Algoritmos Avançados - Randomized Algorithm

Tiago Santos

***Abstract* –The objective of this assignment was to count the number of letters in a text file using 3 different methods: exact counter, fixed probability counter 1/2, and Frequent-Count. Tests were conducted to evaluate the effectiveness of each method.**

I. Introduction

This report investigates the application of methods to determine the number of occurrences of a specific event. In this instance, we will count the number of occurrences of each letter in a text file. To achieve this, we will employ three distinct approaches: the exact count method, the fixed probability counter with a 1/2 probability, and the frequent count technique. The evaluation of each algorithm's efficiency focuses on measuring its execution time and the success rate in accurately identifying the correct number of letters.

II. Problem contextualization

This report delves into the exploration of methods designed to ascertain the occurrences of each letter within a given text file. The focus is on three distinct approaches: the exact count method, the fixed probability counter with a 1/2 probability, and the Misra & Gries algorithm for the frequent count technique. These methods were chosen due to their varying degrees of complexity and computational requirements. By comparing and contrasting these methodologies, we aim to provide valuable insights into their applicability and performance characteristics. This exploration is essential not only for theoretical understanding but also for practical considerations, as the choice of the counting method can significantly impact the feasibility and effectiveness of subsequent analyses.

III. Implementation

For this project, three algorithms were employed: exact count, fixed probability counter, and the Misra & Gries frequent count algorithm. Each of these algorithms serves as a distinct method for determining the number of occurrences of each letter in a given text file.

A. Exact Count Algorithm

The exact count method is a straightforward approach where each occurrence of a letter is meticulously counted without any approximations. This algorithm provides a baseline for accuracy, serving as a reference point for evaluating the performance of the other methods.

B. Fixed Probability Counter (1/2) Algorithm

The fixed probability counter introduces a probabilistic element to the counting process. In this case, each letter encountered has a 1/2 probability of being counted. This approach aims to strike a balance between accuracy and computational efficiency, making it particularly interesting for scenarios where a high level of precision is not essential, but a reduction in computational resources is desired.

C. Misra & Gries Frequent Count Algorithm

The Misra & Gries frequent count algorithm [1] is a well-established method known for its efficiency in counting frequent elements in a stream of data. Specifically adapted for our project, this algorithm maintains a compact representation of the most frequent letters encountered, offering a trade-off between accuracy and reduced memory requirements. The Misra & Gries algorithm provides an intriguing alternative, particularly when dealing with large datasets where memory efficiency is critical.This algorithm provides, for any letter, l, a frequency estimate satisfying

$$f_l - \frac{m}{k} \leq f_l^* \leq f_l$$

were m is the length of the data stream or, in this case, the total number of letters in the text. If some letter has $f_l > \frac{m}{k}$, its counter $A[l]$ will be positive, i.e., no item with frequency $\frac{m}{k}$ is missed.

IV. Code

A. Exact Count

This method is the simplest among the three employed algorithms and it was incorporated into the function exact_counter. through each character in the text file, the function updates a dictionary, letter_count, keeping track of the count for each encountered letter. In the end, the function returns the dictionary which will be useful for comparing the results with the other methods

```
1 def exact_counter(text):
2     letter_count = {}
3     for char in text:
4         letter_count[char] = letter_count.get(
      char, 0) + 1
5     return letter_count
```

B. Fixed Probability Counter (1/2) Algorithm

This method is similar to the previous one in the sense that it iterates through the text file and incorporates a randomized element to determine which letters are counted. In the case of the Fixed Probability Counter (1/2) Algorithm, each letter's inclusion in the count is determined by a 50% probability. This means that right after reading a letter from the text file, a random probability is generated, and if this probability is greater than 0.5, the respective letter is considered for inclusion in the count.

```
def fixed_counter(text):
    letter_count = {}
    for char in text:
        prob = random.randint(0, 1)
        if prob>0.5:
            letter_count[char] = letter_count.
    get(char, 0) + 1
    return letter_count

#after calling the function
for key in letter_count:
        letter_count[key] = letter_count[key]
    * 2
```

C. Misra & Gries Frequent Count Algorithm

The Misra & Gries Frequent Count Algorithm is designed to efficiently identify the k most frequent items in a stream of data. This method is implemented in freq_counter, providing an adaptive and memory-efficient approach to dynamically updating the most frequent items as the data stream unfolds. The code dynamically updates a dictionary, **frequent_items**, as it processes each character in the text. The algorithm adapts by incrementing counts for existing items, adding new ones if below the specified limit, and employing a decrement-and-delete strategy to maintain a compact representation of the k most frequent items. The resulting dictionary is then returned, providing a memory-efficient summary of the most frequent items in the data stream.

```
def freq_counter(text,k):
    frequent_items = {}

    for char in text:
        if char in frequent_items:
            frequent_items[char] += 1
        elif len(frequent_items) < k - 1:
            frequent_items[char] = 1
        else:
            for key in list(frequent_items.
    keys()):
                frequent_items[key] -= 1
                if frequent_items[key] == 0:
                    del frequent_items[key]

    return frequent_items
```

V. Results and Discussion

To have a better understanding of these methods, several tests were conducted to evaluate their performance in counting the occurrences of letters in a given text as well as the success rate. To be able to compare different results two different text files were used: loremipsum.txt Latin language and short (2807 characters) and Book_text.txt English and large (42710) taken from Project Gutenberg website [2].

A. Exact Count

The exact count method served as a benchmark for accuracy. By iterating through each character in the text, this method accurately determined the frequency of each letter, providing a baseline for comparison.

Letter	Value
E	348
I	269
U	254
S	241
T	235
A	226
N	178
L	159
R	154
M	147
O	113
C	106
D	90
P	83
V	48
Q	46
G	42
B	33
H	17
F	17
J	1

Time: 0.00088909

(a) Latin

Letter	Value
E	4314
T	3080
A	2798
O	2510
H	2311
N	2220
I	2185
S	2088
R	1860
D	1531
L	1336
U	948
W	913
C	839
M	763
F	703
G	699
Y	612
B	509
P	473
K	342
V	312
J	60
Q	41
X	33
Z	5
Ï	1

Time: 0.00184366

(b) English

Fig. 1: Comparison between time values using Exact Count

As expected from the results seen in 4, due to the difference in the number of characters between the two files, the times taken for the exact count method exhibit notable variations. The method performed efficiently on the smaller 'Loremipsum.txt' file, completing in 0.00013284 seconds. However, for the larger 'Book_text.txt' file, the execution time increased to 0.00187009 seconds, reflecting the proportional impact of dataset size on method efficiency.

B. Fixed Probability Counter (1/2) Algorithm

The fixed probability method introduced an element of randomness. For each character in the text, a random probability was assigned, and if the generated value exceeded a threshold (in this case, 0.5), the letter was counted.

To better understand the performance and variability of the fixed probability counter method, a series of experiments were conducted. It involved processing the same input text using the fixed probability counter method in 10 iterations. The correctness of the method was compared against the exact count results. For each iteration, the letter counts were recorded, and statistical measures such as the average, minimum, and max-

imum counts were computed. Additionally, the execution time for each iteration was captured to evaluate the computational efficiency of the fixed method.

To quantify the accuracy of the fixed method, percentage error was introduced. This measures the relative deviation of the average count from the exact count, providing insights into the precision of the probabilistic approach. The percentage error was calculated for each letter and included in the results.

Results:

Letter	Correct	Average	Min	Max	Percentage Error
E	348	349	316	374	0.29%
I	269	273	250	292	1.49%
U	254	248	222	278	2.36%
S	241	239	222	264	0.83%
T	235	231	204	256	1.70%
A	226	226	192	244	0.00%
N	178	171	142	192	3.93%
L	159	164	142	186	3.14%
R	154	151	136	170	1.95%
M	147	152	128	174	3.40%
O	113	117	104	136	3.54%
C	106	107	92	118	0.94%
D	90	93	84	106	3.33%
P	83	83	72	102	0.00%
V	48	43	36	48	10.42%
Q	46	47	32	54	2.17%
G	42	43	30	50	2.38%
B	33	32	26	38	3.03%
H	17	16	12	28	5.88%
F	17	17	12	22	0.00%
J	1	1	0	2	0.00%

Fig. 2: Results from fixed probability method with Loremipsum.txt

Results:

Letter	Correct	Average	Min	Max	Percentage Error
E	4314	4286	4242	4326	0.65%
T	3080	3076	2990	3140	0.13%
A	2798	2788	2696	2886	0.36%
O	2510	2526	2406	2602	0.64%
H	2311	2294	2214	2420	0.74%
N	2220	2221	2154	2280	0.05%
I	2185	2187	2110	2258	0.09%
S	2088	2101	2048	2156	0.62%
R	1860	1858	1818	1924	0.11%
D	1531	1559	1498	1598	1.83%
L	1336	1333	1262	1384	0.22%
U	948	930	902	966	1.90%
W	913	906	850	940	0.77%
C	839	832	792	862	0.83%
M	763	761	704	822	0.26%
F	703	707	654	740	0.57%
G	699	694	642	746	0.72%
Y	612	610	586	638	0.33%
B	509	510	478	556	0.20%
P	473	465	430	496	1.69%
K	342	342	304	370	0.00%
V	312	310	284	340	0.64%
J	60	60	46	70	0.00%
Q	41	37	30	48	9.76%
X	33	32	26	48	3.03%
Z	5	4	2	8	20.00%
Ï	1	1	0	2	0.00%

Fig. 3: Results from fixed probability method with Book_text.txt

In the results obtained using the fixed probability method it's evident that the counts vary across multiple runs due to the inherent randomness introduced. With the average values, we can see a more stable representation of the letter frequencies. Comparing these results with the exact count reveals the trade-off between accuracy and variability. While the exact count provides precise frequencies, the fixed probability method introduces randomness, resulting in varied counts across runs.

Times:
Time 1: 0.00220540
Time 2: 0.00231029
Time 3: 0.00179742
Time 4: 0.00174239
Time 5: 0.00180887
Time 6: 0.00189841
Time 7: 0.00191361
Time 8: 0.00193247
Time 9: 0.00182238
Time 10: 0.00227771

(a) Latin

Times:
Time 1: 0.01920341
Time 2: 0.01850391
Time 3: 0.01951303
Time 4: 0.01954973
Time 5: 0.02045614
Time 6: 0.01937242
Time 7: 0.01859224
Time 8: 0.01878266
Time 9: 0.01883965
Time 10: 0.01982845

(b) English

Fig. 4: Comparison between time values using Fixed Probability Counter

The percentage errors for both input files were relatively low, indicating that the fixed probability counter method maintained a reasonable level of accuracy. However, it's essential to note that the percentage errors varied across different letters, highlighting the impact of randomness on individual character counts.

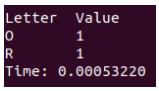
As for the execution time, just like in the Exact Count method, it's clear that the bigger the text the longer the execution time. We can also see that the time it took for the Fixed method to complete its task is longer than the Exact Count, even though in theory it should have been shorter due to not having to iterate through all of the text file. This may be attributed to the specific implementation of the Fixed method. Notably, the use of random.randint(0, 1) to decide whether to count a character introduces a randomness factor that, in practice, contributes to an increased overall execution time. The generation and utilization of random numbers in the decision-making process are key factors influencing the observed performance outcome.

C. Misra & Gries Frequent Count Algorithm

The frequent count method, based on the Misra & Gries algorithm, aimed to estimate the most frequent items in the text while operating under memory constraints. The algorithm maintained a set of at most k-1 items with the highest counts, allowing for a trade-off between accuracy and memory usage.

In order to better understand this method, different values for k, representing the number of frequent items, were used to explore its behavior under varying conditions.

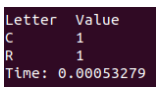
Using Loremipsum.txt first with small values of k(e.g., 3 and 5) the algorithm is not able to find many of the most frequent values. As the value of k increases (e.g.,



Letter	Value
O	1
R	1

Time: 0.00053220

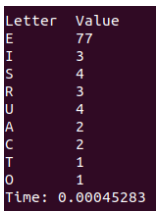
(a) $k = 3$



Letter	Value
C	1
R	1

Time: 0.00053279

(b) $k = 5$



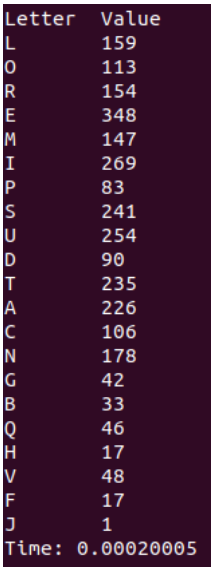
Letter	Value
E	77
I	3
S	4
R	3
U	4
A	2
C	2
T	1
O	1

Time: 0.00045283

(c) $k = 10$

Fig. 5: Comparison Between Frequent Counter for $k = 3$, *5, and 10, for Loremipsum.txt*

10 and 22), the algorithm performs much better in capturing the most frequent items efficiently, getting closer or even exact, for the case of k=22, to the exact value. This occurrence may be attributed to the algorithm's ability to adapt and refine its results as k increases. When k is small, the Misra & Gries algorithm has a limited capacity to identify and maintain counts for a larger set of frequent items. As a result, it might overlook or miscount some of the most frequent items in the dataset. However, as k becomes larger, the algorithm adjusts its counters to capture a greater number of frequent items.

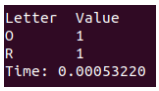


Letter	Value
L	159
O	113
R	154
E	348
M	147
I	269
P	83
S	241
U	254
D	90
T	235
A	226
C	106
N	178
G	42
B	33
Q	46
H	17
V	48
F	17
J	1

Time: 0.00020005

Fig. 6: Results from Frequent Counter method for k=22 with Loremipsum.txt

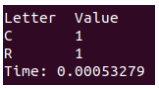
For Book_text.txt, the results from the Misra & Gries algorithm exhibit a similar trend, yet with some variations compared to Loremipsum.txt. With small values of k (e.g., k=5 and k=10), the algorithm struggles to accurately identify and maintain counts for the most frequent items. The output is limited and may not capture the complete set of highly frequent characters. As k increases (e.g., k=20 and k=28), the algorithm demonstrates a noticeable improvement. This behavior is consistent with the adaptability of the Misra & Gries algorithm; as k grows, the algorithm can allocate more counters to capture a larger number of frequent



Letter	Value
O	1
R	1

Time: 0.00053220

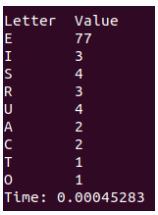
(a) $k = 5$



Letter	Value
C	1
R	1

Time: 0.00053279

(b) $k = 10$

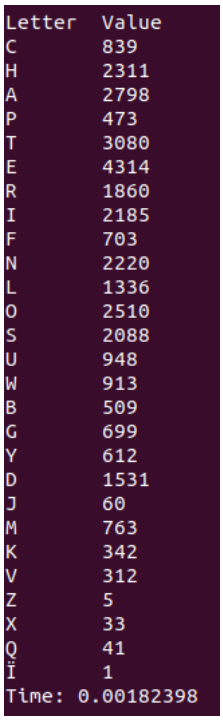


Letter	Value
E	77
I	3
S	4
R	3
U	4
A	2
C	2
T	1
O	1

Time: 0.00045283

(c) $k = 20$

Fig. 7: Comparison Between Frequent Counter for $k = 5$, *10, and 20 for Book_ text.txt*



Letter	Value
C	839
H	2311
A	2798
P	473
T	3080
E	4314
R	1860
I	2185
F	703
N	2220
L	1336
O	2510
S	2088
U	948
W	913
B	509
G	699
Y	612
D	1531
J	60
M	763
K	342
V	312
Z	5
X	33
Q	41
Ï	1

Time: 0.00182398

Fig. 8: Results from Frequent Counter method for k=28 with Book_text.txt

items, resulting in counts that closely align with the exact values.

VI. Usage

Run the Exact Count method:

```
python3 main.py name_file.txt "method"
```

Replace "method" with one of the available methods: "exact" and "fixed"

To run the Frequent Count method the user must choose the k (number of most frequent items).

```
python3 main.py name_file.txt freq k
```

To run the experiment version with times and other data about the methods just replace the "main.py" with "experiment.py".

VII. Conclusion

After exploring these different methods, we can see that some results were not what we expected in the-

ory, as the Fixed Probability method shows unexpectedly longer execution times, possibly due to the random decision-making process in the method. As for the Misra & Gries algorithm, with different k values approaching the exact count accuracy as k increases. However, its efficiency varies with dataset characteristics, as seen with larger text files.

References

[1] Misra & Gries Frequent count algorithm `https://en.wikipedia.org/wiki/Misra%E2%80%93Gries_summary`

[2] Project Gutenberg, The Curse of the Reckaviles by Walter S. Masterman `https://www.gutenberg.org/ebooks/72629`