard-Koeffizienten.

- Wenn $J(S_u, S_v) \geqslant t$, dann werden S_u und S_v in r fixierten Sketch-Komponenten mit Wahrscheinlichkeit mindestens t^r übereinstimmen.
- Wähle deshalb r größtmöglich, so dass $\frac{k}{r} \cdot t^r \approx 1$ und zerlege die k Sketch-Komponenten in $\frac{k}{r}$ Gruppen der Größe r.
  - Wenn $J(S_u, S_v) \geqslant t$, dann erwarten wir **Gleichheit** auf mindestens einer Gruppe.
  - Viele Kandidatenpaare können ausgeschlossen werden, die (hoffentlich) wenigen verbleibenden Kandidatenpaare können dann vollständig durchforstet werden.

Bloom Filter

Bloom Filter

- Sei U ein Universum.
- Konstuiere eine möglichst **platzeffiziente** Datenstruktur, die
  - **Lookup**-Anfragen beantwortet
  - und **Insert**- sowie **Remove**-Operationen ausführt.
- Wir verzichten darauf, die Elemente abzuspeichern,
  - erlauben deshalb, dass „**falsche positive**“ Antworten gegeben werden,
  - garantieren aber, dass jede **negative Antwort korrekt** ist.

Die Arbeitsweise eines Bloom Filters

Wir arbeiten mit einem Booleschen Array der Länge m und k Hash-Funktionen

$$h_1, \ldots, h_k : U \to \{1, \ldots, m\}.$$

- Anfänglich sind alle Zellen von B auf Null gesetzt.
- Ein Element $x \in U$ wird **eingefügt**, indem das Array B an den Stellen $\mathbf{h_1(x)}, \ldots, \mathbf{h_k(x)}$ auf Eins gesetzt wird.
- Wie wird **lookup**(x) ausgeführt?
  - Überprüfe B an den Stellen $h_1(x), \ldots, h_k(x)$.
  - Wenn B an **allen** Stellen den Wert 1 besitzt, dann wird die Vermutung "x **wurde eingefügt**" ausgegeben und sonst die definitive Ausgabe "x **nicht vorhanden**" getroffen.

Fehlerwahrscheinlichkeiten

Wenn die Operation **lookup**(x) ausgeführt wird:

- Eine negative Antwort ist immer richtig,
- eine positive Antwort ist möglicherweise falsch.

Wie groß ist die Wahrscheinlichkeit $\mathbf{q_{n,m,k}}$ einer falschen positiven Antwort, wenn

- eine Menge X von **n** Elementen eingefügt wurde,
- wir mit einem Booleschen Array der Größe **m** und
- **k** zufällig und unabhängig voneinander ausgewürfelten Hashfunktionen $h_1, \ldots, h_k$ arbeiten?

Wie groß ist $q_{n,m,k}$?

- Angenommen, wir haben die Elemente aus X in das Array B gehasht.
- Wie groß ist die Wahrscheinlichkeit $\mathbf{p_{n,m,k}}$, dass B an einer fixierten Position i eine Null speichert?

- Hashfunktion h_j trifft Position i mit Wahrscheinlichkeit $(\frac{m-1}{m})^n$ nicht.
- Für unabhängig voneinander ausgewürfelte Hashfunktionen folgt

$$p_{n,m,k} = (\frac{m-1}{m})^{kn} = (1 - \frac{1}{m})^{kn}.$$

- Es ist $e^{x/(1+x)} \leqslant 1 + x \leqslant e^x$ für $x > -1$ und deshalb

$$e^{-(kn/m)\cdot(1/(1-1/m))} \leqslant p_{n,m,k} = (1 - \frac{1}{m})^{kn} \leqslant e^{-kn/m}.$$

Wie groß ist $q_{n,m,k}$?

Wir betrachten die Anfrage **lookup**(y), wobei y **nicht** gespeichert sei.

- Die Wahrscheinlichkeit, dass B –für eine fixierte Zahl j– in Position $h_j(y)$ eine Eins speichert, ist
$$1 - (1 - \frac{1}{m})^{kn}.$$
- Sei $Q_{n,m,k} = [1 - (1 - \frac{1}{m})^{kn}]^k$. Ist $q_{n,m,k} \leqslant Q_{n,m,k}$?
  - Leider nicht, denn Unabhängigkeit liegt nicht vor:
    Zum Beispiel kann dieselbe Position in B mgl. zweimal erreicht werden, zum Beispiel wenn $h_1(y) = h_2(y)$ gilt.
  - Wenn n, m groß und k nicht zu groß ist, folgt aber $q_{n,m,k} \approx Q_{n,m,k}$.

$$q_{n,m,k} \approx \left(1 - (1 - \frac{1}{m})^{kn}\right)^k \approx (1 - e^{-kn/m})^k.$$

Eine optimale Wahl von k

- Es ist $q_{n,m,k} \approx (1 - e^{-kn/m})^k$.
- Wähle k so, dass
  die Fehlerwahrscheinlichkeit $q_{n,m,k}$ möglichst klein ist.

- Einerseits "sollte" k möglichst groß sein, da
  die Wahrscheinlichkeit wächst eine Null zu erwischen.
- Andererseits "sollte" k möglichst klein sein, da dann eine
  Einfüge-Operation nur wenige neue Einsen einträgt.
- Minimiere $(1 - e^{-kn/m})^k$, bzw $\ln(1 - e^{-kn/m})^k = k \cdot \ln(1 - e^{-kn/m})$.
  - Setze $p = e^{-kn/m}$.
  - Dann ist $k \cdot \ln(1 - e^{-kn/m}) = k \cdot \ln(1 - p) = -\frac{m}{n} \cdot \ln(p) \cdot \ln(1 - p)$.
  - $p = 1/2$ minimiert den Ausdruck: $p = e^{-kn/m} = \frac{1}{2}$ führt auf

$$k = \ln(2) \cdot m/n$$

mit der Fehlerwahrscheinlichkeit $q_{n,m,k} \approx (1 - p)^k = 2^{-k}$.

Zusammenfassung

- Für Mengen mit **n** Elementen,
- einem Array der Länge **m** und
- $\mathbf{k} = \ln(\mathbf{2}) \cdot \mathbf{m}/\mathbf{n}$ Hashfunktionen

ist die Wahrscheinlichkeit einer falschen positiven Antwort $\leqslant 2^{-k}$.

- Bloom-Filter erreichen eine Fehlerwahrscheinlichkeit von höchstens 2^{-k} für $k = \ln(2) \cdot m/n$ und damit für $m = \frac{k \cdot n}{\ln(2)}$.
  - Bloom Filter erreichen eine Fehlerwahrscheinlichkeit von höchstens $1/128$ mit einem Booleschen Array mit gut $10 \cdot n$ Einträgen,
  - bzw eine Fehlerwahrscheinlichkeit von höchstens $\frac{1}{n^r}$ mit Arrays der Größe $\Theta(r \cdot n \log_2 n)$.

Die Remove Operation und zählende Bloomfilter

Die Remove-Operation für Bloom-Filter

Wenn wir das Array B an den Positionen $h_1(x), \ldots, h_k(x)$ auf Null setzen und wenn $\{h_1(x), \ldots, h_k(x)\} \cap \{h_1(y), \ldots, h_k(y)\} \neq \emptyset$, dann wird auch y entfernt!

Verwende **zählende Bloom-Filter**:

- Statt einem Boole'schen Array benutze ein Array von Zählern.
- Elemente werden durch das Hochsetzen (bzw. Heruntersetzen) der entsprechenden Zähler eingefügt (bzw. entfernt).

Anwendungen

Bloom Filter: Anwendungen (1/2)

- **Verteiltes Web Caching:**
  - Proxies speichern Webseiten in ihren Caches und kooperieren, um zwischengespeicherte Webseiten abzurufen:

    Der zuständige Proxy ermittelt, ob ein Cache eines anderen Proxies die nachgefragte Seite enthält und bittet diesen Proxy um Lieferung.
  - Wie helfen Bloom Filter?
    - ★ Jeder Proxy speichert in einem Bloom-Filter ab, welche Seiten zwischengespeichert wurden und sendet seinen aktualisierten Bloom-Filter in regelmäßigen Abständen an alle Kollegen.
    - ★ Nur selten wird ein Proxy aufgrund des Phänomens falscher Positiver vergebens kontaktiert.

Bloom Filter: Anwendungen (2/2)

- **Ermittlung aggressiver Flüsse**:
  - Aggressive Flüsse sind Start-Ziel Verbindungen, die Datenstaus durch den Transfer von besonders vielen Paketen verschärfen oder sogar verursachen.
  - Ein Router kann die Menge aller Start-Ziel Paare, deren Pakete er befördert, durch einen zählenden Bloom-Filter repräsentieren.

    Für eine Start-Ziel Verbindung x inkrementiere die Zähler aller Positionen $h_1(x), \ldots, h_k(x)$.
  - Überschreitet der minimale der k Werte einen vorgegebenen Schwellenwert T, dann wird x als “möglicherweise” aggressiv gebrandmarkt.
  - Alle aggressiven Flüsse werden erkannt; allerdings können auch unschuldige Flüsse als aggressiv eingeschätzt werden.