Bloom filter

Assignment 2
Algorithms for Big Data

Introduction

Bloom filters are probabilistic data structures designed for fast and memory-efficient membership testing. They are widely used in various applications where fast lookups are crucial and false positives can be tolerated.

Features:

- Space Efficiency: Bloom filters offer high space efficiency by using a compact array of bits to represent the presence of elements.

- Unlike traditional data structures, they do not store the actual elements, resulting in lower memory requirements. ● Probabilistic Nature: Bloom filters may yield false positives but not false negatives.

Confusion Matrix (in example of lookup function)

True Positive (TP): Correctly predicted as positive.

- Correctly predicting that a word exists in the document

False Positive (FP): Incorrectly predicted as positive.

- Incorrectly predicting that a word exists in the document

True Negative (TN): Correctly predicted as negative.

- Correctly predicting that a word does not exist in the document

False Negative (FN): Incorrectly predicted as negative.

- Incorrectly predicting that a word does not exist in the document

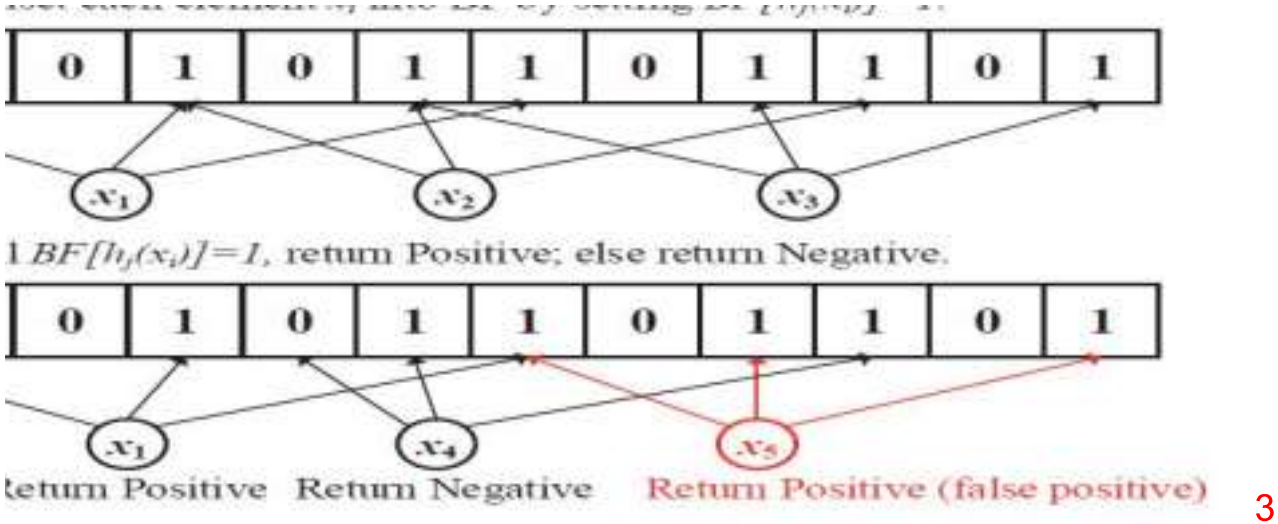
Applications (where FPs are acceptable)

1. **Membership Testing:**
   Test whether an element is "possibly" a member of a set or "definitely" not.
2. **Caching:**
   determine if a requested item is present in a cache or not. Filter out items that are definitely not in the cache. 3. **Web Search Engines:**
   Filter out documents that have already been indexed.
4. **Databases:**
   Determining if a record exists in a database
5. **Network Protocols:**
   Filtering out malicious IP addresses. Google Chrome used a Bloom filter to identify malicious URLs
6. **DNA Sequence Analysis:**
   De-duplication of sequences, filtering out redundant or repeated sequences, and identifying potentially novel or unique sequences.

Implementation

1. **Hashing**
   Choose k independent hash functions, each of which maps an input element to one of the m bit positions. The output of these hash functions should have a uniform distribution across the bit array.
2. **Insert**
   To insert an element into the Bloom filter, apply each hash function to the element and set the corresponding bits in the bit array to 1.
3. **lookup**
   Apply each hash function to the element and check if the corresponding bits in the bit array are all set to 1. If any of the bits are 0, the element is definitely not in the set. If all bits are 1, it is highly probable that the element is in the set.

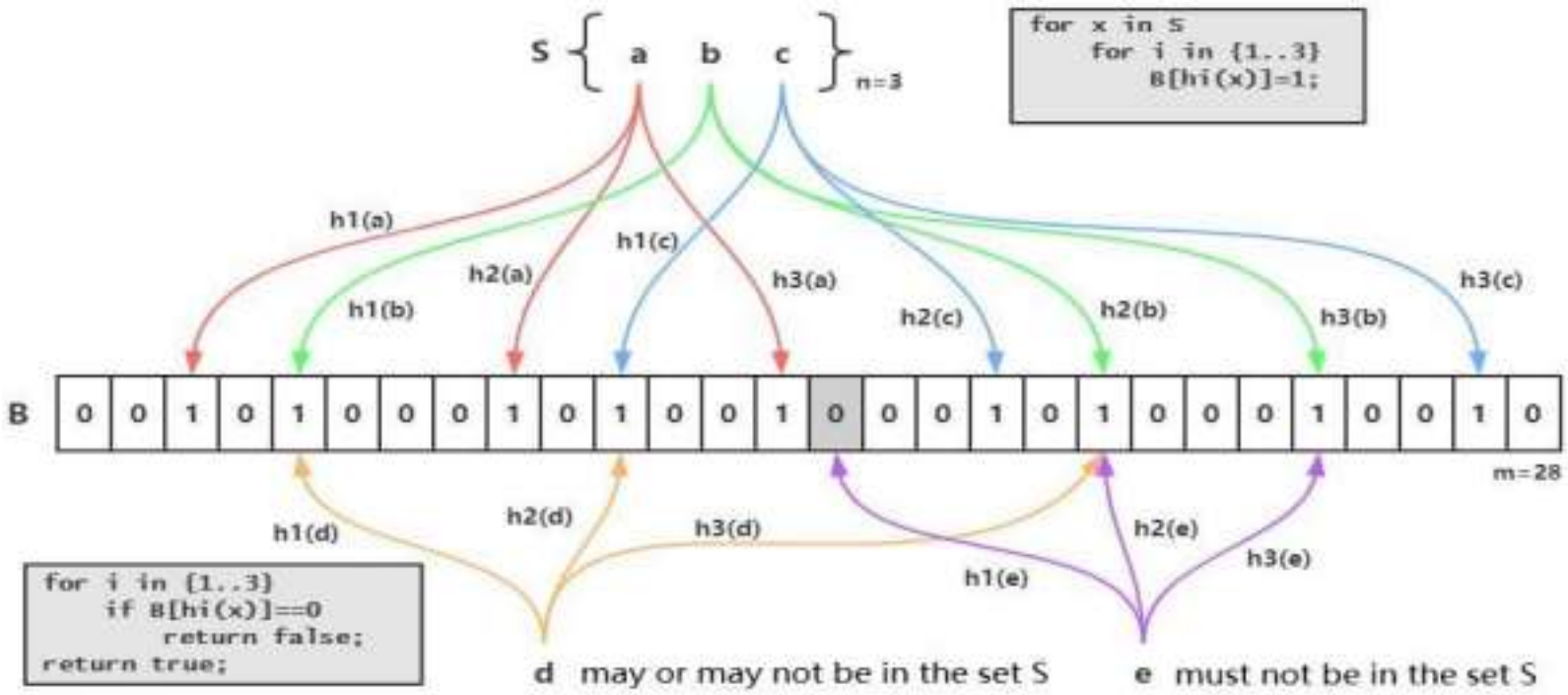
Example 1



3

0 6

Example 2


S { a b c } n=3
for x in S
for i in {1..3}
B[hi(x)]=1;
h1(a)
h1(b)
h2(a)
h1(c)
h3(a)
h2(c)
h2(b)
h3(b)
h3(c)
B 0 0 1 0 1 0 0 0 1 0 1 0 0 1 0 0 0 1 0 1 0 0 0 1 0 0 1 0
m=28
h1(d)
h2(d)
h3(d)
h1(e)
h2(e)
h3(e)
for i in {1..3}
if B[hi(x)]==0
return false;
return true;
d may or may not be in the set S
e must not be in the set S

Assignment 2

Problem: searching in a set of files

Input: One word (a query about a topic)

Output: List of files that contain the word (files related to the topic)

Requirements for Assignment 2

- Task 1: Universal Hash Functions
- Task 2: Keywords Extraction
- Task 3: Bloom Filter Creation
- Task 4: Lookup Mechanism
- Task 5: Measuring False Positive Rate

Assignment file contains the instructions.