Bloom Filter

Jambura Anna, Pürstinger Kathrin,
Schnappauf Franziska, Thiele Coco

23. Februar 2026

Zusammenfassung

passend auszufüllen

1 Funktionsweise und Mathematische Grundlagen

1.1 Aufbau

Der Bloomfilter besteht auf einem m-stelligen Bitarray, welches initial mit Nullen befüllt wird. Weiters werden k unabhängige Hashfunktionen definiert. Diese verwendet man um die Elemente der gewünschten Menge zu hashen. Abhängig von ihrem Hashwert werden die Elemente dann an der entsprechenden Position im Array eingefügt. Um also jedes Element erfolgreich einzufügen, muss die Hasfunktion $\bmod m$ angewandt werden. Somit erreicht man die Indizes 0 bis $m - 1$. [TRL12]

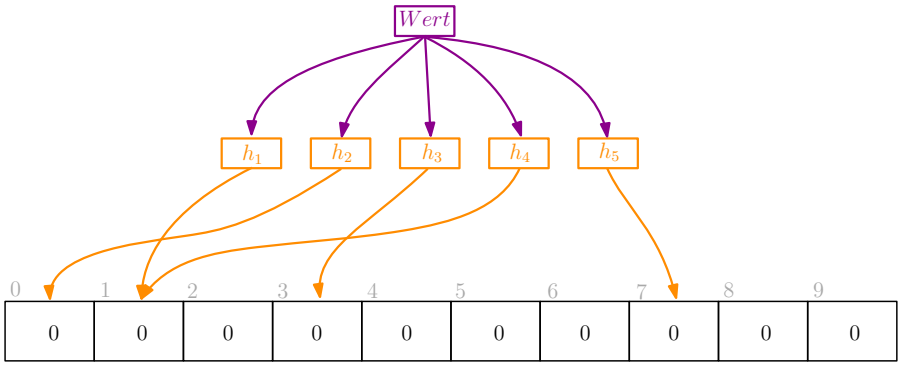

Abbildung 1: Visualisierung eines Bloomfilters

Da die Hashfunktionen keinem Sicherheitsstandard entsprechen, müssen keine kryptographischen Eigenschaften gelten. Kryptographische Eigenschaften bedeutet, minimale Eingabeänderungen müssen zu einer maximalen Änderung des Hashwerts führen. Die Eingabe darf nicht mittels der Hashfunktion wiederhergestellt werden können und zwei Eingaben haben fast unmöglich den selben Hashwert.
Für Bloomfilter verwendet man schnelle und einfache Hashfunktionen, da die Effizienz im Vordergrund steht.

1.2 Einfügen/Suchen

Einfügen
Eine Menge S wird nun wie folgt in einem Bloomfilter eingefügt:
Für jedes Element $x \in S$ werden die Hash Werte aller k Hash Funktionen berechnet. Nun wird an diesen Positionen im Array die 0 auf eine 1 gesetzt. Sollte an einer dieser Positionen bereits eine 1 stehen, wird dies ignoriert. Dieser Vorgang wird für alle n Elemente der Menge S wiederholt.

1.2.1 Beispiel Einfügen

Betrachte folgende Menge $S = \{2, 4, 9\}$ und einen Bloomfilter der Länge $m = 10$ mit $k = 3$ Hashfunktionen.
Als beispielhafte Hashfunktionen verwenden wir: $h_1(x) = x \bmod 10$ $h_2(x) = (2x+3) \bmod 10$ und $h_3(x) = (3x + 7) \bmod 10$.
Nun berechnen wir die Hashwerte für jedes Element der Menge S:

- Für $x = 2$:

$$h_1(2) = 2 \bmod 10 = 2$$
$$h_2(2) = (2 \cdot 2 + 3) \bmod 10 = 7$$
$$h_3(2) = (3 \cdot 2 + 7) \bmod 10 = 3$$

- Für $x = 4$:

$$h_1(4) = 4 \bmod 10 = 4$$
$$h_2(4) = (2 \cdot 4 + 3) \bmod 10 = 1$$
$$h_3(4) = (3 \cdot 4 + 7) \bmod 10 = 9$$

- Für $x = 9$:

$$h_1(9) = 9 \bmod 10 = 9$$
$$h_2(9) = (2 \cdot 9 + 3) \bmod 10 = 1$$
$$h_3(9) = (3 \cdot 9 + 7) \bmod 10 = 4$$

Nun fügt man die Elemente in den Bloomfilter ein. Für das erste Element 2 werden die Positionen 2, 7 und 3 auf 1 gesetzt. Daraus resultiert der folgende Bloomfilter:

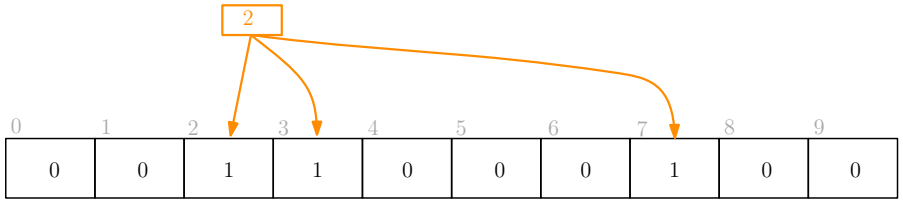

Abbildung 2: Bloomfilter nach Einfügen des Elements 2

Für das zweite Element 4 werden die Positionen 4, 1 und 9 auf 1 gesetzt.

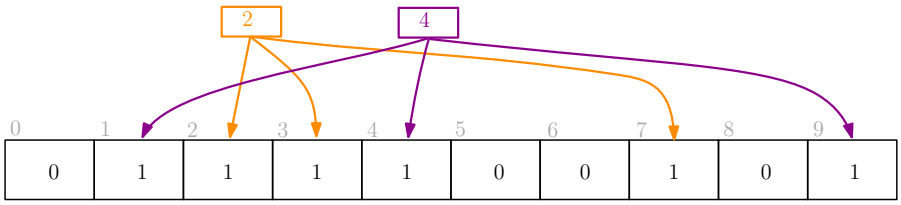

Abbildung 3: Bloomfilter nach Einfügen des Elements 4

Für das dritte Element 9 werden die Positionen 9, 1 und 4 auf 1 gesetzt. Da die Positionen 1, 4 und 9 bereits auf 1 gesetzt wurden, ändert sich der Bloomfilter nicht weiter.

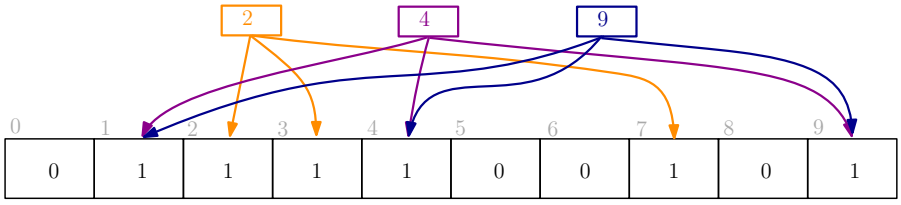

Abbildung 4: Bloomfilter nach Einfügen des Elements 9

Suchen

Um ein Element x in einem Bloomfilter zu suchen, werden dieselben Hashfunktionen wie beim Einfügen verwendet. Die Hashwerte werden berechnet und an den entsprechenden Positionen im Array geprüft. Wenn alle Positionen auf 1 gesetzt sind, so ist das Element wahrscheinlich in der Menge enthalten. Wenn mindestens eine Position auf 0 gesetzt ist, so ist das Element sicher nicht in der Menge enthalten. [TRL12]

1.2.2 Beispiel Suchen

Betrachten wir den zuvor erstellten Bloomfilter und suchen nach dem Element 4. Berechnen wir die Hashwerte für 4:

$$h_1(4) = 4 \bmod 10 = 4$$
$$h_2(4) = (2 \cdot 4 + 3) \bmod 10 = 1$$
$$h_3(4) = (3 \cdot 4 + 7) \bmod 10 = 9$$

Nun prüfen wir die Positionen 4, 1 und 9 im Bloomfilter. Alle drei Positionen sind auf 1 gesetzt, daher ist das Element 4 wahrscheinlich in der Menge enthalten.

Betrachten wir nun das Element 5 und berechnen die Hashwerte:

$$h_1(5) = 5 \bmod 10 = 5$$
$$h_2(5) = (2 \cdot 5 + 3) \bmod 10 = 3$$
$$h_3(5) = (3 \cdot 5 + 7) \bmod 10 = 2$$

Nun prüfen wir die Positionen 5, 3 und 2 im Bloomfilter. Die Position 2 ist auf 0 gesetzt, daher ist das Element 5 sicher nicht in der Menge enthalten.

Ein wichtiger Aspekt des Bloomfilters ist, dass er fälschlicherweise angeben kann, dass ein Element in der Menge enthalten ist, obwohl es tatsächlich nicht vorhanden ist. Dies wird als *False Positive* bezeichnet. Wenn alle Positionen, die durch die Hashfunktionen eines Elements angegeben werden, auf 1 gesetzt sind, obwohl das Element nicht in der Menge enthalten ist, führt dies zu einem False Positive. Ein Beispiel hierfür wäre das Element 12:

$$h_1(12) = 12 \bmod 10 = 2$$
$$h_2(12) = (2 \cdot 12 + 3) \bmod 10 = 7$$
$$h_3(12) = (3 \cdot 12 + 7) \bmod 10 = 3$$

Die Positionen 2, 7 und 3 sind alle auf 1 gesetzt, obwohl das Element 12 nicht in der Menge enthalten ist. Daher würde der Bloomfilter fälschlicherweise angeben, dass 12 in der Menge enthalten ist.

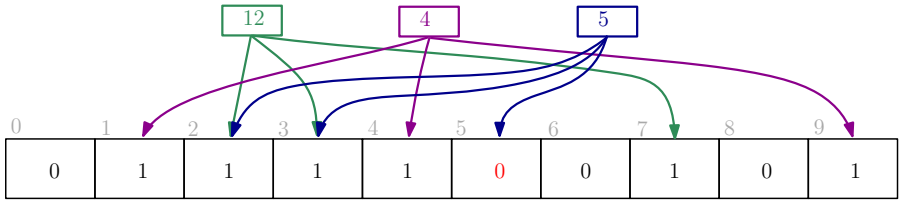

Abbildung 5: Suchen nach den Elementen 4, 5 und 12 im Bloomfilter

1.3 Formeln zur Evaluierung

1.3.1 False Positive Probability

Zur Erstellung des optimalen Bloomfilter ist es wichtig, die False Positive Probability (FPP) zu berechnen. Diese gibt an, wie wahrscheinlich es ist, dass der Bloomfilter fälschlicherweise angibt, dass ein Element in der Menge enthalten ist, obwohl es tatsächlich nicht vorhanden ist. Laut [TRL12] entsteht die Formel zur Berechnung aus folgenden Komponenten:

Unter der Annahme, dass die Hashfunktionen unabhängig und gleichverteilt sind, ergibt sich die Wahrscheinlichekeit dass ein bestimmtes der m Bits nicht gesetzt ist durch:

$$1 - \frac{1}{m} \tag{1}$$

Weiters werden nun die k Hashfunktionen mitbetrachtet, immer noch für den Fall, dass ein bestimmtes Bit nicht gesetzt ist.

$$\left(1 - \frac{1}{m}\right)^k \tag{2}$$

Nun werden die n Elemente der Menge S betrachtet, welche in den Bloomfilter eingefügt werden.

$$\left(1 - \frac{1}{m}\right)^{kn} \tag{3}$$

Die Wahrscheinlichkeit, dass ein bestimmtes Bit auf 1 gesetzt ist, ergibt sich aus der Gegenwahrscheinlichkeit:

$$1 - \left(1 - \frac{1}{m}\right)^{kn} \tag{4}$$

Da es bei Bloomfiltern um Membership-Tests geht, muss die Wahrscheinlichkeit berechnet werden, dass alle k Positionen eines Elements auf 1 gesetzt sind, obwohl das Element nicht in der Menge enthalten ist.

$$\left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \tag{5}$$

Aus der Formel lässt sich schließen, dass je **größer** m gewählt wird, desto **kleiner** wird die False Positive Probability. Je **größer** n gewählt wird, desto **größer** wird die False Positive Probability.

Da die False Positive Probability so klein wie möglich gehalten werden soll, ist auch die Wahl der Anzahl Hashfunktionen von großer Bedeutung. Setzt man die Formel für die False Positive Probability gleich 0 und löst sie nach k auf, erhält man die optimale Anzahl an Hashfunktionen:

$$k_{opt} = \frac{m}{n} \ln 2 \approx \frac{9m}{13n} \tag{6}$$

Literatur

[TRL12] Sasu Tarkome, Christian Esteve Rothenberg, and Eemil Lagerspetz. Theory and practice of bloom filters for distributed systems. *IEEE Communications Surveys & Tutorials*, 14(1):131–155, 2012.