Bloom Filters

Burton Howard Bloom [1970]

Course Website: Bloom Filters Survey by A. Broder and M. Mitzenmacher

Bloom Filter

- Space-efficient “Probabilistic Dictionary”

Bloom Filter

- Space-efficient “Probabilistic Dictionary”
- Maintain the “fingerprints” of the elements of a set S

Bloom Filter

- Space-efficient “Probabilistic Dictionary”
- Maintain the “fingerprints” of the elements of a set S

- Operations:

Bloom Filter

- Space-efficient “Probabilistic Dictionary”
- Maintain the “fingerprints” of the elements of a set S

- Operations:

  BF_Insert(x) : $S \leftarrow S \cup \{x\}$

Bloom Filter

- Space-efficient “Probabilistic Dictionary”
- Maintain the “fingerprints” of the elements of a set S

- Operations:

  BF_Insert(x) : $S \leftarrow S \cup \{x\}$

  BF_Search(x) :

Bloom Filter

- Space-efficient "Probabilistic Dictionary"
- Maintain the "fingerprints" of the elements of a set S

- Operations:

BF_Insert(x) : $S \leftarrow S \cup \{x\}$

"No"

BF_Search(x) :

Bloom Filter

- Space-efficient "Probabilistic Dictionary"
- Maintain the "fingerprints" of the elements of a set S

- Operations:

BF_Insert(x) : $S \leftarrow S \cup \{x\}$

BF_Search(x) : → "No" $\Rightarrow x \notin S$

Bloom Filter

- Space-efficient "Probabilistic Dictionary"
- Maintain the "fingerprints" of the elements of a set S

- Operations:

BF_Insert(x) : $S \leftarrow S \cup \{x\}$

BF_Search(x) :
- "No" $\Rightarrow x \notin S$
- "Probably Yes"

Bloom Filter

- Space-efficient "Probabilistic Dictionary"
- Maintain the "fingerprints" of the elements of a set S

- Operations:

BF_Insert(x) : $S \leftarrow S \cup \{x\}$

BF_Search(x) :

"No" $\Rightarrow x \notin S$

"Probably Yes" $\Rightarrow$ "Probably" in $x \in S$

Bloom Filter

- Space-efficient "Probabilistic Dictionary"
- Maintain the "fingerprints" of the elements of a set S

- Operations:

BF_Insert(x) : $S \leftarrow S \cup \{x\}$

BF_Search(x) :

- "No" $\Rightarrow x \notin S$
- "Probably Yes" $\Rightarrow$ "Probably" in $x \in S$ (but perhaps not!)

Bloom Filter

- Space-efficient “Probabilistic Dictionary”
- Maintain the “fingerprints” of the elements of a set S

- Operations:

BF_Insert(x) : $S \leftarrow S \cup \{x\}$

BF_Search(x) :

→ “No” $\Rightarrow x \notin S$

→ “Probably Yes” $\Rightarrow$ “Probably” in $x \in S$

(but perhaps not!)

Can have False Positives!

Example Application: Malicious URLs

- A web browser wants to check if an URL entered by an user is malicious or not

Example Application: Malicious URLs

- A web browser wants to check if an URL entered by an user is malicious or not
- Huge set S of malicious URLs - say 10 Million URLs

Example Application: Malicious URLs

- A web browser wants to check if an URL entered by an user is malicious or not
- Huge set S of malicious URLs - say 10 Million URLs (S $\approx$ 500 MB total)

Example Application: Malicious URLs

- A web browser wants to check if an URL entered by an user is malicious or not
- Huge set S of malicious URLs - say 10 Million URLs (S $\approx$ 500 MB total)
    - Don’t want to store huge file on user side!

Example Application: Malicious URLs

- A web browser wants to check if an URL entered by an user is malicious or not
- Huge set S of malicious URLs - say 10 Million URLs (S $\approx$ 500 MB total)
  - Don't want to store huge file on user side!
- Store only a Bloom Filter of S on the user side.

Example Application: Malicious URLs

- A web browser wants to check if an URL entered by an user is malicious or not
- Huge set S of malicious URLs - say 10 Million URLs (S $\approx$ 500 MB total)
  - Don't want to store huge file on user side!
- Store only a Bloom Filter of S on the user side.
- To check if a URL is in S :

Example Application: Malicious URLs

- A web browser wants to check if an URL entered by an user is malicious or not
- Huge set S of malicious URLs - say 10 Million URLs (S $\approx$ 500 MB total)
  - Don't want to store huge file on user side!
- Store only a Bloom Filter of S on the user side.
- To check if a URL is in S :

BF_Search(URL)

Example Application: Malicious URLs

- A web browser wants to check if an URL entered by an user is malicious or not
- Huge set S of malicious URLs - say 10 Million URLs (S $\approx$ 500 MB total)
  - Don’t want to store huge file on user side!
- Store only a Bloom Filter of S on the user side.
- To check if a URL is in S :

BF_Search(URL) $\rightarrow$ “No”

Example Application: Malicious URLs

- A web browser wants to check if an URL entered by an user is malicious or not
- Huge set S of malicious URLs - say 10 Million URLs (S ≈ 500 MB total)
  - Don't want to store huge file on user side!
- Store only a Bloom Filter of S on the user side.
- To check if a URL is in S :

BF_Search(URL) → "No" $\Rightarrow$ URL $\notin$ S

Example Application: Malicious URLs

- A web browser wants to check if an URL entered by an user is malicious or not
- Huge set S of malicious URLs - say 10 Million URLs (S $\approx$ 500 MB total)
  - Don’t want to store huge file on user side!
- Store only a Bloom Filter of S on the user side.
- To check if a URL is in S :

BF_Search(URL) → “No” $\Rightarrow$ URL $\notin$ S

visit the website

Example Application: Malicious URLs

- A web browser wants to check if an URL entered by an user is malicious or not
- Huge set S of malicious URLs - say 10 Million URLs (S $\approx$ 500 MB total)
  - Don't want to store huge file on user side!
- Store only a Bloom Filter of S on the user side.
- To check if a URL is in S :

BF_Search(URL)

"No" $\Rightarrow$ URL $\notin$ S
visit the website

"Probably Yes"

Example Application: Malicious URLs

- A web browser wants to check if an URL entered by an user is malicious or not
- Huge set S of malicious URLs - say 10 Million URLs (S ≈ 500 MB total)
  - Don't want to store huge file on user side!
- Store only a Bloom Filter of S on the user side.
- To check if a URL is in S :

BF_Search(URL) → "No" ⇒ URL $\notin$ S
visit the website

BF_Search(URL) → "Probably Yes" ⇒ "Probably" URL $\in$ S

Example Application: Malicious URLs

- A web browser wants to check if an URL entered by an user is malicious or not
- Huge set S of malicious URLs - say 10 Million URLs (S ≈ 500 MB total)
  - Don’t want to store huge file on user side!
- Store only a Bloom Filter of S on the user side.
- To check if a URL is in S :

BF_Search(URL)

“No” ⇒ URL ∉ S
visit the website

“Probably Yes” ⇒ “Probably” URL ∈ S
Issue warning to the user

Example Application: Malicious URLs

- A web browser wants to check if an URL entered by an user is malicious or not
- Huge set S of malicious URLs - say 10 Million URLs (S ≈ 500 MB total)
  - Don't want to store huge file on user side!
- Store only a Bloom Filter of S on the user side.
- To check if a URL is in S :

BF_Search(URL) → "No" ⇒ URL $\notin$ S
visit the website

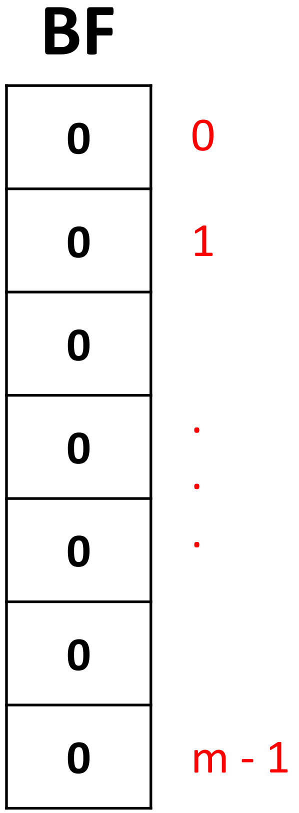
BF_Search(URL) → "Probably Yes" ⇒ "Probably" URL $\in$ S
Issue warning to the user

- Can accomplish this using a **BF** of size ≈ 10 MB, with False Positive rate just 2%

Bloom Filter

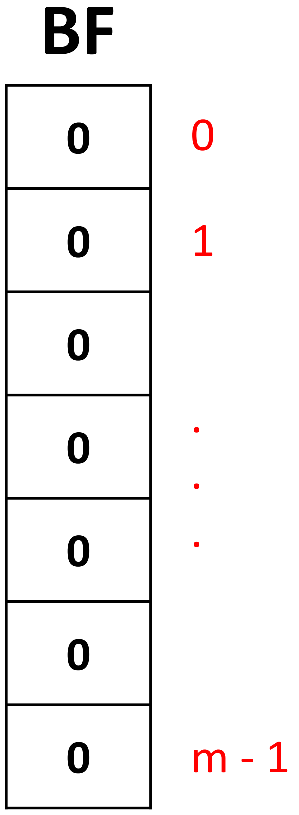
Bloom Filter

- Array **BF**[0 … m-1] of m bits, initially all 0’s



Bloom Filter

- Array **BF**[0 … m-1] of m bits, initially all 0’s
- t **independent** hash functions $h_1, h_2, \ldots, h_t$



Bloom Filter

- Array **BF**[0 … m-1] of m bits, initially all 0's

- t **independent** hash functions $h_1, h_2, \ldots, h_t$

  $h_i : U \to \{0, 1, \ldots, m-1\}$

BF	
0	0
0	1
0	
0	.
0	.
0	.
0	m - 1

Bloom Filter

- Array **BF**[0 … m-1] of m bits, initially all 0’s

- t **independent** hash functions $h_1, h_2, \ldots, h_t$

  $h_i : U \to \{0, 1, \ldots, m-1\}$

  h_i satisfying **SUHA**

BF

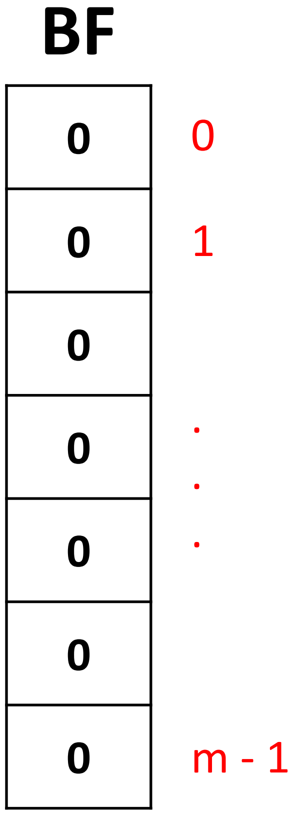
0	0
0	1
0	
0	.
0	.
0	.
0	m - 1

Bloom Filter

- Array **BF**[0 … m-1] of m bits, initially all 0's
- t **independent** hash functions $h_1, h_2, \ldots, h_t$

  $h_i : U \rightarrow \{0, 1, \ldots, m-1\}$

  h_i satisfying **SUHA**



SUHA: Every element is equally likely to hash into any of the m slots of **BF,** independent of where the other elements have hashed to.

Example BF[0 … 7] with $t = 2$: h_1 and h_2

	BF
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0

Example BF[0 … 7] with $t = 2$: h_1 and h_2

INSERTS

BF

BF	
0	0
0	1
0	2
0	3
0	4
0	5
0	6
0	7

Example BF[0 … 7] with $t = 2$: h_1 and h_2

INSERTS

BF_Insert(x_1)

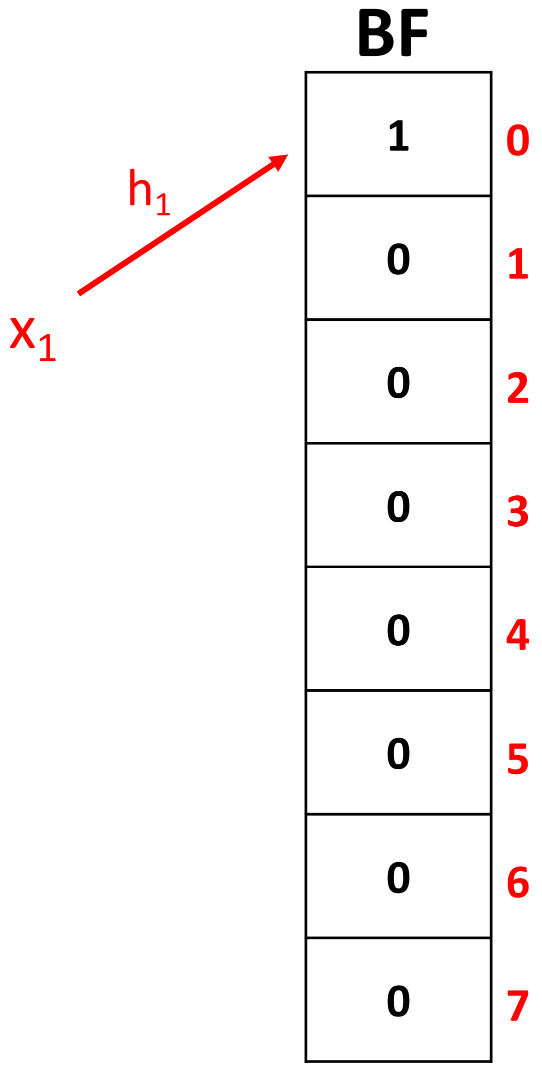
BF

BF	
0	0
0	1
0	2
0	3
0	4
0	5
0	6
0	7

Example BF[0 … 7] with $t = 2$: h_1 and h_2

INSERTS

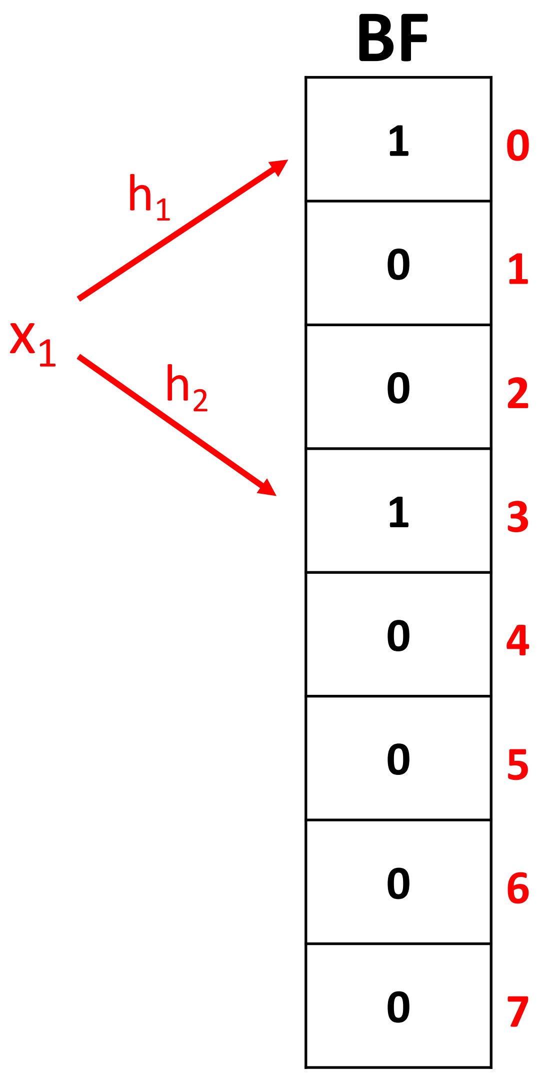
BF_Insert(x_1)



Example BF[0 … 7] with $t = 2$: h_1 and h_2

INSERTS

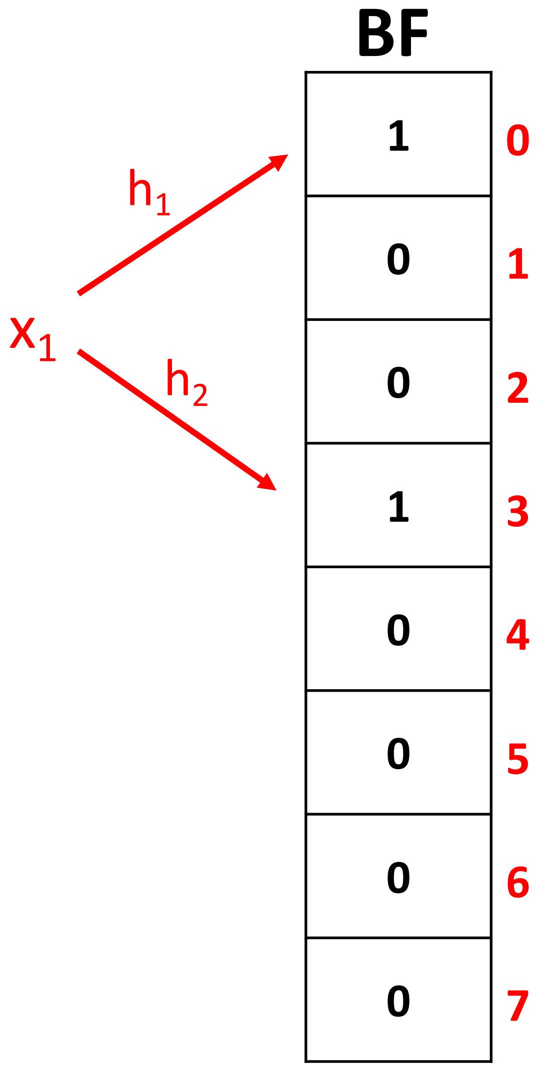
BF_Insert(x_1)



Example BF[0 … 7] with $t = 2$: h_1 and h_2

INSERTS

BF_Insert(x_1)



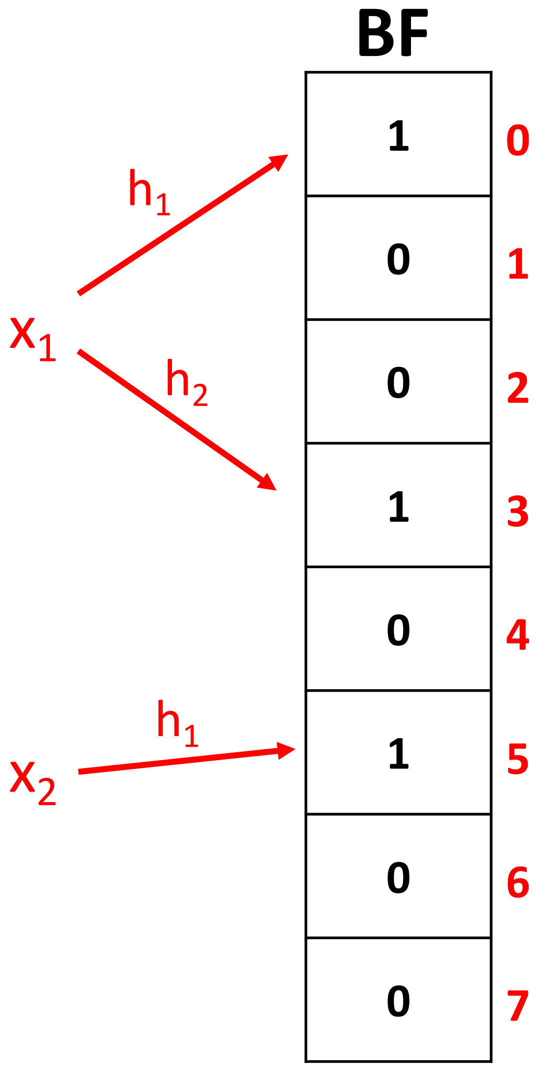
BF_Insert(x_2)

Example BF[0 … 7] with $t = 2$: h_1 and h_2

INSERTS

BF_Insert(x_1)

BF_Insert(x_2)

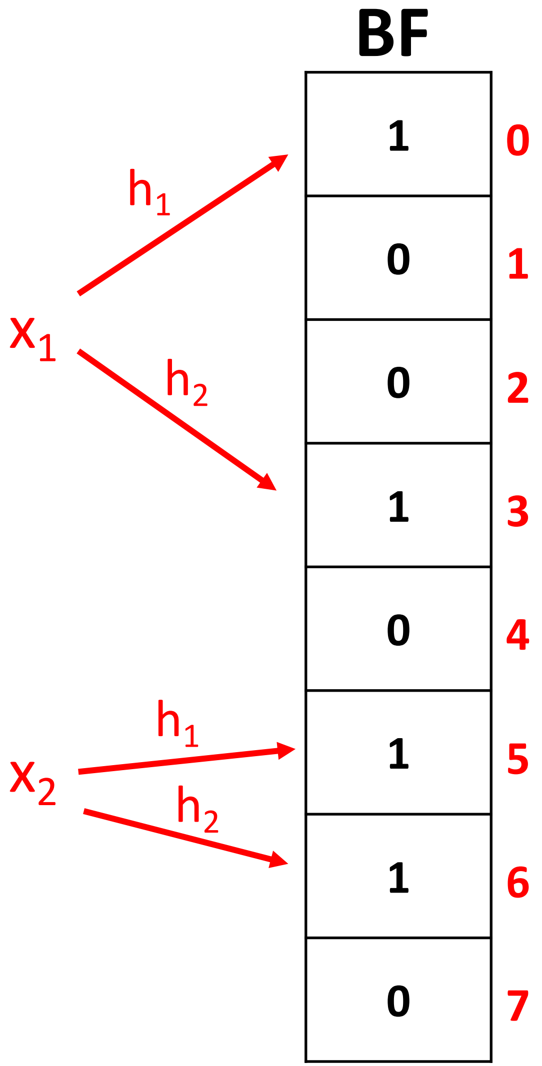


Example BF[0 … 7] with $t = 2$: h_1 and h_2

INSERTS

BF_Insert(x_1)

BF_Insert(x_2)



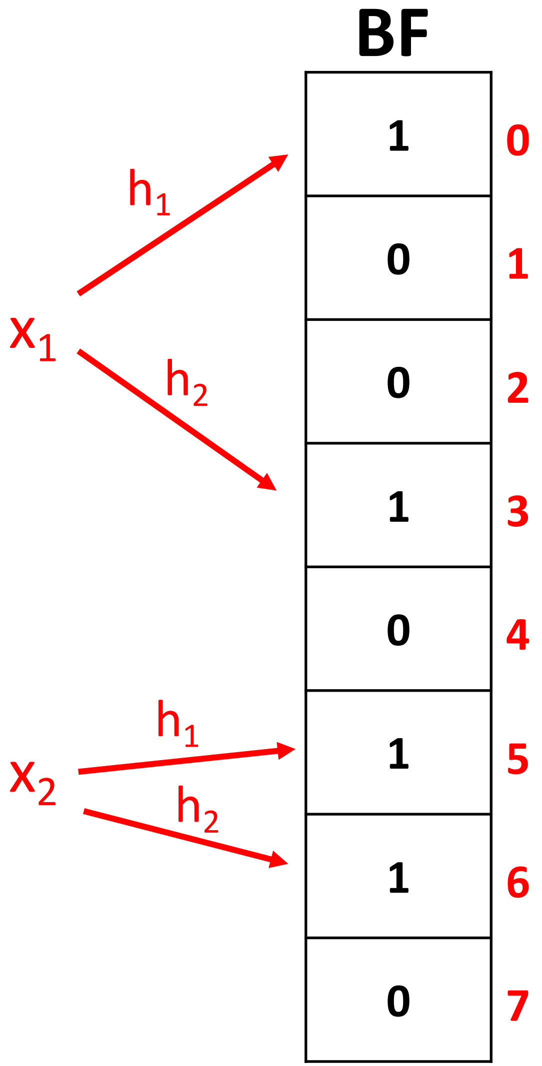
Example BF[0 … 7] with $t = 2$: h_1 and h_2

INSERTS

BF_Insert(x_1)

BF_Insert(x_2)

BF_Insert(x_3)



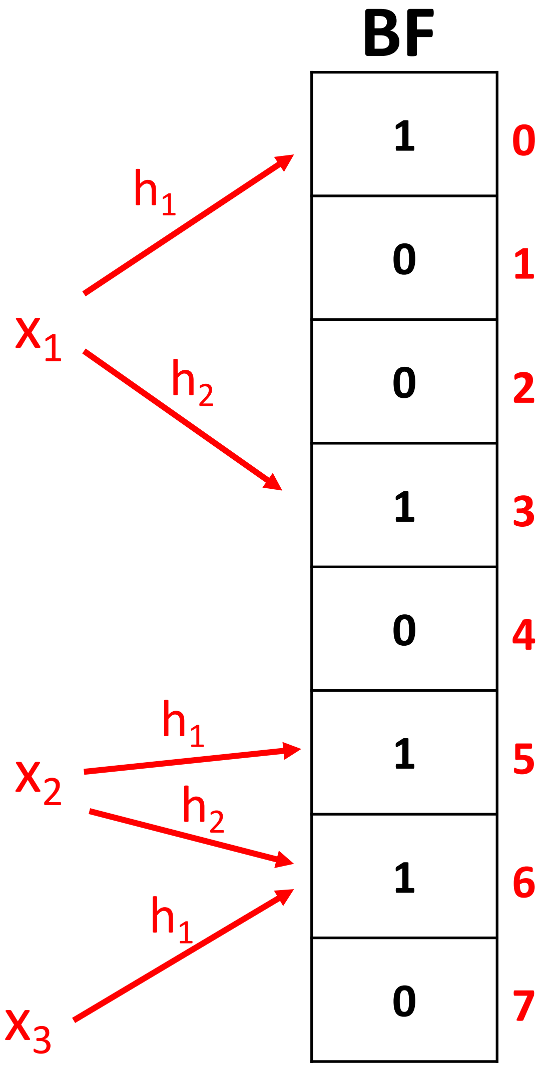
Example BF[0 … 7] with $t = 2$: h_1 and h_2

INSERTS

BF_Insert(x_1)

BF_Insert(x_2)

BF_Insert(x_3)



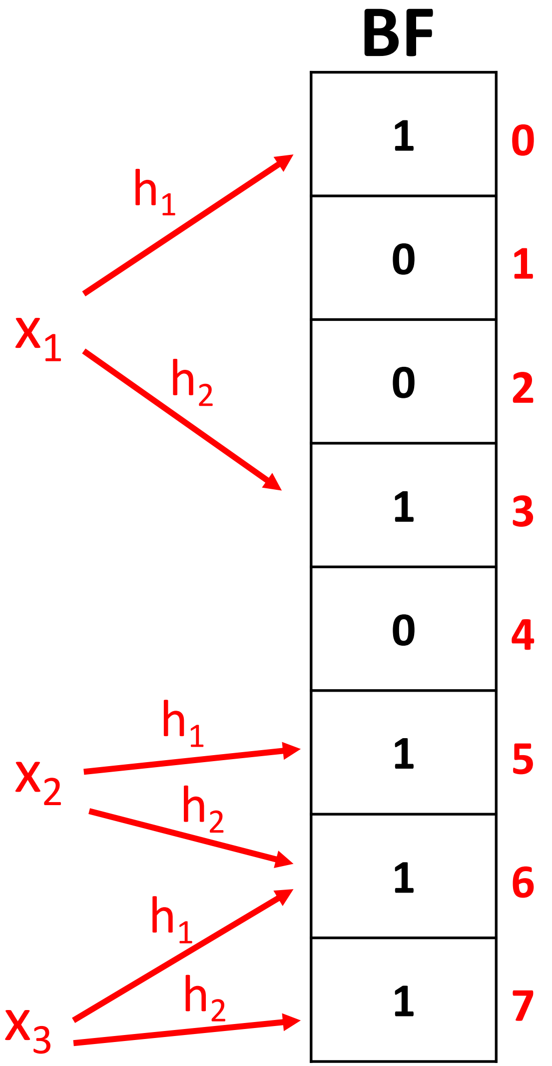
Example BF[0 … 7] with $t = 2$: h_1 and h_2

INSERTS

BF_Insert(x_1)

BF_Insert(x_2)

BF_Insert(x_3)



Example BF[0 … 7] with $t = 2$: h_1 and h_2

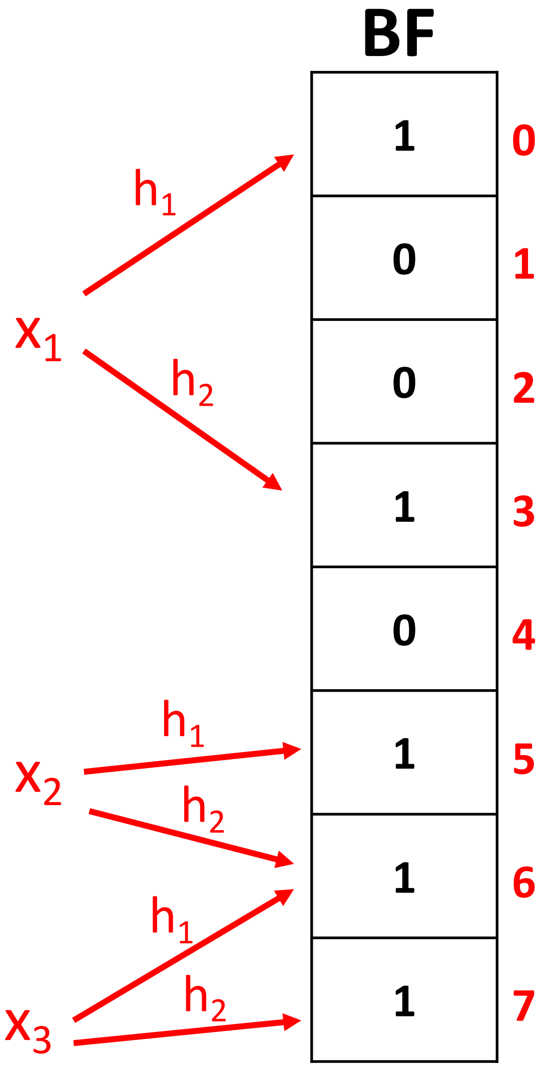
INSERTS

SEARCHES

BF_Insert(x_1)

BF_Insert(x_2)

BF_Insert(x_3)



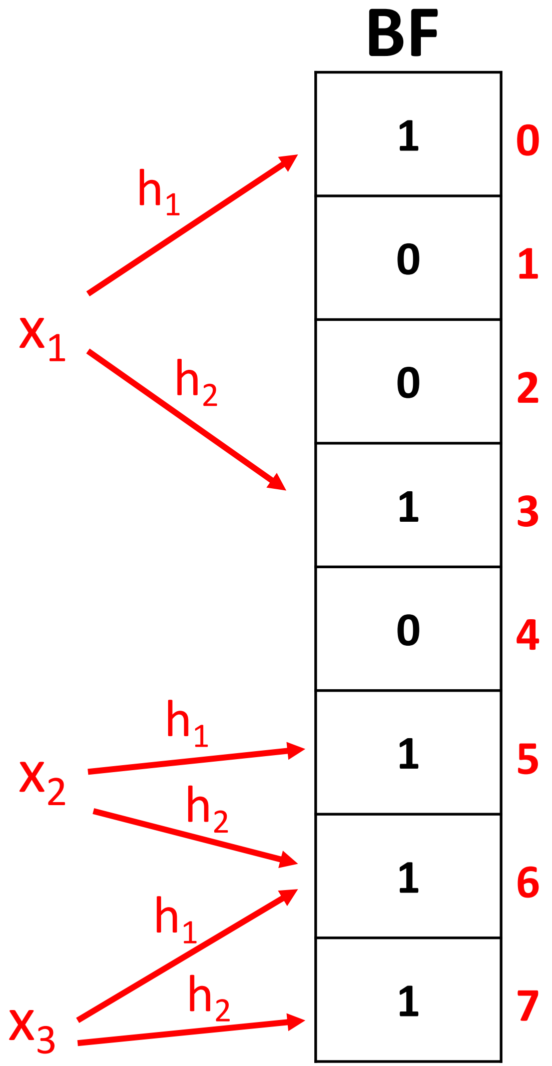
Example BF[0 … 7] with $t = 2$: h_1 and h_2

INSERTS

BF_Insert(x_1)

BF_Insert(x_2)

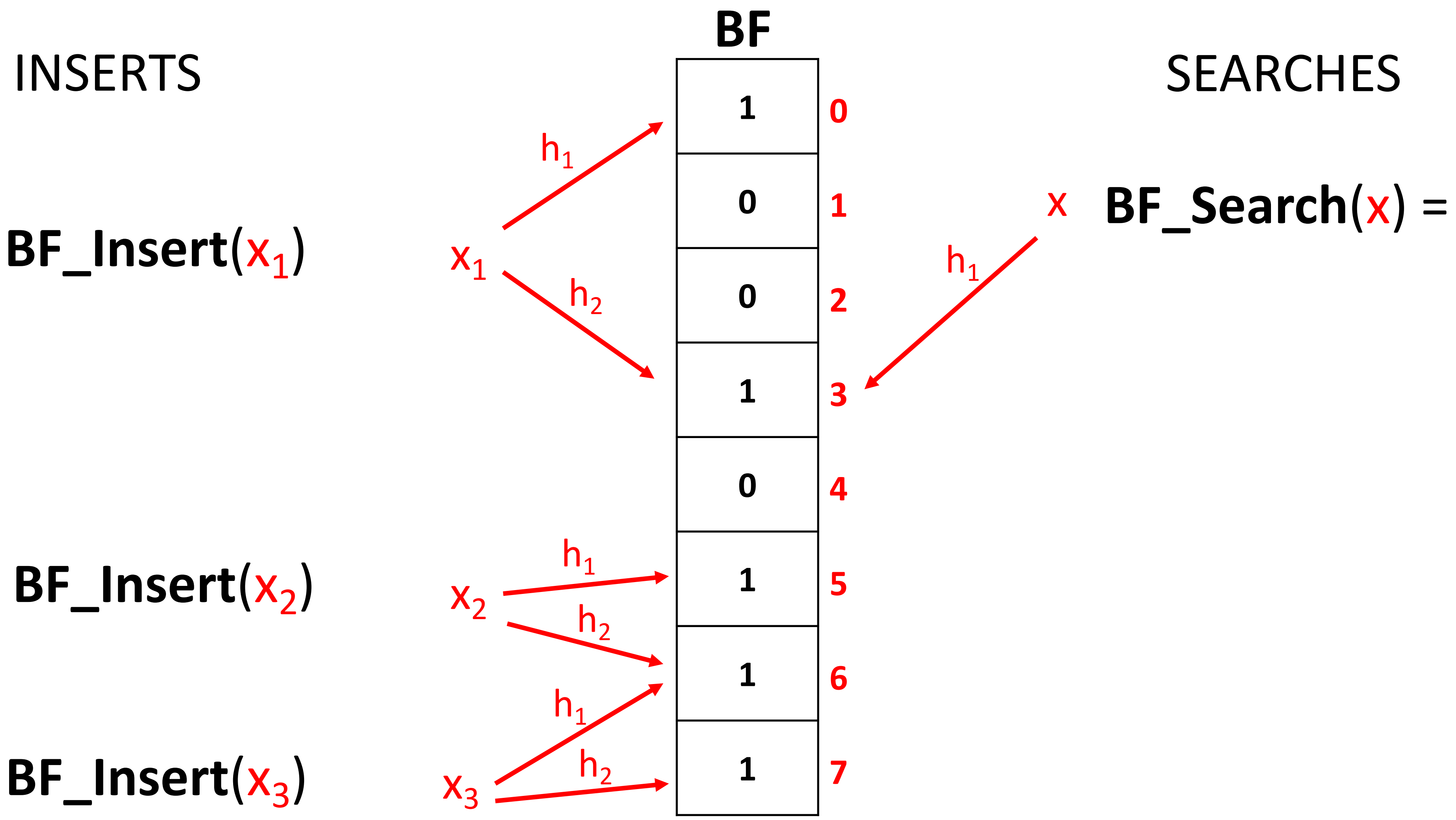
BF_Insert(x_3)



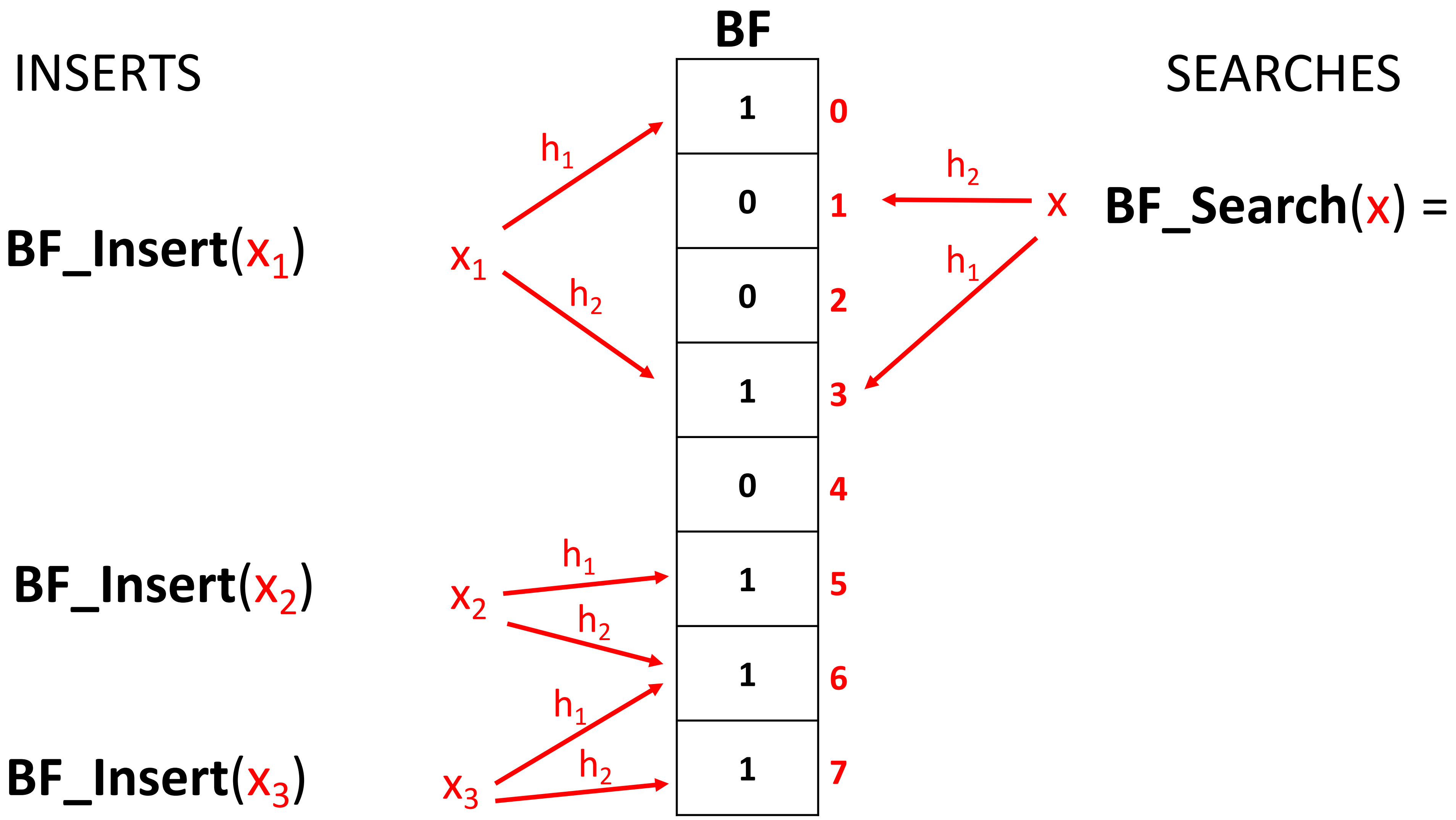
SEARCHES

BF_Search(x) =

Example BF[0 … 7] with $t = 2$: h_1 and h_2



Example BF[0 … 7] with $t = 2$: h_1 and h_2



Example BF[0 … 7] with $t = 2$: h_1 and h_2

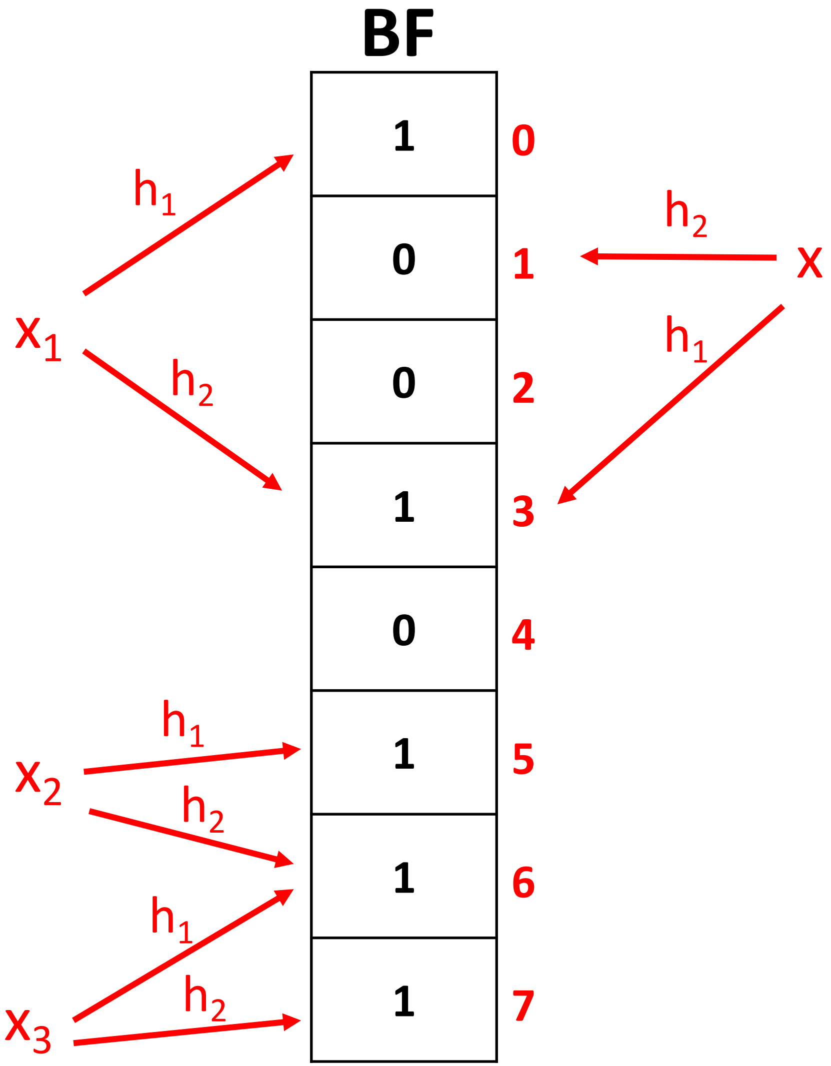
INSERTS

SEARCHES

BF_Insert(x_1)

BF_Insert(x_2)

BF_Insert(x_3)



BF_Search(x) = "No"

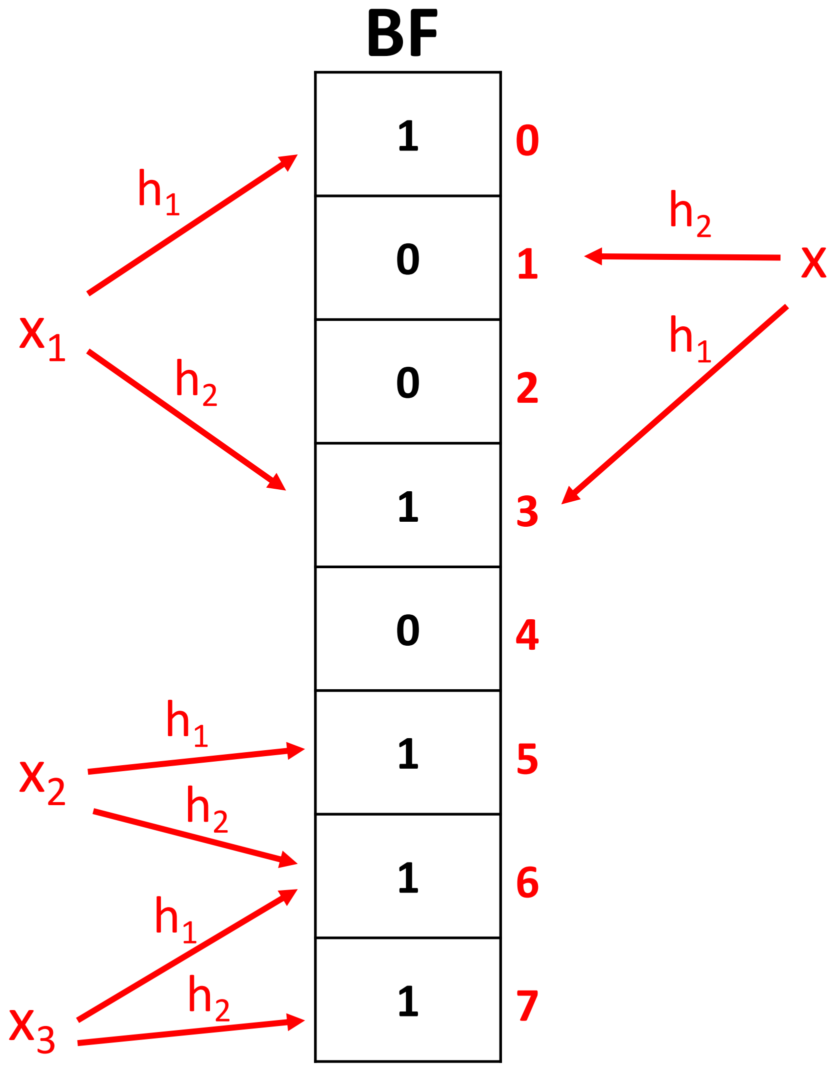
Example BF[0 … 7] with $t = 2$: h_1 and h_2

INSERTS

BF_Insert(x_1)

BF_Insert(x_2)

BF_Insert(x_3)



SEARCHES

BF_Search(x) = “No”

$\Rightarrow x \notin x_1, x_2, x_3$

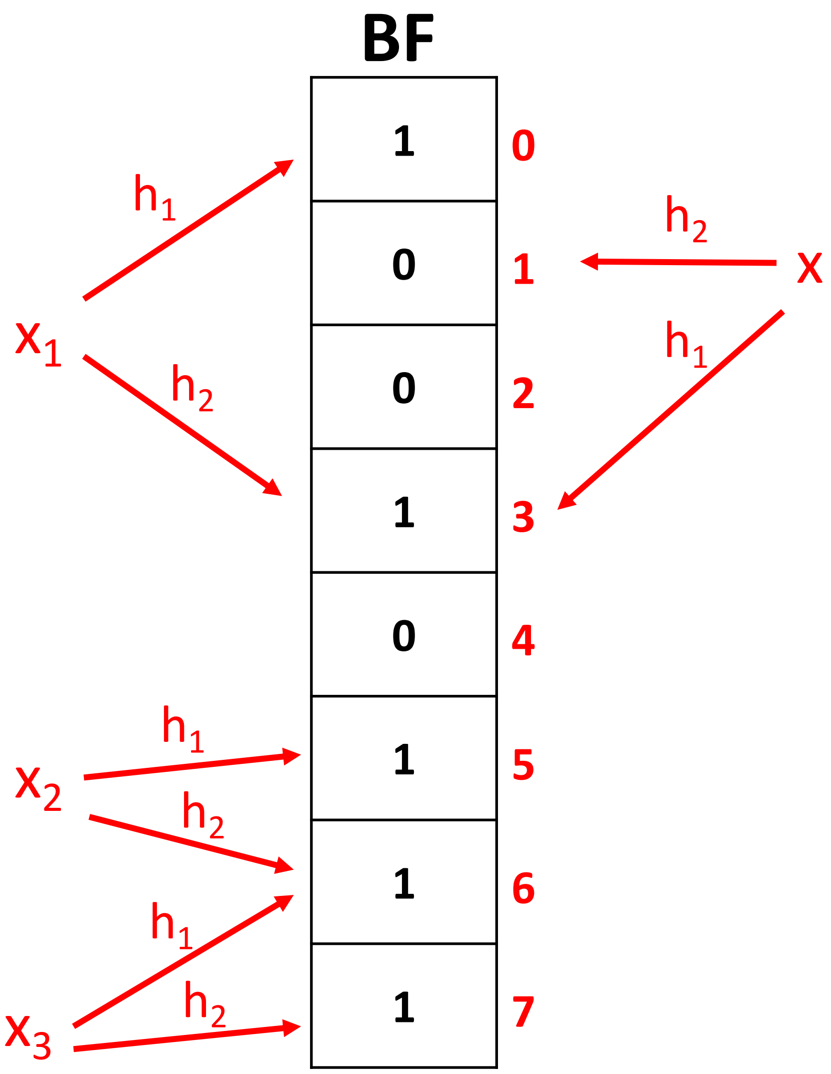
Example BF[0 … 7] with $t = 2$: h_1 and h_2

INSERTS

BF_Insert(x_1)

BF_Insert(x_2)

BF_Insert(x_3)



SEARCHES

BF_Search(x) = "No"

$\Rightarrow x \notin x_1, x_2, x_3$

BF_Search(x') =

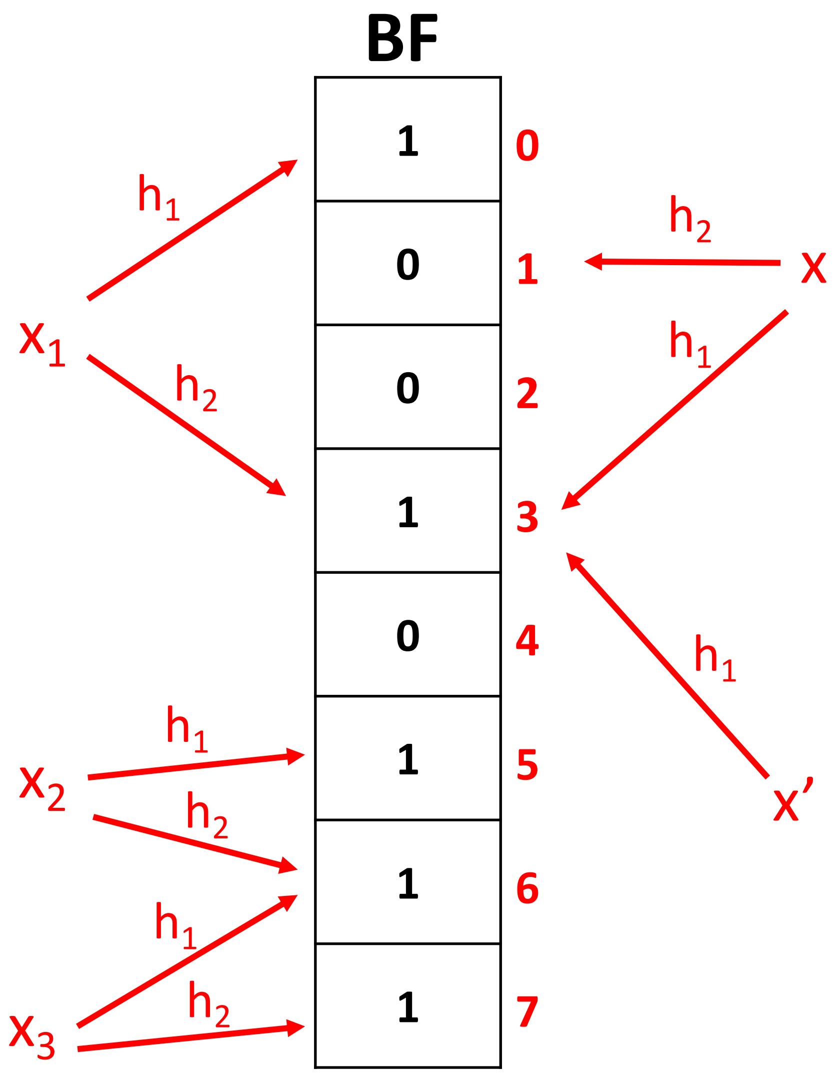
Example BF[0 … 7] with $t = 2$: h_1 and h_2

INSERTS

BF_Insert(x_1)

BF_Insert(x_2)

BF_Insert(x_3)



SEARCHES

BF_Search(x) = "No"

$\Rightarrow x \notin x_1, x_2, x_3$

BF_Search(x') =

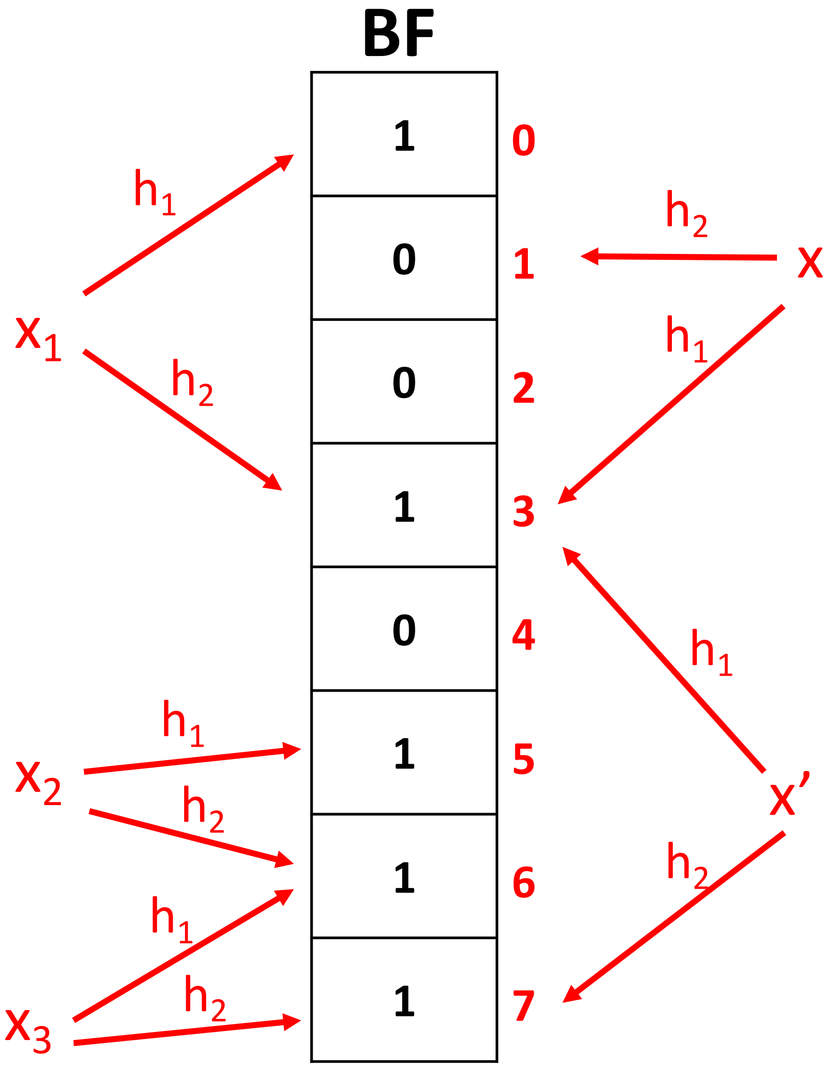
Example BF[0 … 7] with $t = 2$: h_1 and h_2

INSERTS

BF_Insert(x_1)

BF_Insert(x_2)

BF_Insert(x_3)



SEARCHES

BF_Search(x) = "No"

$\Rightarrow x \notin x_1, x_2, x_3$

BF_Search(x') =

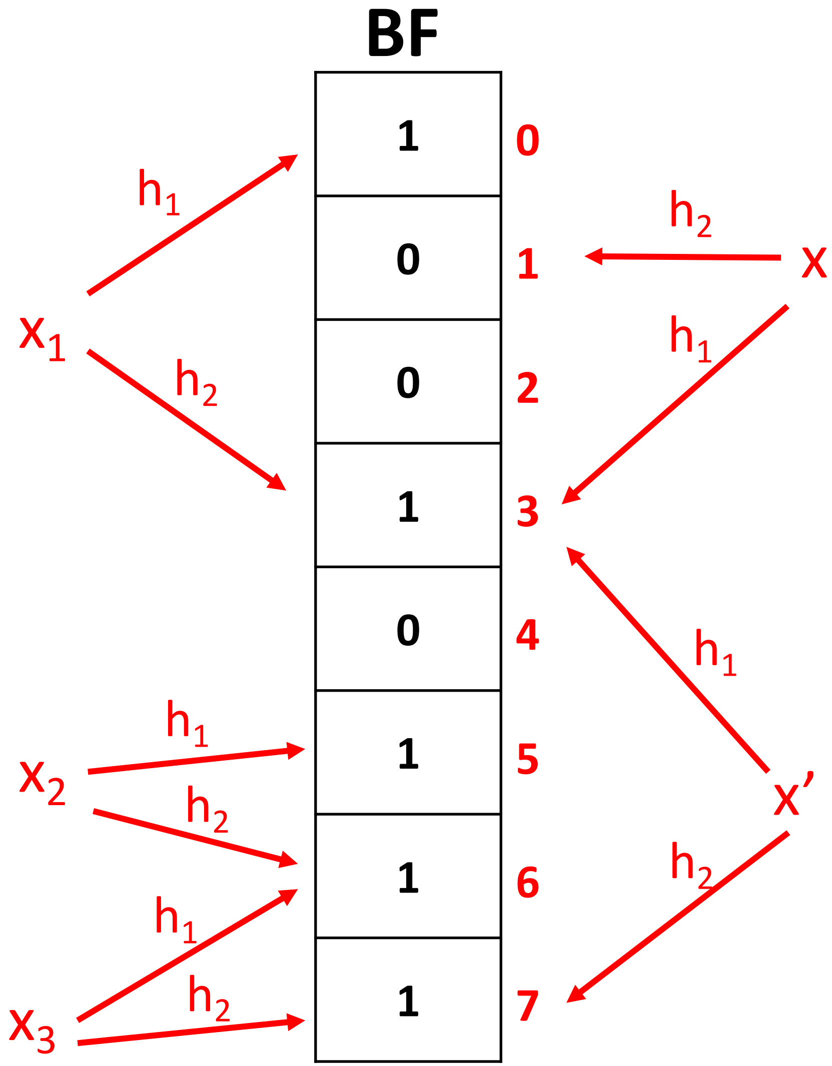
Example BF[0 … 7] with $t = 2$: h_1 and h_2

INSERTS

BF_Insert(x_1)

BF_Insert(x_2)

BF_Insert(x_3)



SEARCHES

BF_Search(x) = "No"

$\Rightarrow x \notin x_1, x_2, x_3$

BF_Search(x') = "Prob. Yes"

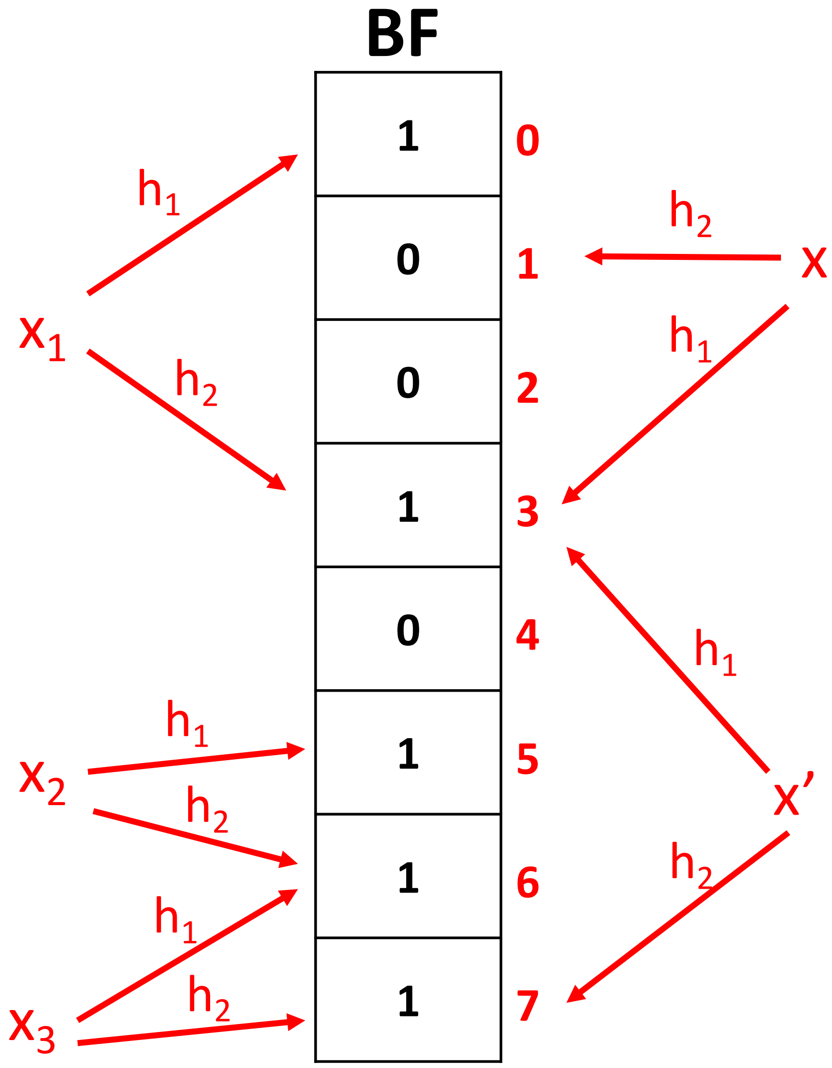
Example BF[0 … 7] with $t = 2$: h_1 and h_2

INSERTS

BF_Insert(x_1)

BF_Insert(x_2)

BF_Insert(x_3)



SEARCHES

BF_Search(x) = “No”

$\Rightarrow x \notin x_1, x_2, x_3$

BF_Search(x’) = “Prob. Yes”

False Positive !

“Fingerprint” of x are the indices $h_1(x), h_2(x), \ldots, h_t(x)$

“Fingerprint” of x are the indices $h_1(x), h_2(x), \ldots, h_t(x)$

BF_Insert(x) [Insert the “fingerprint” of x in **BF**]

"Fingerprint" of x are the indices $h_1(x), h_2(x), \ldots, h_t(x)$

BF_Insert(x) [Insert the "fingerprint" of x in **BF**]

for $i = 1$ to t :

BF[$h_i(x)$] = 1

“Fingerprint” of x are the indices $h_1(x), h_2(x), \ldots, h_t(x)$

BF_Insert(x) [Insert the “fingerprint” of x in **BF**]

for $i = 1$ to t :

    BF[$h_i(x)$] = 1

BF_Search(x) [Search for “fingerprint” of x in **BF**]

"Fingerprint" of x are the indices $h_1(x), h_2(x), \ldots, h_t(x)$

BF_Insert(x) [Insert the "fingerprint" of x in **BF**]

for $i = 1$ to t :

  BF[$h_i(x)$] = 1

BF_Search(x) [Search for "fingerprint" of x in **BF**]

for $i = 1$ to t :

  if **BF**[$h_i(x)$] = 0 then return "No"

"Fingerprint" of x are the indices $h_1(x), h_2(x), \ldots, h_t(x)$

BF_Insert(x) [Insert the "fingerprint" of x in **BF**]

for i = 1 to t :

  BF[$h_i(x)$] = 1

BF_Search(x) [Search for "fingerprint" of x in **BF**]

for i = 1 to t :

  if **BF**[$h_i(x)$] = 0 then return "No"

return "Probably Yes"

Probability of False Positive

Probability of False Positive

Setup:

- Insert x_1, x_2, …, x_n into an empty BF[0 … m-1]
  with t independent hash functions h_1, h_2, …, h_t each satisfying SUHA.

Probability of False Positive

Setup:

- Insert $x_1, x_2, \ldots, x_n$ into an empty BF[0 … m-1] with t independent hash functions $h_1, h_2, \ldots, h_t$ each satisfying SUHA.

- Do **BF_Search**(x) for $x \notin x_1, x_2, \ldots, x_n$

Probability of False Positive

Setup:

- Insert $x_1, x_2, \ldots, x_n$ into an empty BF[0 … m-1] with t independent hash functions $h_1, h_2, \ldots, h_t$ each satisfying SUHA.

- Do **BF_Search**(x) for $x \notin x_1, x_2, \ldots, x_n$

We would like to compute:

- Pr[false positive]

Probability of False Positive

Setup:

- Insert x_1, x_2, …, x_n into an empty BF[0 … m-1]
  with t independent hash functions h_1, h_2, …, h_t each satisfying SUHA.

- Do **BF_Search**(x) for $x \notin x_1, x_2, \ldots, x_n$

We would like to compute:

- Pr[false positive] = Pr [**BF_Search**(x) = "Probably Yes"]

Probability of False Positive

Setup:

- Insert $x_1, x_2, \ldots, x_n$ into an empty BF[0 … m-1]
  with t independent hash functions $h_1, h_2, \ldots, h_t$ each satisfying SUHA.

- Do **BF_Search**(x) for $x \notin x_1, x_2, \ldots, x_n$

We would like to compute:

- Pr[false positive] = Pr [**BF_Search**(x) = "Probably Yes"]

We first compute:

- For an arbitrary index i of **BF**, Pr[**BF**[i] = 0] after inserting $x_1, x_2, \ldots, x_n$

Probability of False Positive

Consider an arbitrary index i of the **BF**

Probability of False Positive

Consider an arbitrary index i of the **BF**

After inserting $x_1, x_2, \ldots, x_n$,

$\Pr\left[\mathbf{BF}[i] = 0\right] =$

Probability of False Positive

Consider an arbitrary index i of the **BF**

After inserting $x_1, x_2, \ldots, x_n$,

$$\Pr\left[\mathbf{BF}[i] = 0\right] = \Pr\left[\bigcap_{k=1}^{n} \bigcap_{j=1}^{t} h_j(x_k) \neq i\right]$$

Probability of False Positive

Consider an arbitrary index i of the **BF**

After inserting $x_1, x_2, \ldots, x_n$,

$$\Pr\left[\mathbf{BF}[i] = 0\right] = \Pr\left[\bigcap_{k=1}^{n} \bigcap_{j=1}^{t} h_j(x_k) \neq i\right]$$

By SUHA and independence of h_i s : these events are mutually independent!

Probability of False Positive

Consider an arbitrary index i of the **BF**

After inserting $x_1, x_2, \ldots, x_n$,

$$\Pr\Big[\mathbf{BF}[i] = 0\Big] = \Pr\Big[\bigcap_{k=1}^{n} \bigcap_{j=1}^{t} h_j(x_k) \neq i\Big]$$

$$= \prod_{k=1}^{n} \prod_{j=1}^{t} \Pr\Big[h_j(x_k) \neq i\Big]$$

By SUHA and independence of h_i s : these events are mutually independent!

Probability of False Positive

Consider an arbitrary index i of the **BF**

After inserting $x_1, x_2, \ldots, x_n$,

$$\Pr\left[\mathbf{BF}[i] = 0\right] = \Pr\left[\bigcap_{k=1}^{n} \bigcap_{j=1}^{t} h_j(x_k) \neq i\right]$$

$$= \prod_{k=1}^{n} \prod_{j=1}^{t} \underbrace{\Pr\left[h_j(x_k) \neq i\right]}_{1 - 1/m}$$

Because of SUHA !

Probability of False Positive

Consider an arbitrary index i of the **BF**

After inserting $x_1, x_2, \ldots, x_n$,

$$\Pr\left[\mathbf{BF}[i] = 0\right] = \Pr\left[\bigcap_{k=1}^{n} \bigcap_{j=1}^{t} h_j(x_k) \neq i\right]$$

$$= \prod_{k=1}^{n} \prod_{j=1}^{t} \underbrace{\Pr\left[h_j(x_k) \neq i\right]}_{1 - 1/m}$$

Because of SUHA !

$$= (1 - 1/m)^{nt}$$

Probability of False Positive

Consider an arbitrary index i of the **BF**

After inserting $x_1, x_2, \ldots, x_n$,

$\Pr\left[\mathbf{BF}[i] = 0\right] = (1 - 1/m)^{nt}$

Probability of False Positive

Consider an arbitrary index i of the **BF**

After inserting $x_1, x_2, \ldots, x_n$,

$\Pr\left[\mathbf{BF}[i] = 0\right] = (1 - 1/m)^{nt}$ $\left[1 - y \approx e^{-y} \quad \text{(for small } y = 1/m\text{)}\right]$

Probability of False Positive

Consider an arbitrary index i of the **BF**

After inserting $x_1, x_2, \ldots, x_n$,

$\Pr\left[\mathbf{BF}[i] = 0\right] = (1 - 1/m)^{nt}$ $\left[1 - y \approx e^{-y}\right.$ (for small $y = 1/m$) $\left.\right]$

$\approx (e^{-1/m})^{nt}$

Probability of False Positive

Consider an arbitrary index i of the **BF**

After inserting $x_1, x_2, \ldots, x_n$,

$\Pr\left[\mathbf{BF}[i] = 0\right] = (1 - 1/m)^{nt}$ $\left[1 - y \approx e^{-y} \quad \text{(for small } y = 1/m\text{)}\right]$

$\approx (e^{-1/m})^{nt} = e^{-nt/m}$

Probability of False Positive

Consider an arbitrary index i of the **BF**

After inserting $x_1, x_2, \ldots, x_n$,

$\Pr\left[\mathbf{BF}[i] = 0\right] = (1 - 1/m)^{nt}$ $\left[1 - y \approx e^{-y} \quad \text{(for small } y = 1/m\text{)}\right]$

$\approx (e^{-1/m})^{nt} = e^{-nt/m}$

$\Pr\left[\mathbf{BF}[i] = 1\right] =$

Probability of False Positive

Consider an arbitrary index i of the **BF**

After inserting $x_1, x_2, \ldots, x_n,$

$\Pr\left[\mathbf{BF}[i] = 0\right] = (1 - 1/m)^{nt}$ $\left[1 - y \approx e^{-y} \quad (\text{for small } y = 1/m)\right]$

$\approx (e^{-1/m})^{nt} = e^{-nt/m}$

$\Pr\left[\mathbf{BF}[i] = 1\right] = 1 - \Pr\left[\mathbf{BF}[i] = 0\right]$

Probability of False Positive

Consider an arbitrary index i of the **BF**

After inserting $x_1, x_2, \ldots, x_n,$

$$\Pr\big[\mathbf{BF}[i] = 0\big] = (1 - 1/m)^{nt} \qquad \big[1 - y \approx e^{-y} \quad \text{(for small } y = 1/m\text{)}\big]$$

$$\approx (e^{-1/m})^{nt} = e^{-nt/m}$$

$$\Pr\big[\mathbf{BF}[i] = 1\big] = 1 - \Pr\big[\mathbf{BF}[i] = 0\big]$$

$$\approx 1 - e^{-nt/m}$$

Probability of False Positive

Lemma 1: After inserting x_1, x_2, …, x_n into a **BF** of size m with t hash functions,

for every index i, $\Pr\left[\mathbf{BF}[i] = 1\right] \approx 1 - e^{-nt/m}$

Probability of False Positive

Lemma 1: After inserting x_1, x_2, …, x_n into a **BF** of size m with t hash functions, for every index i, $\Pr\left[\mathbf{BF}[i] = 1\right] \approx \underbrace{1 - e^{-nt/m}}_{q}$

Probability of a False Positive

Do **BF_Search**(x) for $x \notin x_1, x_2, \ldots, x_n$

Pr[false positive] = Pr [**BF_Search**(x) = "Probably Yes"]

Probability of a False Positive

Do **BF_Search**(x) for $x \notin x_1, x_2, \ldots, x_n$

Pr[false positive] = Pr [**BF_Search**(x) = "Probably Yes"]

$$= \Pr\left[\mathbf{BF}[h_1(x)] = 1 \cap \mathbf{BF}[h_2(x)] = 1 \cap \ldots \cap \mathbf{BF}[h_t(x)] = 1\right]$$

Probability of a False Positive

Do **BF_Search**(x) for $x \notin x_1, x_2, \ldots, x_n$

Pr[false positive] = Pr [**BF_Search**(x) = "Probably Yes"]

$$= \Pr\Big[\mathbf{BF}[h_1(x)] = 1 \cap \mathbf{BF}[h_2(x)] = 1 \cap \ldots. \cap \mathbf{BF}[h_t(x)] = 1\Big]$$

Not independent !

Probability of a False Positive

Do **BF_Search**(x) for $x \notin x_1, x_2, \ldots, x_n$

Pr[false positive] = Pr [**BF_Search**(x) = "Probably Yes"]

$$= \Pr\left[\mathbf{BF}[h_1(x)] = 1 \cap \mathbf{BF}[h_2(x)] = 1 \cap \ldots \cap \mathbf{BF}[h_t(x)] = 1\right]$$

Probability of a False Positive

Do **BF_Search**(x) for $x \notin x_1, x_2, \ldots, x_n$

Pr[false positive] = Pr [**BF_Search**(x) = "Probably Yes"]

$$= \Pr\left[\mathbf{BF}[h_1(x)] = 1 \cap \mathbf{BF}[h_2(x)] = 1 \cap \ldots \cap \mathbf{BF}[h_t(x)] = 1\right]$$

$$\approx \Pr\left[\mathbf{BF}[h_1(x)] = 1\right] . \Pr\left[\mathbf{BF}[h_2(x)] = 1\right] . \ldots . \Pr\left[\mathbf{BF}[h_t(x)] = 1\right]$$

Probability of a False Positive

Do **BF_Search**(x) for $x \notin x_1, x_2, \ldots, x_n$

Pr[false positive] $=$ Pr [**BF_Search**(x) = "Probably Yes"]

$= \Pr\left[\mathbf{BF}[h_1(x)] = 1 \cap \mathbf{BF}[h_2(x)] = 1 \cap \ldots \cap \mathbf{BF}[h_t(x)] = 1\right]$

$\approx \Pr\left[\mathbf{BF}[i_1] = 1\right] . \Pr\left[\mathbf{BF}[i_2] = 1\right] . \ldots . \Pr\left[\mathbf{BF}[i_t] = 1\right]$

Probability of a False Positive

Do **BF_Search**(x) for $x \notin x_1, x_2, \ldots, x_n$

Pr[false positive] = Pr [**BF_Search**(x) = "Probably Yes"]

$$= \Pr\left[\mathbf{BF}[h_1(x)] = 1 \cap \mathbf{BF}[h_2(x)] = 1 \cap \ldots \cap \mathbf{BF}[h_t(x)] = 1\right]$$

$$\approx \underbrace{\Pr\left[\mathbf{BF}[i_1] = 1\right]}_{q} \cdot \underbrace{\Pr\left[\mathbf{BF}[i_2] = 1\right]}_{q} \cdot \ldots \cdot \underbrace{\Pr\left[\mathbf{BF}[i_t] = 1\right]}_{q}$$

By Lemma 1 !

Probability of a False Positive

Do **BF_Search**(x) for $x \notin x_1, x_2, \ldots, x_n$

Pr[false positive] = Pr [**BF_Search**(x) = "Probably Yes"]

$$= \Pr\left[\mathbf{BF}[h_1(x)] = 1 \cap \mathbf{BF}[h_2(x)] = 1 \cap \ldots \cap \mathbf{BF}[h_t(x)] = 1\right]$$

$$\approx \underbrace{\Pr\left[\mathbf{BF}[i_1] = 1\right]}_{q} \cdot \underbrace{\Pr\left[\mathbf{BF}[i_2] = 1\right]}_{q} \cdot \ldots \cdot \underbrace{\Pr\left[\mathbf{BF}[i_t] = 1\right]}_{q}$$

By Lemma 1 !

$$\approx q^t$$

Probability of a False Positive

Do **BF_Search**(x) for $x \notin x_1, x_2, \ldots, x_n$

Pr[false positive] = Pr [**BF_Search**(x) = "Probably Yes"]

$$= \Pr\left[\mathbf{BF}[h_1(x)] = 1 \cap \mathbf{BF}[h_2(x)] = 1 \cap \ldots \cap \mathbf{BF}[h_t(x)] = 1\right]$$

$$\approx \underbrace{\Pr\left[\mathbf{BF}[i_1] = 1\right]}_{q} \cdot \underbrace{\Pr\left[\mathbf{BF}[i_2] = 1\right]}_{q} \cdot \ldots \cdot \underbrace{\Pr\left[\mathbf{BF}[i_t] = 1\right]}_{q}$$

By Lemma 1 !

$$\approx q^t = \left(1 - e^{-nt/m}\right)^t$$

Probability of a False Positive

Pr[false positive] $\approx \left(1 - e^{-nt/m}\right)^t$

Probability of a False Positive

Pr[false positive] $\approx \left(1 - e^{-nt/m}\right)^t$

$$\frac{m}{n}$$

Probability of a False Positive

$$\Pr[\text{false positive}] \approx \left(1 - e^{-nt/m}\right)^t$$

$$\frac{m}{n}$$

# of elements

Probability of a False Positive

$$\Pr[\text{false positive}] \approx \left(1 - e^{-nt/m}\right)^t$$

# of bits in BF

$$\frac{m}{n}$$

# of elements

Probability of a False Positive

Pr[false positive] $\approx \left(1 - e^{-nt/m}\right)^{t}$

# of bits in BF

$\frac{m}{n}$ (# of bits per element)

# of elements

Probability of a False Positive

Pr[false positive] $\approx \left(1 - e^{-nt/m}\right)^t$

$\frac{m}{n}$ (# of bits per element)

Probability of a False Positive

Pr[false positive] $\approx \left(1 - e^{-nt/m}\right)^t$

Fix the ratio $\frac{m}{n}$ (# of bits per element)

Probability of a False Positive

Pr[false positive] $\approx \left(1 - e^{-nt/m}\right)^t$

Fix the ratio $\frac{m}{n}$ (# of bits per element)

Find t (i.e. # of hash functions) which minimizes Pr[false positive]

Probability of a False Positive

Pr[false positive] ≈ $(1 - e^{-nt/m})^t$

Fix the ratio $\frac{m}{n}$ (# of bits per element)

Find t (i.e. # of hash functions) which minimizes Pr[false positive]

Find derivative of $(1 - e^{-nt/m})^t$ w.r.t t and set it to 0

Probability of a False Positive

Pr[false positive] $\approx \left(1 - e^{-nt/m}\right)^t$

Fix the ratio $\frac{m}{n}$ (# of bits per element)

Find t (i.e. # of hash functions) which minimizes Pr[false positive]

Optimal $t = (\log_e 2)\,\frac{m}{n} = (0.69)\,\frac{m}{n}$

Probability of a False Positive

Pr[false positive] $\approx \left(1 - e^{-nt/m}\right)^t \;\; = \;\; 0.62^{\frac{m}{n}}$

with optimal t

Fix the ratio $\frac{m}{n}$ (# of bits per element)

Find t (i.e. # of hash functions) which minimizes Pr[false positive]

Optimal $t = (\log_e 2)\,\frac{m}{n} = (0.69)\,\frac{m}{n}$

Example

- Want a Bloom Filter for a set S of $n = 10$ Million URLs

Example

- Want a Bloom Filter for a set S of n = 10 Million URLs

- Can allocate 8 bits per URL

Example

- Want a Bloom Filter for a set S of n = 10 Million URLs

Much less than space required to store 1 full URL

- Can allocate 8 bits per URL

Example

- Want a Bloom Filter for a set S of n = 10 Million URLs

- Can allocate 8 bits per URL

Example

- Want a Bloom Filter for a set S of n = 10 Million URLs

- Can allocate 8 bits per URL

  Size of BF = m

Example

- Want a Bloom Filter for a set S of n = 10 Million URLs

- Can allocate 8 bits per URL

  Size of BF = $m = 8n = 8 * 10$ Million bits $\approx$ 10 MB

Example

- Want a Bloom Filter for a set S of n = 10 Million URLs

- Can allocate 8 bits per URL

  Size of BF = $m = 8n = 8 * 10$ Million bits $\approx$ 10 MB

  The t that minimizes Pr[false positive] is:

Example

- Want a Bloom Filter for a set S of n = 10 Million URLs

- Can allocate 8 bits per URL

  Size of BF = m = 8n = 8 * 10 Million bits ≈ 10 MB

  The t that minimizes Pr[false positive] is:

  $t \approx (0.69)\,\frac{m}{n}$ hash functions

Example

- Want a Bloom Filter for a set S of $n = 10$ Million URLs

- Can allocate 8 bits per URL

Size of BF = $m = 8n = 8 * 10$ Million bits $\approx$ 10 MB

The t that minimizes Pr[false positive] is:

$$t \approx (0.69)\frac{m}{n} = 0.69 * 8 = 5.52 \text{ hash functions}$$

Example

- Want a Bloom Filter for a set S of n = 10 Million URLs

- Can allocate 8 bits per URL

Size of BF = m = 8n = 8 * 10 Million bits ≈ 10 MB

The t that minimizes Pr[false positive] is:

$$t \approx (0.69)\frac{m}{n} = 0.69 * 8 = 5.52 \text{ hash functions}$$

With this t, Pr[false positive] ≈ $0.62^{\frac{m}{n}}$

Example

- Want a Bloom Filter for a set S of $n = 10$ Million URLs

- Can allocate 8 bits per URL

Size of BF = $m = 8n = 8 * 10$ Million bits ≈ 10 MB

The t that minimizes Pr[false positive] is:

$$t \approx (0.69)\frac{m}{n} = 0.69 * 8 = 5.52 \text{ hash functions}$$

With this t, Pr[false positive] $\approx 0.62^{\frac{m}{n}} = 0.62^{8}$

Example

- Want a Bloom Filter for a set S of n = 10 Million URLs

- Can allocate 8 bits per URL

Size of BF = $m = 8n = 8 * 10$ Million bits $\approx$ 10 MB

The t that minimizes Pr[false positive] is:

$$t \approx (0.69)\,\frac{m}{n} = 0.69 * 8 = 5.52 \text{ hash functions}$$

With this t, Pr[false positive] $\approx 0.62^{\frac{m}{n}} = 0.62^8 \approx 2\,\%$!!