Misra & Gries Algorithm

Rafael Gomes de Sá, 104552, rafael.sa@ua.pt, MEI

Resumo **– O presente relatório apresenta uma implementação do algoritmo de Misra & Gries para obter os itens mais frequentes de um conjunto de dados de grandes dimensões.**

Foi efetuada uma implementação direta e modular do algoritmo, recorrendo a classes e módulos do Python. Para simular *data streams* foram utilizados ficheiros de texto de obras literárias.

Foi realizada uma sequência de testes ao algoritmo, de forma a comparar os resultados obtidos com as contagens exatas dos itens, e analisar a influência do parâmetro *k* nos resultados dos testes computacionais. Por último, verificou-se que a implementação respeitava as propriedades do algoritmo.

Abstract **- This report presents an implementation of the Misra & Gries algorithm to obtain the most frequent items of a large data set.**

A direct and modular implementation of the algorithm has been performed, using the classes and modules from Python. To simulate data streams, text files of literary works were used.

A sequence of tests was performed on the algorithm, to compare the results obtained with the exact counts of the items, and to analyze the influence of the parameter *k* on the results of the computational tests. Finally, it was verified that the implementation complied with the properties of the algorithm.

I. INTRODUCTION

The frequent items problem is to process a stream of items and find all items occurring more than a given fraction of the time. It is one of the most heavily studied problems in data stream algorithms. Informally, given a sequence of items, the problem is simply to find those items which occur most frequently. Typically, this is formalized as finding all items whose frequency exceeds a specified fraction of the total number of items [1].

The Misra & Gries algorithm is a simple algorithm that solves the frequent items problem and can be viewed as a generalization of the Majority algorithm to track multiple frequent items. This algorithm stores at most $k-1$ item-counter pairs, and each new item is compared against the stored items. If the item exists, then the corresponding counter is incremented, or else, if there is space for new counters, the new item is allocated with the counter set to one. If all $k-1$ counters are allocated to distinct items, then all are decremented by one and counters equal to zero are deleted. Any item which occurs more than $\frac{n}{k}$ times must be stored by the algorithm when it terminates [1].

The time cost of the algorithm is dominated by the $O(1)$ dictionary operations per update, and the cost of decrementing counts. The experimental studies conducted on the Misra & Gries algorithm have shown that the algorithm is accurate and fast to execute [1].

Many applications rely directly or indirectly on finding the frequent items, and implementations are in use in large scale industrial systems. The question of tracking approximate counts for a large number of possible objects arises in a number of settings. Many applications have arisen in the context of the Internet, such as tracking the most popular source or the most popular queries to a search engine. Finding approximate counts of items is also needed within other stream algorithms, such as approximating the entropy of a stream [1].

II. ALGORITHM IMPLEMENTATION

To implement the Misra & Gries algorithm a class, *FrequentCount*, was created. This class receives the parameter k when initialized, which can be changed later. The method that obtains the most frequent items receives a string, which consists of the data stream, and runs the algorithm, which was implemented in a modular way. To store the counters, Python's dictionary data structure was used. The class also allows obtaining the counter value of a certain item, and if the item does not exist in the dictionary, the value returned is zero.

A. Auxiliary Classes and Modules

To be able to test the algorithm using text files from literary works, it was necessary to create an auxiliary class and module.

The *ExactCount* class allows to obtain the exact counts of the frequencies, being these frequencies stored in a dictionary. This class allows reading the counts from an already existing file or performing the counts from a certain data stream and storing them in a file previously indicated. Like the *FrequentCount* class, this class allows obtaining the counter value of an item, which is returned zero if it does not exist in the dictionary. This class also allows to

obtain the x items with higher frequency, and the items with frequency higher than $\frac{n}{k}$.

The *file processing* module handles the processing of text files, providing the method that processes the file and returns a string with only the letters converted to lower case. In addition, it allows you to write a dictionary for a file, which will be useful for saving the exact counts of the frequency of the letters.

III. TEST SEQUENCE ANALYSIS

To test the developed algorithm, a sequence of tests was performed, varying the value of the parameter and the data stream. For each test, a set of metrics was recorded, which will be analyzed in more detail next, to better understand the behavior of the algorithm.

The literary works used to simulate the data stream were:

- *The Adventures of Sherlock Holmes*: 431485 letters and 32 distinct letters.
- *King James Bible*: 3224153 letters and 26 distinct letters.

These works allow the analysis of the behavior for smaller and larger data streams. The parameter *k* varied between 2 and the number of distinct items of each work minus one.

A. Execution Time Analysis

The results regarding the execution time of the tests performed are shown in figure 1. The execution time is given in seconds.

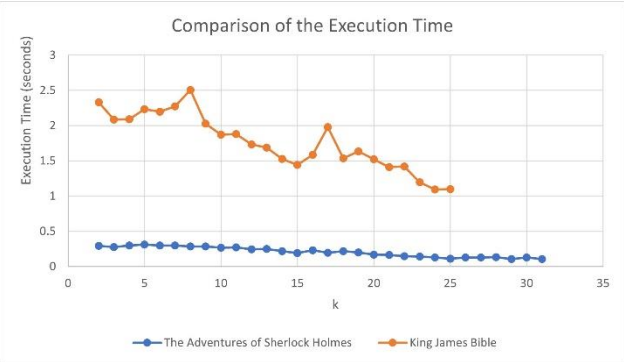

Fig. 1 - Test results regarding the execution time

Through the graphical analysis, it is possible to conclude that the algorithm is fast to execute, even when checking millions of items, as in the case of *King James Bible*. In both books there is a tendency to decrease the execution time with the increase of the *k* parameter, which is explained by the smaller need to decrement and eliminate counters, since it is possible to store more counters simultaneously.

B. Counters Position Analysis

The results regarding the comparison between the position of the items in the estimates and the position in the exact counts are shown in figure 2. Figure 2 gives the percentage of items whose position in the estimate is equal to the position in the exact counts.

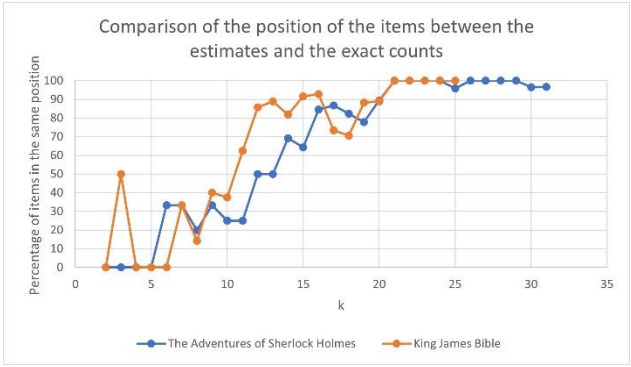

Fig. 2 - Test results regarding the position of the counters

By analyzing the graph, it is possible to observe a growing trend in the percentage of items with the same position with the increase of parameter *k*. This behavior is expected to occur, since the higher the value of *k*, the higher the accuracy of the counters and, consequently, of the final estimates will be. The *King James Bible* shows, in most cases, a higher percentage, since it has fewer distinct items than *The Adventures of Sherlock Holmes*.

C. Counters Value Analysis

The results regarding the comparison between the frequencies of the items obtained by the algorithm and the exact frequencies are shown in figure 3. The figure shows the sum of the discrepancies between the two values.

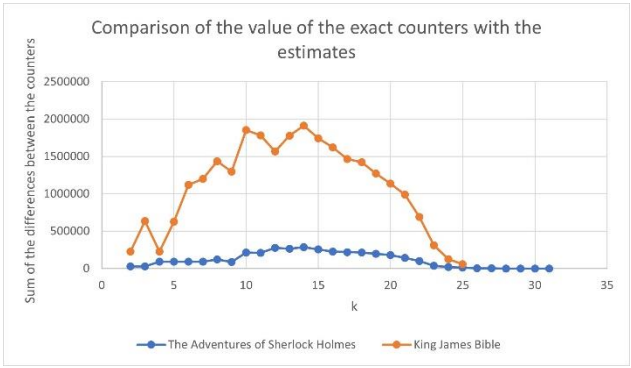

Fig. 3 - Test results regarding the discrepancies between the counters

Through the graphical analysis it is possible to conclude that there is an initial increase in the discrepancy between the counters, since as the value of *k* increases, it begins to store more counters, although the counts are not yet accurate because there is a need to decrement them. For higher values of *k*, the trend reverses, and the discrepancy begins to decrease, meaning that besides the estimate have almost as many counters as distinct items, their counts are also similar to the exact counts. The difference between both literary works is due to the difference in number of items, but the trend observed occurs in both.

In addition, the mean percentage of the difference between the estimated frequency and the exact frequency of the items was also analyzed. The results are shown in figure 4.

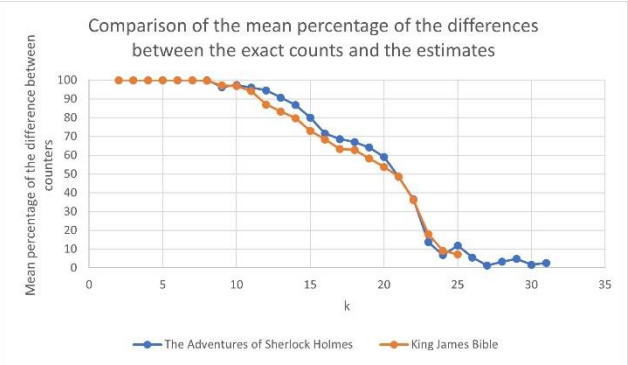

Fig. 4 - Test results regarding the mean percentage of discrepancy between counters

By analyzing the graph, it is possible to verify that as the value of k increases, the difference, in relative terms, between the frequency counts of the items will be smaller. As in the previous graphic, when the value of k is close to the number of distinct items, the difference between the counters tends to be zero.

D. Verification of the Algorithm Properties

To verify if the algorithm was properly implemented, it was verified if the results obtained in the tests were following the properties of the algorithm.

Figure 5 shows the number of items returned by the algorithm, to verify if, at most, $k - 1$ items are returned.

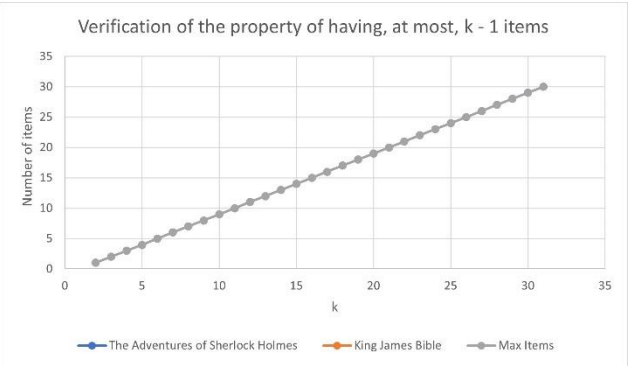

Fig. 5 - Test results regarding the number of items returned by the algorithm

The grey line represents the maximum number of items that the algorithm could return and, as can be seen on the graphic, the number of items returned in both books practically overlap the maximum number, never exceeding this limit. This behavior was the one that would be expected.

Figure 6 shows the percentage of mandatory items returned by the algorithm. This mandatory is related to the frequency of the item and serves to verify if all items with a frequency higher than $\frac{n}{k}$ were returned by the algorithm.

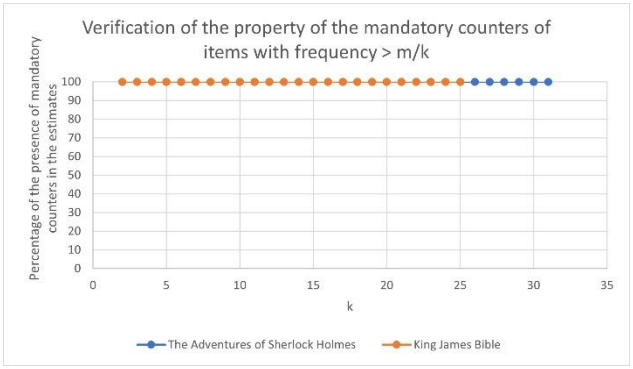

Fig. 6 - Test results regarding the presence of mandatory items returned by the algorithm

As it is possible to observe from the graphic, all results returned by the algorithm contained 100% of the items with a frequency higher than $\frac{n}{k}$. This behavior was the expected and confirms the property of the algorithm.

IV. CONCLUSIONS

In short, the implementation of the algorithm has proven its accuracy and speed of execution, concluding that it has a good speed-efficiency ratio.

The parameter k has a great influence on the results, being that the higher the value of k, the more accurate the results will be. That has a memory cost since it will be necessary to store more counters.

Finally, it is concluded that the algorithm is indicated to find the most frequent items in large data sets, but the frequency counts of the item are only minimally reliable for k values close to the number of distinct items in the data set.

REFERENCES

[1] Cormode, G. (2016). Misra-Gries Summaries. In M. Y. Kao (eds) *Encyclopedia of Algorithms*. Springer, New York, NY. https://doi.org/10.1007/978-1-4939-2864-4_572