Hypothesis A1 - Most Frequent Items Count: Lowercase Strings

Luís Fonseca nº 89066

***Resumo* – No presente relatório é feito um estudo a um algoritmo de *stream* de dados, cujo objetivo é encontrar os itens mais frequentes. Assim tenta-se perceber o funcionamento do mesmo e em que cenários este é mais bem aplicado. São analisados vários dados de forma a fazer um estudo sobre diversos ângulos.**

***Abstract* – This report studies a data stream algorithm; whose goal is to find the most frequent items. Thus, we seek to have a better understanding on how it works and when is it a best fit for a situation.**

I. CONTEXT

No mundo atual, a informação chega até nós numa *stream* contínua de grandes quantidades de dados. Desta forma, torna-se cada vez mais necessário processar os dados enquanto estes surgem, de forma a ter um melhor entendimento do que está a acontecer. Assim, foram desenvolvidos ao longo do tempo vários algoritmos que respondam a esta necessidade. Um deles é o aqui estudado neste relatório, o algoritmo de **Space-Saving Count**. Como o nome sugere, este algoritmo permite poupar espaço em memória, uma vez que apenas um número máximo de k valores são guardados. Este visa a responder ao problema de frequências, isto é, de quais os itens mais frequentes dum conjunto de dados.

A nível de funcionamento é conceptualmente bastante simples. Após garantir a existência de uma estrutura de dados devida, capaz de guardar um par <key, value>, a ocorrência de um item é registada no mesmo. Se não houver mais capacidade no contador para um novo item, este substitui o mais antigo, e a contagem é incrementada em 1.

O pseudocódigo da figura 1 pode ajudar a entender melhor o funcionamento.

Algorithm 3: SPACESAVING(k)

```
n ← 0;
T ← ∅;
foreach i do
    n ← n + 1;
    if i ∈ T then c_i ← c_i + 1;
    else if |T| < k then
        T ← T ∪ [i];
        c_i ← 1;
    else
        j ← arg min_{j∈T} c_j;
        c_i ← c_j + 1;
        T ← T ∪ [i] \ [j];
```

Figura 1 - Pseudocódigo do Algoritmo

II. CONSIDERATIONS

No contexto do estudo da hipótese proposta, foi analisada a influencia de `k` e `n` neste algoritmo. A primeira variável corresponde ao número de itens a guardar na estrutura de dados, enquanto que a segundo corresponde ao tamanho da sequência a ser alvo de contagem.

Adicionalmente, para além do estudo de sequências com distribuições de dados variadas (neste caso, gaussianas), também se abordou de forma sucinta o comportamento para uma distribuição uniforme.

Embora de forma não tão extensiva, em oposição a relatórios anteriores, foram usadas duas métricas para a avaliação do desempenho do algoritmo. Estas foram inspiradas em [1] e [2] e são *precision* e *recall*.

As fórmulas para o cálculo das mesmas são as seguintes:

$$precision = \frac{number\ of\ retrieved\ frequent\ elements}{number\ of\ retrieved\ elements}$$

$$recall = \frac{number\ of\ retrieved\ frequent\ elements}{total\ number\ of\ frequent\ elements\ in\ the\ dataset}$$

Note-se que é considerado frequent elements, os elementos mais comuns, ou seja os que aparecem mais vezes, e tendo em conta este algoritmo, os que aparecem (definido de forma arbitrária) mais de $\varepsilon \times n$ das vezes, com $\varepsilon = 1/k$. Desta forma, precision indicará a razão de itens frequentes contados entre todos os obtidos, e recall indicará de todos os frequentes, quantos foram contados. A figura 2 [3] pode ajudar a perceber o raciocínio.

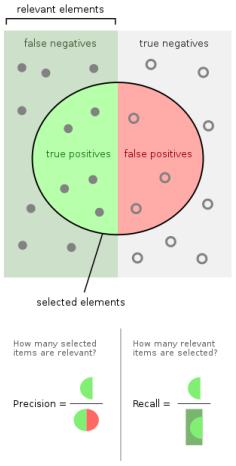


Figura 2 - Intuição de *Precision* e *Recall*

III. NORMAL DISTRIBUITION

No sentido de perceber como se comportava o algoritmo face a um contexto não "homogéneo", foram geradas sequências a partir de uma distribuição "pseudo-"normal – uma vez que é feito um pequeno ajuste aquando da geração de sequências. Tal permite não só que haja caracteres (muito) mais frequentes que outros, como se pode entender como uma representação mais fidedigna de um cenário real.

A. Variating K

Como referido, uma das variáveis em estudo é o K, ou seja, o número de elementos a guardar na estrutura de dados.

I. n = 1000

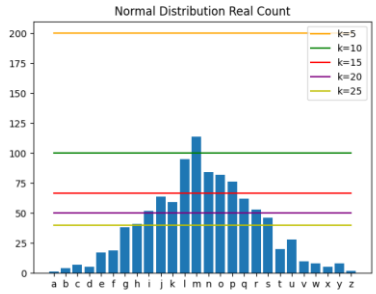


Figura 3 - Distribuição de Caracteres para Sequencia Pequena

Isto leva-nos ao gráfico anterior, onde se apresenta, para uma sequência de tamanho n=1000, a distribuição de caracteres gerados. Através da notória distribuição Gaussiana da mesma, facilmente se entende a discrepância entre ocorrências de caracteres existente. As linhas apresentadas, representam os valores do *threshold* $\varepsilon \times n$, com $\varepsilon = 1/k$, a partir do qual as ocorrências são consideradas "frequentes", para cada k.

Somos assim trazidos para os seguintes gráficos.

O primeiro ponto que se pode observar, principalmente para **k**=5, é que a soma das contagens vai até **n**=1000. Adicionalmente, a média dos valores é, portanto **n/k**. Tal não se apresenta como uma surpresa, uma vez que são propriedades do algoritmo já conhecidas, mas servem para conferir confiança à implementação desenvolvida.

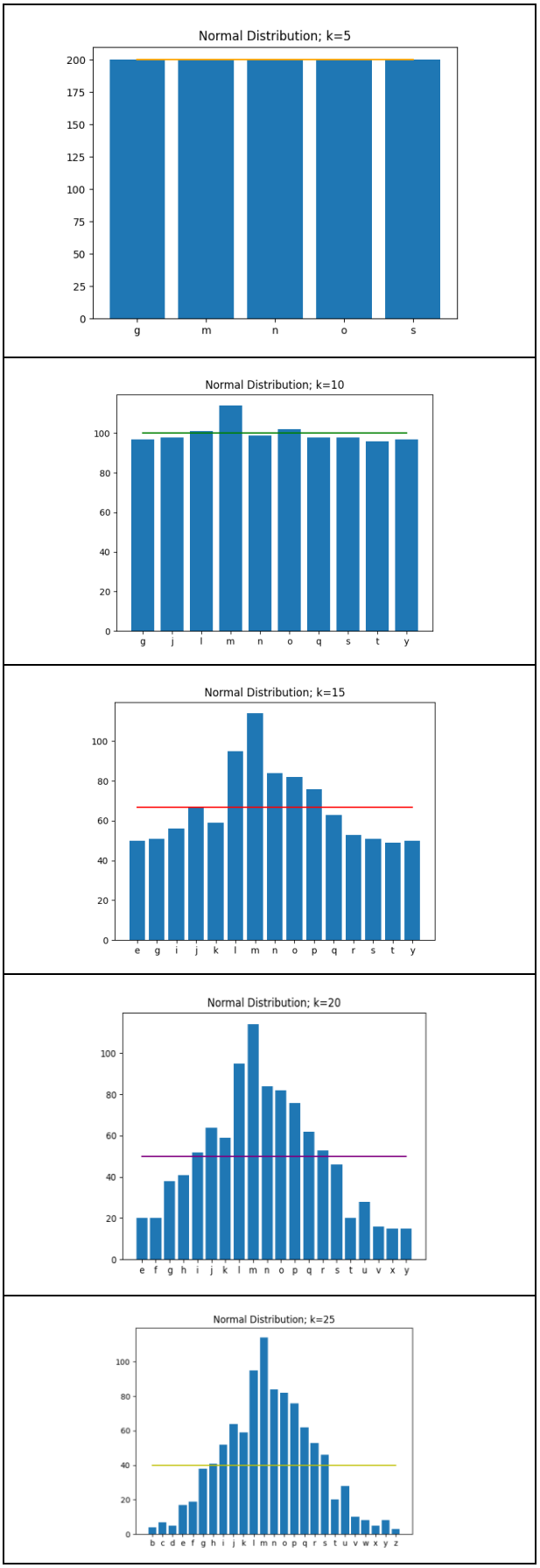


Figura 4 – Resultados Da Contagem para Sequencia Pequena

Posto isto, nota-se que à medida que se aumenta o valor de **k**, os resultados obtidos vão-se tornando cada vez mais próximos dos reais, estando cada vez mais bem definida a curva gaussiana graficamente.

Assim, é torna-se obvio que **k** influencia os resultados obtidos.

Para k demasiado pequenos (k=5), nenhum valor real ultrapassou o *threshold*. Contudo, apesar de estarem abaixo do referido *threshold*, alguns dos valores mais frequentes, são já capturados pelo contador, no caso as letras "m", "n" e "o"; correspondendo as 2 letras restantes a valores intermédios – evidenciando assim a eventual "fraqueza" do algoritmo, que dá bastante importância aos valores finais / mais recentes da sequência, acabando por os sobrestimar. Porém, como nenhum valor ultrapassou o *threshold*, nenhum resultado se pode considerar confiável, e num cenário real, não seria possível perceber quais os itens mais frequentes.

Posto isto, à medida que se aumenta o valor de k cada vez mais valores *frequentes* vão sendo contabilizados. Isto de tal forma, que é possível perceber as relações de grandeza entre as letras contabilizadas. Noutras palavras, letras *frequentes* com mais ocorrências estão associadas a um maior valor que letras *frequentes* menos ocorrentes, como se pretende.

Obviamente, se $k = |A|$, sendo A o alfabeto em estudo, os resultados obtidos são os exatos, contudo isso tornaria o algoritmo inútil uma vez que não estaria a ir ao encontro do seu propósito.

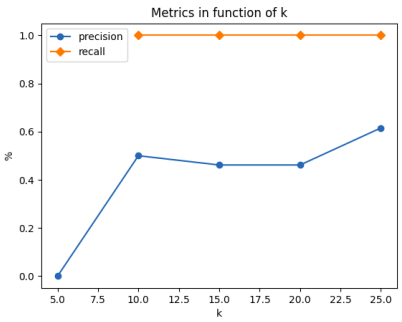


Figura 5 - Métricas para Sequencia Pequena

Finalmente, no que respeita às métricas, obtemos o resultado apresentado na figura 5.

É importante notar que valores não existentes de *recall* acontecem quando não existem ocorrências frequentes nos valores reais, *zerando* o denominador da equação. Além disso, de maneira geral, é impossível que para $|A| > 1$, a *precision* seja igual a 1, uma vez que o número de ocorrências *frequentes* é sempre menor ao de ocorrências totais.

Posto isto, observa-se que uma vez que haja ocorrências frequentes no contador exato, estas ocorrências são também sempre captadas por este contador. Obviamente isto não surge como uma surpresa, uma vez que se estão a considerar como ocorrências *frequentes* aquelas que o algoritmo irá contar certamente. Não obstante tal observação serve para reforçar a ideia de que de facto, letras mais frequentes são capturadas.

No que respeita à precisão, esta tem tendência a subir, uma vez que o número de ocorrências frequentes vai aumentando conforme o k também aumenta.

I. n = 10000000

Para uma sequência de maior tamanho, temos para a seguinte distribuição:

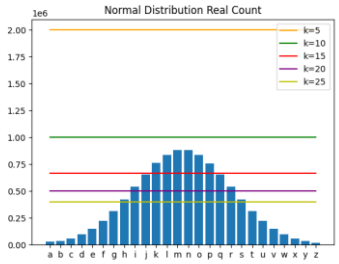
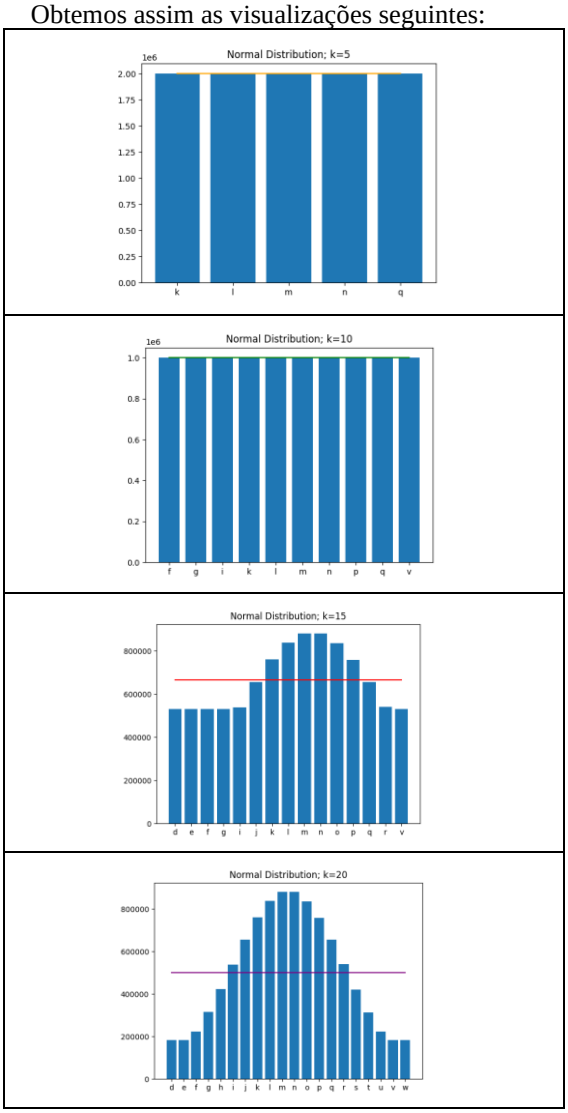


Figura 6 - Distribuição de Caracteres para Sequencia Grande

Obtemos assim as visualizações seguintes:



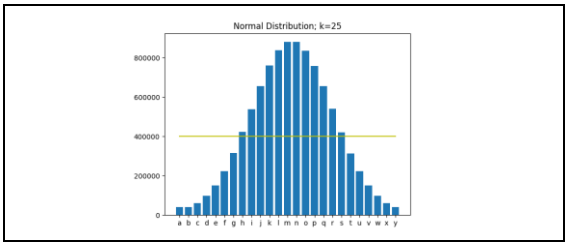


Figura 7 - Resultados Da Contagem para Sequencia Grande

Não procurando repetir o que foi dito anteriormente, nota-se que em geral, as letras mais frequentes são capturadas cada vez melhor à medida que k aumenta.

Para o caso de um k=5 reduzido, apesar de as letras no contador não serem classificáveis em *frequentes* segundo o *threshold* definido, as que se encontram lá inseridas correspondem maioritariamente às mais *frequentes* reais, não sendo, contudo, capturadas as frequências relativas entre elas.

No que diz respeito às métricas, não há também muito a acrescentar. Assim, não são abordados mais gráficos sobre estas métricas, uma vez que as conclusões a retirar são bastante idênticas.

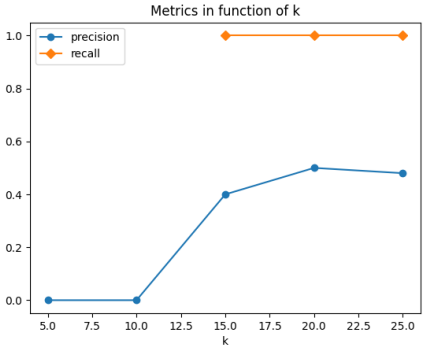
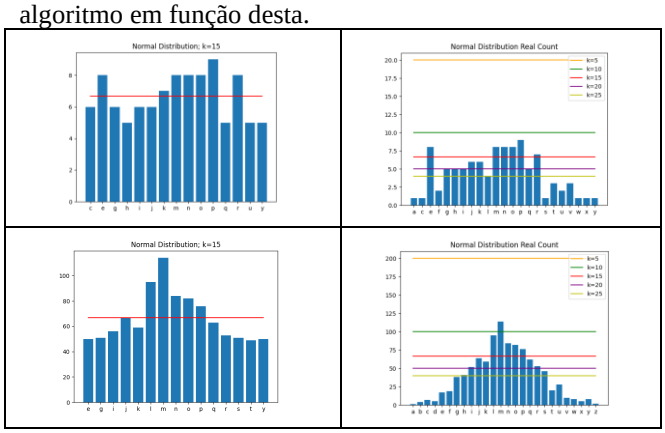


Figura 8 - Métricas para Sequencia Grande

B. *Variating N*

A segunda variável em estudo, foi o tamanho da sequência gerada, para perceber como se comporta o algoritmo em função desta.



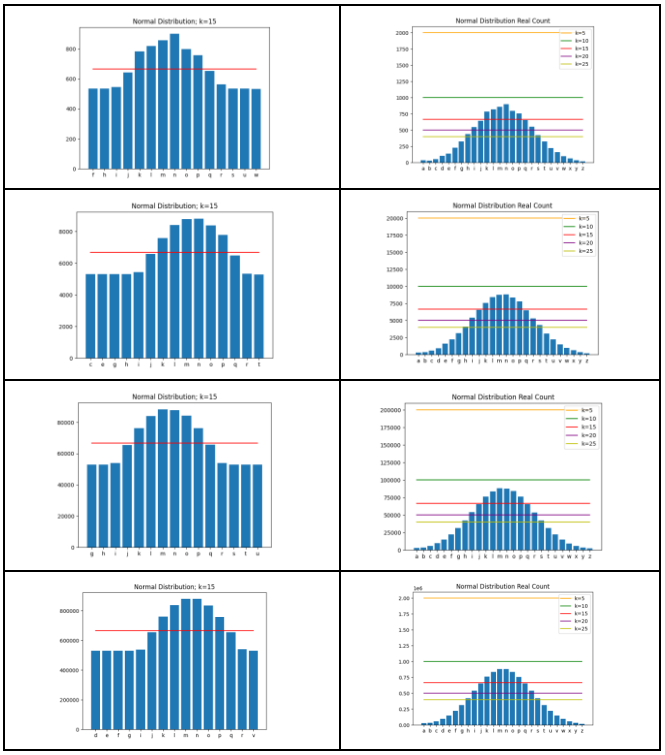


Figura 9 - Resultados em Função de N. Valores obtidos pelo Contador (esquerda) vs. Reais (direita)

Assim, obtemos a figura anterior, onde se fixou um k=15, que num |A| de cerca de 24 letras, apresentou-se como um bom compromisso para a analise.

Observando o conjunto de visualizações, 2 conclusões principais podem ser obtidas:

1. Não parece haver indícios que o tamanho da sequência gerada influencie os resultados obtidos. Desde que se verifiquem ocorrências *frequentes*, estas são detetadas pelo contador, que é capaz não só de manter a razão entre essas contagens igual à real, como apresenta os valores exatos para essas contagens.
2. Relembra-se o papel de k, que está relacionado com o número de elementos contados cuja contagem se sabe exatamente – só os valores acima do *threshold* influenciado por esta variável são contados efetivamente.

V. UNIFORM DISTRIBUITION

Por fim, em jeito de curiosidade, foi também analisado o comportamento do algoritmo para letras com ocorrências distribuídas uniformemente.

A. *Variating k*

Assim, estudou-se também aqui o comportamento em função do k.

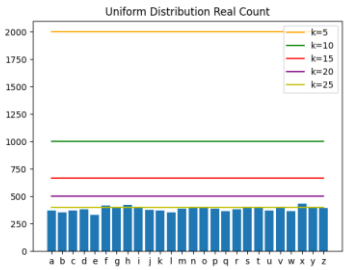
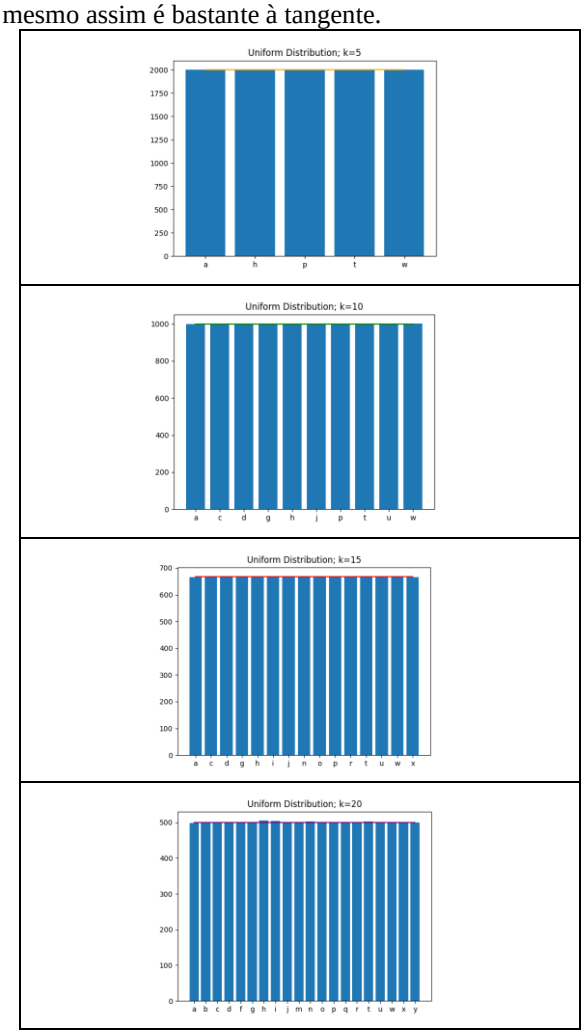


Figura 10 - Distribuição de Caracteres para Sequencia Uniforme

Já aqui, trabalhou-se com 1 sequencia de 10000 caracteres de comprimento. Cuja distribuição se pode ver na figura 10.

Algo bastante importante que se pode notar muito rapidamente, é o facto de que, com nenhum *threshold* é possível classificar contagens reais em *frequentes*, exceto para `k=25`, que é apenas uma unidade inferior a `|A|=26`, e que mesmo assim é bastante à tangente.



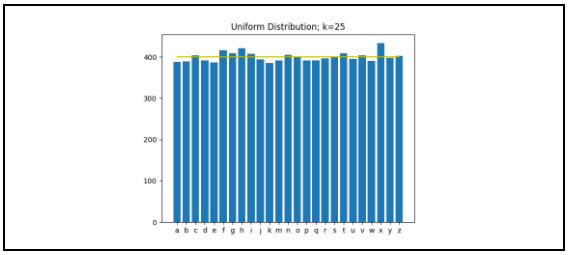


Figura 11 - Resultados Da Contagem para Sequencia Uniforme

Esta observação é novamente suportada nos resultados experimentais obtidos. Noutras palavras, não só não é possível perceber a razão entre ocorrências de letras distintas, como - ao contrário das distribuições anteriores - não são contadas sequer as letras mais *frequentes*, que também apenas o são por uma margem mínima. Isto significa que, num contexto em que há valores com a mesma frequência, o algoritmo é incapaz de indicar quais os mais *frequentes*, uma vez que de facto, não há.

VI. CONCLUSION

Como observado, este algoritmo, tem o objetivo de poupar espaço, registando apenas até `k` contagens em qualquer momento. Desta forma, apresenta o *tradeoff* de não registar as contagens todas, apresentando, contudo, as mais relevantes / frequentes, que é o verdadeiro objetivo do algoritmo. Deve-se então balançar o espaço em memória que se pretende usar contra a quantidade de elementos cuja contagem se pretende saber.

Apesar dos valores acima do várias vezes referido *threshold* serem os únicos cuja contagem é certa, os restantes valores não se devem considerar "lixo". Estes também contribuem com informação útil, no sentido em que se sabe que se encontram abaixo do *threshold* de certeza.

Embora se tenha estudado o comportamento para uma distribuição uniforme, este foi um estudo muito breve, uma vez que as conclusões foram mais fáceis de retirar: este algoritmo não é indicado nesse tipo de situações.

Contudo, é sabido que na natureza / cenários reais, muitas vezes apresentam 1 distribuição normal, logo este algoritmo seria de facto relevante, num contexto de grande fluxo de dados, em que fosse necessário perceber quais os valores mais relevantes, com limitação de processamento / memoria.

VII. HOW TO RUN

Como última nota, indica-se como se deve executar o código desenvolvido.

Este é possível executar de 3 formas:

1. **Gerar Sequencias de Vários Tamanhos**
   Isto irá dar Override aos ficheiros existentes no diretório atual, pelo que caso se queiram guardar estes, se deve ter isso em atenção.
   ```
   python3 generateSequences.py
   ```

2. **Fazer a Contagem destas Sequências segundo do Algoritmo**
   Tendo em conta um dos ficheiros gerados, é feita a contagem segundo o algoritmo, onde são incluídas visualizações gráficas ao longo da execução, tornando-a "síncrona".
   ```
   python3 SpaceSaving.py 1000000-Normal\ Distribution.txt
   ```
3. **Executar tudo**
   Serve para executar os 2 passos anteriores, criando sequencias em memoria e contando-as, e guardando as visualizações em disco. Teve o propósito de facilitar a realização do relatório.
   ```
   python3 ReportSpaceSaving.py
   ```

   Como última nota, no diretório results/ encontram se todos os gráficos gerados a partir deste estudo, sujeitos a Override ao executar o ponto 3 anterior.

REFERENCES

[1] [ONLINE]. AVAILABLE: HTTPS://IMOUMOULIDOU.GITHUB.IO/SPACESAVING_PRESENTATION.PDF.

[2] [ONLINE]. AVAILABLE: HTTP://WWW.CSE.UST.HK/~RAYWONG/COMP5331/REFERENCES/EFFICIENTCOMPUTATIONOFFREQUENTANDTOP-KELEMENTSINDATASTREAMS.PDF.

[3] [ONLINE]. AVAILABLE: HTTPS://EN.WIKIPEDIA.ORG/WIKI/PRECISION_AND_RECALL.