Project 3: Frequent Items Counting
A Study on Memory-Efficient Algorithms

Filipe Pires [85122] & João Alegria [85048]

Advanced Algorithms
Department of Electronics, Telecommunications and Informatics
University of Aveiro

Abstract **– Lorem ipsum ...**

Keywords **– Probabilistic Counter, Count-Min Sketch, Memory-Efficient Algorithms**

I. Problem Contextualization

This report was written for the course of 'Advanced Algorithms', taught by professor Joaquim Madeira for the master's in Informatics Engineering at DETI UA. It describes the work done for the third assignment of the course [*1*]. The chosen hypothesis was "Hipótese A-2 – Contagem dos Itens Mais Frequentes".
Lorem ipsum ...

II. Dataset

Lorem ipsum ...

- *Alice in the Wonderland*, by Lewis Carol - written in English, German, French and Italian
- *A Christmas Carol*, by Charles Dickens - written in English, Finnish, German, Dutch and French
- *King Solomon's Mines*, by H. Rider Haggard - written in English, Finnish and Portuguese
- *Oliver Twist*, by Charles Dickens - written in English, French and German
- *The Adventures of Tom Sawyer*, by Mark Twain - written in English, Finnish, German and Catalan

Lorem ipsum ...

III. Frequent Word Identification

Identifying the most frequent words in a text stream was achieved using two very different strategies. On the first one, the program keeps record of all words that appear in the information stream along with an exact count of each word - this allows it to determine at any time which words are the most frequent. The second, on the other hand, is a far more memory-friendly solution that uses hash functions to avoid storing the pair $(word, counter)$. In this chapter we discuss the implementation of both solutions for the use case of identifying the most frequent words on literary works.

A. Exact Counter

The exact counter is the gold standard when creating alternative solutions that save time, memory or other resource at the cost of precision. It is by the values computed by it that the other counters are evaluated.
Its implementation consists of a regular dictionary of words as keys and counts as values, constantly updated as new words appear on the text stream. There are conditions, however, to update this dictionary. The incoming word must contain at least two characters and cannot belong to a list of stop-words defined *a priori*. This is done to prevent considering words that are irrelevant, i.e. do not offer any practical value to the study such as articles or prepositions.
Stop-words are managed by an external library [*2*], capable of dealing with all languages of our dataset.

B. Count-Min Sketch

The Count–Min Sketch (CM Sketch) is a probabilistic data structure that serves as a frequency table of events in a data stream. This means it works essentially like Bloom-Filters (BFs). However, they are used differently and therefore sized differently: a CM Sketch typically has less entries than the total number of different events (or words in our case) of the stream, related to the desired approximation quality of the sketch, while a counting BF is usually sized to match the total number of different events.
The solution uses hash functions to map word occurrences to frequencies, but saves memory space at the expense of over-counting some words due to collisions. The effect of these phenomena is addressed ahead.

The implementation of our CM Sketch was taken from the resources made available by the course's professor. As it served perfectly for our purposes, we found it unreasonable to reach our own implementation at the risk of adding imperfections that could compromise our study. Nevertheless, we offer a simple description of how it works and how we introduced it to the use case of count words.
If the same conditions defined in the Exact Counter are verified for the current word on the text stream, the word is passed to the *update()* method of the `CountMinSketch` class. This method then applies all its hash functions to the word and increments the values on the hash tables on the respective entries.

As the word counts either have the exact count or suffer from over-counting, the value considered as more trustworthy is the one whose result from hashing and updating returned the smallest value - the one with less occurrences of collisions - (hence the name count-MIN sketch). It is this value that is returned when the class instance is queried.

`CountMinSketch` receives two regulatory parameters that allow us to control the overestimation factor. These are `m`, the size of the hash tables, and `d`, the number of hash tables. The first regulates how significant is the overestimation since only the smallest value is returned by *query()* so the more tables there are, the smallest will be the difference between the real count and the one returned. The second reduces the probability of collisions since more hash tables means more complex hashes less likely to be equal. Two alternative parameters could have been used, `delta` and `epsilon`, that regulate the probability of query error and the query error factor and internally define `m` and `d` but in our case we sticked with the first two.

IV. Literary Study

The fist step taken when identifying the most frequent words on a text corpus is detecting the language of the text. This is common to both solutions and done even before they are executed, as it is later used to distinguish translations.

Lorem ipsum ...

V. Results & Discussion

Lorem ipsum ...

VI. Conclusion

Lorem ipsum ...

References

1. Joaquim Madeira, "3o trabalho - algoritmos probabilísticos", `https://elearning.ua.pt/pluginfile.php/1514391/mod_resource/content/0/AA_1920_Trab_2.pdf`.
2. Python SW Foundation, "Pypi - stop-words", `https://pypi.org/project/stop-words/`.