

ESTIMATION AND DATA ANALYSIS
WITH APPLICATIONS

Bloom Filter

Approccio Set-Membership

Studente: Salvatore Corvaglia
Professore: Gianfranco Parlangeli

Indice

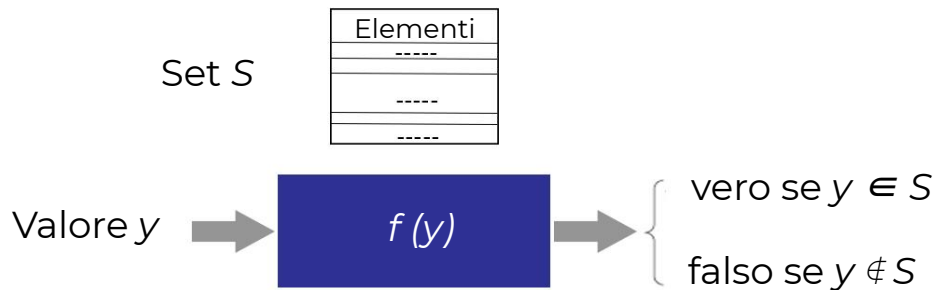
- Approccio Set-Membership
- Filtro Bloom
- Probabilità di Falsi Positivi
- Caratteristiche
- Casi d'uso
- Password Weak
- Conclusioni

Approccio Set-Membership

Approccio Set-Membership

Le metodologie Set-Membership permettono di utilizzare strutture di modello approssimate e di valutare gli effetti dovuti a un numero finito di dati.

Forniscono non un singolo modello, ma un insieme di modelli “ammissibili”, spesso indicato come “modello di incertezza”.

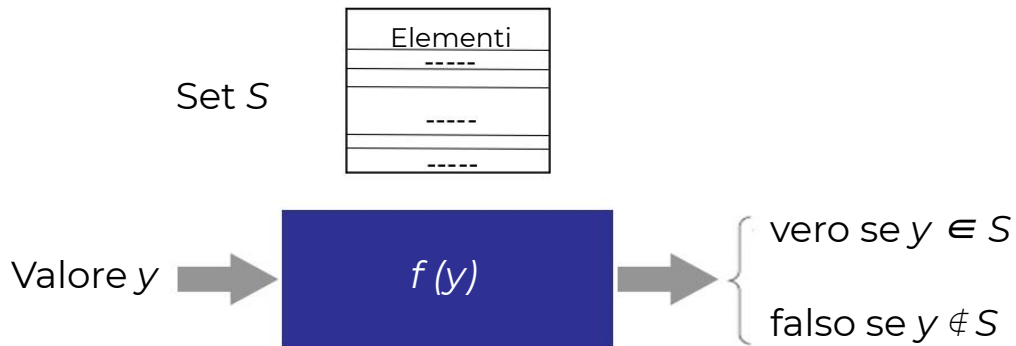


La teoria Set Membership permette di quantificare l'errore di approssimazione.

La funzione approssimante ottenuta risulta ottimale, nel senso che rende minimo l'errore di approssimazione sulla base delle assunzioni fatte.

Approccio Set-Membership

Dato un insieme $S = \{x_1, x_2, \dots, x_n\}$ di n elementi scelti da un universo U molto grande, si vuole rispondere a query del tipo: l'elemento y è nell'insieme S ?

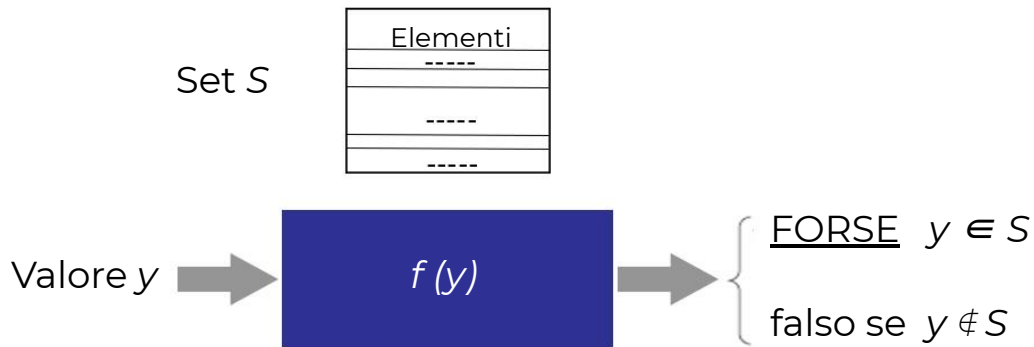


Implementare la funzione f che restituisce vero o falso a seconda della presenza o meno di y nell'insieme dato.

- ridurre tempo e spazio per la rappresentazione degli elementi
- approssimazione del problema

Approccio Set-Membership

Se si è disposti a rinunciare a un certo grado di accuratezza nelle operazioni di calcolo proprio per migliorare in termini di efficienza sulla rappresentazione dell'insieme.



Approccio Set-Membership

Problema: scegliere una rappresentazione degli elementi di S in modo da mantenere limitato il tempo di risposta delle query e ottimizzare lo spazio per la rappresentazione degli elementi, anche rinunciando a una completa correttezza del risultato, ammettendo così dei falsi positivi.

Quindi dobbiamo consentire alcune probabilità di errore:

- Falsi positivi: $y \in S$
- Falsi negativi: $y \notin S$



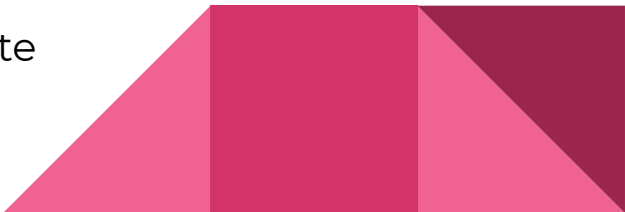
Falsi positivi / Falsi negativi

- **Falso positivo (errore di primo tipo):** indica che è stato erroneamente segnalato come vero (positivo al test) qualcosa che in realtà non lo è.

Un esempio è un antivirus che considera erroneamente dannoso un programma innocuo, generando un falso allarme.

- **Falso negativo (errore di secondo tipo):** indica che è stato erroneamente segnalata come assente una caratteristica che in realtà è presente.

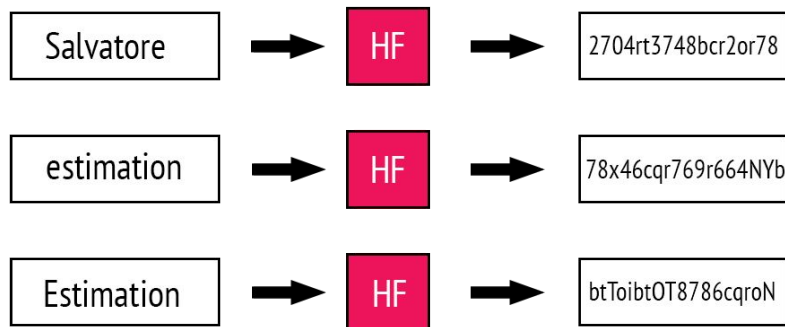
Un esempio è un filtro antispam che lascia erroneamente passare una mail indesiderata.



Funzioni di hash

Le funzioni di hash molto semplicemente sono funzioni matematiche che permettono di ridurre una qualunque stringa di testo (indipendentemente dalla sua lunghezza) in una nuova stringa avente precise caratteristiche, tra cui un numero di caratteri predefinito.


In altre parole, a partire da un input X sarà possibile generare un'uscita Y che avrà delle caratteristiche ben definite.



=
79054025
255fb1a2
6e4bc422
aef54eb4

Utilizzi delle funzioni di hash

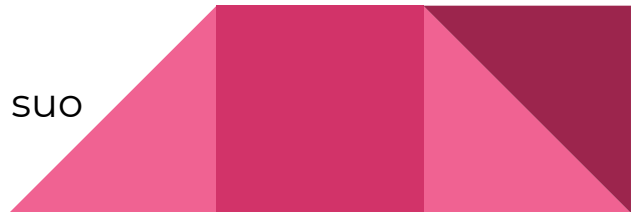
Gli hash sono una sorta di “impronta digitale” a lunghezza fissa di un messaggio e sono davvero utili in diversi settori applicativi, per esempio:

- **Controllo degli errori:** la funzione di hash viene utilizzata come un checksum/controllo per identificare eventuali errori di trasmissione dei dati.
 - **Integrità dei dati:** in questo caso la funzione di hash viene utilizzata per garantire che un messaggio non sia stato modificato da un eventuale attaccante.
 - **Verifica dell'integrità di un messaggio:** documenti e firme digitali
 - **Memorizzazione di password e autenticazione:** quando un sistema informatico deve memorizzare una password, questa non viene salvata in chiaro per motivi di sicurezza, né viene cifrata per evitare che sia possibile risalire alla password originale. In questi casi si memorizza l'hash della password.
- 

Proprietà delle funzioni di hash

La funzione di hash ideale deve avere alcune proprietà fondamentali:

- Deve identificare univocamente il messaggio, cioè non è possibile che due messaggi differenti pur essendo simili abbiano lo stesso valore di hash.
- Deve essere deterministico, in modo che lo stesso messaggio si traduca sempre nello stesso hash.
- Deve essere semplice e veloce calcolare un valore hash da un qualunque tipo di dato.
- Deve essere molto difficile generare un messaggio dal suo valore hash se non provando tutti i messaggi possibili.




Approccio Set-Membership: Applicazioni

Approccio Set-Membership: Applicazioni

1. Elaborazione del segnale

- Classificazione di segnali brevi per la morfologia derivazionale del parlato
- Equalizzazione adattiva per sistemi multicanale
- Stima delle frequenze armoniche variabili nel tempo
- Identificazione di frequenze audio

2. Teoria dell'informazione

- Decodifica simbolo per simbolo per la trasmissione di dati binari
 - Riduzione del modello per i canali di trasmissione
 - Strutture dati probabilistiche
- 

Approccio Set-Membership: Applicazioni

3. Rilevamento dei guasti

- Individuazione e isolamento dei guasti negli impianti chimici
- Test non distruttivi di strutture e materiali

4. Elaborazione delle immagini e computer vision

- Rilevamento e classificazione delle funzioni tomografiche
- Ricostruzione dell'immagine
- Tecniche di risoluzione delle collisioni

5. Produzione di semiconduttori

- Filtraggio non lineare

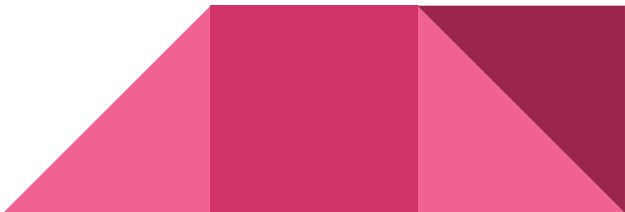


Approccio Set-Membership: Applicazioni

6. Ecologia

- Previsione dell'eutrofizzazione del lago
- Previsione del fabbisogno energetico nelle serre
- Stabilità a lungo termine dei laghi poco profondi

7. Robotica mobile e navigazione autonoma

- Localizzazione dei robot in tempo reale
 - Modellazione dell'ostacolo
 - Osservatori robusti per il tracciamento dei robot
 - Localizzazione e mappatura simultanee
- 

Strutture dati probabilistiche

Strutture dati probabilistiche

Una struttura dati è un modo per organizzare un insieme di informazioni con delle precise regole per aggiungere o togliere elementi, passare da un elemento all'altro, individuare il primo o l'ultimo elemento ecc...

Le strutture dati probabilistiche sono generalmente utilizzate per guadagnare spazio ed efficienza. Sono particolarmente utili quando si lavora con un set di dati molto grande e in generale il tasso di errore può essere controllato.

Le strutture dati probabilistiche sono compatibili con lo streaming e possono essere facilmente parallelizzate.
Sono utilizzate per risolvere una varietà di problemi:



Strutture dati probabilistiche

- Confronto di set di dati (Simhash algorithm)
- Test di appartenenza (Filtro Bloom)
- Problemi di stima della cardinalità (HyperLogLog algorithm)
- Database probabilistici (BlinkDB)
- Count-min sketch (CM sketch)
- Feature hashing (Machine Learning)




Filtro Bloom

Filtro Bloom

Il filtro Bloom, creato da Burton Howard Bloom nel 1970, è una struttura dati probabilistica efficiente usata per testare se un elemento appartiene a un insieme o no. Questi filtri meritano una descrizione approfondita, poiché sono piuttosto importanti ma poco conosciuti.

Una query quindi può ritornare “può darsi che ci sia dentro l'insieme (forse!)” oppure “sicuramente non c'è dentro l'insieme”.

Due caratteristiche fondamentali dei filtri Bloom sono la presenza di falsi positivi e l'assenza di falsi negativi, cioè se la ricerca di un elemento nel set dà esito negativo allora l'elemento sicuramente non appartiene al set, se invece la ricerca dà esito positivo allora esiste una probabilità (piccola) che in realtà l'elemento non esista.




Filtro Bloom

Un filtro Bloom è una matrice di m bit che rappresenta un insieme $S = \{x_1, x_2, \dots, x_n\}$ di n elementi. Inizialmente tutti i bit nel filtro sono impostati a 0. L'idea chiave è usare k funzioni di hash $[h_i(x) \mid 1 \leq i \leq k]$ per mappare gli elementi $x \in S$ su numeri casuali uniformi nell'intervallo $(1, \dots, m)$.

NB: l'algoritmo MD5 è una scelta popolare per le funzioni di hash.

Un elemento $x \in S$ viene inserito nel filtro impostando i bit $h_i(x) = 1 \mid 1 \leq i \leq k$.

Il punto debole dei filtri Bloom è la possibilità di un falso positivo. I falsi positivi sono elementi che non fanno parte dell'insieme S ma che sono riportati nel set dal filtro.



Filtro Bloom

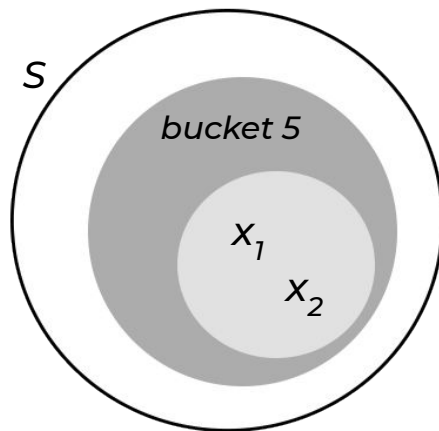
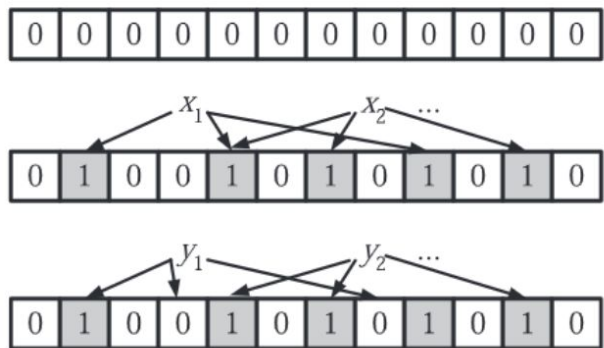
Più elementi vengono aggiunti all'insieme maggiore è la probabilità di falsi positivi, mentre la sua dimensione rimane costante.

I filtri Bloom sono quindi particolarmente utili per operazioni in cui si desideri scartare elementi non rilevanti e mantenere, seppur con un margine di errore, tutti gli altri.

Un filtro Bloom codifica in modo compatto le informazioni del set in un vettore di bit che può quindi essere interrogato in merito all'appartenenza al set.



Filtro Bloom



Filtro Bloom


Il funzionamento è il seguente:

- 1)** In fase di inizializzazione si definisce la dimensione del filtro (n), numero di bit, e il numero delle funzioni di hash (k). Ricordiamo che le funzioni di hash trasformano un valore in un numero che in questo caso è utilizzato come indice dell'array di bit.
- 2)** Per ogni elemento del set, le k funzioni di hash producono un numero casuale e i bit corrispondenti a tali indici sono posti pari a **1**.
- 3)** Per testare l'esistenza di un elemento nel set si determinano con le stesse funzioni di hash gli indici degli array. Se tutti contengono il valore **1**, allora l'elemento appartiene al set; se anche soltanto un valore è pari a **0**, allora l'elemento non appartiene al set.



Proprietà di un Filtro Bloom Standard

Proprietà di un Filtro Bloom Standard

- La quantità di spazio necessaria per memorizzare un filtro Bloom è molto piccola rispetto all'intero set.
 - Il tempo necessario per verificare se un elemento è presente o meno è indipendente dal numero di elementi presenti nel set.
 - La rimozione di un elemento dal filtro Bloom Standard è impossibile perché i falsi negativi non sono consentiti.
 - I filtri Bloom non supportano la cancellazione di elementi, infatti il ripristino dei bit potrebbe causare falsi negativi.
- 

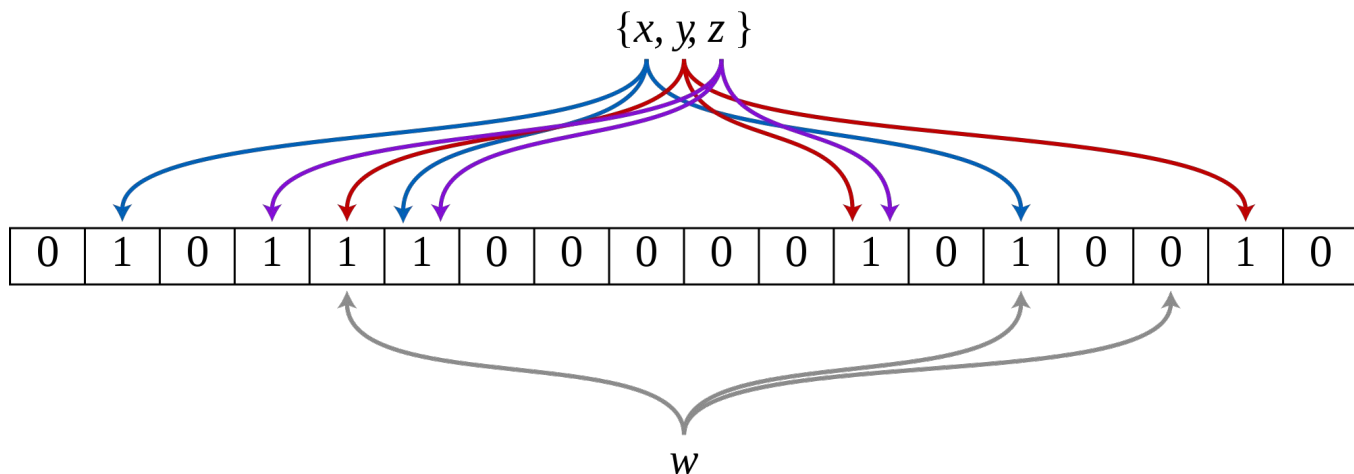
Proprietà di un Filtro Bloom Standard

- I filtri Bloom possono essere facilmente dimezzati ed essere utilizzati per approssimare l'intersezione tra due set.
- Se due filtri Bloom rappresentano gli insiemi S_1 e S_2 con lo stesso numero di bit e lo stesso numero di funzioni di hash, un filtro Bloom che rappresenta l'unione di questi due set può essere ottenuto prendendo l'operatore OR (disgiunzione logica \vee) dei due vettori bit dei filtri originali.



Descrizione dell'algoritmo

Descrizione dell'algoritmo



Ecco un esempio di un filtro Bloom che rappresenta l'insieme $\{x, y, z\}$. Per questa figura, $m = 18$ e $k = 3$. Le frecce colorate mostrano le posizioni nell'array di bit a cui ogni elemento è mappato. L'elemento w non è nell'insieme $\{x, y, z\}$ perché ha una funzione di hash su una posizione di matrice di bit contenente 0.

Descrizione dell'algoritmo

Un filtro Bloom **vuoto** è un array di bit con dimensione m tutti impostati su 0 . Ci devono anche essere k diverse funzioni di hash, ognuna delle quali mappa o esegue l'hashing di un elemento impostato su una delle posizioni m dell'array con una distribuzione casuale uniforme.

$\{x, y, z\}$

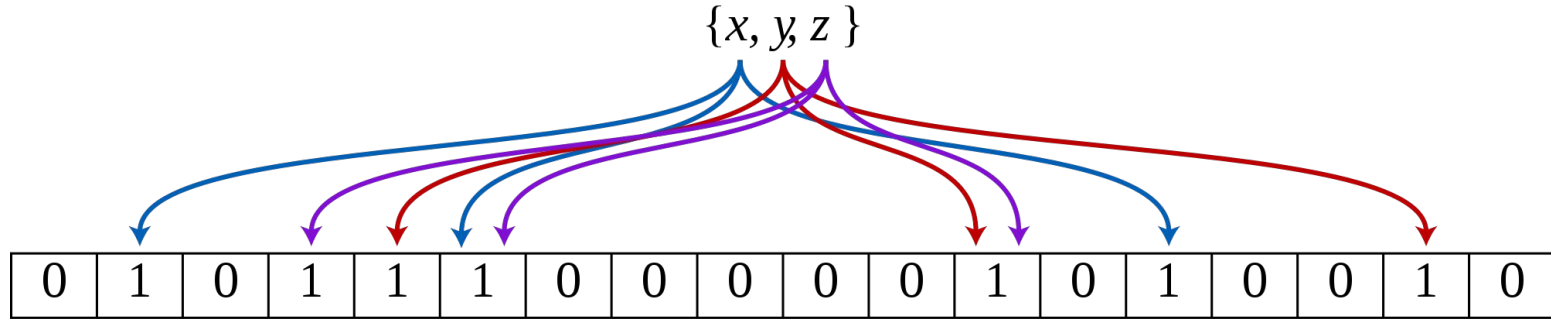
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

w



Descrizione dell'algoritmo

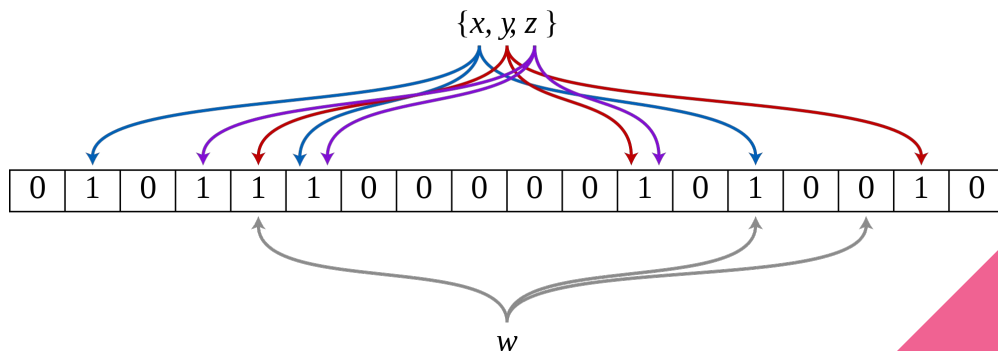
Per **aggiungere** un elemento bisogna alimentarlo con ciascuna delle funzioni di hash k per ottenere le posizioni dell'array. Si va a impostare a **1** i bit in tutte queste posizioni.



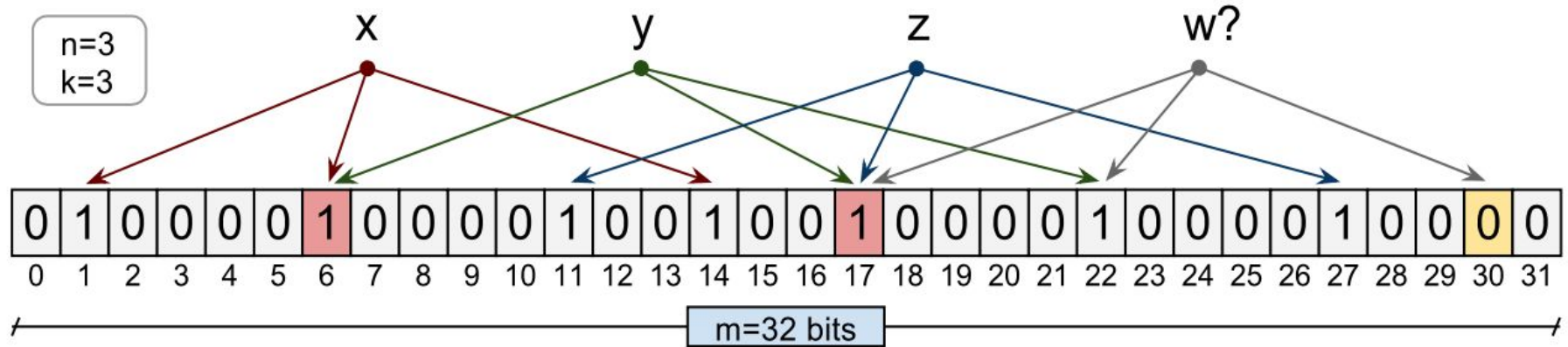
w

Descrizione dell'algoritmo

Si va a **testare** se un elemento è nel set. Se uno qualsiasi dei bit in queste posizioni è **0**, l'elemento non è sicuramente nel set, in caso contrario tutti i bit sarebbero stati impostati su **1**. Se tutti sono **1** allora l'elemento è nell'insieme, oppure i bit sono stati casualmente impostati su **1** durante l'inserimento di altri elementi, risultando così un falso positivo. In un filtro Bloom Standard non c'è modo di distinguere i due casi, ma tecniche più avanzate possono risolvere questo problema.



Descrizione dell'algoritmo



Qui ad esempio x e y condividono l'indice di array del bit 6. Mentre inseriamo più elementi in questo array, questa sovrapposizione aumenterà con possibili falsi positivi. Questo accade quando stiamo interrogando un elemento che non è stato inserito.

Descrizione dell'algoritmo

Algorithm 1: build

Data: S, H, B

Result: B inizializzato

```
begin
  for each s in S do
    for each h in H do
      B[h(s)]=1;
    end
  end
end
```

Algorithm 2: search

Data: H, B, y (the query)

Result: "found", "not found"

```
begin
  for each h in H do
    if B[h(y)]=0 then
      return "not found";
    end
  end
  return "found";
end
```

Esempio Jason Davies

Esempio Jason Davies

Operazioni

Il filtro Bloom di base supporta due operazioni: **test** e **inserimento**.

Il **test** viene utilizzato per verificare se un dato elemento è nell'insieme o meno e restituisce:

- *Falso* se l'elemento non è sicuramente nel set.
- *Vero* se l'elemento è probabilmente nel set.

L'**inserimento** semplicemente aggiunge un elemento al set.

La rimozione è impossibile senza introdurre falsi negativi, ma ci sono delle estensioni al filtro Bloom Standard che consentono proprio la rimozione.



Esempio Jason Davies

Applicazioni

Se l'elemento non è nel filtro Bloom allora sappiamo per certo che non è necessario eseguire la ricerca. D'altra parte se si trova nel filtro Bloom quando eseguiremo la ricerca ci aspetteremo che fallisca una parte del tempo (il tasso di falsi positivi).

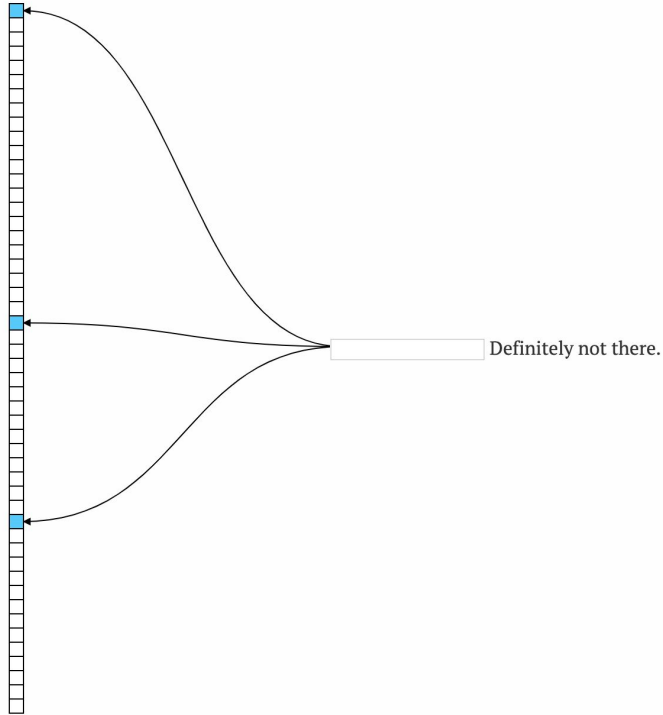
Dimostrazione

Jason ha creato un .js e ha impostato $m = 50$ e $k = 3$.



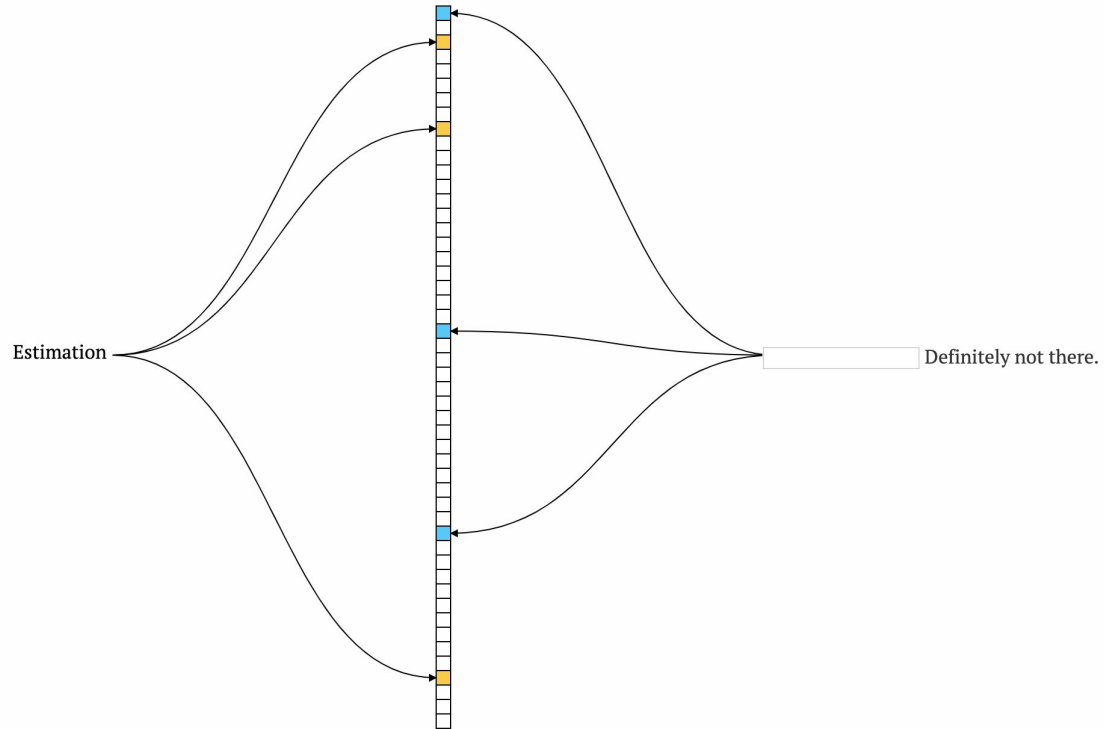
Esempio Jason Davies

Key:



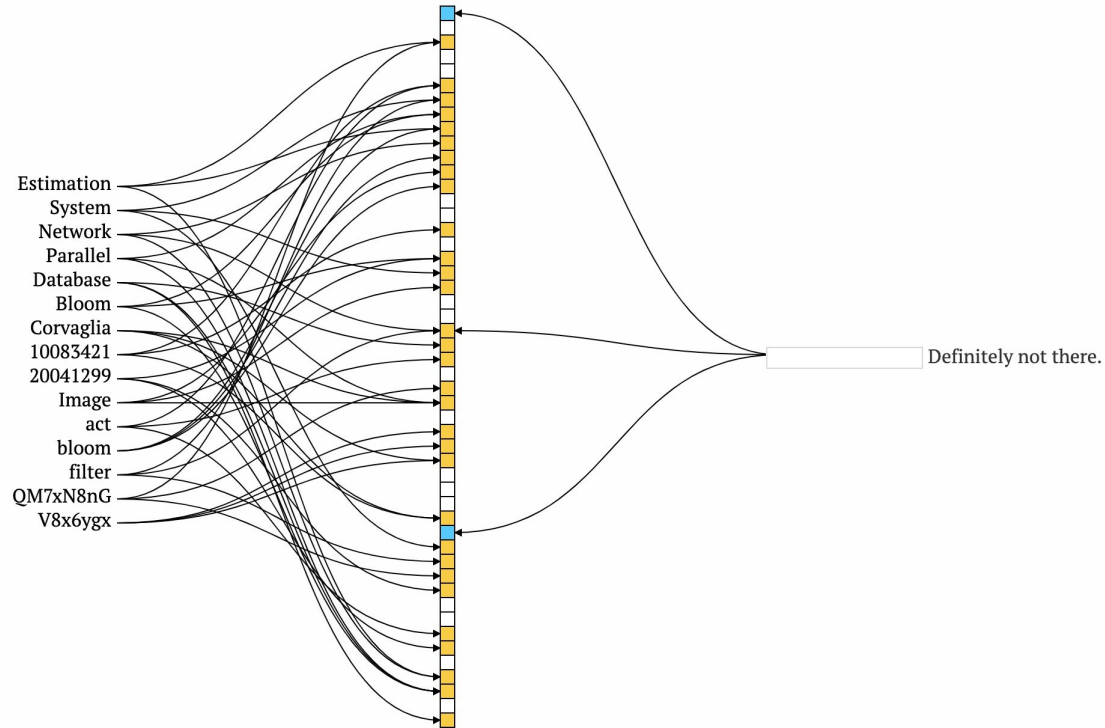
Esempio Jason Davies

Key:



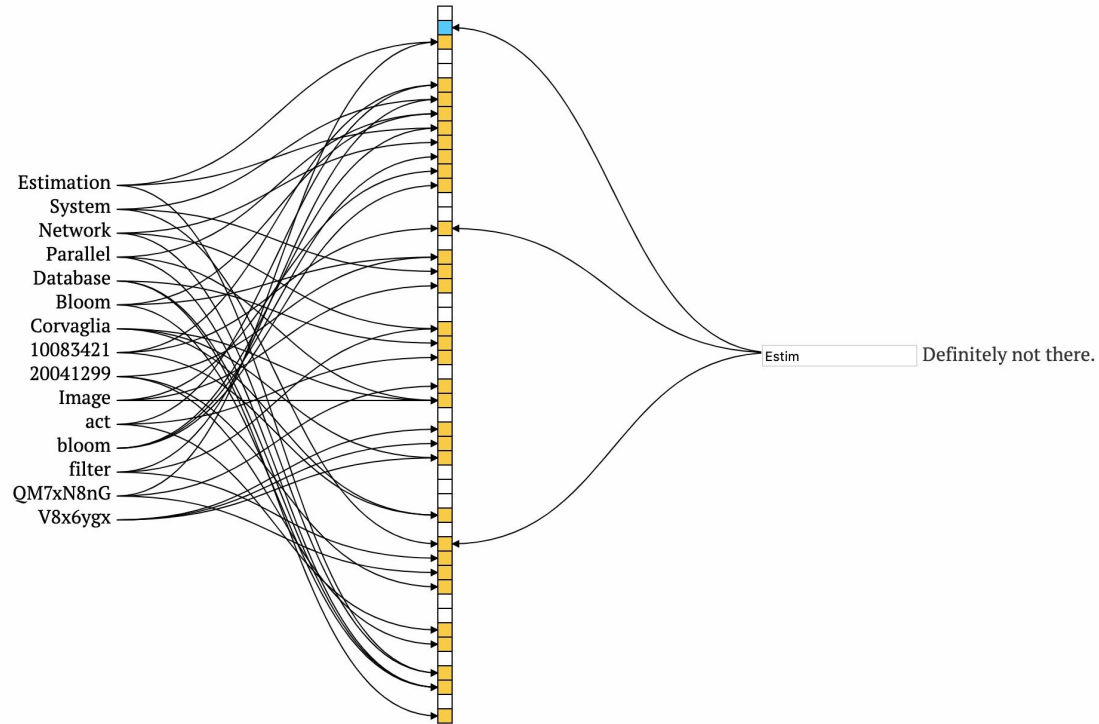
Esempio Jason Davies

Key:



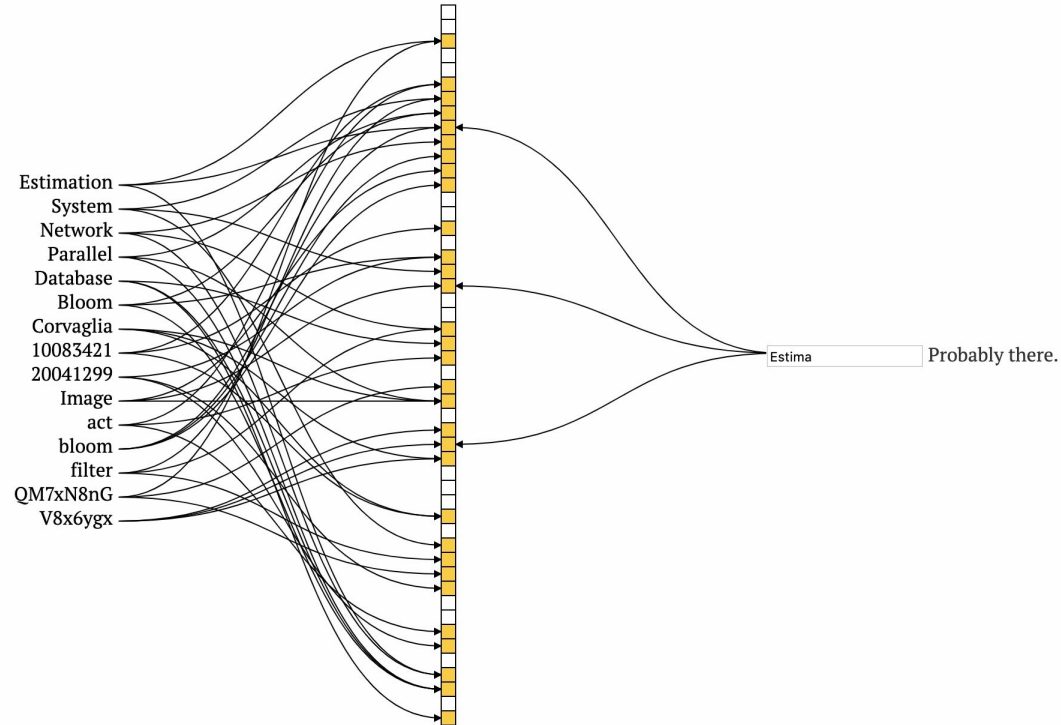
Esempio Jason Davies

Key:



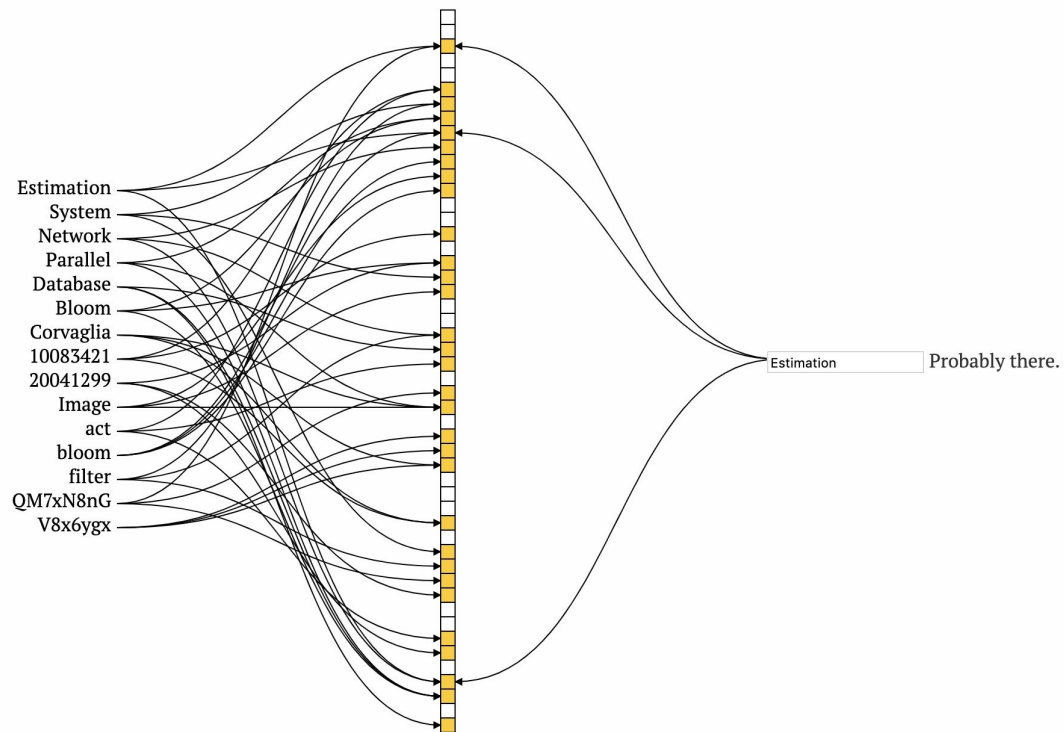
Esempio Jason Davies

Key:



Esempio Jason Davies

Key:



Tipi di Filtri Bloom

Tipi di Filtri Bloom

Ci sono oltre 20 varianti del filtro Bloom e queste variazioni includono funzionalità aggiuntive come il conteggio, la cancellazione, la popolarità, l'utilizzo, il falso negativo e il tipo di output (booleano, frequenza,...).

I filtri Bloom in genere vengono applicati in situazioni di appartenenza di un elemento per un set sufficientemente grande in una piccola quantità di tempo.

Sono state proposte numerose varianti del filtro Bloom che risolvono alcune delle limitazioni della struttura originale, tra cui i multiset e l'efficienza in termini di spazio. Alcune varianti presentano anche un'operazione di decremento per rimuovere elementi da un set.



Tipi di Filtri Bloom

Filter	Key feature	C	D	P	FN	Output
Standard Bloom filter	Is element x in set S ?	N	N	N	N	Boolean
Adaptive Bloom filter	Frequency by increasing number of hash functions	Y	N	N	N	Boolean
Bloomier filter	Frequency and function value	Y	N	N	N	Freq., $f(x)$
Compressed Bloom filter	Compress filter for transmission	N	N	N	N	Boolean
Counting Bloom filter	Element frequency queries and deletion	Y	Y	N	M	Boolean or freq.
Decaying Bloom filter	Time-window	Y	Y	N	N	Boolean
Deletable Bloom filter	Probabilistic element removal	N	Y	N	N	Boolean
Distance-sensitive Bloom filters	Is x close to an item in S ?	N	N	N	Y	Boolean
Dynamic Bloom filter	Dynamic growth of the filter	Y	Y	N	N	Boolean
Filter Bank	Mapping to elements and sets	Y	Y	M	N	x , set, freq.
Generalized Bloom filter	Two set of hash functions to code x with 1s and 0s	N	N	N	Y	Boolean
Hierarchical Bloom filter	String matching	N	N	N	N	Boolean
Memory-optimized Bloom filter	Multiple-choice single hash function	N	N	N	N	Boolean
Popularity conscious Bloom filter	Popularity-awareness with off-line tuning	N	N	Y	N	Boolean
Retouched Bloom filter	Allow some false negatives for better false positive rate	N	N	N	Y	Boolean
Scalable Bloom filter	Dynamic growth of the filter	N	N	N	N	Boolean
Secure Bloom filters	Privacy-preserving cryptographic filters	N	N	N	N	Boolean
Space Code Bloom filter	Frequency queries	Y	N	M	N	Frequency
Spectral Bloom filter	Element frequency queries	Y	Y	N	M	Frequency
Split Bloom filter	Set cardinality optimized multi-BF construct	N	N	N	N	Boolean
Stable Bloom filter	Has element x been seen before?	N	Y	N	Y	Boolean
Variable-length Signature filter	Popularity-aware with on-line tuning	Y	Y	Y	Y	Boolean
Weighted Bloom filter	Assign more bits to popular elements	N	N	Y	N	Boolean

Per ogni variante questa tabella va a indicare il tipo di output (booleano, frequenza, valore), se il conteggio (C), la cancellazione (D) e la consapevolezza della popolarità (P) sono supportati (Sì/No/Forse), e se vengono introdotti i falsi negativi (FN).

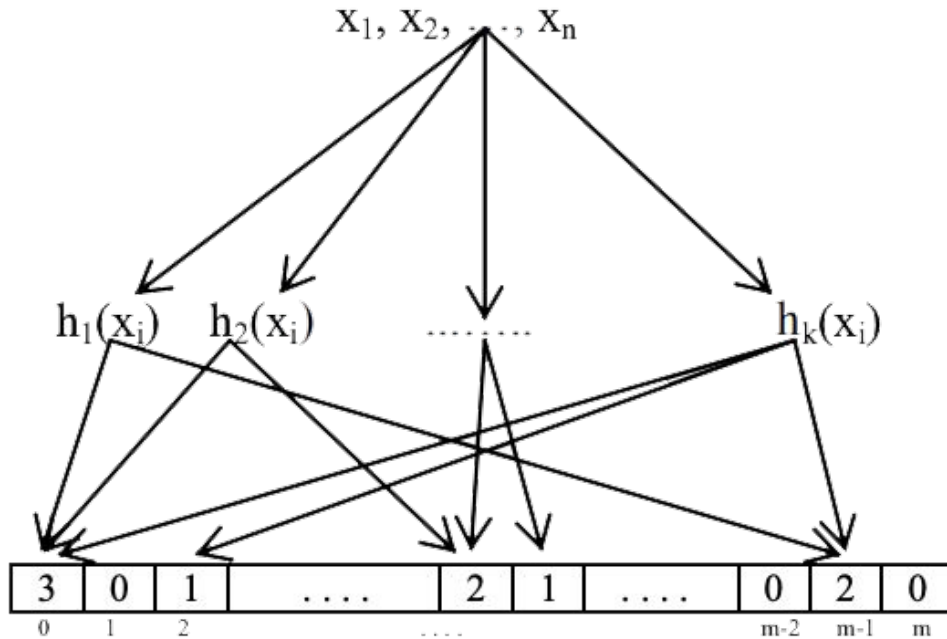
Tipi di Filtri Bloom

- **Counting Bloom Filter:** forniscono un modo per implementare un'operazione di eliminazione su un filtro Bloom senza ricreare nuovamente il filtro. In un Counting Bloom Filter le posizioni dell'array (bucket) sono estese da un singolo bit a un contatore n bit. In effetti i normali filtri Bloom possono essere considerati come Counting Filter con una dimensione del bucket di un bit. L'operazione di inserimento viene estesa per incrementare il valore dell'array.

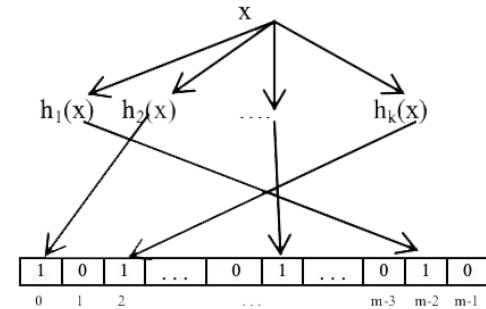
L'operazione di cancellazione consiste nel decrementare il valore di ciascuno dei rispettivi bucket che solitamente è di 3 o 4 bit.



Tipi di Filtri Bloom



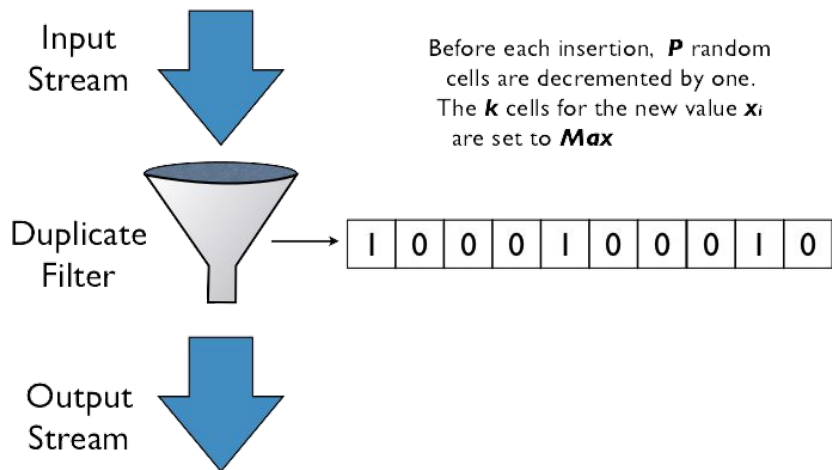
Set Operation in a Counting Bloom Filter



Set Operation in a Standard Bloom Filter

Tipi di Filtri Bloom

Stable (Time-Based) Bloom Filter



- **Stable Bloom Filter:** particolarmente utile nelle applicazioni di streaming dei dati. L'idea è che, poiché non è possibile memorizzare l'intera cronologia di un flusso di dati (che può essere infinito), gli Stable Bloom Filter rimuovono continuamente le informazioni obsolete per lasciare spazio a elementi più recenti. Poiché queste informazioni sono sfrattate, il filtro introduce dei falsi negativi.

Tipi di Filtri Bloom

- **Compressed Bloom Filter:** l'uso di questo filtro riduce il numero di bit trasmessi e il tasso di falsi positivi. I tempi di calcolo sono maggiori, sia per la compressione che per la decompressione.
- **Generalized Bloom Filter:** in questo filtro il valore iniziale dei bit dell'array non è più limitato a 0. Per verificare se l'elemento appartiene o no all'insieme, controlliamo se i bit sono tutti impostati e ripristinati.




Probabilità di Falsi Positivi

Probabilità di Falsi Positivi

Andiamo a ricavare il tasso di probabilità di falsi positivi di un filtro Bloom e il numero ottimale di funzioni di hash.

Partiamo dal presupposto che una funzione di hash seleziona ogni posizione dell'array con uguale probabilità. Se m è il numero di bit nell'array e k è il numero di funzioni di hash, allora la probabilità che un certo bit non sia impostato a 1 da una determinata funzione di hash durante l'inserimento di un elemento è quindi:

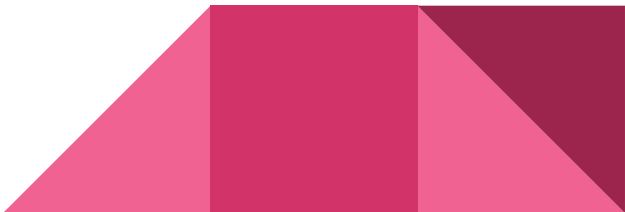
$$1 - \frac{1}{m}$$


Probabilità di Falsi Positivi

La probabilità che non sia impostata su 1 da nessuna delle funzioni di hash è:

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

Se abbiamo inserito n elementi la probabilità che un certo bit sia ancora 0 è:

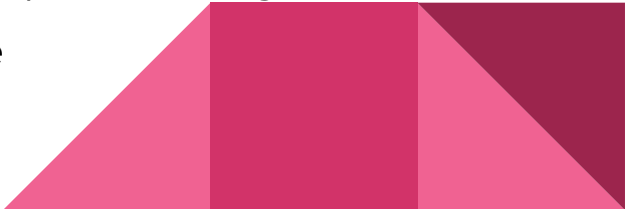
$$p = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-kn/m}$$


Probabilità di Falsi Positivi

La probabilità che sia 1 è quindi

$$p = 1 - \left(1 - \frac{1}{m}\right)^{kn}$$

Ora proviamo a vedere l'appartenenza a un elemento che non è nel set. Ciascuna delle posizioni dell'array calcolate dalle funzioni di hash è 1 con una probabilità già definita sopra. La probabilità che tutti siano 1 e che affermerebbe che l'elemento è nell'insieme è:



Probabilità di Falsi Positivi

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

Questo non è strettamente corretto in quanto è indipendente dalla probabilità che ogni bit è impostato. Tuttavia abbiamo che la probabilità di falsi positivi diminuisce quando m (il numero di bit nell'array) aumenta al crescere di n (il numero di elementi inseriti). Il numero di funzioni di hash k deve essere un numero intero positivo.

Per un dato m e n , il valore di k che minimizza la probabilità è:



Probabilità di Falsi Positivi

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

che dà:

$$2^{-k} \approx 0.6185^{m/n}$$

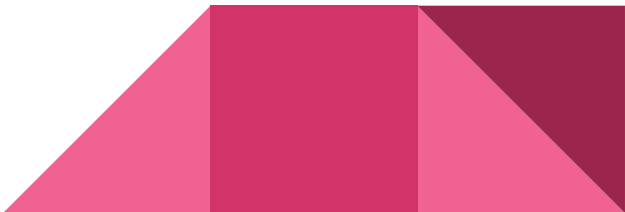


Probabilità di Falsi Positivi

Il numero richiesto di bit m , dato n e una probabilità di falsi positivi desiderata p , può essere calcolato sostituendo il valore ottimale di k nell'espressione di probabilità già considerata prima:

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

che può essere semplificata:

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


Probabilità di Falsi Positivi


Questo risulta in:

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

quindi il numero ottimale di bit per elemento è

$$\frac{m}{n} = -\frac{\log_2 p}{\ln 2} \approx -1.44 \log_2 p$$

con il numero corrispondente di funzioni di hash k :

$$k = -\frac{\ln p}{\ln 2} = -\log_2 p$$



Probabilità di Falsi Positivi

Ciò significa che per una data probabilità p , la lunghezza di un filtro Bloom m è proporzionale al numero di elementi n che vengono filtrati.

La formula già descritta prima è asintotica (cioè applicabile come $m, n \rightarrow \infty$), la probabilità di falsi positivi per un filtro Bloom finito con m bit, n elementi e k funzioni di hash è al massimo:

$$\left(1 - e^{-\frac{k(n+0.5)}{m-1}}\right)^k$$

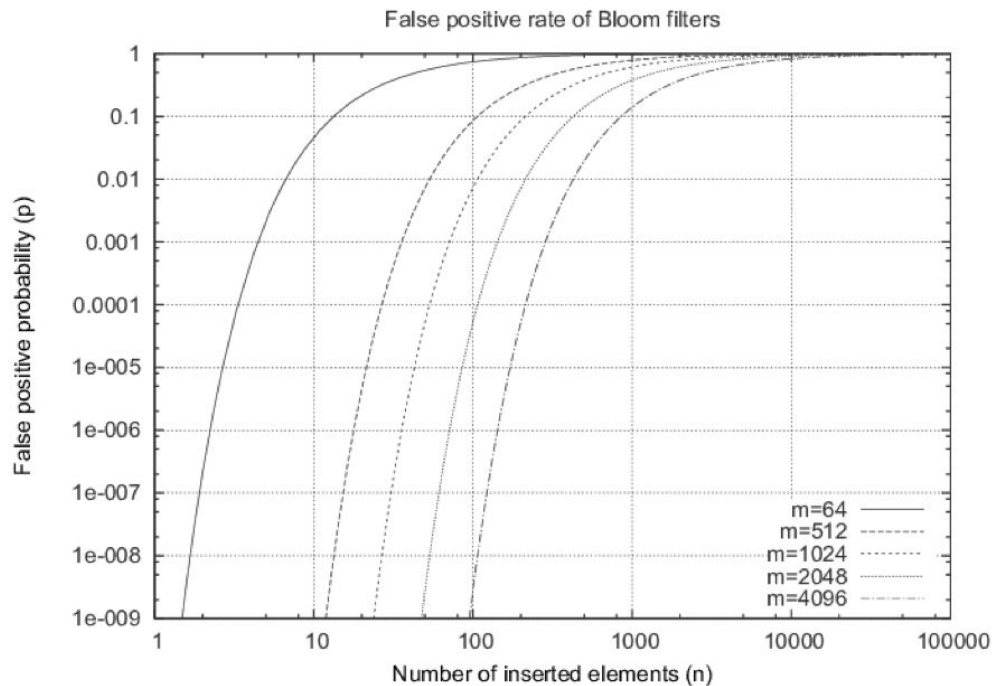
Small Subset Queries and Bloom Filters Using
Ternary Associative Memories, with Applications
(Goel and Gupta)



Probabilità di Falsi Positivi

m/n	k	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$
2	1.39	0.393	0.400						
3	2.08	0.283	0.237	0.253					
4	2.77	0.221	0.155	0.147	0.160				
5	3.46	0.181	0.109	0.092	0.092	0.101			
6	4.16	0.154	0.0804	0.0609	0.0561	0.0578	0.0638		
7	4.85	0.133	0.0618	0.0423	0.0359	0.0347	0.0364		
8	5.55	0.118	0.0489	0.0306	0.024	0.0217	0.0216	0.0229	
9	6.24	0.105	0.0397	0.0228	0.0166	0.0141	0.0133	0.0135	0.0145
10	6.93	0.0952	0.0329	0.0174	0.0118	0.00943	0.00844	0.00819	0.00846
11	7.62	0.0869	0.0276	0.0136	0.00864	0.0065	0.00552	0.00513	0.00509
12	8.32	0.08	0.0236	0.0108	0.00646	0.00459	0.00371	0.00329	0.00314
13	9.01	0.074	0.0203	0.00875	0.00492	0.00332	0.00255	0.00217	0.00199
14	9.7	0.0689	0.0177	0.00718	0.00381	0.00244	0.00179	0.00146	0.00129
15	10.4	0.0645	0.0156	0.00596	0.003	0.00183	0.00128	0.001	0.000852
16	11.1	0.0606	0.0138	0.005	0.00239	0.00139	0.000935	0.000702	0.000574

Probabilità di Falsi Positivi



Questa figura rappresenta la probabilità di falsi positivi p in funzione del numero di elementi n nel filtro e della dimensione del filtro stesso m .

È stato assunto un numero ottimale di funzioni di hash $k = (m/n) \ln 2$.

Probabilità di Falsi Positivi (Esempio)

Probabilità di Falsi Positivi (Esempio)

Se poniamo di avere 100.000.000 di elementi nel set (n) e di utilizzare 8 bit per elemento (m), avremo:

$$\begin{aligned}n &= 100.000.000 \\m &= 100.000.000 * 8 \\k &= 8 * \ln 2 = 5,545177444 \approx 6\end{aligned}$$

Quindi la probabilità di falsi positivi è:

$$p = \left(1 - e^{(-6 * (\frac{1}{8}))}\right)^6 = 2,158\%$$



Probabilità di Falsi Positivi (Esempio)

Esiste quindi una probabilità di poco più del 2% di avere dei falsi positivi utilizzando 6 funzioni di hash (k).

Lo spazio occupato dal filtro sarà di 800 milioni di bit, cioè circa 95 *MB*.

Se ipotizziamo che ciascun elemento dell'insieme sia costituito da una stringa di 50 byte, lo spazio occupato dall'intero set sarebbe di circa 4.6 *GB*.

Il filtro consente perciò di eseguire ricerche utilizzando circa il 2% dello spazio dell'intero set.


Approssimare il numero di elementi in un filtro Bloom

Approssimare il numero di elementi in un filtro Bloom

Swamidass & Baldi (2007) hanno dimostrato che il numero di elementi in un filtro Bloom può essere approssimato con la seguente formula:

$$X^* = -\frac{N \ln \left[1 - \frac{X}{N} \right]}{k}$$

dove X^* è la stima del numero di elementi nel filtro, N è la lunghezza del filtro, k è il numero di funzioni di hash per elemento e X è il numero di bit impostato su 1.




Unione e intersezione di insiemi

Unione e intersezione di insiemi

I filtri Bloom rappresentano un modo per rappresentare in modo compatto un insieme di elementi, infatti possono essere utilizzati per approssimare le dimensioni dell'intersezione e dell'unione di due set. Swamidass & Baldi (2007) hanno dimostrato che per due filtri Bloom di lunghezza m , la loro stima è

$$A^* = -\frac{N \ln \left[1 - \frac{A}{N} \right]}{k}$$

e

$$B^* = -\frac{N \ln \left[1 - \frac{B}{N} \right]}{k}$$


Unione e intersezione di insiemi

La dimensione della loro unione può essere stimata come:

$$A^* \cup B^* = - \frac{N \ln \left[1 - \frac{A}{N} \right]}{k}$$

dove $A^* \cup B^*$ è il numero di bit impostato su 1 in uno dei due filtri di Bloom. E l'intersezione può essere stimata come:

$$A^* \cap B^* = A^* + B^* - A^* \cup B^*$$

usando le tre formule insieme.



Vantaggi nello spazio e nel tempo

Vantaggi nello spazio e nel tempo

Pur rischiando di avere falsi positivi, i filtri Bloom hanno un forte vantaggio in termini di spazio rispetto ad altre strutture dati per la rappresentazione di insiemi, come alberi binari di ricerca, tabelle hash o matrici.

Un filtro Bloom con un errore dell'1% e un valore ottimale di k richiederebbe solo circa 9,6 *bit* per elemento, indipendentemente dalla dimensione degli elementi. Il tasso di falsi positivi dell'1% può essere ridotto di un fattore 10 aggiungendo solo circa 4,8 *bit* per elemento.



Vantaggi nello spazio e nel tempo

Le tabelle hash ottengono un forte vantaggio in termini di spazio e di tempo se iniziano a ignorare le collisioni.

I filtri Bloom hanno anche una proprietà insolita, cioè che il tempo necessario per aggiungere elementi o per verificare se un elemento è nel set è una costante fissa $O(k)$, completamente indipendente dal numero di elementi già presenti nel set.



Dimensione e numero di funzioni di hash del filtro Bloom

Dimensione e numero di funzioni di hash del filtro Bloom

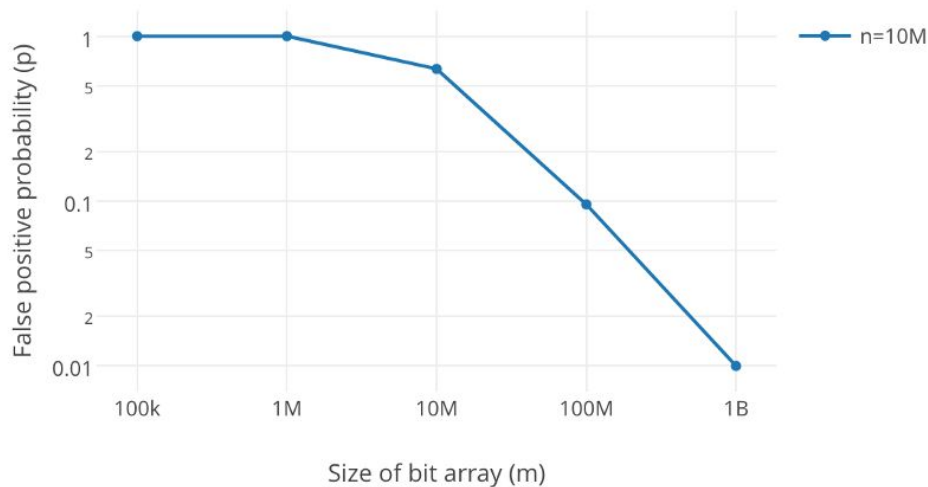
Se la dimensione del filtro Bloom è troppo piccola, molto velocemente tutti i campi di bit diventeranno **1** e quindi il nostro filtro Bloom restituirà falso positivo per ogni input. Quindi la dimensione del filtro Bloom è una decisione molto importante da prendere in considerazione. Un filtro più grande avrà meno falsi positivi rispetto a uno più piccolo.

Un altro parametro importante è quante funzioni di hash usare. Più funzioni di hash vengono utilizzate, più lento sarà il filtro Bloom e più rapidamente verrà riempito. Se ne abbiamo troppo pochi tuttavia potremmo subire troppi falsi positivi.



Dimensione e numero di funzioni di hash del filtro Bloom

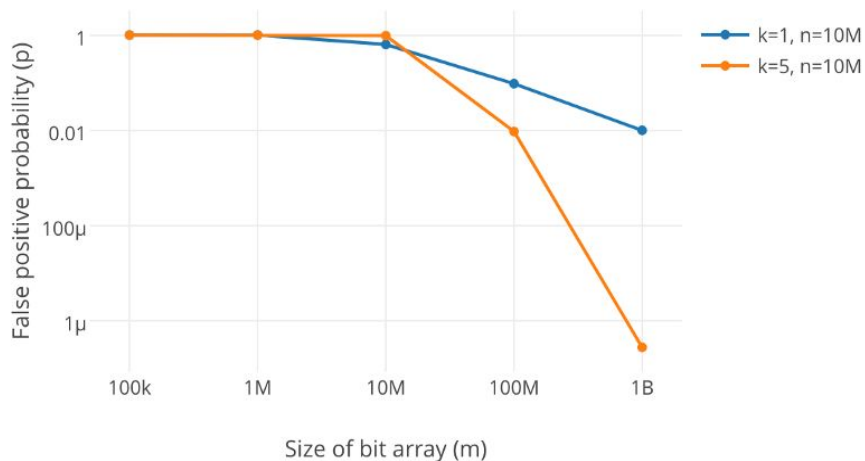
False positive probability v. Size of bit array



Tracciando la probabilità di falsi positivi rispetto alla dimensione dell'array per un numero fisso di elementi n da inserire, possiamo vedere che il tasso di falsi positivi diminuisce bruscamente all'aumentare delle dimensioni del campo dei bit. Dal grafico a fianco, per $p < 0.01$, avremmo bisogno di un array di dimensione **100x** il numero di valori n se usassimo solo una funzione di hash.

Dimensione e numero di funzioni di hash del filtro Bloom

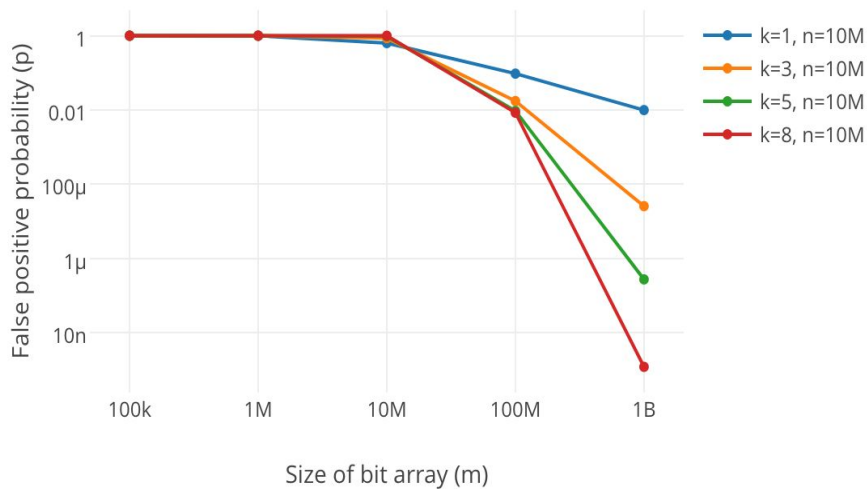
False positive probability v. Size of bit array



Un'idea importante alla base di un filtro Bloom è che se usiamo più funzioni di hash k , lo spazio richiesto può essere notevolmente ridotto.

Questo grafico ci suggerisce che l'aumento del numero di funzioni di hash k da 1 a 5 ha raggiunto $p < 0,01$ in uno spazio 10 volte più piccolo.

Dimensione e numero di funzioni di hash del filtro Bloom



A seconda del numero di valori da inserire e dalla percentuale massima consentita di falsi positivi, è possibile arrivare a un numero ottimale di funzioni di hash.

Qui possiamo vedere 4 funzioni di hash ($k = 1$, $m = 1B$), ($k = 3$, $m \approx 100M$), ($k = 5$, $m = 100M$), ($k = 8$, $m = 100M$). In questo caso ($n = 10M$, $p < 0,01$) la nostra scelta ideale sarebbe ($k = 5$, $m = 100M$) in quanto riduce al minimo sia i requisiti di spazio che di calcolo.



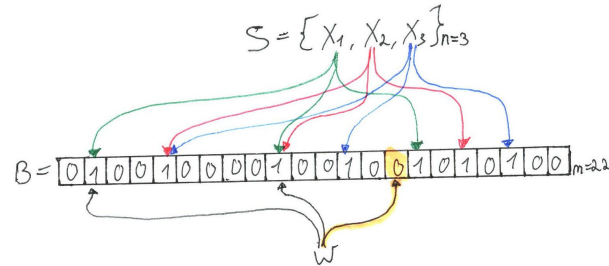
Casi d'uso

Bloom Filter


Casi d'uso

Il filtro Bloom, poiché può rappresentare l'intero universo di elementi ed è ottimo in termini di efficienza di spazio e tempo, viene utilizzato in molte aree della tecnologia informatica.

Alcuni casi d'uso veramente interessanti dei filtri Bloom sono elencati di seguito.



Casi d'uso

- **Spell Checking:** è possibile utilizzare il filtro Bloom per implementare il correttore ortografico utilizzando un dizionario predefinito con un numero elevato di parole.
 - **Password Protection:** per fornire sicurezza alle password, le parole usate frequentemente sono vietate. Per la corrispondenza di questi tipi di parole viene implementato il filtro Bloom, utilizzato per impedire la scelta di password deboli grazie a un dizionario di password facilmente ipotizzabili.
 - **Attack Checking:** il filtro Bloom è utilizzato da Google nel suo browser Chrome per identificare gli URL malevoli/ostili (primo controllo dell'URL)
- 

Casi d'uso

- **Metadata Finding:** quando ci sono molti dati nei file, il filtro Bloom utilizza una matrice per trovare un set di file per eseguire un test di ricerca. Gran parte della memoria viene utilizzata quando è necessario registrare una grande quantità di dati nei server.
- **Spam Mail Control:** il filtro Bloom blocca lo spam e protegge la rete da minacce recapitate via mail. Molte volte le mail vanno nella cartella spam e inoltre è necessario spostarle alla posta in arrivo. Grazie a questo filtro il problema è risolto.



Casi d'uso


- **Refining Web Search Results:** i filtri Bloom sono estremamente utili per perfezionare i risultati di ricerca restituiti dai motori di ricerca. Questa tecnica comporta la rimozione o il raggruppamento di tutti i documenti duplicati nei risultati presentati all'utente.
- **Web caching:** l'Internet Cache Protocol (ICP), protocollo molto leggero per la localizzazione di oggetti web e per la condivisione della cache, utilizza il filtro Bloom. Le query (le interrogazioni da parte di un utente per compiere determinate operazioni sui dati) vengono inviate ad altri server per ottenere una risposta sui dati. L'ICP introduce la cache per la comunicazione tra server proxy (un computer che si posiziona tra un client e un sito/pagina web che si vuole visitare, facendo da tramite tra i due).



Casi d'uso

- **Distributed Systems:** il filtro Bloom è ampiamente utilizzato in ambiente distribuito con memorizzazione di chiavi per la ricerca (BitTorrent, Skype,...)
 - **Packet Routing:** la classificazione del pacchetto può essere facilmente risolta con filtri Bloom. Utilizzando il filtro Bloom si può migliorare l'instradamento dei pacchetti attraverso la rete in modo efficiente.
 - **Crypto wallets:** la piattaforma decentralizzata Ethereum utilizza i filtri Bloom per trovare rapidamente i log sulla sua blockchain. Bitcoin li utilizza per accelerare la sincronizzazione dei wallet e anche per migliorare la sicurezza del portafoglio.
- 

Casi d'uso

- **Database:** i database non relazionali Google BigTable, Apache HBase, Apache Cassandra e Postgresql utilizzano i filtri Bloom per ridurre le ricerche su disco per righe o colonne inesistenti.
 - **Molecular biology:** nella branca della biologia molecolare si utilizzano i filtri Bloom per la classificazione delle sequenze di DNA con una successione di lettere che rappresentano la struttura primaria di una molecola di DNA.
 - Rilevamento di truffe negli stream Pay-Per-Click delle reti pubblicitarie online (**AdSense**).
- 

Casi d'uso

- **URL Shorteners:** abbreviazione degli indirizzi web, una tecnica utilizzata per abbreviare lunghi indirizzi web in link composti da pochi caratteri.

Rebrandly

bitly

cuttly

I servizi di abbreviazione di URL come **Bit.ly** vengono presi di mira dai criminali informatici per attirare gli utenti in trappole di phishing o malware.

Per prevenire il problema, questi tool applicano i filtri Bloom per rilevare URL dannosi.




(Password Weak)

(Password Weak)

La scelta di password definite deboli è un problema comune quando un sistema utilizza username e password come credenziali per l'autenticazione, infatti possono compromettere il sistema. Sono stati proposti molti approcci per impedire all'utente di selezionare password deboli o indovinate.

È stato selezionato un dataset di password deboli con 300.000 password. Per questo esperimento sono stati considerati due approcci: quello sperimentale e quello teorico, che si confronteranno per avere una buona conoscenza della versione implementata del filtro Bloom. Lo scopo consiste nel calcolare il tasso di falsi positivi più basso.

(International Journal of Computer Applications Technology and Research)
Warsaw (Poland)




(Password Weak)

È stato utilizzato un elenco di 30.000 password sicure per calcolare il tasso di falsi positivi dal punto di vista empirico.

Il primo passo consiste nel calcolare il tasso di falsi positivi basato su diverse funzioni di hash per avere un'idea di base su alcuni numeri ottimali di k .

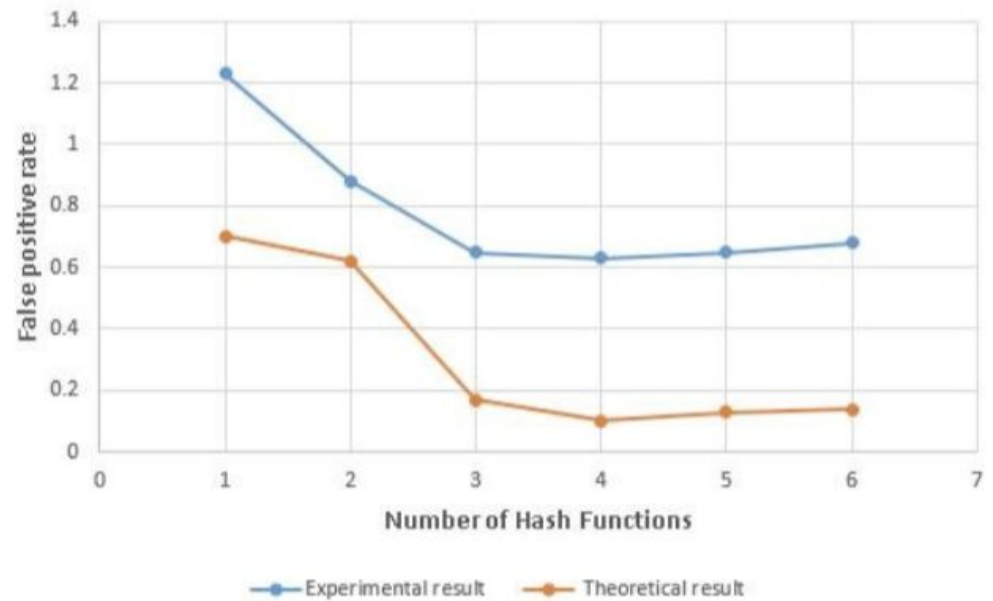
Usando più funzioni di hash può essere considerato un modo per raggiungere il più basso tasso di falsi positivi, questo è vero, ma in alcuni punti l'aggiunta di più funzioni di hash al sistema non funziona e il tasso di falsi positivi ricomincia a crescere. In pratica le funzioni di hash producono uscite sufficientemente distribuite in modo uniforme, come MD5, CRC32, SHA1, SHA2 e FNV che sono utili per la maggior parte dei filtri probabilistici come il filtro Bloom.



K	Experimental result	Theoretical result
1	1.23	0.7
2	0.88	0.62
3	0.65	0.17
4	0.63	0.1
5	0.65	0.13
6	0.68	0.14

La tabella mostra il risultato esatto per entrambi i casi.

(Password Weak)

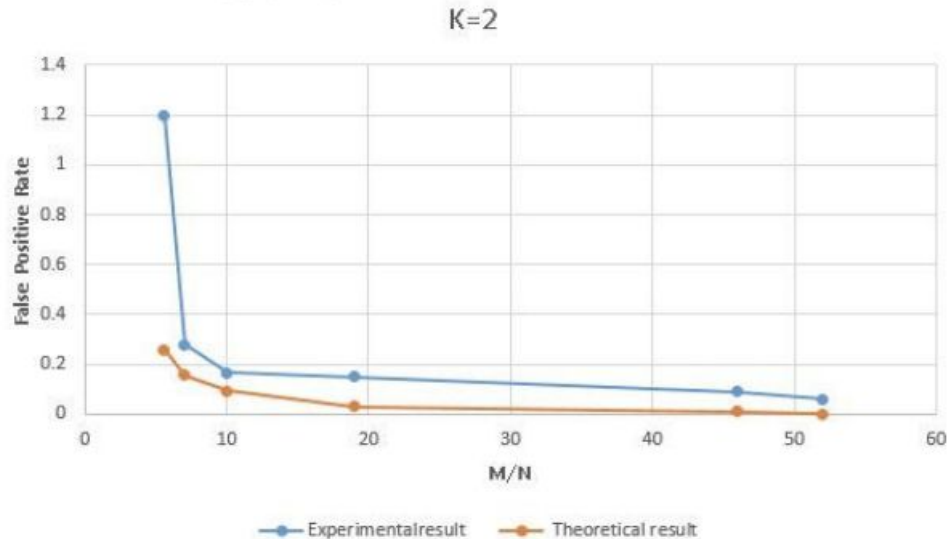


Per il primo esperimento è stato selezionato $m/n = 6$ e sono state applicate diverse funzioni di hash k .

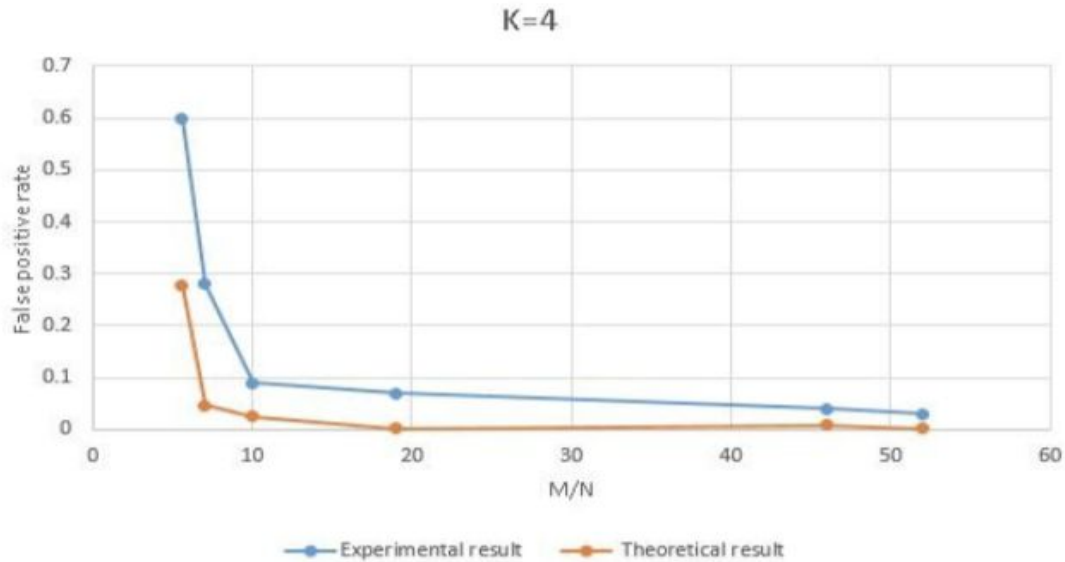
Il risultato sperimentale ha un errore un pò più alto rispetto a quello teorico, ma quasi entrambi seguono lo stesso orientamento. Dal punto di vista del risultato teorico, $K = 4$ gioca una soluzione ottimale per i dati e quella sperimentale conferma questo risultato.

(Password Weak)

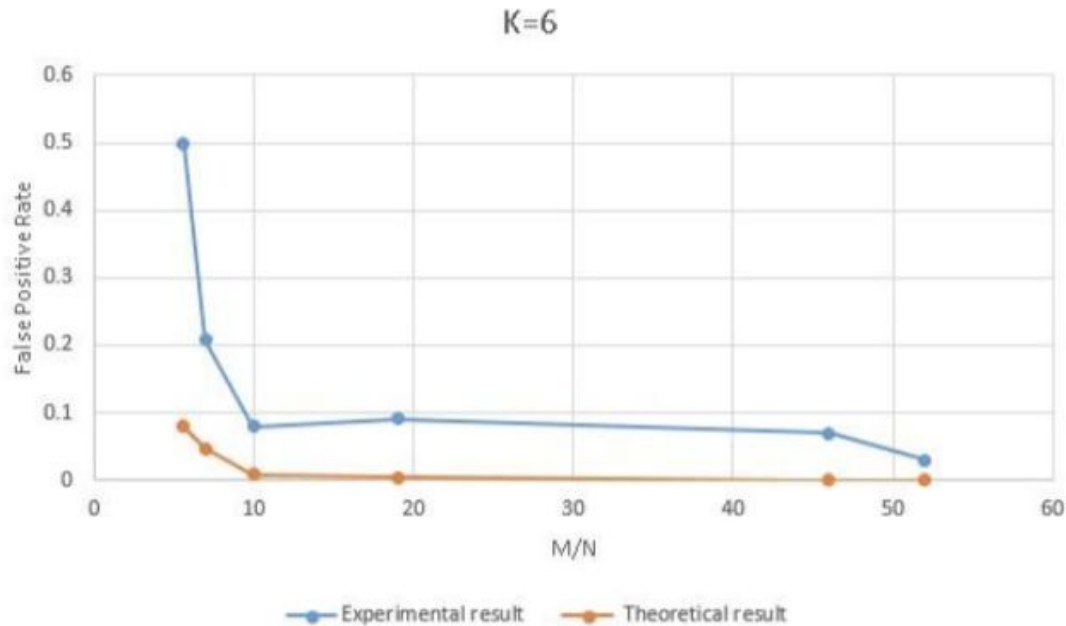
Per il prossimo passaggio il tasso di falsi positivi viene calcolato in base alle diverse dimensioni del filtro Bloom m . Le tre figure successive mostrano i diversi valori dei falsi positivi mentre si applicano un numero diverso di funzioni di hash k del filtro Bloom.



(Password Weak)

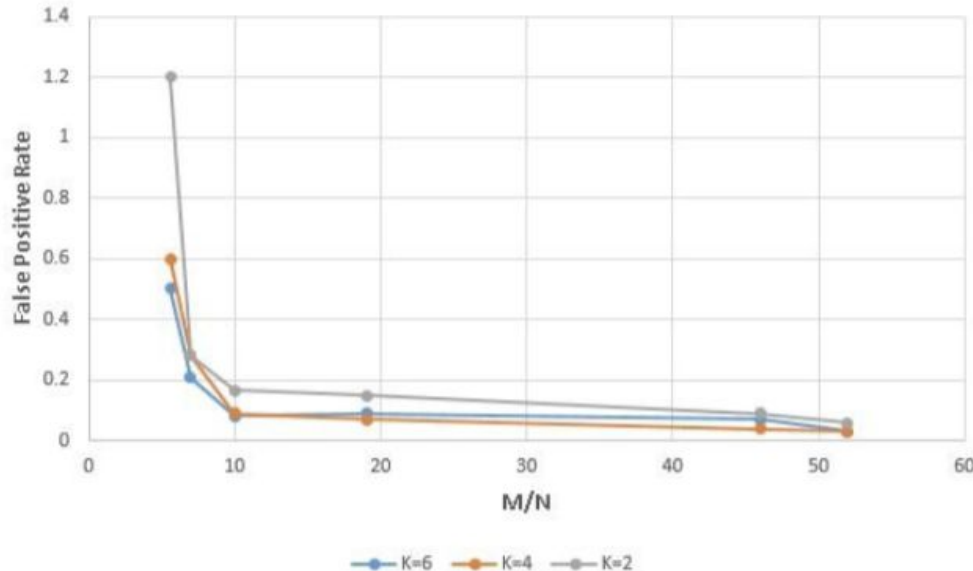


(Password Weak)



(Password Weak)

Nella figura qui sotto, il tasso di falsi positivi più basso può essere ottenuto con un filtro Bloom con $m/n = 50$.



Poiché non abbiamo tanta sensibilità, possiamo prendere $K = 4$ con $m/n = 10$ come soluzione ottimale.

Conclusioni

Il filtro Bloom è una struttura dati compatta per la rappresentazione probabilistica di un set al fine di supportare le query di appartenenza.

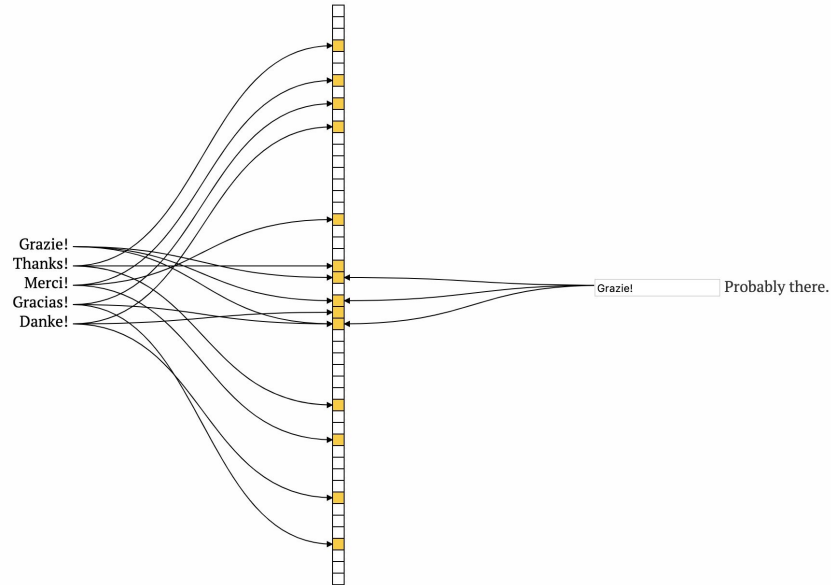
Ha un forte vantaggio rispetto ad altre strutture dati per la rappresentazione di insiemi in termini di spazio e tempo, ma come ci si può aspettare c'è sempre un compromesso e in questo caso è il falso positivo.

Il filtro Bloom non memorizza gli elementi stessi, questo è il punto cruciale. Non usiamo un filtro Bloom per verificare se un elemento è presente, lo usiamo per verificare se non è certamente presente, proprio perché non garantisce falsi negativi, esattamente il motivo per cui è chiamato probabilistico.



Grazie per l'attenzione!

Key:



Bibliografia

- Bloom Filters: Is element x in set S ? - Abhishek Tiwari
- Putze, F.; Sanders, P.; Singler, J. (2007) - "Cache, Hash and Space-Efficient Bloom Filters"
- Pagh, Anna; Pagh, Rasmus; Rao, S. Srinivasa (2005) - "An optimal Bloom filter replacement"
- Kirsch, Adam; Mitzenmacher, Michael (2006) - "Less Hashing, Same Performance: Building a Better Bloom Filter"
- Yes/no Bloom filter: A way of representing sets with fewer false positives
- Theory and Practice of Bloom Filters for Distributed Systems - Sasu Tarkoma, Christian Esteve Rothenberg, Eemil Lagerspetz