

BloomFilter

A Idee des Bloom-Filters

Ein BloomFilter liefert schnell eine Antwort, ob ein Wert bereits vorgekommen ist oder nicht.

Der Filter liefert also zwei verschiedene Antworten:

- Mit hoher Wahrscheinlichkeit enthalten
- Definitiv nicht enthalten

Gerade das ein Wert nicht bekannt ist, kann genutzt werden, um schnelle Entscheidungen zu treffen.

| Vor- und Nachteile | |
|---|--|
| Vorteil | Nachteil |
| <ul style="list-style-type: none"> • sehr performant • Treffergenauigkeit über die Grösse des Index konfigurierbar, dabei besteht kein Risiko, dass ein Treffer nicht gefunden wird • kompaktes Speichern der Daten (oft wird nur circa 1/8 der Ausgangs-Datenmenge benötigt) • Implementierungen in allen relevanten Programmiersprachen verfügbar • leicht parallelisierbar (über Aufteilung des Suchindex auf mehrere Server/CPU-Kerne) | <ul style="list-style-type: none"> • keinerlei Ähnlichkeitssuche, keine Fehler-toleranz • Suchergebnisse werden nicht sortiert • Risiko, sogenannte False Positives zu erhalten, das heisst, es können Datensätze zurückgegeben werden, die den Suchbegriff nicht enthalten |

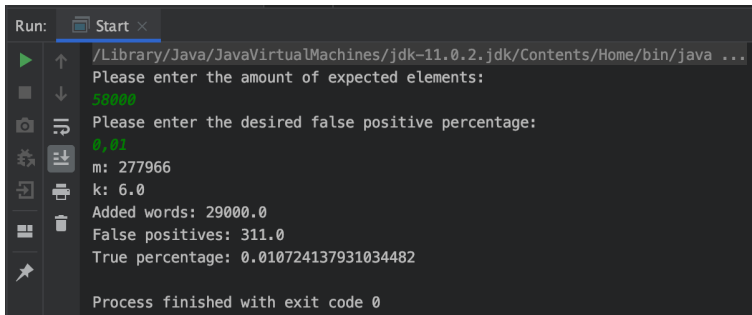
B Praxisbeispiel



Ethereum benutzt einen Bloom Filter, um Logs schnell und effizient in der Ethereum Blockchain finden zu können.

Im Ethereum-System müssen Events, einschliesslich historischer Events, leicht und ohne unnötigen Aufwand gefiltert und gesucht werden können. Gleichzeitig ist der Speicherplatz teuer, dass nicht viele Daten gespeichert werden sollen, wie z. B. die Liste der Transaktionen und die von ihnen erstellten Protokolle. Wenn ein Block generiert oder verifiziert wird, wird die Adresse eines Protokollierungsvertrags einem Bloom Filter hinzugefügt, der im Blockheader enthalten ist. Die eigentlichen Protokolle sind aus Platzgründen nicht in den Blockdaten enthalten. Wenn nun alle Protokolleinträge durchsucht werden, kann der Bloom Filter überprüfen, ob relevante Protokolle vorhanden sind.

C Testen der Fehlerwahrscheinlichkeit



```
Run: Start x
/Library/Java/JavaVirtualMachines/jdk-11.0.2.jdk/Contents/Home/bin/java ...
Please enter the amount of expected elements:
58000
Please enter the desired false positive percentage:
0.01
m: 277966
k: 6.0
Added words: 29000.0
False positives: 311.0
True percentage: 0.010724137931034482
Process finished with exit code 0
```

Die Fehlerwahrscheinlichkeit wird in unserem Programm folgendermassen berechnet: Zuerst kann der Benutzer eine bestimmte anzahl Wörter angeben, die in seinem Datensatz vorhanden sind und eine gewünschte Fehler-toleranz in Prozent.

Der Bloom Filter berechnet aus diesen beiden Werten die geeignete Bit Array Grösse und die Anzahl Hash Funktionen.

Das Programm fügt jedes zweite Wort in die Bloom Filter Liste. Beim hinzufügen wird ein Hash Wert generiert, mithilfe der Google Guava Library.

Aus diesem Hash Wert wird ausgelesen, an welcher Position im Bit Array eine 1 gesetzt werden soll.

Nachdem jedes zweite Wort hinzugefügt wurde, wird die andere Hälfte der Wörter im Datensatz verglichen. Dazu wird wieder aus jedem Wort ein Hash Wert generiert und es wird überprüft, ob an der Stelle im Bit Array das Bit noch auf 0 gesetzt ist, was bedeutet dass das Wort nicht vorhanden ist.

Aus der Anzahl hinzugefügter Wörter und der Anzahl gefundenen Wörtern kann nun die eigentliche Prozentzahl berechnet werden.

Diese liegt bei einem Datensatz von 58000 Wörtern bei einer Genauigkeit von über 99 %.