Most Frequent Letters

Eduardo Santos, nºmec 93107, eduardosantoshf@ua.pt

Abstract – **The objective of this assignment was to identify the most frequent letters in text files using different methods and to evaluate the quality of estimates regarding the exact counts. Three types of counters were implemented: an Exact Counter, a Decreasing Probability Counter, and a Frequent Counter.**

Keywords –**Counting Algorithms, Data Stream Algorithms, Probabilistic Counters, Exact Counter, Decreasing Probability Counter, Frequent Counter**

I. Introduction

A. Decreasing Probability Counter

The aim of a probabilistic counter is to count very large numbers using only a little space to store the counter. Counting a very large number of events using an exact counter will result in a large memory usage, which is something that should be avoided, as memory is expensive, and makes a program less efficient. To mitigate this problem, probabilistic counters were created. So, for each call to an increment method on the counter, its actual value is updated with probability p. Using this method, we are trading accuracy for the ability to count up to very large numbers with little storage space.

Even though nowadays memory is no longer scarce, this approach is still useful when treating/counting massive data volumes, when there is a need for quick and memory-efficient processing.

B. Frequent Counter

A streaming algorithm is an algorithm for processing data streams in which the input is presented as a sequence of items and can be examined in only a few passes, typically just one. The frequent problem happens when, given a sequence of items, we want to identify those which occur most frequently. This can also be expressed as finding all items whose frequency exceeds a specified fraction of the total number of items.

II. Problem Description

The goal of this assignment was to identify the most frequent letters in literary works from Project Gutenberg [1] using different methods and to evaluate the quality of estimates regarding the exact counts.

In order to accomplish that, three different approaches were developed and tested:

- exact counter
- approximate counter - decreasing probability counter with probability $\frac{1}{\sqrt{2}^k}$
- frequent counter - Misra & Gries algorithm to identify frequent items in data streams

The testing involved an analysis of the computational efficiency and limitations of the developed approaches was carried out, in terms of absolute and relative errors, computing the mean, minimum, and maximum of each error, as well as computing the standard deviation and variance. Finally, for each method, the most frequent letters were identified, checking if they were in the same relative order.

III. Implementation Description

A. Main

Running the **main.py**, with the **–help** flag, a few running options are presented.

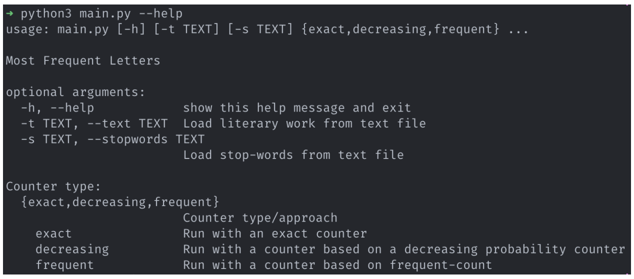

Fig. 1: Help Menu of the Main Program

The **-t** flag represents the filename of the literary work to be read and processed by the program, which can be found inside the **/texts/** directory. The **-s** flag represents the filename of the stop-words to be ignored when processing the text file. The stop-words files can be found inside the **/stop-words/** directory. After the previous parameters, we have to specify the counter type to be computed, using one of the following:

- **exact** - count using the exact counter
- **decreasing** - count using the decreasing probability counter (with probability $\frac{1}{\sqrt{2}^k}$)
- **frequent** - count using the frequent counter (we must also specify the k parameter, using the flag **-k**)

B. Counter

The **counter.py** contains the main logic of the solution, using the **ABC** [3] (Abstract Base Classes) Python module, the **ExactCounter**, **DecreasingProbabilityCounter**, and **FrequentCounter** classes extend the **Counter** parent class, using an OOP (object-oriented programming) model. Inside the **Counter** class there is the *read_letters()* method, which is responsible for parsing the text file, removing the Project Gutenberg file headers and footers, all stop-words and punctuation marks, as well as converting all letters to uppercase.

```
def read_letters(self):
    with open(self.filename, 'r') as file:
        while True:
            line = file.readline()

            # ignore the Project Gutenberg's
                file headers
            if line.strip() in [header.
                value for header in
                Headers]: break

        while line:
            line = file.readline()

            # ignore the Project Gutenberg's
                file footers
            if line.strip() in [footer.value
                for footer in Footers]: break

            for words in line.split():
                # remove all stop-words and
                    punctuation marks
                for word in regex.findall('\p{
                    alpha}+', words):
                    for letter in word:
                        self.parsed_letters.
                            append(letter.
                            upper())
```

C. Exact Counter

Inside the **ExactCounter** class we can find the *compute()* method, which is responsible for counting the exact number of occurrences of each letter from the literary work.

D. Decreasing Probability Counter

The **DecreasingProbabilityCounter** class implements the Decreasing Probability Counter as a probabilistic counter, with the increment being made with probability $\frac{1}{\sqrt{2}^k}$, where k represents the number of occurrences of each letter. If the counter has value k, the algorithm increases the number of occurrences of the letter with the previously mentioned probability. Due to k and the probability being inversely proportional, as k increases, the probability of incrementing the counter will be much smaller. This method allows the counting of a large number of events using a small amount of memory. The estimated value of the counter for each letter can be calculated using the following formula:

$$\frac{\sqrt{2}^k - \sqrt{2} + 1}{\sqrt{2} - 1}$$

E. Frequent Counter

The **FrequentCount** class implements a Frequent Counter as a Data Stream Algorithm. The goal is to establish an estimate for the frequency of any stream letter. The frequent count algorithm implemented was the Misra Gries algorithm. This uses a parameter k that controls the quality of the results given. It uses a *letter: counter* dictionary, with at most *(k - 1)* counters, at any time. 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.

F. Tests

The **tests.py** contains the logic of the tests that are used to compute the results for the Decreasing Probability and Frequent counters, comparing both to the Exact Counter. The obtained results will be discussed in the next section.

IV. Results and Discussion

As mentioned in the previous subsection, some tests were developed and used to compare both algorithms. Regarding the Decreasing Probability Counter, the following measures were computed: absolute error (mean, minimum, and maximum), relative error (mean, minimum, and maximum), standard deviation, and variance. All measures were calculated as an average from one hundred trials, and the comparison was made using the estimated count from each letter from the text. Regarding the Frequent Counter, due to being a deterministic algorithm, the results are always the same. So, one run of the tests gives the results needed for comparison. For this counter, the k most frequent letters were computed. For both tests, the literary work used was *Crime and Punishment, by Fyodor Dostoevsky* in both English and Spanish languages. Inside the **/texts/** folder, *The Metamorphosis, by Franz Kafka* can also be found, in both English and German languages.

A. Decreasing Probability Counter

The figure 2 shows the results for the *Crime and Punishment, by Fyodor Dostoevsky* text, in English, which has, without the stop-words, *874218* letters.

Letter	Value	Exact Value	Expected Value	MeanAE	MinAE	MaxAE	MeanRE	MinRE	MaxRE	StandDev	Variance
E	31	103213	105208	35502.1	8662.0	213221.0	34.4%	8.4%	206.6%	4.763E+04	2.269E+09
T	30	79320	85618	28849.3	213.0	144432.0	36.4%	0.3%	182.1%	4.350E+04	1.892E+09
A	30	73042	76034	26758.7	6065.0	243392.0	36.6%	8.3%	333.2%	4.019E+04	1.615E+09
O	30	70888	74802	22325.1	8219.0	87328.0	31.5%	11.6%	123.2%	2.793E+04	7.799E+08
N	29	62120	60009	20846.8	6183.0	49755.0	33.6%	10.0%	80.1%	2.519E+04	6.347E+08
I	29	61534	62852	18160.0	5597.0	96682.0	29.5%	9.1%	157.1%	2.340E+04	5.476E+08
H	29	55491	63030	21416.6	446.0	168261.0	38.6%	0.8%	303.2%	3.489E+04	1.217E+09
S	29	52753	52811	18515.0	3184.0	105463.0	35.1%	6.0%	199.9%	2.413E+04	5.825E+08
R	28	47324	47317	16780.5	7771.0	64551.0	35.5%	16.4%	136.4%	2.194E+04	4.813E+08
D	28	38670	37264	12641.7	883.0	73205.0	32.7%	2.3%	189.3%	1.715E+04	2.942E+08
L	27	35375	37274	13669.6	4178.0	76500.0	38.6%	11.8%	216.3%	1.953E+04	3.815E+08
U	27	27270	27016	10594.3	698.0	51837.0	38.8%	2.6%	190.1%	1.397E+04	1.951E+08
M	26	22102	23737	8869.8	2326.0	33835.0	40.1%	10.5%	153.1%	1.141E+04	1.303E+08
Y	26	20955	21968	6027.1	1179.0	34982.0	28.8%	5.6%	166.9%	8.757E+03	7.668E+07
W	26	20726	19228	7030.7	950.0	18827.0	33.9%	4.6%	90.8%	8.353E+03	6.978E+07
C	26	18277	18779	5372.1	1499.0	21276.0	29.4%	8.2%	116.4%	6.872E+03	4.722E+07
G	26	17988	18427	6008.5	1788.0	37949.0	33.4%	9.9%	211.0%	8.461E+03	7.158E+07
F	25	16822	16194	5542.7	2839.0	22731.0	32.9%	16.9%	135.1%	7.114E+03	5.061E+07
P	25	13548	14072	4625.0	435.0	14420.0	34.1%	3.2%	106.4%	5.918E+03	3.503E+07
B	24	12008	11534	3902.2	1975.0	15960.0	32.5%	16.4%	132.9%	5.010E+03	2.510E+07
V	24	10995	11302	3838.4	1108.0	16973.0	34.9%	10.1%	154.4%	5.141E+03	2.643E+07
K	24	9622	10128	3338.7	265.0	10154.0	34.7%	2.8%	105.5%	4.210E+03	1.772E+07
X	18	1303	1268	410.7	68.0	1168.0	31.5%	5.2%	89.6%	5.221E+02	2.726E+05
Z	17	1086	1098	420.9	149.0	2409.0	38.8%	13.7%	221.8%	5.787E+02	3.349E+05
Q	16	777	795	285.8	96.0	1694.0	36.8%	12.4%	218.0%	3.803E+02	1.446E+05
J	16	749	725	255.2	124.0	998.0	34.1%	16.6%	133.2%	3.188E+02	1.016E+05
Ï	13	222	231	73.7	5.0	395.0	33.2%	2.3%	177.9%	1.023E+02	1.047E+04
É	6	17	17	4.6	1.0	20.0	27.0%	5.9%	117.6%	5.914E+00	3.498E+01
Ô	3	4	4	1.1	1.0	2.0	26.2%	25.0%	50.0%	1.055E+00	1.114E+00
Æ	3	4	3	1.1	1.0	2.0	28.0%	25.0%	50.0%	1.034E+00	1.068E+00
Ê	2	3	3	0.3	0.0	1.0	9.0%	0.0%	33.3%	4.440E-01	1.971E-01
Ä	2	3	3	0.3	0.0	1.0	11.7%	0.0%	33.3%	4.770E-01	2.275E-01
Â	2	2	2	0.0	0.0	0.0	0.0%	0.0%	0.0%	0.000E+00	0.000E+00
Î	1	1	1	0.0	0.0	0.0	0.0%	0.0%	0.0%	0.000E+00	0.000E+00
Ç	1	1	1	0.0	0.0	0.0	0.0%	0.0%	0.0%	0.000E+00	0.000E+00
Ö	1	1	1	0.0	0.0	0.0	0.0%	0.0%	0.0%	0.000E+00	0.000E+00
Ü	1	1	1	0.0	0.0	0.0	0.0%	0.0%	0.0%	0.000E+00	0.000E+00
È	1	1	1	0.0	0.0	0.0	0.0%	0.0%	0.0%	0.000E+00	0.000E+00

Execution time (for 100 trials): 114.32s

Fig. 2: Results obtained using the Decreasing Probability Counter, in English

From these results, there are a few conclusions we can take. For each letter, the expected (estimated) value has a high deviation from the real value. This can be confirmed by the mean absolute and relative error columns. Taking, for example, the letter E, the mean relative error is *34.4%*, meaning that the accuracy of the results is less than *70%*, which I believe is not a bad result, considering that this algorithm only increments each letter's counter with probability $\frac{1}{\sqrt{2}^k}$. There is also a relatively high standard deviation, and thus, variance. On the other hand, memory-wise this method is much more efficient than the exact count. It is worth mentioning that there are some letters whose mean relative error is *0.0%*, translating into a *100%* accuracy.

Letter	Value	Exact Value	Expected Value	MeanAE	MinAE	MaxAE	MeanRE	MinRE	MaxRE	StandDev	Variance
E	31	107588	114717	37403.9	4287.0	208846.0	34.8%	4.0%	194.1%	5.297E+04	2.806E+09
A	30	98850	93768	33212.7	13025.0	124902.0	33.6%	13.2%	126.4%	3.975E+04	1.580E+09
O	30	72334	73831	24623.2	6773.0	244100.0	34.0%	9.4%	337.5%	4.016E+04	1.613E+09
S	29	59077	60648	20582.2	3140.0	99139.0	34.8%	5.3%	167.8%	2.815E+04	7.923E+08
N	29	55842	55472	18042.8	95.0	102374.0	32.3%	0.2%	183.3%	2.480E+04	6.150E+08
R	28	52476	49974	17250.3	3461.0	59399.0	32.9%	6.6%	113.2%	2.143E+04	4.594E+08
I	28	46367	44838	13627.5	6814.0	32740.0	29.4%	14.7%	70.6%	1.594E+04	2.541E+08
D	28	40815	41240	12547.2	1262.0	71060.0	30.7%	3.1%	174.1%	1.697E+04	2.881E+08
L	28	38938	39630	13039.5	615.0	119278.0	33.5%	1.6%	306.3%	2.055E+04	4.222E+08
T	28	36760	39874	14647.0	2793.0	75115.0	39.8%	7.6%	204.3%	1.902E+04	3.617E+08
U	27	36230	34102	11954.7	3323.0	42877.0	33.0%	9.2%	118.3%	1.515E+04	2.294E+08
C	27	30774	34300	12279.1	2806.0	48333.0	39.9%	9.1%	157.1%	1.560E+04	2.432E+08
M	26	23963	24023	7963.2	4005.0	31974.0	33.2%	16.7%	133.4%	9.551E+03	9.121E+07
P	26	21500	21029	7248.9	1724.0	34437.0	33.7%	8.0%	160.2%	9.287E+03	8.625E+07
B	25	14030	13536	4977.5	47.0	25523.0	35.5%	0.3%	181.9%	6.659E+03	4.434E+07
H	24	11025	10444	3963.9	1138.0	16943.0	36.0%	10.3%	153.7%	4.836E+03	2.339E+07
V	24	10332	10587	3319.1	445.0	17636.0	32.1%	4.3%	170.7%	4.740E+03	2.247E+07
Q	24	9651	9489	3252.8	236.0	29902.0	33.7%	2.4%	309.8%	5.135E+03	2.636E+07
G	23	7601	7616	2699.2	610.0	12175.0	35.5%	8.0%	160.2%	3.510E+03	1.232E+07
Y	23	7226	7477	2450.0	235.0	12550.0	33.9%	3.3%	173.7%	3.362E+03	1.130E+07
Ó	23	7086	6988	1963.8	95.0	6897.0	27.7%	1.3%	97.3%	2.623E+03	6.881E+06
F	23	6954	6458	2020.0	37.0	12822.0	29.0%	0.5%	184.4%	2.768E+03	7.659E+06
Í	23	6858	7463	2574.9	133.0	12918.0	37.5%	1.9%	188.4%	3.455E+03	1.194E+07
J	21	4578	4302	1416.8	365.0	9405.0	30.9%	8.0%	205.4%	1.922E+03	3.696E+06
É	21	3961	3993	1217.3	466.0	5926.0	30.7%	11.8%	149.6%	1.634E+03	2.670E+06
Z	21	3492	3465	1116.4	3.0	6395.0	32.0%	0.1%	183.1%	1.587E+03	2.519E+06
Á	21	3487	3479	1016.3	8.0	6400.0	29.1%	0.2%	183.5%	1.458E+03	2.127E+06
K	20	2875	2836	972.1	404.0	4116.0	33.8%	14.1%	143.2%	1.236E+03	1.529E+06
Ñ	18	1445	1444	485.8	210.0	2050.0	33.6%	14.5%	141.9%	5.861E+02	3.435E+05
X	18	1415	1457	487.5	180.0	2080.0	34.5%	12.7%	147.0%	6.278E+02	3.942E+05
Ú	17	892	922	348.7	19.0	1579.0	39.1%	2.1%	177.0%	4.662E+02	2.173E+05
Ü	7	32	32	11.2	5.0	44.0	34.9%	15.6%	137.5%	1.390E+01	1.931E+02
W	6	17	16	5.7	1.0	20.0	33.3%	5.9%	117.6%	6.627E+00	4.392E+01
Û	2	2	2	0.0	0.0	0.0	0.0%	0.0%	0.0%	0.000E+00	0.000E+00
Ï	1	1	1	0.0	0.0	0.0	0.0%	0.0%	0.0%	0.000E+00	0.000E+00
Ö	1	1	1	0.0	0.0	0.0	0.0%	0.0%	0.0%	0.000E+00	0.000E+00
Ä	1	1	1	0.0	0.0	0.0	0.0%	0.0%	0.0%	0.000E+00	0.000E+00

Execution time (for 100 trials): 109.13s

Fig. 3: Results obtained using the Decreasing Probability Counter, in Spanish

Let's take a look at the results for the same literary work, but this time in Spanish, in figure 3. Comparing both languages, and even though they are different, we can see that, for the top 10 most frequent letters, the letters E, T, A, O, N, I, S, R, and D are common in both, despite not all of them having the same relative order.

Letter	Value
E	31
T	30
A	30
O	30
N	29
I	29
H	29
S	29
R	28
D	28

(a) English

Letter	Value
E	31
A	30
O	30
S	29
N	29
R	28
I	28
D	28
L	28
T	28

(b) Spanish

Fig. 4: Comparison Between Top 10 Most Frequent Letters in English and Spanish

In 4, we can see a direct comparison between the top 10 most frequent letters in both English and Spanish languages.

B. Frequent Counter

Both figures 5 and 6 show the comparisons made for the frequent counter, on both English and Spanish languages, with k as 5, 10, and 20. One thing worth mentioning is that, for higher values of k, the algorithm gives us more accurate results, as this literally work is large.

Letter	Value	Exact Value
D	1	38670
C	0	18277
R	0	47324
I	0	61534
M	0	22102
E	0	103213
A	0	73042
N	0	62120
P	0	13548
U	0	27270
S	0	52753
H	0	55491
T	0	79320
B	0	12008
Y	0	20955
F	0	16822
O	0	70888
V	0	10995
K	0	9622
L	0	35375
G	0	17988
W	0	20726
X	0	1303
J	0	749
Q	0	777
Z	0	1086
Ï	0	222
É	0	17
Ô	0	4
Î	0	1
Ê	0	3
Æ	0	4
Ä	0	3
Ç	0	1
Ö	0	1
À	0	2
Ü	0	1
È	0	1

Execution time: 4.88s

(a) k = *5*

Letter	Value	Exact Value
E	35	103213
R	1	47324
D	1	38670
S	1	52753
T	1	79320
C	0	18277
I	0	61534
M	0	22102
A	0	73042
N	0	62120
P	0	13548
U	0	27270
H	0	55491
B	0	12008
Y	0	20955
F	0	16822
O	0	70888
V	0	10995
K	0	9622
L	0	35375
G	0	17988
W	0	20726
X	0	1303
J	0	749
Q	0	777
Z	0	1086
Ï	0	222
É	0	17
Ô	0	4
Î	0	1
Ê	0	3
Æ	0	4
Ä	0	3
Ç	0	1
Ö	0	1
À	0	2
Ü	0	1
È	0	1

Execution time: 4.8s

(b) k = *10*

Letter	Value	Exact Value
E	78975	103213
T	55082	79320
A	48804	73042
O	46650	70888
N	37882	62120
I	37297	61534
H	31255	55491
S	28515	52753
R	23086	47324
D	14432	38670
L	11143	35375
U	3061	27270
W	33	20726
F	9	16822
G	7	17988
M	6	22102
Y	5	20955
P	1	13548
B	1	12008
C	0	18277
V	0	10995
K	0	9622
X	0	1303
J	0	749
Q	0	777
Z	0	1086
Ï	0	222
É	0	17
Ô	0	4
Î	0	1
Ê	0	3
Æ	0	4
Ä	0	3
Ç	0	1
Ö	0	1
À	0	2
Ü	0	1
È	0	1

Execution time: 4.39s

(c) k = *20*

Fig. 5: Comparison Between Frequent Counter for k = *5, 10, and 20, in English*

Letter	Value	Exact Value
Y	1	7226
P	0	21500
R	0	52476
O	0	72334
D	0	40815
U	0	36230
C	0	30774
E	0	107588
B	0	14030
A	0	98850
L	0	38938
S	0	59077
Ó	0	7086
N	0	55842
T	0	36760
H	0	11025
I	0	46367
V	0	10332
F	0	6954
G	0	7601
M	0	23963
W	0	17
K	0	2875
Í	0	6858
Á	0	3487
Ú	0	892
Ñ	0	1445
Z	0	3492
J	0	4578
Q	0	9651
X	0	1415
É	0	3961
Ü	0	32
Ï	0	1
Ö	0	1
Ä	0	1
Û	0	2

Execution time: 4.3s

(a) k = *5*

Letter	Value	Exact Value
A	17	98850
E	9	107588
O	3	72334
Y	2	7226
S	1	59077
K	1	2875
P	0	21500
R	0	52476
D	0	40815
U	0	36230
C	0	30774
B	0	14030
L	0	38938
Ó	0	7086
N	0	55842
T	0	36760
H	0	11025
I	0	46367
V	0	10332
F	0	6954
G	0	7601
M	0	23963
W	0	17
Í	0	6858
Á	0	3487
Ú	0	892
Ñ	0	1445
Z	0	3492
J	0	4578
Q	0	9651
X	0	1415
É	0	3961
Ü	0	32
Ï	0	1
Ö	0	1
Ä	0	1
Û	0	2

Execution time: 4.26s

(b) k = *10*

Letter	Value	Exact Value
E	83599	107588
A	74861	98850
O	48345	72334
S	35088	59077
N	31853	55842
R	28487	52476
I	22378	46367
D	16826	40815
L	14950	38938
T	12771	36760
U	12246	36230
C	6786	30774
M	177	23963
Y	4	7226
P	2	21500
G	2	7601
B	1	14030
Ó	0	7086
H	0	11025
V	0	10332
F	0	6954
W	0	17
K	0	2875
Í	0	6858
Á	0	3487
Ú	0	892
Ñ	0	1445
Z	0	3492
J	0	4578
Q	0	9651
X	0	1415
É	0	3961
Ü	0	32
Ï	0	1
Ö	0	1
Ä	0	1
Û	0	2

Execution time: 3.91s

(c) k = *20*

Fig. 6: Comparison Between Frequent Counter for k = *5, 10, and 20, in Spanish*

In both languages, taking for example k = *20*, and as seen on the decreasing probability counter, some of the most frequent letters are the same, although in a different relative order. Compared with the exact count, if a letter has a frequency larger or equal to $\frac{m}{k}$, the order in which it appears on the table is the same as in the exact count.

V. Conclusion

This assignment allowed for a better understanding of the difference between probabilistic counter and data stream algorithms. Both the decreasing probability counter and the frequent counter can be good solutions to reducing memory usage, even though they have different applications, as the use cases differ between the two. Regarding future work, it would be relevant to compare the algorithms explored with different ones, to have an even better term of comparison and knowledge about this type of counting methods.

References

[1] Project Gutenberg. (1971-2021). Welcome to Project Gutenberg.https://www.gutenberg.org/

[2] Python Software Foundation. (Dec 22, 2022). re — Regular expression operations. https://docs.python.org/3/library/re.html

[3] Python Software Foundation. (Dec 22, 2022). abc — Abstract Base Classes. https://docs.python.org/3/library/abc.html

[4] Wikipedia contributors. (Dec 24, 2022). Approximate counting algorithm. Wikipedia. https://en.wikipedia.org/wiki/Approximate_counting_algorithm

[5] Wikipedia contributors. (Dec 24, 2022). Approximate counting algorithm. Wikipedia. https://en.wikipedia.org/wiki/Approximate_counting_algorithm

[6] Wikipedia contributors. (Jan 2, 2023). Streaming algorithm. Wikipedia. https://en.wikipedia.org/wiki/Streaming_algorithm#:~:text=In\%20computer\%20science\%2C\%20streaming\%20algorithms,passes\%20(typically\%20just\%20one)