**Universal College of Engineering, Kaman**

**Department of Computer Engineering**

**Subject: Big Data Analytics Laboratory**

**Experiment No: 05**

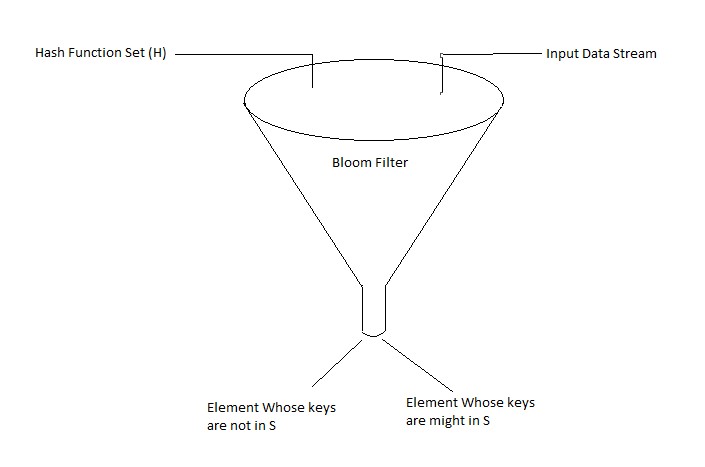
|  |  |  |  |
| --- | --- | --- | --- |
| **Roll No: 45** | **Name: Kartik A Parmar** | **Div: A** | **Batch: A-2** |

**Aim:** ToImplementing Bloom Filter using Python.

**Theory:**

**What is Bloom Filter?**

A Bloom filter is a space-efficient probabilistic data structure that is used to test whether an element is a member of a set. For example, checking availability of username is set membership problem, where the set is the list of all registered username. The price we pay for efficiency is that it is probabilistic in nature that means, there might be some False Positive results. False positive means, it might tell that given username is already taken but actually it’s not.



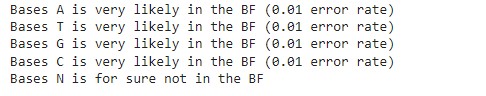
**Properties of Bloom Filters**

* Unlike a standard hash table, a Bloom filter of a fixed size can represent a set with an arbitrarily large number of elements.
* Adding an element never fails. However, the false positive rate increases steadily as elements are added until all bits in the filter are set to 1, at which point all queries yield a positive result.
* Bloom filters never generate false negative result, i.e., telling you that a username doesn’t exist when it actually exists.
* Deleting elements from filter is not possible because, if we delete a single element by clearing bits at indices generated by k hash functions, it might cause deletion of few other elements. Example – if we delete “geeks” (in given example below) by clearing bit at 1, 4 and 7, we might end up deleting “nerd” also Because bit at index 4 becomes 0 and bloom filter claims that “nerd” is not present.

**Code:**

|  |
| --- |
| pip install bloom-filter pip install pybloom-live # Import BF libraryfrom bloom\_filter import BloomFilter  # Define BF dna\_bf = BloomFilter(max\_elements=5, error\_rate=0.01)  # Add bases to BF bases = ["A", "T", "G", "C"] for base in bases:  assert base notin dna\_bf # Ensure the base is not in the BF initially dna\_bf.add(base) # Mark the base as seen  # Query BF with all the bases that were added + 'N'for base in bases + ["N"]: if base in dna\_bf:  print(f"Bases {base} is very likely in the BF (0.01 error rate)") else:  print(f"Bases {base} is for sure not in the BF") |

**Output:**



**Conclusion:** Hence We have successfully implemented Bloom Filter algorithm using Python.

from bloom\_filter import BloomFilter

# Define the Bloom filter

dna\_bf = BloomFilter(max\_elements=5, error\_rate=0.01)

# Add bases to the Bloom filter

bases = ["A", "T", "G", "C"]

for base in bases:

assert base not in dna\_bf # Ensure the base is not in the BF initially

dna\_bf.add(base) # Mark the base as seen

# Query the Bloom filter with all the bases that were added + 'N'

for base in bases + ["N"]:

if base in dna\_bf:

print(f"Base {base} is very likely in the Bloom filter (0.01 error rate)")

else:

print(f"Base {base} is for sure not in the Bloom filter")