Bloom Filter

Introduction

Bloom filters have become indispensable in big data algorithms, offering a compact and probabilistic data structure for efficient approximate membership queries. With the exponential growth of data volumes, traditional methods struggle to scale and deliver optimal efficiency. Bloom filters provide a space-efficient solution for tasks like set membership testing, duplicate elimination, and cache management. They offer a trade-off between space usage and query accuracy, making them ideal for scenarios where fast, memory-efficient solutions are required. Bloom filters' low memory footprint, constant time complexity for queries, and parallel processing capabilities make them efficient for handling massive datasets in real-time or distributed environments. Their usability and efficiency have solidified their role in big data processing, facilitating tasks such as duplicate detection, set operations, and cache management, ensuring scalable and efficient data processing in the face of big data challenges.

Problem Description

Bloom filters are widely used in search engines to handle keyword-based searches efficiently. In this context, a bloom filter is employed to store the presence of keywords in a large collection of documents or files. By using a set of hash functions and a bit array, the bloom filter can quickly determine whether a keyword is potentially present in a file or not. This allows the search engine to filter out irrelevant documents early in the search process, significantly reducing the computational overhead. Although bloom filters may introduce a small probability of false positives, they provide a powerful and space-efficient technique for narrowing down the search space and improving search efficiency in large-scale information retrieval systems.

In this assignment, you will implement a simple search mechanism using bloom filters. The mechanism would return a boolean array representing whether a word is possibly present in a set of files or if it is definitely not present.

The next section explains the required tasks for implementing the Bloom filter in the context of the search mechanism. The tasks are designed to provide a step-by-step approach that aligns with the actual implementation process. While the tasks may appear simplified, they serve the purpose of ensuring that you can focus on each step individually and understand the significance of their contributions while also facilitating a thorough evaluation. For this assignment, we have chosen to use a 64-bit Bloom filter with five hash functions as a simple example to illustrate the key parameters of a Bloom filter. The size of the filter refers to the total number of bits used to represent the filter. The number of entries corresponds to the number of elements that will be inserted into the filter. In our case, the elements are a set of keywords extracted from text files (32 files). Those files represent the search space. The false positive rate determines the probability of incorrectly identifying an element as being in the filter when it is not, and it is used to evaluate the effectiveness and performance of the Bloom filter.

Requirments

Task 1: Universal Hash Functions

- Implement five universal hash functions. Each hash function should take an input `string` and return a hash code of `integer` in the range [0,63].

- Aggregate the five hash functions in one function that takes one input `string` and returns an array of five integers (one for each hash function).
- Submission of this task:
  A python file named "*hashing.py*" takes input `string` as a command line argument and prints five integers separated by commas.
- Example:
  $ `python hashing.py algorithm`
  `10,20,30,40,50`

Task 2: Keywords Extraction

- The attached zip file *DATA* contains 100 text files named 0.txt, 1.txt, ..., 99.txt.
- You are assigned to only 32 files inside *DATA*.
- The names (numbers) of your assigned files can be found in the attached CSV file "*assigned_files.csv*". Where each roll number has 32 randomly selected integers representing the assigned files.
- Write a Python script to extract keywords from text files.
- Run your script on only the 32 files assigned to you.
- Store the extracted keywords of each file in a separate text file with the same name (same number) but in a separate folder. Name the new folder "*Keywords*". The keywords stored in the file should be separated by commas
- Submission of this task:
  The folder *Keywords* containing 32 files.
- Example:
  The file 4.txt inside the folder *Keywords* should contain keywords extracted from the file 4.txt inside *DATA*. Keywords should be stored as follows:
  `Digital,Negroponte,second,months,Chinese,Cambodia,`

Task 3: Bloom Filter Creation

- Create a Bloom filter of 64 bits for every file assigned to you inside *DATA*.
- A filter for a file in *DATA* is constructed by inserting all keywords in its corresponding file in the folder *Keywords*.
- Store the constructed bloom filter of each file in a separate text file with the same name (same number) but in a separate folder. Name the new folder "*Filters*". The bloom filter should be stored as a string of 0s and 1s.
- Submission of this task:
  The folder *Filters* containing 32 files.
- Example:
  The file 4.txt inside the folder *Filters* should contain a bloom filter for the file 4.txt inside *DATA*, which is constructed using keywords stored in the file 4.txt in the folder *Keywords*. A bloom filter should be stored as a string of 64 characters (0s and 1s):
  `0001000110000001010100000010000001000111001000000001110011001000`

Task 4: Lookup Mechanism

- Implement a lookup function that takes a keyword as input and returns an array of 32 boolean values.
- Values in the boolean array are set to 'True' if the keyword is possibly present and 'False' if it is definitely not present.
- Submission of this task:
  A python file named "*lookup.py*" takes input `string` (query) as a command line argument and prints an array of 32 bits (a string of 0s and 1s).
- Example:
  $ `python lookup.py algorithm`
  `00010000001000001000000000000000`
  This means that the word algorithm is possibly present in the 4th, 11th, and 17th files of your assigned files and definitely not present in all other assigned files.

Task 5: Measuring False Positive Rate

- The attached file *testing_queries.csv* contains 100 words that do not exist in the text files inside *DATA*.
- The 100 words will be used for testing queries to calculate the false positive rates.
- Use the lookup mechanism to check each word against the 32 Bloom filters.
- For every filter, count the queries that return 'True' (the word is possibly present). Such cases are false positives.
- Calculate the false positive rate for each filter as the ratio of false positives to the total number of testing queries.
- Submission of this task:
  A text file named "*false_positive.txt*" containing 32 false positives rates for the 32 bloom filters.

Submission Format and Attachments

You are requested to submit a single zip file named as your roll number.
The zip file should contain the following.

1. The file *hashing.py*
2. The folder *Keywords*: containing 32 text files (0.txt, 10.txt, ...).
3. The folder *Filters*: containing 32 text files(0.txt, 10.txt, ...).
4. The file *lookup.py*
5. The file *false_positive.txt*: containing 32 false positive rates.

The following are attached to this assignment.

1. The zip file *DATA*: containing 100 text files.
2. The file *assigned_files.csv*
3. The file *testing_queries.csv*

Other Instructions

- Given that the assignment involves many customized values (inputs, constants, and functions), it is hard to overlook instances of plagiarism. Generate your own submissions and avoid the consequences of plagiarism.
- Follow the mentioned formats and names in your submissions
- Command line arguments are values assigned while running the program, and they are obtained from the array `sys.argv` in Python. In your final submissions, use command line arguments, do not use hard-coded inputs or inputs that require user interaction.
- Not following the instructions will drop your submissions from the evaluation process.