Bloom Filter

Yannick Koller
Mike Gilgen

November 2022

1 Einleitung

Ein Bloom Filter, benannt nach seinem Erfinder Burton Howard Bloom, ist eine auf Wahrscheinlichkeit basierende Set Datenstruktur. Anders als herkömmliche Set Datenstrukturen erlaubt der Bloom Filter kein entfernen von Elementen und liefert kein garantiertes Ergebniss für die Zugehörigkeit eines Elements. Dies erlaubt es einem Bloom Filter Speichereffizienter zu sein als herkömmliche Set Datenstrukturen.

2 Vor- und Nachteile

Vorteil	Nachteil
Effizient	Fixe Grösse
Keine false negatives	Möglichkeit von false positives
Einfügen immer möglich	Kann überfüllen
	Löschen von Elementen nicht möglich

Die Möglichkeit von false positives ist sogenannten "Hash-Overlaps" geschuldet. Dabei kommt es bei nicht in der Menge vorhandenen Elementen zu den selben Hashwerten wie bei einem anderen, bereits in der Menge enthaltenem Element. Für alle gesuchten Elemente, die sich nicht in der Menge befinden, kann man dafür eine klare Aussage machen, dass sie nicht in der Menge ist. Daraus folgen zwei Aussagen die diese Datenstruktur zurückgibt wenn man die Zugehörigkeit eines Elements abfragt:

- Is probably in set (positive)
- Is definitely not in set (negative)

3 Praxisbeispiel

Der Webbrowser Google Chrome verwendete einen Bloom Filter um potenziell gefährliche URLs (wie z.B. Spam oder Malware) zu erkennen. Malwarescans werden auf Googles eigenen Servers gemacht was diese Operation Zeit- und Rechenaufwändig macht. Damit nicht jede Seite einem solchen Malwarescan unterzogen wird, wird zuerst der Bloom Filter befragt, ob eine URL potenziell gefährlich ist. Nur bei einem positiven Resultat wird die URL auf Googles eigenen Servern einem Malewarescan unterzogen.

4 Funktionsweise

Folgende Variablen werden für einen Bloom Filter benötigt:

- Erwartete Anzahl an Elementen n
- Gewünschte Fehlerquote p
- Anzahl benötigte Bits m
- Anzahl unterschiedliche Hashfunktionen k

Die optimalen Werte für Variablen m und k lassen sich aus n und p mit folgenden Formeln errechnen:

$$m = -\frac{n \ln p}{(\ln 2)^2} \tag{1}$$

$$k = \frac{m}{n} \ln 2 \tag{2}$$

Die Herleitung der Formeln sind auf Wikipedia auffindbar.

Sind Variablen m und k bekannt, kann ein neuer, leerer Bloom Filter erstellt werden.

pseudo Code

```
// defining the Arrays
HashFunctions[] hashFunctions;
boolean[] filter;

// defining the Hashalgorithms and build an empty filter
boolean[] buildFilter(int n, double p){
    int m = (int) -((n * Math.log(p)) / (Math.log(2) * Math.log(2)));
    int k = (int) ((m / n) * Math.log(2));

    hashFunctions = new HashFunction[k];
    // feeding the filter
```

```
        for( int i = 0; i < k; i++ ) {
            hashFunctions[i] = Hashing.<hashAlogrithm>(RandomInt);
        }
        filter = new boolean[m] // empty filter
    }
```

Nun sind alle benötigten Komponenten des Bloom Filter verfügbar und der Filter lässt sich mit folgender Methode mit Elementen befüllen:

pseudo Code

```
    void add(E element){
        for( HashFunction hashFunction : hashFunctions ){
            filter[hashFunction(element)] = true;
        }
    }
```

Die Abfrage, ob ein Element möglicherweise im Bloom Filter enthalten ist, funktioniert fast analog dem Einfügen:

pseudo Code

```
    boolean mightContain(E element){
        for( HashFunction hashFunction : hashFunctions ){
            if( !filter[hashFunction(element)] ){
                return false;
            }
        }
        return true;
    }
```

Sobald an einer Stelle im Filter der Wert $false$ angetroffen wird, kann mit 100% Sicherheit gesagt werden, dass ein Element nicht im Bloom Filter enthalten ist.

5 Erkenntnisse aus unserem Projekt

Wir testen unseren Bloom Filter mit der erhaltenen Wörterliste. Dabei fügen wir nur die Hälfte aller Wörter in den Bloom Filter ein. Die erwartete False-Positive Wahrscheinlichkeit haben wir mit $p = 0,2$ initialisert.
Mit der anderen Hälfte überprüfen wir die Korrektheit des Bloom Filters. Dabei zählen wir die False Positives und Vergleich deren Anteil im vergleich zur Totalanzahl an Wörtern. In unseren Versuchen variierte die Anzahl der False Positives zwischen 5600 und 5900 und die False Positives Rate schwankte um $0,2 \pm 0,01$ herum. Die False Positives Rate entspricht somit ziemlich genau der erwarteten Wahrscheinlichkeit von $p = 0,2$.

```
Tested words: 29055
False positives: 5799
False positives rate: 0.199587
```

Figure 1: Bloom Result