Text Comparison Detection Application

An Implementation of a Revised Levenshtein Distance Algorithm and KMP Algorithm

A Research Presented to the

Faculty of the College of Computer and Information Sciences,

Polytechnic University of the Philippines

In Partial Fulfillment for the Course

COSC 30033 - Design and Analysis of Algorithms

Allan Christian D. Tejada

Allen Carl Delas Alas

John Jeffrei Tumbaga

Leinard Rapiza

Mathew James Muyco

Nathanael Stephen Memis

BSCS 2-1

Proponents

Table of Contents

Introduction

Today, it is virtually impossible to avoid writing using technology. Whether it's emails, important documents, essays, or poetry, we are dependent on our devices if we hope to communicate, submit assignments, or work with it. As long as you have your phone on you, you can jot down an idea. There's no need to find a flat surface or a working pen. No one will look at you strangely for typing into your phone, so even if you are on a crowded bus or at a party, you can subtly get some writing in.

We can also write together using online documents with no distance hindrance. Special in our time were at a distance learning curriculum because of the pandemic. So, writing collaboratively in an online documents app and website is very helpful in writing a group assignment, research, and presentation etc.

But what if the user has a long document such as a thesis paper or a research paper and someone edited it without permission, or the user just wants to track the changes on the document. The researcher finds it very hard to search the little adjustment in a very long document.

So, using String Matching Algorithm and Levenshtein Distance Algorithm the researcher creates a text comparison website where the user can easily track every change on his document. And if the user is a programmer, he can also use this to track all the changes in his codes. Using a string matching algorithm, specifically the Knuth–Morris–Pratt (KMP) algorithm where it

searches every text in the document and finds a mismatch and uses the Levenshtein algorithm to show the changes in the original and the edited documents.

Review of Related Literature

The following related studies have been used as a guide in this study, Furthermore, referencing similar studies on Levenshtein Distance Algorithm and Knuth–Morris–Pratt Algorithm proved paramount in the development of the web application.

The application of string similarity is very extensive, and the algorithm based on Levenshtein Distance is particularly classic, but it is still insufficient in the aspect of universal applicability and accuracy of results. (Zhang, Hu, Bian, 2017)

Furthermore, (Haldar, Mukhopadhyay, 2011) also declared that Levenshtein distance is a simple metric which can be an effective string approximation tool. After observing the effectiveness of this method, an improvement has been made to this method by grouping some similar looking alphabets and reducing the weighted difference among members of the same group. The results showed marked improvement over the traditional Levenshtein distance technique.

As plagiarism takes up a big problem in society, due to the fact that plagiarism is essentially theft and fraud committed simultaneously. It is considered theft because the writer takes ideas from a source without giving proper credit to the author.

(Su et al., 2009) studied the use of Levenshtein Distance and Smith-Waterman Algorithm in Plagiarism Detection, based on the results, they indicated the practicality of such improvement using Levenshtein distance and Smith-Waterman algorithm and to illustrate the efficiency gains. In the future, it would be interesting to explore appropriate heuristics in the area of text comparison.

Lastly, based on the study of (Aung, 2019) that focused on the comparison of Levenshtein Distance Algorithm and Needleman-Wunsch Distance Algorithm for String Matching, as the score value of both algorithms depend on the input size, the results suggest that Levenshtein Distance Algorithm has better accuracy than the Needleman-Wunsch Distance Algorithm for the comparison of average score, Levenshtein Distance Algorithm has better accuracy than the Needleman-Wunsch Distance Algorithm for both cases. And as a time, complexity of both algorithms, the Needleman-Wunsch Distance Algorithm has more complexity than the Levenshtein Distance Algorithm. But it could be seen that some data for the Levenshtein Distance Algorithm also has a long execution time measured in seconds. But for the comparison of average time, the Needleman-Wunsch Distance Algorithm also has a longer execution time than the Levenshtein Distance Algorithm.

Objectives of The Project

General Objective:

The main objective of this study is to compare text that involves revising and suggestions about the content of a text.

Specific Objective:

- Develop an application that improves the accuracy of language.
- Develop an application that improves the flow of the text.
- Develop an application that improves the overall readability of the text.
- Develop an application that checks the comparison of two text data.

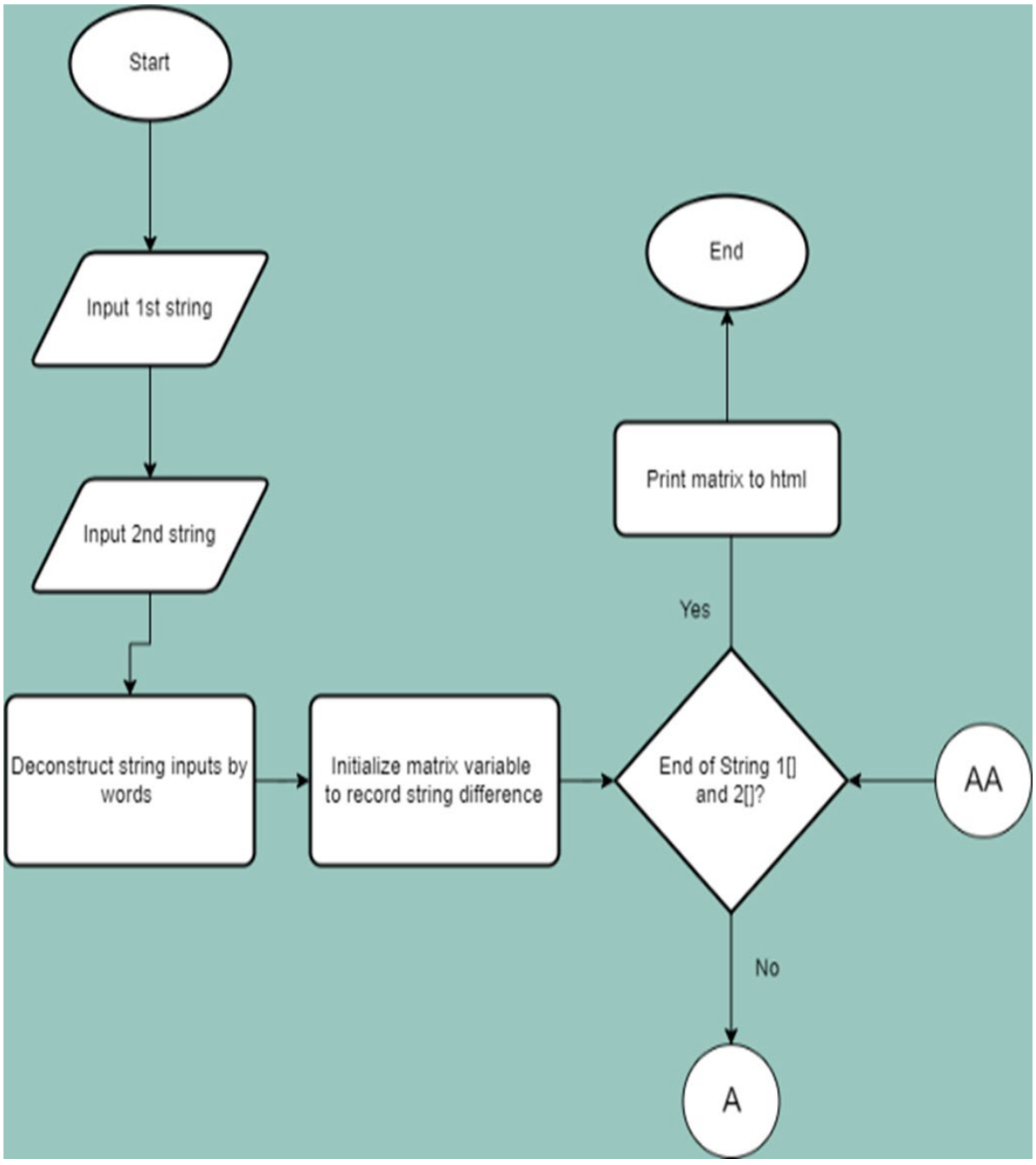
Scope and Limitations

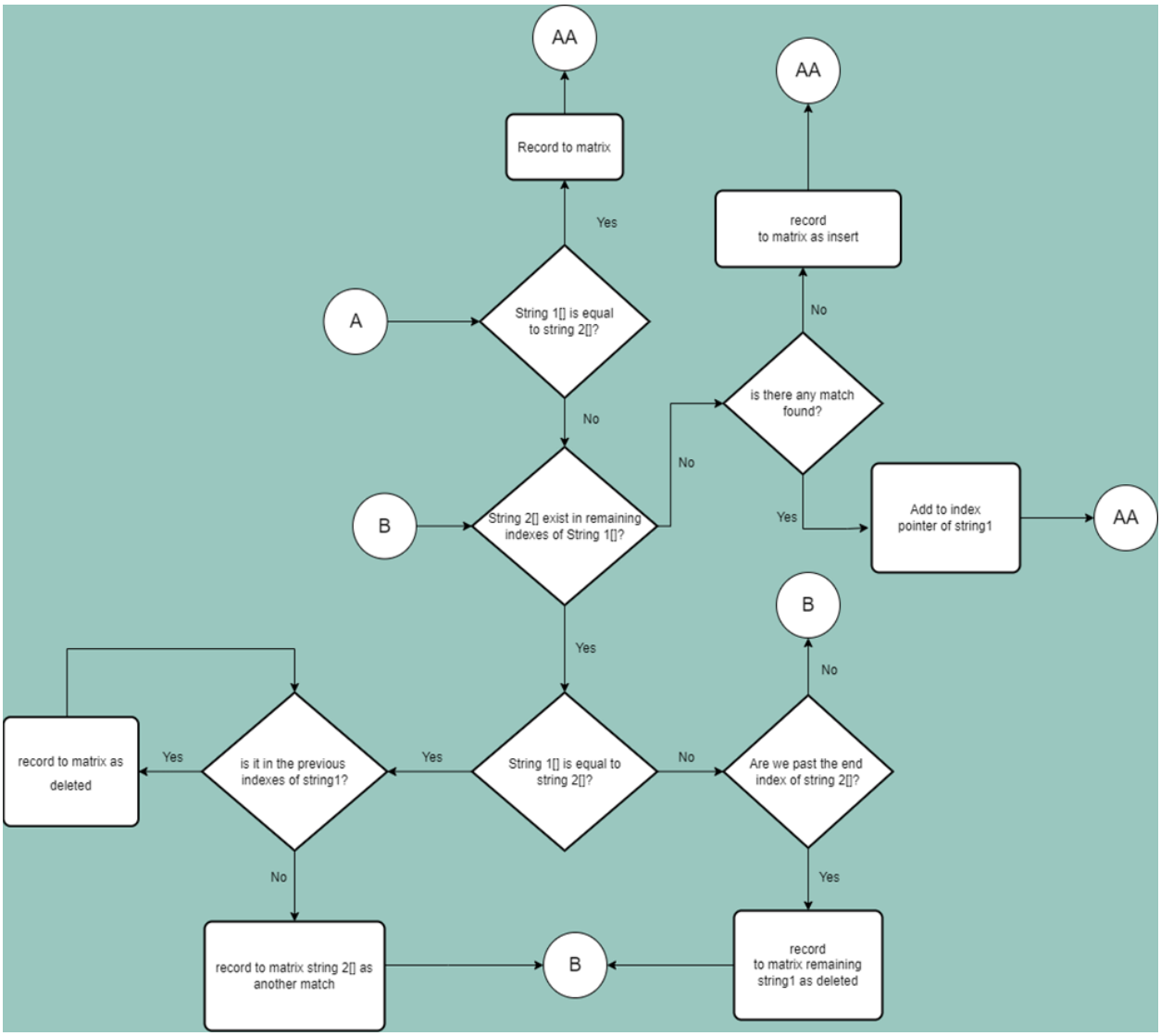
This algorithm will mainly focus on tracking and comparing the revised texts from an originally online written work. It could be that one individual is the rightful owner of a document, code, or any online writing.

The algorithm prevents other people who have access to your document/code from plagiarizing it. For the writer of a text can now track and scan the edited words, phrases, or even sentences on his original work.

As per the innovative mechanics of this algorithm, the process is only limited to online documents and has yet to work on physical documents because the system needs to have access first to the original file before it can track any other changes.

System Design and Methodology





The figure above shows the process of the program from inserting strings to showing its differences using Knuth-Morris-Pratt String Matching Algorithm and the concept of Levenshtein Algorithm. When the string is inserted, the program here will deconstruct the strings into substrings and checks it word by word to find the matches and differences between the two strings, then once the program is executed successfully, it will display the result.

If n = 0 | O(1)

If n > 0 |

Best Case: O(n)

Worst Case: O(mn)

Knuth-Morris-Pratt String Matching Algorithm is used to compare characters, but in our case, we use it to compare words instead of single characters to make it more understandable for the other users. According to the scaler.com, this algorithm was the first-ever string-matching algorithm that ran in linear time; it means when it scans a string, it does not back-track, meaning it executes in much less time compared to a basic algorithm. To further explain this, we provide an example.

We have here two strings, string 1 and string 2

String 1 = “I am a string”

String 2 = “string”

The program here will deconstruct these strings into substrings

String 1= [“I”, “am”, “a”, “string”]

String 2 = [“string”]

Then the program will compare the indexes of string 1 and string 2

```
if(string1[0] == string2[0]) {
        it will be recorded to the matrix
}
else if (string1[1] == string2[0]) {
        it will be recorded to the matrix
```

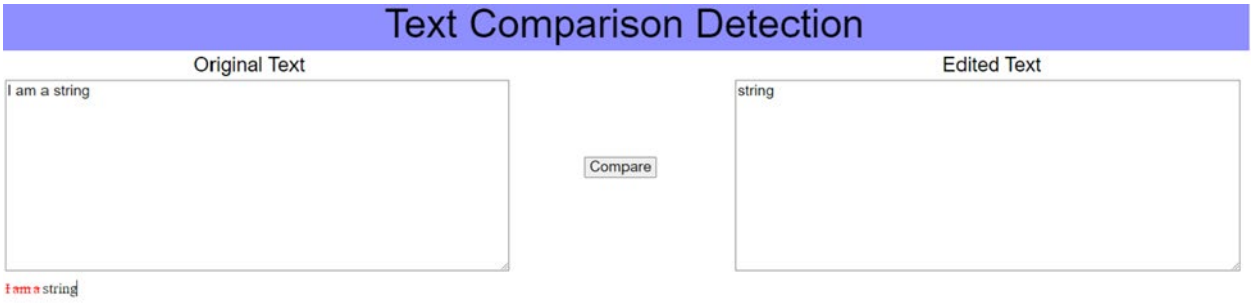
```
}
else if (string1[2] == string2[0]) {
        it will be recorded to the matrix
}
else if (string1[3] == string2[0]) {
        it will be recorded to the matrix
}
```

The matrix here is supposed to hold the recorded substring values.

Matrix string 1 = [“I”, “am”, “a”, “string”]

Matrix string 2 = [“null”, “null”, “null”, “string”]

When there is no match in compared indexes, it will store “null”, this will help identify if the text is deleted or inserted. In the case of matrix above, the null value is stored in the 2nd matrix holder; therefore, it will be identified as deleted. After identifying the difference between the two strings, the program will now display the result below. As for the example above, the result will be: “I am a string”.



As for the user interface of our program, we didn't put too much details on it for the user to easily navigate the program. We limit the functionalities of the program to just insert, compare, and delete the text the user inputs to the text box area. The researcher focused on the effectiveness of the program to satisfy whoever will use the program.

Schedule of Activities

The figure shows the planned time frame for the whole project, from Conceptualization until the day of the Final Presentation.

	May 27, 2022	Jun-03	Jun-10	Jun-17	Jun-24	Jul-01	Jul-08	Jul-15	Jul-22	Jul-29
		week 1	week 2	week 3	week 4	week 5	week 6	week 7	week 8	week 9
1	**Conceptualization**									
1.1	Brainstorming									
1.2	Problem Formulation									
2	**Project Planning**									
2.1	Distribution of Work									
2.1	Communication Platform									
2.1	Program Structure									
3	**Program Creation**									
3.1	Program Structure Foundation									
3.2	Main Algorithm									
3.2	Website Frontend									
3.2	Initial Testing									
3.2	Initial Debug									
3.2	Final Testings									
3.2	Final Debug									
4	**Documentation**									
5	**Final Presentation**									

Conclusion

The Levenshtein Distance Algorithm and KMP Algorithm used is effective in checking comparison between two sentences while holding a best-case time complexity of O(n) and worst-case time complexity of O(mn), however, the implementation can still be developed to handle checking of tokens letter by letter instead of word by word. Zhang, Hu, Bian, (2017) argued in their study that the application of string similarity is very extensive, and the algorithm based on Levenshtein Distance is particularly classic, but it is still insufficient in the aspect of universal applicability and accuracy of results. In conclusion, it aligns with this argument due to the fact that there is a much more effective use of Levenshtein Distance Algorithm and KMP algorithm in an application that compares texts, lastly, the researchers acknowledge the effectivity of Levenshtein Distance Algorithm and KMP Algorithm effective in checking comparisons between texts

References

https://thedrawingboardcanada.com/2016/12/14/the-benefits-of-writing-using-technology/

https://www.geeksforgeeks.org/kmp-algorithm-for-pattern-searching/

https://stackoverflow.com/questions/15303631/what-are-some-algorithms-for-comparing-how-similar-two-strings-are#:~:text=S%C3%B8rensen%E2%80%93Dice%20Coefficient%20%3A%20A%20similarity%20algorithm%20that%20computes,it%20is%20similar%20to%20the%20Levenshtein%20distance%20algorithm.

https://www.scaler.com/topics/data-structures/kmp-algorithm/

https://www.cuelogic.com/blog/the-levenshtein-algorithm

https://www.youtube.com/watch?v=4SP_AY7GGxw

file:///C:/Users/Finn/Downloads/KhinMoeMyintAung.pdf

https://ieeexplore.ieee.org/abstract/document/8054419/metrics#metrics

https://ieeexplore.ieee.org/abstract/document/4160958

https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-019-2819-0#Sec36

https://arxiv.org/abs/1101.1232

https://ieeexplore.ieee.org/abstract/document/4603758

https://tinyurl.com/3an6p6wp