Third Project of Advanced Algorithms – Most Frequent Letters

João Fonseca (103154)

Resumo **- Este projeto para a unidade curricular de Algoritmos Avançados aborda o desenvolvimento de diferentes métodos de contagem para identificar as letras mais frequentes em ficheiros de texto, mais especificamente em obras literárias. Produzimos três algoritmos diferentes: um contador exato, um contador probabilístico com uma probabilidade fixa de 1/16 e um contador com poupança de espaço para identificar itens frequentes num fluxo de dados.**

Para avaliar os algoritmos, utilizámos Otelo de William Shakespeare em 5 línguas diferentes (DE, EN, FI, FR, PT). Analisámos a eficiência computacional de cada abordagem, juntamente com os valores dos contadores e os erros absolutos e relativos das abordagens probabilística e de poupança de espaço.

Concluímos que o algoritmo probabilístico pode fornecer uma aproximação ao contador exato e prevê com precisão o valor e a classificação correta para cada token no documento. Por outro lado, embora não tenha o melhor desempenho para valores exatos do contador ou classificações, a abordagem de poupança de espaço é capaz de identificar os tokens mais frequentes no documento, sendo mais eficiente em termos de complexidade de espaço do que a anterior.

Abstract **- This project for the Advanced Algorithms course delves into the development of different counter methods for identifying the most frequent letters in text files, more specifically literary works. We produced three different algorithms: an exact counter, a probabilistic counter with a fixed probability of 1/16, and a space-saving counter for identifying frequent items in a data stream.**

To benchmark the algorithms, we used Othello by William Shakespeare in 5 different languages (DE, EN, FI, FR, PT). We analysed the computation efficiency of each approach, along with the counter values, and absolute and relative errors of both the probabilistic and space-saving approaches.

We conclude that the probabilistic algorithm can provide an approximation for the exact counter, and closely predicts the correct value and rank for each token in the document. On the other hand, while not performing the best for accurate counter values or rankings, the space-saving approach is able to identify the most frequent tokens in the document, being more efficient in terms of space complexity than the previous.

I. Introduction

In the context of the 3rd project of the Advanced Algorithms course (2023/2024), it was proposed the development of counting methods to identify the most frequent letters in text files [1]. Three different approaches were considered: an exact counter, an approximate (or probabilistic) counter with a fixed probability of 1/16, and a space-saving algorithm to identify frequent items in data streams [2].

Each algorithm's computational performance was analysed, along with the counter results, absolute and relative errors, and average values. The relative order of the letters obtained in each counter was also considered for analysis, and also if the most frequent letters are similar for the same literary work in different languages.

The source for the letters was Othello by William Shakespeare, obtained from the Project Gutenberg in German (DE), English (EN), Finish (FI), French (FR) and Portuguese (PT). Every file was processed, so that it became a continuous stream of uppercase letters, to be consumed more easily by each counter algorithm.

II. Development environment

The folder structure of the project is the following.

- benchmarks/
- documents/
- excel/
- src/
  - benchmark.py
  - comparison.py
  - counter_exact.py
  - counter_prob.py
  - counter_space.py
  - process.py
  - utils.py
- comparisons.txt
- punctuations.txt
- stop_words.txt
- report.pdf

The benchmarks folder contains the results from the algorithms' performance (time taken) and results of the counters.

The documents folder contains the downloaded literary works from Project Gutenberg (raw folder) and the processed versions used for the experiments (proc folder).

The excel folder contains the analysis the of the benchmark data and generation of charts.

The src folder contains the Python scripts that were used in the project.

- The process.py script is used to transform the Project Gutenberg files into a stream of uppercase letters. It removes the file headers; extracts the lines with content; removes stop words (from stop_words.txt), punctuation (from punctuation.txt), numbers, and all whitespaces.
- The utils.py script contains general functions that allowed the code in the other scripts to be more concise. There are functions to return a sorted dictionary by the keys and by the values, to write and read files, and the write benchmark results.
- The counter_*.py scripts contain the implementation of the counter approaches considered in the project: exact, probabilistic (fixed on 1/16) and space-saving [2].
- The benchmark.py script is responsible for executing the experiments over the letter streams, and outputs the results for each algorithm in a CSV format for a later analysis in excel.
- The comparison.py script is responsible for reading and comparing the results from the benchmarks. It displays the top-K results and average counter value for each approach, and compares the exact counter results to the probabilistic and space-saving methods, so that we can determine the absolute and relative errors of each.

III. Analysis of the Algorithms

A. Computational Efficiency

Exact Counter: In terms of time it is $\boldsymbol{O(n)}$ – linear, since it iterates though all n tokens in the sequence. In terms of space it is $\boldsymbol{O(m * b)}$ – where $\boldsymbol{m}$ is the number of counters (or unique tokens in the sequence), because every token is considered, and $\boldsymbol{b}$ is the number of bits used to store each counter's value. If the counter's value surpasses the maximum value of the counter, the number of bits needed to represent the value needs to be increased.

The exact counter was slower for French, faster for Finnish, and had a similar performance for the remaining languages. It can be noticed a small correlation between the time taken and the number of letters in each document, since the expect performance of the algorithm is linear.

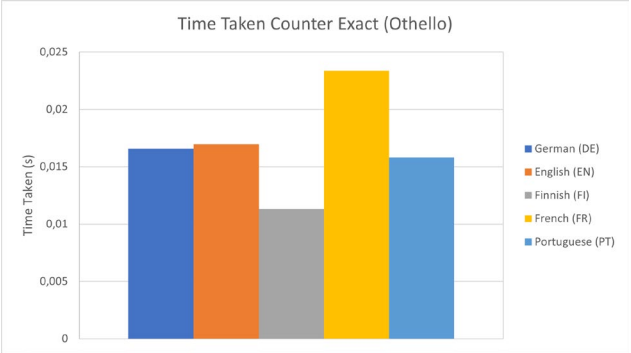

Fig. 1 – Time Taken Counter Exact (Othello).

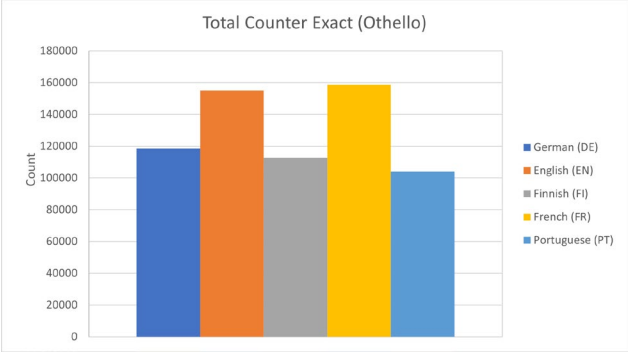

Fig. 2 – Total Counter Exact (Othello).

Probabilistic Counter: In terms of time it is $\boldsymbol{O(n)}$ – linear, since it iterates though all n tokens in the sequence. In terms of space it is $\boldsymbol{O(\frac{1}{16} * m * b)}$ **-** where $\boldsymbol{m}$ is the number of counters (or unique tokens in the sequence), because every token is considered, $\boldsymbol{b}$ is the number of bits used to store each counter's value. It is similar to the exact counter, however it only considers a token for the counter with a 1/16 probability, therefore in theory it can handle token streams 16 times larger than the exact counter, with a similar amount of space occupied, however sacrificing a bit of accuracy in return. The algorithm was run 10 times, so the results we considered refer to the average of these 10 runs.

The probabilistic counter was slower for German and French and faster for Portuguese, and had a similar performance for the remaining languages. There is a strong correlation between the time taken and the predicted counts, where we can conclude that when a token is considered for the counts, it takes more time to process the letter that ignoring it.

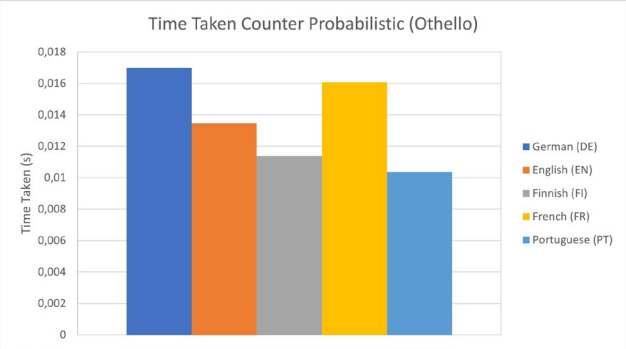

Fig. 3 – Time Taken Counter Probabilistic (Othello).

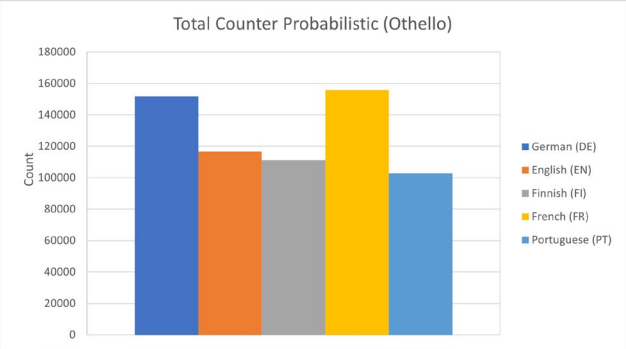

Fig. 4 – Total Counter Probabilistic (Othello).

Space-saving Counter [2]: In terms of time it is $\boldsymbol{O(n)}$ – linear, since it iterates though all n tokens in the sequence. In terms of space it is $\boldsymbol{O(k * b)}$ - where $\boldsymbol{k}$ is the number of counters considered, and $\boldsymbol{b}$ is the number of bits used to store each counter's value. If the goal is to identify the most frequent tokens in a stream without much accuracy in the counter and taking a very small amount of space, this method is ideal compared to the exact and probabilistic counters. When $\boldsymbol{k}$ increases, so does the accuracy of the results, however in return it occupies more space.

The space-saving counter was slower for German and French and faster for Portuguese and Finnish. This can be attributed to the higher number and variety or letters in German and French. When the number of k-counters is smaller, the algorithm takes more time. Therefore, time taken and space occupied by counters are inversely proportional.

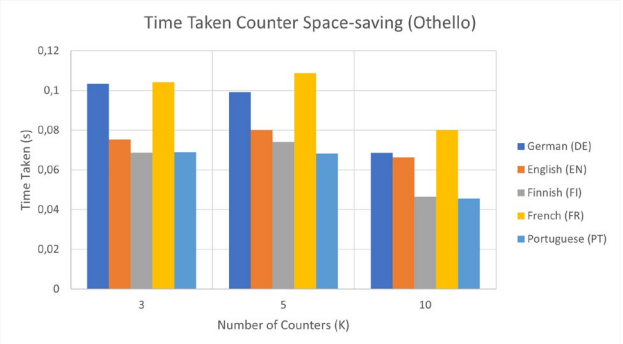

Fig. 5 – Time Taken Counter Space-saving, for k={3, 5, 10} (Othello).

B. Comparison of Probabilistic and Space-saving Counters to the Exact Counter

To evaluate the quality of the results obtained from the probabilistic and space-saving counters, it was performed a comparison between those results and the exact results, for the same documents. The comparison.py script read the benchmark results from each algorithm and performed an analysis of the results.

Firstly, we extracted the top-10 letters of each counter, for each language. To make the report concise, let's consider just the results for English and Portuguese versions of Othello (for German, French and Finnish, you can check the comparions.txt file).

	English (EN)			Portuguese (PT)		
Rank	**Exact**	**Prob.**	**S. Saving**	**Exact**	**Prob.**	**S. Saving**
#1	E	E	E	E	E	E
#2	O	O	O	A	A	A
#3	T	T	C	O	O	O
#4	A	A	T	S	S	S
#5	I	I	F	R	R	N
#6	S	N	I	I	I	H
#7	H	S	L	N	N	L
#8	N	H	N	D	M	P
#9	R	R	R	M	D	T
#10	L	L	V	T	U	C

Table. 1 – Top-10 results for English and Portuguese (Othello). Caption is as follows. Green: letter matches position in exact counter; Yellow: letter is the exact counter rank, but is not in the correct position; Red: letter is not in the exact counter rank.

We can observe that the probabilistic approach outperforms the space-saving approach in most cases, making it a more viable choice. It is more accurate in terms of the correct ranking of the letters. The most frequent letter in all languages is "E" (except for Finnish, which is "A").

	English (EN)			Portuguese (PT)		
	Exact	**Prob.**	**S. Saving**	**Exact**	**Prob.**	**S. Saving**
Counter Values	29 4557 14023	80 4486 13568	11602 11846 14023	2 3151 14236	32 3210 14144	9398 10396 14235
Absolute Errors	-	49 163 455	2421 7027 12280	-	81 183 400	0 6730 10842
Relative Errors	-	1% 8% 52%	17% 1400% 12100%	-	0.6% 10% 25%	0% 638% 2828%

Table. 2 – Counter values, absolute errors and relative errors for the algorithms (Minimum, average, maximum).

After analysing the counter values, absolute errors and relative errors, we can conclude that the most accurate method for counting (excluding the exact counter of course), is the probabilistic approach, which has an average relative error of 8% for English and 10% for Portuguese.

The space-saving method reveals inaccurate in predicting the ranking of the letters, therefore it has a very high error values. However, its purpose is to identify frequent tokens rather than their position, so it wouldn't be fair to consider it a bad approach – it just has a different purpose.

C. Counter Results for the Exact and Probabilistic Counters

The chart with these values can be viewed in the appendix section of this report, since the image was to wide to properly be included in the middle. Fig. I refers to the exact counter results, Fig. II to the probabilistic counter results.

VI. Conclusion

We conclude that a probabilistic method for counting token streams provides more accurate results than a space-saving approach, when compared to an exact counter approach. However, the probabilistic method's goal is to estimate the letter frequency, whereas the space-saving approach is to identify frequent letters, no matter their actual counts. Choosing between either to improve the performance of our counting system comes down to the sacrifices we are willing to make: if we want an approximation of the frequency over a very long stream of data, we should prefer a probabilistic counter; if we only want to approximately identify the most common tokens instead, we can opt for a space-saving approach, in case the memory space is valuable.

References

[1] Joaquim Madeira: Advanced Algorithms Materials. In: E-Learning (Dec 2023).

[2] Metwally, A., Agrawal, D., Abbadi, A.E.: Efficient computation of frequent and top-k elements in data streams. In: International Conference on Database Theory (2005).

APPENDIX

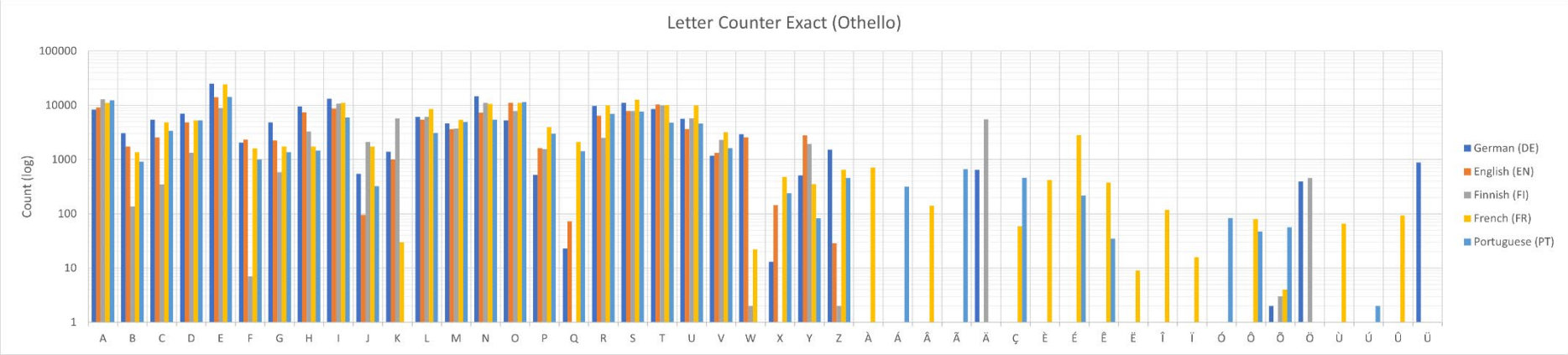

Fig. I – Letter Counter Exact (Othello). The scale is logarithmic, to allow for an easier perception of small counter values.

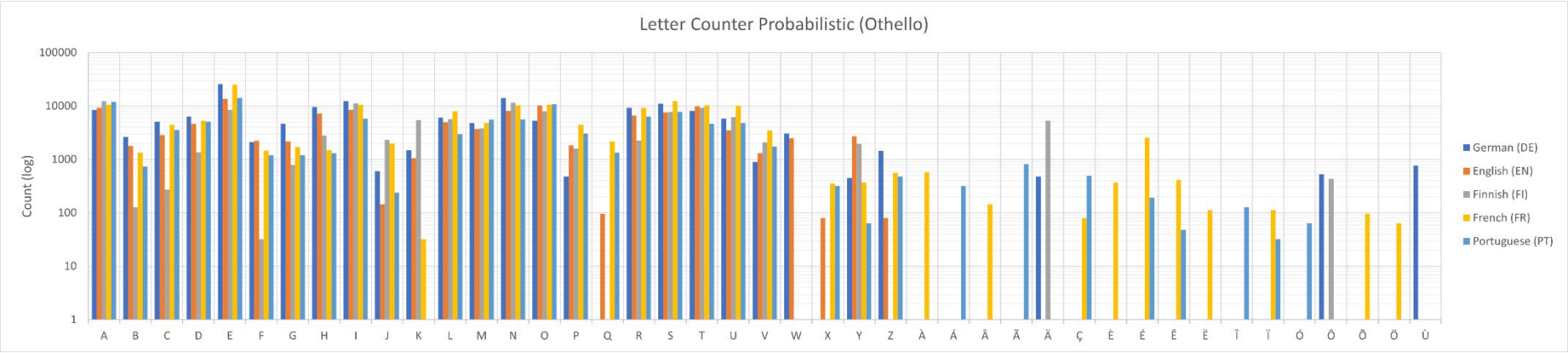

Fig. II – Letter Counter Probabilistic (Othello). The scale is logarithmic, to allow for an easier perception of small counter values.