COSC 320 – 001

Analysis of Algorithms

2020 Winter Term 2

Project Topic Number: #4

String Matching Problem - Milestone 4

Group Lead:

Matthew Borle - 42615435

Group Members:

Barry Feng - 12981156

Guy Kaminsky - 65234775

Ross Morrison - 36963411

Abstract.

We first began this final milestone by completing the Rabin Karp (RBK) algorithm. As with the previous milestone, we used the already made algorithm from GeeksForGeeks but had to create our individualized main function to allow the RBK implementation to work with our dataset. Once the RBK algorithm worked with our main method and detected the number of identical words from one document to the master document we created a python file called final implementation. In this file, we partitioned our string matching code into five main chunks for ease in reading and debugging the code. With this partitioned python file, we took our RBK implementation and integrated it with our KMP implementation from the previous milestone. While we had both algorithms working, we also had to make changes to the main function. These changes included adding a conditional check to combine the results of both algorithms. This check would determine if a document plagiarises. Given this completion of the implementation, we followed this up by gathering data on the running time of RBK individually and of our final implementation. We then took the data and plotted it. Finally, with everything complete, we created a video explaining our program and the data collected.

Implementation.

For this milestone, we followed a similar design philosophy to the third milestone. Ross and Guy began by implementing the Rabin-karp algorithm. The same dataset was used to load and test the algorithm. The input file was split into multiple parts but this time it was split by word instead of sentences. To complete our combined algorithm the scores given by the KMP and Rabin-karp implementation were compared to predetermined values to check if the given input document was plagiarized. When a plagiarized document is found, the index of the document in the dataset folder is printed.

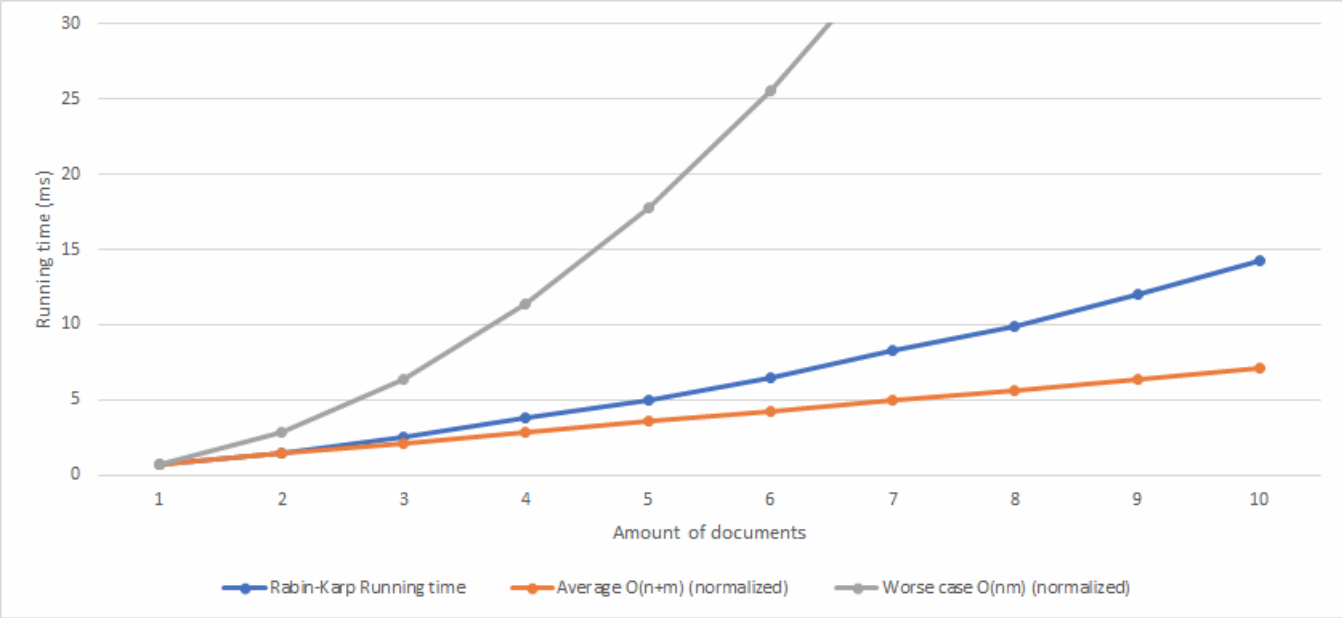
Reference: https://www.geeksforgeeks.org/rabin-karp-algorithm-for-pattern-searching/

Results.

Rabin-Karp:

Rabin-Karp Running time (Round 1)	0.728947	0.715929	0.815086	0.963212	0.969306	1.091273	1.16795	1.219662	1.263106	1.423157
Rabin-Karp Running time (Round 2)	0.681732	0.725373	0.884117	0.929683	0.996733	1.115678	1.191187	1.266536	1.345001	1.46061
Rabin-Karp Running time (Round 3)	0.717487	0.805737	0.832003	0.976434	1.011793	1.046293	1.188771	1.235035	1.38663	1.399952

Documents	1	2	3	4	5	6	7	8	9	10
Rabin-Karp Running time	0.709389	1.498026	2.531207	3.825772	4.963052	6.506489	8.278452	9.923286	11.98421	14.27906
Average O(n+m)	2	4	6	8	10	12	14	16	18	20
Average O(n+m) (normalized)	0.709389	1.418777	2.128166	2.837554	3.546943	4.256332	4.96572	5.675109	6.384497	7.093886
Worse case O(nm)	1	4	9	16	25	36	49	64	81	100
Worse case O(nm) (normalized)	0.709389	2.837554	6.384497	11.35022	17.73471	25.53799	34.76004	45.40087	57.46048	70.93886



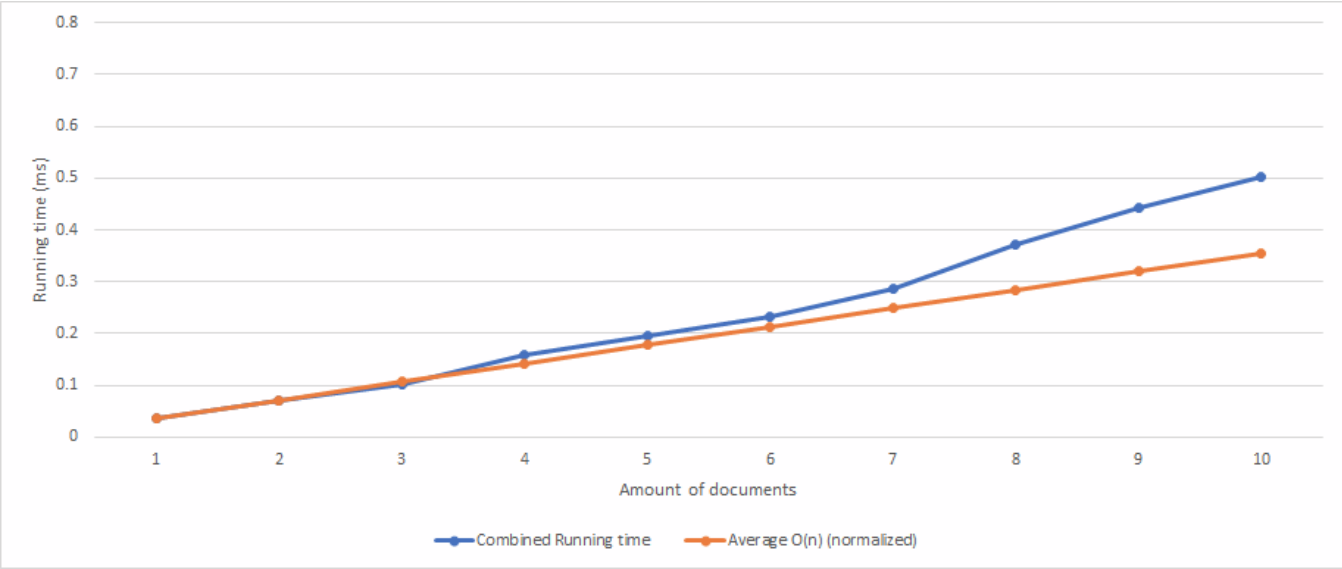
Once again we implemented a timer in our algorithm similar to the last milestone and had it run through one to ten plagiarized documents. However this time we ran each number of plagiarized documents through our implemented algorithm 3 times and got the average running time in milliseconds. We did this so the time would be more consistent and any deviations in a fast or slow round would not affect our result. From a mathematical analysis, our Rabin-Karp algorithm matched our research and had an average of O(n+m) running time and worst-case O(nm) running time. O(n+m) running time and O(nm) were normalized to the first point and then allowed to grow as usual since a python timer records in milliseconds.

The running time of our algorithm is what we expected as it ran better than the worst case of O(nm) and ran close to the average running time of O(n+m). Since our Rabin-Karp uses hashes for data structure we did expect it to be close to the average running time. We do not believe that the constant values of our algorithm nor did our choice of data structure affect the result

Final algorithm:

Combined algorithm Running time (Round 1)	0.032522	0.032147	0.034142	0.034197	0.035533	0.03938	0.039504	0.042237	0.055989	0.050251
Combined algorithm Running time (Round 2)	0.031721	0.033041	0.034488	0.034573	0.036058	0.037623	0.039711	0.054743	0.046873	0.050783
Combined algorithm Running time (Round 3)	0.042307	0.039437	0.033516	0.050327	0.045982	0.038771	0.043365	0.042462	0.044722	0.049211

Documents	1	2	3	4	5	6	7	8	9	10
Combined Running time	0.035517	0.06975	0.102146	0.158796	0.195956	0.231548	0.286023	0.371846	0.442753	0.500817
Average O(n)	1	2	3	4	5	6	7	8	9	10
Average O(n) (normalized)	0.035517	0.071033	0.10655	0.142067	0.177584	0.2131	0.248617	0.284134	0.319651	0.355167



Our final algorithm was an implementation of Rabin-Karp combined with KMP. As usual, we had it run through one to ten plagiarized documents to check for plagiarism. Rabin-Karp runs at an average of O(n+m) running time with worst-case O(nm), and KMP runs worse case O(n). Our combined algorithm runs at an average O(n).

This is what we expected as the average between n+m and n is approximately 1.5n which equates to a big O-notation of O(n). We do not believe the implementation of our algorithm affected our constant values. As we can see in the plot, our final algorithm does indeed run at O(n) as it closely follows the line, with some slight deviations due to external factors.

In terms of Big-O notation, the final implementation runs better than KMP and worse than Rabin Karp when comparing our final application to the individualized KMP and Rabin Karp algorithms. However, our complete implementation checks both words and sentences for plagiarism.

Unexpected Cases/Difficulties.

The first challenge faced in this milestone arose during the implementation of the RBK algorithm. The return value of this algorithm provided the number of times words have been used in one document compared to the master document. This result alone does not help us to determine if a document plagiarises. To combat this issue, we took the return value and divided it by the number of words the master document has. This new result gives us a percentage of how many times a word matches. We then added a threshold to determine when to flag for suspicion. If this percentage passes, we will increment the counter to act as a weight, which in the end will be the determining factor if a document plagiarizes.

An issue that came into account while creating this program is the lack of feedback between milestone three and this last milestone. Specifically, we were not confident if our KMP implementation was well received and how the teacher assistants (TA's) will test our string matching code. In terms of

adjusting to the former issue, we just looked over our KMP code before starting the final implementation to make sure we are happy with how it works. For the latter, we modified our code to allow the TA's to change the file path with ease when testing our program. That way they only need to change a couple of lines rather than every instance where we used our dataset.

Task Separation and Responsibilities.

In terms of task separation, we continued to implement peer programming techniques. Specifically, Guy Kaminsky wrote the abstract. Then, Guy Kaminsky and Ross Morrison wrote the implementation for RBK and the combined implementation of RBK and KMP. Next, Barry Feng and Matt Borle were in charge of the results section. They used the data collected from all of the implementations and created the data plots using excel. Furthermore, Matt Borle and Guy Kaminsky filmed the video showing the choices we made during the project, how our algorithms work, their implementation, and how they run. Lastly, everyone contributed to the unexpected cases and difficulties section.