# Einleitung

* Wenn BF sage, denken manche bestimmt an so etwas (klick)
* Aber nein ich habe keine Endung verschluckt
* Erzähle euch heute wirklich etwas über BF aber die werden so hier geschrieben

# Gliederung

* Vorlesen

# Prinzipielle Idee

* Was ist BF?
* Erfunden 1970 von Burton Bloom
* Überprüfung ob Elemente in bestimmter vorher eingespeicherter Menge enthalten oder eben nicht
* Prinzip auf Grafik erkennbar
* Dabei Elemente selbst bzw der Inhalt aber nicht gespeichert sondern quasi merkt nur dass schonmal gesehen
* Ähnlich wie Fingerabdruckdatenbank 🡪 weiß dass den Fingerabdruck schon gesehen
* Dadurch ist sehr platzsparend und effizient
* Haken an der Sache
* Probabilistisch = Hat mit Wahrscheinlichkeit zu tun
* Bedeutet dass Aussagen nicht immer verlässlich sondern nur zu bestimmter WK
* Dabei unterscheiden: Wenn nein sagt, stimmt sicher
* Bei „ja“ nur zu bestimmter WK
* So entsteht Falsch Positiv Rate (FPP)

# Funktionsweise und Operationen

## Einfügen

* Array voller 0 mit Länge m
* N Elemente einfügen
* K unabhängige, gleichmäßig verteilte Hash Fkt
* Eignet bspw Murmur
* Wenn X als Element einfügen will bildet k Hashs davon
* Ergebnisse auf Arraylänge heruntergebrochen
* Positionen dann mit 1 ORed
* Hängt von Anzahl Hash Fkt ab, die größten Teil der Zeit ausmachen
* Diagramm

## Prüfen

* Element wird genau so gehasht auf Arraylänge
* Positionen an Stellen getestet (Grafik)
* Soabld nur eine 0 dabei, ist sicher nicht Teil der Menge gewesen
* Wenn alles 1 ist nur mit bestimmter WK Teil gewesen
* Kann durch Uneindeutigkeit der Hash Fkt auch durch andere gesetzt sein
* Somit FPP
* Auch das Überprüfen ist von k abhängig

## Parameterwahl

* Natürlich die Frage wie bestimmt k und die anderen Parameter, sodass optimal ausnutzt
* Grafik zeigen und Variablennamen verdeutlichen
* Wenn von Gleichverteilung der Hash Fkt ausgeht erhält folgende Formel für WK 0 zu sein
* Somit ergibt WK für k Einsen, was falsch positivem Ergebnis entspricht weil so alle Stellen zufällig durch fremde Elemente auf 1 gesetzt sind
* Diese Formel ergibt grob eine andere
* FPP ist natürlich an sich unerwünscht und deshalb will minimieren
* Umgestellt ergibt dafür folgende Formel für Länge des Arrays die zu wählen gilt
* Dabei sieht auch linearen Platzzusammenhang von zu wählender Länge und einzufügenden Elementen
* Wenn diese Länge hat kann dann optimale Anzahl Hash Fkt berechnen
* Das entspricht auch realistischem Szenario
* Hat maximale Fehlerrate gegeben und weiß zumindest grob wie viele Elemente einfügen will
* Daraus dann optimales m und k berechnen und BF laufen lassen
* Das schauen wir uns jetzt in der Praxis an
* Vorführen

# Vorführen

* Definierten Werte zeigen -🡪 wie gesagt FPP fest und Anzahl einzufügender Elemente
* Struct zeigen
* Init Fkt
* Einfügen
* Prüfen
* Dann auf FF eingehen

# Vergleich der Varianten

* Jetzt schauen mal an wie die Varianten im Vergleich arbeiten
* Diagramme auswerten

# Anwendungen

* Was kann man mit diesem Prinzip nun anfangen?
* Wo helfen einem Aussagen die falsch sein können?
* Klingt vllt erstmal absurd aber gibt sehr viele Anwendungen
* Allgemein kann sagen: Wo schnell Mitgliedschaft eines Elements aus großer Menge geprüft werden soll und wo dabei kleiner Fehlerenateil nicht verheerend ist
* Im speziellen finden dafür oft Anwendung in NW gefunden
* Routing
  + Anfrage für Ressource und soll schnellstmöglich zu Server der sie hat
  + Wo geht lang?
  + Speicher für alle Nachbarn/Links BF mit Ressourcen die dort gespeichert
  + Somit schnell prüfen
  + Wenn kein direkter Nachbar hat, geht immer Ebene tiefer wo Dokumente der Nachbarn der Nachbar sind usw. bis gefunden und dann schickt weiter oder wenn nicht findet, gibt an deterministischen Algorithmus weiter, der länger braucht
* Loop Prevention
  + BF in Header der Pakete hinzu
  + Jeder Node hat Bloom Mask mit 1 und 0 drin (äquivalent zu seinen Hash Werten)
  + ORed das mit rumgeschickten BF
  + Wenn ändert, war hier noch nicht, wenn nicht ändert liegt Loop nahe
  + Oder FP 🡪 schlecht
* Trace Route
  + BF in Paket mit drin
  + Jeder Router trägt dort Hash Wert seiner IP vom Output Interface ein
  + Kaum Zeitaufwand, da vorher berechnet werden kann und dann nur OR
  + Rückverfolgung über Überprüfung der Nachbar IP Adressen
  + Findet so immer Vorgänger 🡪 rekursiv zurückwandern
* DDoS
  + Einmal darüber und dann IP ignorieren
  + Oder zum White- bzw. Blacklisten von IP Adressen
* Web Cache
  + Speichert URLs in BF ein die gecached hat
  + Somit schnelle Überprüfung möglich
  + Je nachdem anfragen oder schicken
* Weitere Anwendungen gehen in Richtung Sicherheit/Security
* IDS
  + Bekannte gefährliche Strings von Viren (Signatures genannt)
  + Diese in verschiedenen Längen gestückelt in verschiedenen BF gespeichert
  + Eintreffende Daten werden dann ebenso zerstückelt und jeweils auf Membership in einem der BF geprüft
  + Wenn enthalten markiert als potentiell gefährlich
  + Folgt Überprüfung mit deterministischem Verfahren da ja auch FP sein könnte
* Datenbanken
  + Differential Files werden in DB verwendet um Änderungen abzufangen und dann gesammelt durchzuführen
  + BF dabei verwendet um vorherzusagen ob Anfrage auf DF oder in DB geht, weil das eben noch nicht geändert wurde
  + Zeit sparen, weil Suche einschränkt
* Weitere praktische und alltägliche Anwendungen
* Rechtschreibprüfung
  + Alle Wörter einer Sprache einspeichern in BF (deutsch zB 500 000)
  + Prüft dann geschriebenes Wort auf Membership
  + Gibt keine Vorschläge zur Korrektur aber guter erster Indikator
  + Ähnlich wie iDS
* LPM
  + Wieder ähnlich IDS für Prefixe genutzt
  + Zerteilt IP Adresse in bitweise Stücke
  + Jeweils BF für die eingespeicherten Prefixe mit bestimmten Längen
  + So wieder schnell überprüfen ob zB in BF mit Prefixlänge 6 Match und 7 nicht mehr, dann weiß dass da ist
* Ähnlichkeitsanalysen bei Websuchen
  + Teilt Seiten in Chunks
  + Diese in BF gespeichert
  + Vergleicht BF mit AND
  + Wenn viele 1 übrig bleiben ist Seite ähnlich
  + Somit Suchergebnisse verbessern oder zumindest gruppieren

# Varianten

* Aus den Anwendungen heraus könnt euch denken dass spezielle BF gebraucht werden mit angepassten Eigenschaften
* SO könnte zB sinnvoll sein Elemente wieder entfernen zu können
* Wie bei Web Cache, wenn Seite aus Cache fällt
* Wäre blöd immer ganzen BF neu berechnen zu müssen oder eben dass FPP steigt
* Darum gibt etliche Varianten die teilweise ganz angepasste Verbesserungen mitbringen, teilweise aber auch allgemeine Egft verbessern
* 3 davon vorstellen

## Counting BF

* Häufiger und simpler BF
* Addressiert eben jenes Problem, dass nicht löschen kann
* Statt je nur ein Bit, werden Zähler verwendet (grafiK)
* Somit immer ++ oder –
* Platzbedarf steigt aber Löschen als neue Funktion machbar

## Variable Increment

* Einen schritt weiter geht VIBF
* Basiert auf Idee des Counting BF
* Erhöht die Zähler aber nicht mehr um 1 sondern um speziell ausgewählte Zahlen aus Bh reihen
* Haben Eigenschaften dass Summen dieser Zahlen distinkt sind und somit feststellbar ob ein Summand Teil der Summe gewesen sein kann
* Somit zusätzliche Überprüfung mgl nicht nur anhand Position im Array wo 0 oder nicht 0 sondern auch noch anhand davon ob Summand Teil der Summe sein kann (Grafik)
* Ebenfalls löschen mgl und noch größerer Platzbedarf

## Deletable BF

* Ähnlich wie meine eigene Implementierung
* Extraspeicher für b Bereiche (Grafik)
* Aufpassen bei Eintragen der Elemente
* Wenn an Position bereits ne 1 wird ganzer Bereich markiert
* Wenn 0 dann nicht
* So ermöglicht teilweise löschen
* Überprüft bei Löschanfrage ob der Bereich aus dem etwas löschen soll mit 1 oder 0 markiert
* Wenn noch 0 kann ja bedenkenlos löschen da somit nicht anderes Element löscht
* An sich reicht somit sogar 1 freier Bereich pro zu löschendes Element, aber je mehr desto suaberer hält BF natürlich
* Wie gesagt gibt noch zahlreiche weitere Varianten die unmöglich alle behandeln kann und die viel zu speziell sind

# Zusammenfassung

* Kommen wir nun aber zu einem Allgemeinen Abschluss
* Seit Erfindung in 1970 auf jeden Fall viel Verwendung gefunden in Literatur und Praxis
* Beliebt durch Einfachheit, Effizienz und Vielseitigkeit
* Somit eben auch ständig neue Weiterentwicklungen und Varianten
* Welche Beste? Keine eindeutige Antwort
* Sicherlich ein oder andere, die in meisten Fällen besser als andere aber allgemein wohl keine ultimative Antwort
* Problem natürlich einmal die FPP, die aber durch geeignete Paramterwahl drücken kann
* Viel schwerer wiegt natürlich, dass Infos selbst gar nicht speichert
* Benötigt immer weitere Datenstruktur die Infos selbst speichert und was es da für Möglichkeiten gibt haben gehört oder hören noch
* Als abschließende Frage und auch Ausblick in die Zukunft stellt natürlich ob man die positiven Eigenschaften des BF mit Speicherfkt verknüpfen kann
* Daran wird natürlich immer weiter geforscht, an der Ultimativen Datenstruktur
* Dazu muss natürlich sagen, dass quasi selbst widerspricht weil die positiven Egft aus dem Nichtspeichern entstehen
* Aber wer weiß, ob es nicht doch einen Weg geben kann
* Vor 100 Jahren hätte auch niemand das Internet für möglich gehalten
* Bis so weit ist, müssen mit BF und seinen Varianten zufrieden geben