Identificação das letras mais frequentes num ficheiro de texto usando diferentes metadologias

Gonçalo Freitas
Mestrado em Engenharia Computacional

***Abstract* –In this project, we use 3 methods, *Exact Counter*, *Probabilistic Counter* (with fixed probability) and *Lossy Counter*, to study the most frequent letters in a text file, evaluating each of these methods through various metrics such as accuracy and mean square error. From the results obtained it was possible to conclude that, if we do not encounter memory problems, the *Exact Counter* method is the best approach, as it always returns the exact values. However, if we are facing a memory problem or at risk of being, both *Probabilistic Counter* and *Lossy Counter* methods, are good approaches, as they return sufficiently satisfactory results and with a execution time not much longer than the *Exact Counter* method.**

***Resumo* –Neste projeto, procuramos utilizar 3 métodos, *Exact Counter*, *Probabilistic Counter* (de probabilidade fixa) e *Lossy Counter*, para estudar as letras mais frequentes num ficheiro de texto, avaliando cada um destes métodos através de várias métricas como a exatidão e o erro quadrático médio. A partir dos resultados obtidos foi possível concluir que, caso não nos deparemos com problemas de memória, o método de *Exact Counter* é a melhor abordagem, visto retornar sempre os valores exatos. No entanto, caso estejamos perante um problema de memória ou em risco de o estar, ambos os métodos de *Probabilistic Counter* e *Lossy Counter*, são boas abordagens, pois apresentam resultados suficientemente satisfatórios e com um tempo de execução não muito mais elevado que o método *Exact Counter*.**

***Palavras chave* –Contadores exatos, Contadores aproximados/probabilísticos, *Data Streams*, Exatidão, Tempo de Execução, Erro Quadrático Médio**

I. Introdução

Este projeto tem como objetivo identificar as letras mais frequentes num ficheiro de texto. Neste caso, os ficheiros texto utilizados serão livros, de duas línguas diferentes, obtidos a partir do *Project Gutenberg*, [1]. Irão ser usados, para o estudo deste problema, três métodos diferentes, explicados em II, que abordam diferentes metodologias enumerados de seguida:

- Contadores exatos (*Exact Counters*) - II-A
- Contadores aproximados/probabilísticos (*Probabilistic Counters*) - II-B
- *Lossy Counter* - II-C

Em relação aos livros analisados neste projeto, estes são os seguintes:

- O Livro de Cesário Verde (Português), [2]
- Os Lusíadas (Português), [3]
- The Expedition of Humphry Clinker (Inglês), [4]

Para além de ser analisado cada livro individualmente, será também analisado o conjunto dos três.

Tal como referido, estes livros foram retirados do *Project Gutenberg*, [1], contudo, antes de aplicar os métodos é necessário formatar os ficheiros de texto obtidos a partir do site. Esta formatação deu-se em três simples passos, onde no primeiro removemos todos os cabeçalhos relativos ao *Project Gutenberg* que se encontram no inicio do ficheiro e no fim do texto relativo ao livro. No segundo passo, removemos todos os sinais de pontuação e, no terceiro passo, convertemos todos as letras para letra maiúscula, convertendo também letras com acentos e outras formatações para a respetiva letra maiúscula, por exemplo, $\tilde{a}$, $\acute{A}$, $\hat{a}$ e a serão todas convertidas para A.

II. Métodos

A. Exact Counter

O método de *Exact Counter* é o método mais simples de entender e de implementar mas também é aquele com uma maior maior exigência computacional devido a ter uma complexidade computacional, $\mathcal{O}(n)$, e de memória $\mathcal{O}(m)$, onde n representa o tamanho do conjunto de elementos ao qual se pretende aplicar este método e m o número de elementos diferentes no conjunto.

O algoritmo deste método é, tal como referido, bastante simples e a sua ideia é a seguinte. Imaginemos que temos um dicionário tradicional, D, com uma chave (key) e um valor ($value$) associado a ela. Neste caso, key representará o elemento e $value$ o número total de vezes que o elemento está presente no conjunto, isto é, a sua frequência.

Dado então um conjunto de elementos, C, de tamanho n com m elementos diferentes iremos precisar de m pares $(key, value)$, pois não existem chaves repetidas,

i.e., iguais, num dicionário. Para proceder à contagem dos elementos no conjunto, analisamos todos os elementos presentes neste, um a um, e, caso o elemento já esteja presente em D incrementamos o *value* correspondente por 1, caso contrário, inicializamos um novo par $(key, 1)$ em D. Desta forma, o algoritmo para implementar este método pode ser descrito como de seguida:

Algorithm 1 ExactCounter(C)

```
D ← ∅
for each E ∈ C do
    if E ∈ D then
        D[E] ← D[E] + 1
    else
        D[E] ← 1
    end if
end for
```

B. Probabilistic Counter

Em geral, um registo de n-bits pode contar até 2^n valores, [5], assim, um registo de 8 bits apenas pode contar até 256 eventos. Antigamente, como a memória dos computadores era escassa, os cientistas rapidamente se deparavam com problemas de memória quando se queria contabilizar vários eventos. Assim, Robert Morris, [6], ao deparar-se com um problema destes e como apenas estava interessado em contagens aproximadas, desenvolveu um algoritmo probabilístico com uma margem de erro quantificável, também conhecido como *Morris Counter*.

Este algoritmo funciona incrementando o contador (de eventos) com uma probabilidade que depende do atual valor do contador, isto é, dado o evento $n+1$, incrementa-mos o valor do seu contador, C_{n+1}, por 1, com base no valor do contador no evento n, C_n, com uma probabilidade de 2^{-C_n}, assumindo uma base de 2, por exemplo. Assim, no evento $n+2$ o valor do seu contador, C_{n+2}, seria incrementado com probabilidade $2^{-C_{n+1}}$, e assim sucessivamente. Esta condição de incrementação pode ser visualizada na Equação 1, retirada de [7].

$$C_{n+1} = \begin{cases} C_n + 1 & \text{com probabilidade } 2^{-C_n} \\ C_n & \text{com probabilidade } 1 - 2^{-C_n} \end{cases} \quad (1)$$

Desde a invenção deste método por Morris, têm sido criadas várias adaptações deste, tal como o método que foi proposto para este projeto, em que se assume uma de probabilidade fixa 2^k. Assim, ao contrário do que acontece no método apresentado por Morris, a probabilidade de incrementar o contador mantém-se constante ao longo de todo o problema, não dependendo assim do valor do contador no evento anterior. Um diagrama representativo do método *Probabilistic Counter* com probabilidade fixa de $1/2^k$ pode ser visualizado na Fig.1. Para este projeto em específico iremos considerar $k = 3$, ou seja, dado um evento, incrementamos o seu contador com probabilidade $1/2^3 = 1/8$, o que nos leva a conseguir contar até 2^{n+3} valores, com n-bits.

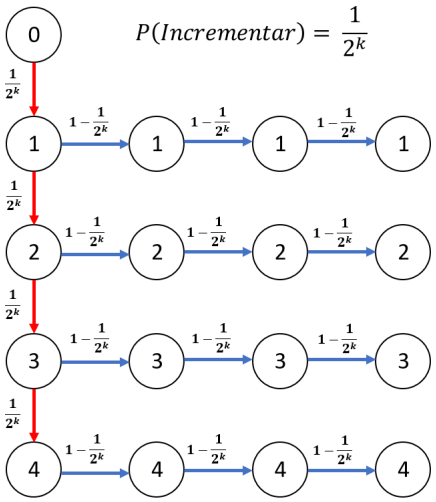

Fig. 1: Diagrama representativo do método *Probabilistic Counter* com probabilidade fixa de $1/2^k$

De notar que, como este método se trata de um método probabilístico, de forma a obter um resultado mais preciso é boa prática repetir o método m vezes e, no fim, calcular média dos contadores nessas m repetições. De notar também que, quanto maior for o valor de m, mais precisos serão os resultados. Por fim, de forma a obter o valor estimado para o contador de um evento basta multiplicar o valor do contador (média das m repetições) por 2^k, $Estimativa = Contador \times 2^k$. [5]

Dado um conjunto de elementos, C, podemos então descrever o algoritmo para este modelo tal como representado no *Algorithm 2*.

Algorithm 2 ProbabilisticCounter($k = 3$, C)

```
function INCREMENT(prob)
    r ← random value between 0 and 1
    if r < prob then
        return 1
    else
        return 0
    end if
end function

D ← ∅
for each E ∈ C do
    if E ∈ D then
        i ← INCREMENT(1/2^k)        ▷ chamada à função
        D[E] ← D[E] + i
    else
        D[E] ← 1
    end if
end for
```

Não pode passar em despercebido que, quanto maior for o valor de k maior poderá ser o erro associado à estimativa mas, no entanto, maior é número de valores que conseguimos contar, com os mesmos n-bits, 2^{n+k}, o que pode ser bastante vantajoso em certos casos.

C. Lossy Counter

O algoritmo *Lossy Counter* é um algoritmo bastante utilizado para identificar os elementos de um grande conjunto de dados, (*data stream*), que apresentam uma maior frequência, isto é, estão mais presentes no conjunto. Este algoritmo divide a *data stream* original em sub-divisões, *'Buckets'*, cada uma com dimensão $w = \lceil \frac{1}{e} \rceil$, onde $e \in (0, 1)$ e representa o parâmetro de erro. Este valor é definido pelo utilizador. Estes *'Buckets'* possuem um índice (inteiro) b, com $1 \leq b \leq \lceil \frac{N}{w} \rceil$, em que N representa o tamanho da *data stream* original. [8]

Para este algoritmo é necessário uma *data structure*, D, que guarde tuplos do tipo (E, f, Δ), onde E representa o elemento, f o limite inferior para a sua frequência e Δ a diferença entre o limite superior e inferior. O parâmetro f representa limite inferior da frequência para um elemento E e nem sempre a frequência exata deste pois, no final da análise de cada *'Bucket'*, algumas entradas de D podem ser eliminadas. [9]

Importante referir que, caso fizermos $f + \Delta$, isto não implica que obteremos o verdadeiro valor correto da frequência do respetivo elemento, mas sim uma sobre-estimativa deste (e nunca uma sub-estimativa), [5]. Partindo dos elementos menos frequentes, para os mais, este parâmetro Δ vai diminuindo à medida que nos aproximamos do elemento mais frequente até que, em alguns casos, este chegar a zero, $\Delta = 0$, que traduz o momento em que o valor de f passa a traduzir o valor exato da frequência do elemento.

Num *'Bucket'*, o algoritmo vai analisando os elementos, $(E_1, E_2, ...)$, incrementando o valor de f por 1 se o elemento já existir em D ou então criando uma nova entrada em D. O valor de Δ apenas é alterado quando uma nova entrada é criada e toma o valor de $b - 1$, onde b representa o índice do *'Bucket'*. [8]

Tal como referido anteriormente, no final da análise de cada *'Bucket'*, alguma entradas de D podem ser eliminadas. Estas eliminações são realizadas com base numa condição. Caso o limite superior de um elemento E, isto é, $f_E + \Delta_E$ for menor que índex do *'Bucket'* atual, b, o elemento E é eliminado de D, libertando assim memória. [10]

Assim, dado um conjunto de elementos, C, o algoritmo que implementa este método pode ser descrito tal como representado no *Algorithm 3*.

Este algoritmo possui, no pior dos casos, uma complexidade computacional (de memória) de $\mathcal{O}(\frac{1}{e} log(eN))$, o que se torna bastante vantajoso aquando a análise de um grande conjunto de elementos. [8]

Algorithm 3 LossyCounting(e, C)

```
w ← ⌈1/e⌉
n ← 0; i ← 1
D ← ∅
for each E ∈ C do
    n ← n + 1
    if E ∈ D then
        f_E ← f_E + 1
    else
        Δ ← i − 1
        D[E] ← (1, Δ)
    end if
    if n == w then
        for all j ∈ D do
            if f_j + Δ_j < i then
                D ← D\{j}
            end if
        end for
        n ← 0
        i ← i + 1
    end if
end for
```

III. Análise de Resultados

De referir que, até ao último ponto, III-G, o valor respetivo à frequência de uma letra, retornado pelo método *Lossy Counter*, representado nas Tabelas apresentadas, será o valor, seguindo a definição apresentada em II-C, do limite inferior do contador, f. Assim, dos pontos III-A até III-F, nas Tabelas apresentadas e referidas em anexo, nesses pontos, o valor apresentado, para o metódo *Lossy Counter*, é então o valor de f. Quanto ao cálculo das exatidão do ranking, para este método, apenas a analisou-se as letras retornadas o que faz com que esta métrica possa ser de 100.0% apesar de o número de letras retornando não ser o mesmo. No entanto, para o erro quadrático médio e a exatidão em frequência assumiu-se que, para as letras não retornadas por este método, o valor da sua frequência é 0.

De forma a obter sempre os mesmo resultados, cada vez que os algoritmos são executados, foi optado fixar a *seed* da função *random*, [11]. Para o valor da *seed*, foi optado usar o número mecanográfico do aluno, neste caso, 98012.

A. Letras mais frequentes

Na Tabela I estão representadas as 5 letras mais frequentes retornadas por cada método, com a respetiva frequência, e tempo de execução de cada algoritmo que implementa o respetivo método, para o livro *O Livro de Cesário Verde*, em que podemos reparar que a ordem é a mesma para todos os métodos. De referir que para o método *Lossy Counter*, se considerou $e = 0.01$, e, para o método *Probabilistic Counter*, repetiu-se a contagem 100 vezes procedendo-se apenas no final ao cálculo da estimativa de cada contador. A partir desta tabela,

também reparamos que a frequência de cada letra, retornada pelo método *Probabilistic Counter*, não corresponde a frequência correta, retornada pelo método *Exact Counter* (e pelo método *Lossy Counter*).

TABELA I: Top 5 letras mais frequentes retornadas por cada método, com respetiva frequência, e tempo de execução de cada algoritmo que implementa o método - *O Livro de Cesário Verde*

Top 5 \ Counter	*Exact*		*Probabilistic* (100)		*Lossy* (0.01)	
1	A	9411	A	9408	A	9411
2	E	8457	E	8485	E	8457
3	0	7109	O	7106	O	7109
4	S	6428	S	6451	S	6428
5	R	4471	R	4482	R	4471
Tempo(s)	0.010		2.661		0.0140	
Exatidão ranking(%)	-		100.0		95.65	

Por fim, a partir da linha 'Tempo(s)', da Tabela I, podemos reparar que o método *Exact Counter* é aquele com um menor tempo de execução enquanto o método *Probabilistic Counter* aquele é aquele cujo tempo de execução é maior.

B. Avaliação os métodos

Uma estudo interessante para qualquer método deste tipo é, analisar algumas métricas de avaliação de desempenho, como o valor médio para um contador, a exatidão (*accuracy*) entre outras apresentadas na Tabela II, para o método *Probabilistic Counter* quando se repete a contagem 100. Uma mais completa versão desta tabela encontra-se no Appendix A, Tabela XI, e, para os restantes métodos, as tabelas com as respetivas métricas encontram-se no mesmo Appendix nas Tabelas XII e XIII para o métodos *Exact Counter* e *Lossy Counter*($e = 0.01$), respetivamente. De referir que, para os valores considerados como corretos, usados para calcular a exatidão, o erro quadrático médio, e o número de contadores e posições no ranking corretas iremos usar os valores retornados pelo método *Exact Counter* pelo que, para este método, não faz sentido calcular estas 4 métricas. As fórmulas para o cálculo das métricas, apresentadas nesta projeto, encontram-se no Appendix B.

TABELA II: Métricas relativas ao método *Probabilistic Counter* (1000) - OCV (*O Livro de Cesario Verde*), OLUS (*Os Lusíadas*), TEOHC (*The Expedition of Humphry* Clinker), ALL (3 livros juntos)

Métrica \ Livro	OLCV	OLUS	TEOHC	ALL
Média	2695.32	10300.45	25816.34	37901.61
Variância	6865722.37	95762143.33	1048095102.15	1045640230.45
Exatidão Ranking (%)	100.0	100.0	100.0	92.31
Número de posições corretas	25	24	26	24
Número de letras	25	24	26	26

A partir da Tabela II podemos concluir que o livro que obteve pior resultados foi o livro que representa a soma dos três livros juntos (ALL), no entanto, no geral, o método apresentou resultados satisfatórios na ordem das letras, como se pode ver pelo parâmetro 'Exatidão Ranking (%)'.

Já a partir da Tabela XIII, podemos analisar que, caso o objetivo do trabalho fosse apenas encontrar ordem correta das letras, não ligando à sua frequência, o método *Lossy Counter* também apresenta resultados satisfatórios, sendo, no entanto, pior, visto apresentar uma menor exatidão neste parâmetro. Caso o objetivo fosse, no entanto, estudar o valor da frequência das X letras mais frequentes, vemos que, analisando a Tabela XIV, que representa a ordem do ranking total retornado por cada algoritmo para o livro *O Livro de Cesário Verde*, o melhor método seria o *Lossy Counter*, visto retornar valores mais próximos dos valores exatos para, pelo menos, $X = 17$. O ranking total retornado por cada algoritmo para os restantes livros pode ser encontrado nos ficheiros de texto submetidos em anexo. No entanto, à medida de que o valor de X aumenta, o método *Probabilistic Counter* começa a obter resultados mais próximos dos corretos, pelo que se deve considerar este método a melhor abordagem para estudar a frequência das $N - X$ letras menos frequentes, onde N representa o número exato de letras no ficheiro.

C. Probabilistic Counter - *Número de repetições*

Como já foi referido em II-B, quanto mais vezes se repetir a contagem, maior será a precisão do método. Isto pode ser visualizado na Tabela III, onde comparamos as letras mais frequentes retornadas pelo método *Probabilistic Counter* quando se repete a contagem $n = 1$, $n = 10$ e $n = 100$ vezes, com as letras mais frequentes retornadas pelo método *Exact Counter*, para o livro *O Livro de Cesário Verde*. A partir desta tabela é então possível concluir que, ao aumentar o valor de n, apesar da ordem das letras mais frequentes se manter, a respetiva frequência aproxima-se mais dos valores corretos, retornados pelo método *Exact Counter*, como se pode ver pela diminuição do erro quadrático médio (EQM), da estimativa da frequência.

TABELA III: Top 5 letras mais frequentes retornadas pelo método *Probabilistic Counter* quando se repete a contagem 1, 10 e 100 vezes, Prob(n), e respetivo erro quadrático médio (EQM), da estimativa da frequência, comparando com as letras mais frequentes retornadas pelo método *Exact Counter* - *O Livro de Cesário Verde*

Top 5 \ Counter	*Exact*		*Prob (1)*		*Prob (10)*		*Prob (100)*	
1	A	9411	A	8976	A	9412	A	9408
2	E	8457	E	8040	E	8411	E	8485
3	0	7109	O	7040	O	7080	O	7106
4	S	6428	S	6344	S	6389	S	6451
5	R	4471	R	4592	R	4488	R	4482
EQM	-		26253.44		938.36		105.4	
Tempo(s)	0.010		0.028		0.287		2.661	

O mesmo poderíamos concluir caso analisássemos as letras menos frequentes retornadas por cada método, ou seja, à medida que aumentamos o valor n a frequência de cada letra aproxima-se mais dos valores corretos,

como se pode ver pela Tabela IV. No entanto, não pode passar despercebido também que à medida que se aumenta o valor de n, o custo para executar o algoritmo aumenta, traduzindo-se num maior tempo de execução, tal como se pode reparar pela linha 'Tempos(s)' na Tabela III.

TABELA IV: Top 5 letras menos frequentes retornadas pelo método *Probabilistic Counter* quando se repete a contagem 1, 10 e 100 vezes, Prob(n), comparando com as letras menos frequentes retornadas pelo método *Exact Counter - O Livro de Cesário Verde*

Counter / Top 5	*Exact*		*Prob (1)*		*Prob (10)*		*Prob (100)*	
1	K	4	K	8	K	3	K	4
2	Y	54	Y	56	Y	48	Y	53
3	X	155	X	136	X	146	X	157
4	J	208	J	256	J	205	J	201
5	Z	344	Z	344	Z	356	Z	346
Nº Letras	25							

D. Probabilistic Counter - *Parâmetro k*

Apesar deste não ser o caso, visto o valor de k não ser muito elevado ($k = 3$), poderíamos nos deparar com um caso em que, para $n = 1$, a ordem das letras não seria a correta e fosse preciso um valor de n maior para obter as letras mais e menos frequentes corretamente. Por exemplo, se consideramos $k = 8$ e $n = 1$, isto é, incrementar o contador com probabilidade $1/2^8$ e apenas repetir o algoritmo 1 vez, como se pode ver pelas Tabelas V e VI, assinalado a amarelo, a ordem das letras mais e menos frequentes, do livro *O Livro de Cesário Verde*, nem sempre é a correta, quando comparando com a ordem retornada pelo método *Exact Counter*. No entanto, quando se considera $k = 8$ e $n = 10$, a ordem destas já é a correta. A partir destas tabelas também podemos analisar a influência que o valor k tem na qualidade dos resultados, em termos do valor da frequência de cada letra, onde reparamos que, quanto menor for o valor de k, mais próximos são os resultados dos valores reais, como se pode ver pelo valor do erro quadrático médio (EQM), da estimativa da frequência, quando comparamos este valor quando assumimos $k = 3$ e $k = 8$. Por fim, é possível concluir, que quanto maior for o valor de k, menor é o tempo de execução do algoritmo que implementa este método, como se pode ver pela linha 'Tempos(s)' apresentada na Tabela V.

E. Número de letras retornadas por cada método

Outro estudo interessante a ser feito é o número de letras que cada método retorna no final da sua execução, isto pois, tal como explicado em II-C, o método *Lossy Counter*, de forma a diminuir o consumo da memória, elimina algumas letras da estrutura de dados usada, neste caso, um dicionário, com base numa condição, já explicada em anteriormente. Esta eliminação leva a que o número total de letras, retornado por este método, seja sempre ser inferior ou igual (dependendo do parâmetro e) ao número exato de letras no ficheiro,

TABELA V: Top 5 letras mais frequentes retornadas pelo método *Probabilistic Counter* quando se repete a contagem $n = 1, 10$ vezes incrementando o contador com uma probabilidade $1/2^k$ com $k = 3, 8$, Prob(n,k), e respetivo erro quadrático médio (EQM), da estimativa da frequência, comparando com as letras mais frequentes retornadas pelo método *Exact Counter - O Livro de Cesário Verde*

Counter / Top 5	*Exact*		*Prob (1,3)*		*Prob (10,3)*		*Prob (1,8)*		*Prob (10,8)*	
1	A	9411	A	8976	A	9412	*E*	9472	A	8883
2	E	8457	E	8040	E	8411	*O*	7936	E	8294
3	0	7109	O	7040	O	7080	*A*	7936	O	7168
4	S	6428	S	6344	S	6389	*R*	5632	S	6707
5	R	4471	R	4592	R	4488	*S*	5376	R	4710
EQM	-		26253.44		938.36		648896.64		62358.28	
Tempo(s)	0.010		0.028		0.287		0.030		0.271	

TABELA VI: Top 5 letras menos frequentes retornadas pelo método *Probabilistic Counter* quando se repete a contagem $n = 1, 10$ vezes incrementando o contador com uma probabilidade $1/2^k$ com $k = 3, 8$, Prob(n,k), comparando com as letras menos frequentes retornadas pelo método *Exact Counter - O Livro de Cesário Verde*

Counter / Top 5	*Exact*		*Prob (1,3)*		*Prob (10,3)*		*Prob (1,8)*		*Prob (10,8)*	
1	K	4	K	8	K	3	K	256	K	25
2	Y	54	Y	56	Y	48	*X*	256	Y	25
3	X	155	X	136	X	146	*F*	256	X	128
4	J	208	J	256	J	205	*Z*	512	J	179
5	Z	344	Z	344	Z	356	*J*	512	Z	588
Nº Letras	25									

dado pelo método *Exact Counter*, tal como podemos ver na Tabela VII, para os livros CV (*O Livro de Cesario Verde*), OLUS (*Os Lusíadas*), TEOHC (*The Expedition of Humphry* Clinker), ALL (3 livros juntos). Para a obtenção dos resultados apresentados nesta tabela foi considerado um parâmetro de erro $e = 0.01$ para o método *Lossy Counter*. Por fim, em relação ao algoritmo que implementa o método *Probabilistic Counter*, neste caso assumindo $n = 100$, concluímos que este retorna sempre o número exato de letras, o que vai de acordo com esperado visto este método não eliminar nenhuma letra do dicionário usado para este problema.

TABELA VII: Número de letras retornadas por cada método, no final da análise - OCV (*O Livro de Cesario Verde*), OLUS (*Os Lusíadas*), TEOHC (*The Expedition of Humphry* Clinker), ALL (3 livros juntos)

Livro / Counter	OLCV	OLUS	TEOHC	ALL
Exact	25	24	26	26
Probabilistic(100)	25	24	26	26
Lossy (0.01)	23	22	22	22

F. Lossy Counter - *Parâmetro e*

Como já vimos, em III-C, como é que os resultados do método *Probabilistic Counter* variam consoante o número de vezes que o método é repetido, podemos analisar também como os resultados do método

Lossy Counter variam consoante o valor do parâmetro de erro, e, referido em II-B, obtendo a Tabela VIII que representa as 5 letras menos frequentes retornadas pelo método *Lossy Counter* quando se considera $e = 0.1, 0.01, 0.001$, Lossy(e), comparando com as letras menos frequentes retornadas pelo método *Exact Counter*, para o livro *O Livro de Cesário Verde*. A partir desta, podemos concluir que quanto mais pequeno for o valor de e mais próximos dos valores reais são os valores retornados pelo método, tanto analisando a ordem das letras, como se pode ver pelo aumento da Exatidão Ranking (%), como analisando o valor da frequência destas, como se pode ver pela diminuição do erro quadrático médio, linha 'EQM', com a diminuição de e. No entanto, analisando a linha 'Nº de letras' vemos que quanto menor for o valor de e, maior memória é necessária devido a ser armazenado no dicionário um maior número de letras. Este comportamento é o esperado pois, como e representa o parâmetro de erro, quanto menor este for, menor é o erro do método, pelo que, mais próximo da solução correta serão os resultados obtidos.

TABELA VIII: Top 5 letras menos frequentes retornadas pelo método *Lossy Counter* quando se considera e como $0.1, 0.01, 0.001$, Lossy(e), comparando com as letras menos frequentes retornadas pelo método *Exact Counter*, com o respetivo número de letras retornado, respetiva exatidão (de ranking) e respetivo erro quadrático médio (EQM) para o valor da frequência das letras - *O Livro de Cesário Verde*

Counter / Top 5	*Exact*		*Lossy (0.1)*		*Lossy (0.01)*		*Lossy (0.001)*	
1	K	4	C	1	J	1	Y	16
2	Y	54	D	1	Z	1	X	155
3	X	155	V	3	Y	1	J	208
4	J	208	I	3	F	662	Z	344
5	Z	344	R	7	Q	687	F	701
Nº Letras	25		9		23		24	
Exatidão Ranking (%)	-		77.78		95.65		100.0	
EQM	-		5521185.22		8351.04		0.66	

G. Lossy Counter - *Parâmetro* Δ

Até aqui apenas analisamos o valor do limite inferior da frequência, f, de cada letra, retornado pelo método o método *Lossy Counter* contudo, tal como referido em II-C, este método possui também um parâmetro Δ, que representa a diferença entre o valor de f e o limite superior, pelo que, este parâmetro, pode se tornar bastante útil visto poder ser usado para obter uma sobre-estimativa do valor exato, calculada como $f + \Delta$. Assim, considerando, por exemplo, $e = 0.01$, e analisando as 10 letras menos frequentes retornadas por este método calculando a respetiva sobre-estimativa para o valor do contador de cada letra obtemos os valores representados na Tabela IX, onde as colunas a vermelho e verde representam os valores de f e Δ, respetivamente.

A partir desta tabela podemos realizar três conclusões. Primeiramente verifica-se que, quando $\Delta \neq 0$, este método realiza uma sobre-estimativa da frequência de cada letra mas nunca uma sub-estimativa, visto que o valor real para a frequência de cada letra é sempre menor que $f + \Delta$ e nunca maior, e, à medida que o valor de Δ diminui, o valor desta sobre-estimativa vai se aproximando cada vez do valor correto. Por fim, podemos concluir que, quando $\Delta = 0$, o valor de f passa a traduzir o valor correto da frequência de cada letra, como se pode ver pelas letras 'G' e 'H'. Para as restante letras, isto é, 'V' e 'P', podemos comparar o valor de f, retornando pelo método *Lossy Counter*, com o valor exato da frequência, a partir da Tabela XIV, onde concluímos que os valores são iguais, tal como esperado pois, para estas letras, temos $\Delta = 0$.

TABELA IX: Top 10 letras menos frequentes, com respetivo limite de frequência inferior, f, e respetiva diferença entre o valor de f e o limite superior, Δ, retornadas pelo método *Lossy Counter* quando se considera e como 0.01, Lossy(e), comparando com as letras menos frequentes retornadas pelo método *Exact Counter* - *O Livro de Cesário Verde*

Counter / Top 5	*Exact*		*Lossy (0.1)*			$f + \Delta$
1	K	4	J	1	673	674
2	Y	54	Z	1	673	674
3	X	155	Y	1	673	674
4	J	208	F	662	47	709
5	Z	344	Q	687	24	711
6	F	701	B	803	65	868
7	Q	704	G	901	0	901
8	B	857	H	951	0	951
9	G	901	V	1090	0	1090
10	H	951	P	1484	0	1484
Nº Letras	25		23			

IV. Livros de dimensões maiores

Tal como realizado nos projetos passados, podemos estimar quanto tempo demoraria cada algoritmo a analisar ficheiros com N letras, usando os valores de tempo de execução de cada algoritmo para ficheiros com um número de letras inferior a N. Posto isto, iremos analisar s palavras, com $s = 10, 100, 1000, ..., 10^8$, de um ficheiro que representa a soma dos três livros, referidos em I. Dado que somando os três livros ficamos com 985714 letras, iremos repetir o ficheiro 102 vezes de forma a podermos analisar 10^8 letras, isto pois, $985714 \times 101 < 10^8 < 985714 \times 102$.

Posto isto, após a análise do ficheiro com de $s = 10, 100, 1000, ..., 10^8$ palavras, guardando o respetivo tempo de execução cada ficheiro, obtendo uma equação aproximada da reta formada por estes, para cada método, realizando um fit polinomial de grau 1, isto é, linear, tendo-se obtido os gráficos representados na Fig. 2, 3 e 4, presentes no Appendix C, para os métodos *Exact Counter*, *Probabilistic Counter* ($n = 100$) e *Lossy Counter* ($e = 0.01$), respetivamente. As respetivas equações dos fits são as seguintes:

$$f_{EC}(x) = (2.481 \times 10^{-7}) \times N + 0.04944$$

$$f_{PC}(x) = (6.503 \times 10^{-5}) \times N + 4.611$$

$$f_{LC}(x) = (4.159 \times 10^{-7}) \times N + 0.1106$$

Usando então estas equações podemos estimar quanto tempo cada algoritmo demoraria para implementar cada método para um ficheiro com N letras, tal como representado na Tabela X, em que reparamos que o método *Exact Counter* é aquele que apresenta tempos de execução menores enquanto que o método *Probabilistic Counter* é o que apresenta maiores valores pelo que, para um ficheiro de grandes dimensões, se deve usar o método *Exact Counter*, a menos que o ficheiro contenha um número de elementos diferentes grande o suficiente para provocar um problema de memória.

TABELA X: Estimativas do tempo de execução, em segundos, dos algoritmos que implementam os três métodos para ficheiros com N letras

Counter / N	*Exact*	*Probabilistic* (100)	*Lossy* (0.01)
$N = 10^{10}$	2.481×10^{3}	6.503×10^{5}	4.159×10^{3}
$N = 20^{10}$	2.5405×10^{6}	6.6591×10^{8}	4.2588×10^{6}
$N = 10^{15}$	2.481×10^{8}	6.503×10^{10}	4.159×10^{8}
$N = 20^{15}$	8.1297×10^{12}	2.1309×10^{15}	1.3628×10^{13}
$N = 10^{20}$	2.481×10^{13}	6.503×10^{15}	4.159×10^{13}
$N = 20^{20}$	2.6015×10^{19}	6.8189×10^{21}	4.3610×10^{19}

V. Aplicações na vida real

Os métodos *Probabilistic Counter* e *Lossy Counter*, na vida real, aparecem, em certos problemas/tarefas, como uma alternativa a métodos exatos/força-bruta que, caso implementados, poderiam levar a problemas de memória. Uma implementação de métodos de contadores exatos na vida real, que poderia resultar num problema de memória, pode ser, por exemplo, caso pretendesse-mos analisar os produtos com mais vendas na plataforma *Amazon*, [12], num determinado dia/semana, isto pois, como existe uma grande quantidade de produtos a ter em conta, isto traduziria numa grande quantidade de contadores diferentes, um para cada produto, o que, por sua vez, levaria a uma grande quantidade de memória necessária, para aplicar este tipo de métodos.

No entanto, os métodos *Probabilistic Counter* e *Lossy Counter* podem apresentar inúmeras outras aplicações como, por exemplo, proteger a privacidade de um conjunto de dados, no caso dos métodos *Probabilistic Counter*, dado estes providenciarem uma informação mais geral sobre o conjunto, em vez de uma informação mais individual de cada elemento, [13]. Já em relação ao método *Lossy Counter*, este encontra aplicações, por exemplo, em computações em que os dados assumem a forma de um fluxo de dados contínuo em vez de um conjunto de dados finito, como *network traffic measurements*, *web server logs*, e *clickstreams*. [8]

VI. Conclusão e Trabalho futuro

Este trabalho permitiu a aprendizagem de metodologias que combatem problemas de memória, o que pode ser bastante útil em grande empresas, bem como permitiu por em prática a escrita de código.

Visto que, nos dias de hoje, problemas de memória são cada vez menos frequentes, devido à crescente capacidade de armazenamento que cada computador possui, é de esperar que os métodos *Probabilistic Counter* e *Lossy Counter* apresentem cada vez menos utilidade. No entanto, caso nos deparássemos com este tipo de problemas, estes métodos seriam uma boa prática para combatê-los, tal como foi possível concluir com este projeto, em que vimos que estes conseguem obter resultados bastante próximos da realidade e com um tempo de execução não muito mais elevado, quando comparando com o método *Exact Counter*.

Analisando agora, os resultados obtidos ao longo do projeto, poderíamos concluir que, para o tema proposto, isto é, identificar as letras X mais frequentes num ficheiro de texto, ambos os métodos apresentaram resultados satisfatórios, sendo o método *Lossy Counter* o melhor, para valores de X pequenos. Já para valores de X grandes, perto do número total de elementos diferentes, o método *Probabilistic Counter* seria a melhor abordagem.

Para trabalho futuro, este passaria por otimizar os algoritmos dos métodos *Probabilistic Counter*(n) e *Lossy Counter*(e), isto é, verificar quais os parâmetros n e e que, dentro da memoria e do tempo de execução disponíveis, maximizam a exatidão ou minimizam o erro quadrático médio do algoritmo, bem como estudar este métodos, aplicados a uma maior quantidade de dados, de forma a observar mais explicitamente a vantagem que estes métodos podem trazer.

Referências

[1] Project gutenberg, https://www.gutenberg.org

[2] O Livro de Cesario Verde by Cesário Verde. *Project Gutenberg.* (2005,8), https://www.gutenberg.org/ebooks/8698

[3] Camões Os Lusíadas by Luís de Camões. *Project Gutenberg.* (2002,7), https://www.gutenberg.org/ebooks/3333

[4] The Expedition of Humphry Clinker by T. Smollett. *Project Gutenberg.* (2000,4), https://www.gutenberg.org/ebooks/2160

[5] Madeira, J. Slides Teóricos - Algoritmos Avançados (40751). (Universidade de Aveiro)

[6] Morris, R. Counting large numbers of events in small registers. *Communications Of The ACM.* **21**, 840-842 (1978)

[7] Labs, Q. Overview of morris's counters. *Blog.*

[8] Manku, G. & Motwani, R. Approximate frequency counts over data streams. *Proceedings Of The VLDB Endowment.* **5**, 1699-1699 (2012)

[9] Cormode, G. & Hadjieleftheriou, M. Methods for finding frequent items in data streams. *The VLDB Journal.* **19**, 3-20 (2009)

[10] Dimitropoulos, X., Hurley, P. & Kind, A. Probabilistic lossy counting. *ACM SIGCOMM Computer Communication Review.* **38**, 5-5 (2008)

[11] Python Documentation, Random-generate pseudo-random numbers, https://docs.python.org/3/library/random.html

[12] Amazon, https://www.amazon.com/ref=nav_logo

[13] Bojko, D., Grining, K. & Klonowski, M. Probabilistic Counters for Privacy Preserving Data Aggregation. (arXiv,2020), https://arxiv.org/abs/2003.11446

Appendix

I. Resultados - Avaliação dos métodos

TABELA XI: Métricas relativas ao método *Probabilistic Counter* (100) - OCV (O Livro de Cesario Verde), OLUS (Os Lusíadas), TEOHC (The Expedition of Humphry Clinker), ALL (3 livros juntos)

Métrica \ Livro	OLCV	OLUS	TEOHC	ALL
Média	2695.32	10300.45	25816.34	37901.61
Variância	6865722.37	95762143.33	456667313.53	1048095102.15
Desvio Padrão	2620.25	9785.81	21369.77	32374.29
Desvio máximo	6712.68	24068.54	56296.65	85350.38
Desvio médio absoluto	2045.39	7665.82	18213.19	27041.06
Erro Quadrático Médio	105.4	732.625	1845.42	2040.46
Exatidão (Contadores)	8.0%	4.17%	0.0%	0.0%
Número de contadores corretos	2	1	0	0
Exatidão Ranking	100.0%	100.0%	100.0%	92.31%
Número de posições corretas	25	24	26	24
Número de letras	25	24	26	26
Máximo valor de contador	9408	34369	82113	123252
Mínimo valor de contador	4	8	337	271
Tempo de execução (s)	2.661	9.764	26.232	39.443

TABELA XII: Métricas relativas ao método *Exact Counter* - OCV (O Livro de Cesario Verde), OLUS (Os Lusíadas), TEOHC (The Expedition of Humphry Clinker), ALL (3 livros juntos)

Métrica \ Livro	OLCV	OLUS	TEOHC	ALL
Média	2692.88	10295.16	25819.53	37912.07
Variância	6846161.78	95520018.97	456482765.01	1048828957.14
Desvio Padrão	2616.51	9773.43	21365.45	32385.62
Desvio máximo	6718.12	24012.83	56255.46	85420.92
Desvio médio absoluto	2041.70	7660.26	18210.50	27045.24
Número de letras	25	24	26	26
Máximo contador	9411	34308	82075	123333
Mínimo contador	4	8	276	1539
Tempo total (s)	0.010	0.033	0.090	0.145

TABELA XIII: Métricas relativas ao método *Lossy Counter* (0.01) - OCV (O Livro de Cesario Verde), OLUS (Os Lusíadas), TEOHC (The Expedition of Humphry Clinker), ALL (3 livros juntos)

Métrica \ Livro	OLCV	OLUS	TEOHC	ALL
Média	2889.13	11024.72	30169.40	44097.22
Variância	6961147.67	97887290.92	416178897.05	990654257.26
Desvio Padrão	2638.39	9893.80	20400.46	31474.66
Desvio máximo	6521.86	23283.27	51905.59	79235.77
Desvio médio absoluto	7656.95	7583.96	17715.02	26349.08
Erro Quadrático Médio	8351.04	279351.90	964501.77	2751229.40
Exatidão (Contadores)	68.0%	70.83%	80.77%	69.23%
Número de contadores corretos	17	17	21	18
Exatidão Ranking	95.65%	95.45%	100.0%	95.45%
Número de posições corretas	22	21	22	21
Número de letras	23	22	23	22
Máximo contador	9411	34308	82075	123333
Mínimo contador	1	1	1	1
Tempo total (s)	0.014	0.052	0.137	0.218

TABELA XIV: Ranking das letras mais frequentes retornadas por cada método, com respetiva frequência, e tempo de execução de cada algoritmo que implementa o método - *O Livro de Cesário Verde*

Top 5 \ Counter	*Exact*		*Probabilistic* (100)		*Lossy* (0.01)	
1	A	9411	A	9408	A	9411
2	E	8457	E	8485	E	8457
3	0	7109	O	7106	O	7109
4	S	6428	S	6451	S	6428
5	R	4471	R	4482	R	4471
6	I	3652	I	3647	I	3652
7	M	3358	M	3372	M	3358
8	N	3264	N	3256	N	3264
9	D	3138	D	3139	D	3138
10	U	2994	T	2994	U	2994
11	T	2861	U	2876	T	2861
12	C	2539	C	2524	C	2539
13	L	2187	L	2185	L	2187
14	P	1484	P	1494	P	1484
15	V	1090	V	1093	V	1090
16	H	951	H	963	H	951
17	G	901	G	898	G	901
18	B	857	B	856	B	803
19	Q	704	Q	703	Q	687
20	F	701	F	690	F	662
21	Z	344	Z	346	Y	1
22	J	208	J	201	Z	1
23	X	155	X	157	J	1
24	Y	54	Y	53	-	-
25	K	4	K	4	-	-
Tempo(s)	0.0100		3.112		0.0140	

II. Fórmulas utilizadas

Dado um dicionário X de tamanho n em que x_i representa o valor da *key* de índice i, as métricas apresentadas neste projeto podem ser calculadas como:

$$\text{Média} \equiv \mu(X) = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$\text{Variância} \equiv \sigma^2(X) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$$

$$\text{Desvio Padrão} \equiv \sigma(X) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$$

$$\text{Desvio máximo} \equiv maxdev(X) = max\{(x_i - \mu)\}$$

$$\text{Desvio absoluto médio} \equiv mad(X) = \frac{1}{n}\sum_{i=1}^{n}|x_i - \mu|$$

$$\text{Erro Quadrático Médio} \equiv EQM(X) = \frac{1}{n}\sum_{i=1}^{n}|x_i - \hat{x_i}|$$

$$\text{Exatidão} \equiv acc(\%) = \frac{Corretos}{Corretos + Errados} \times 100$$

Onde $\hat{x_i}$ representa o valor exato da *key* de índice i e *Corretos* e *Errados* o número de valores corretos e errados do parâmetro que se pretende estudar, respetivamente.

III. Livros de dimensões maiores

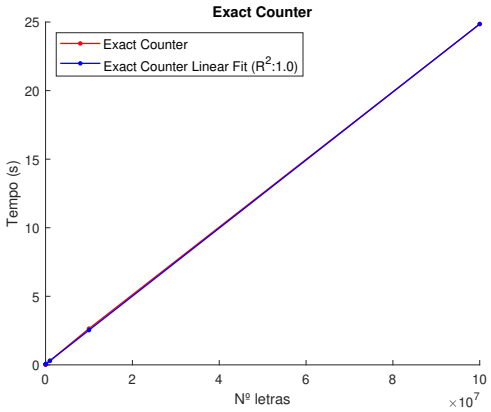

Fig. 2: Tempo de execução do algoritmo que implementa o método *Exact Counter* em função do número de letras + Linear Fit

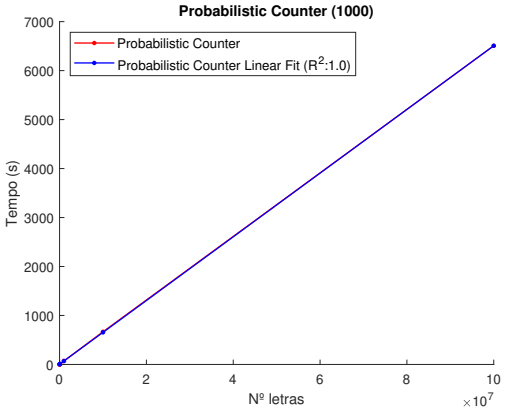

Fig. 3: Tempo de execução do algoritmo que implementa o método *Probabilistic Counter*, repetido 100 vezes, em função do número de letras + Linear Fit

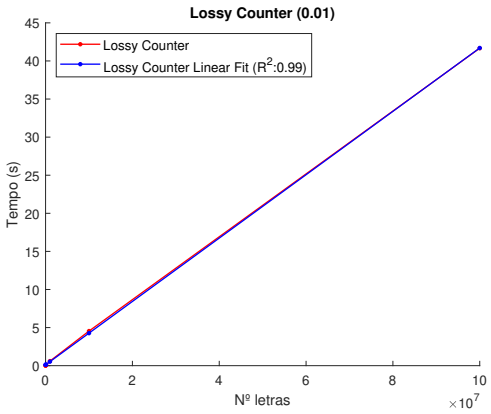

Fig. 4: Tempo de execução do algoritmo que implementa o método *Lossy Counter*, com $e = 0.01$ em função do número de letras + Linear Fit