Plagiarism Checker

20k-0122 Abdeali
20k-0386 Syed Sufyan Imran

January 12, 2022

Contents

0.1 Introduction

The idea for this project was Plagiarism checker, which compares two files given by the User, and looks for similar phrases present in both of these files. User is then told of the similar phrases and shows exactly how much percentage the work is plagiarised from the Source (original) file. The Algorithm being used is called ***KMP*** algorithm also known as **Knuth–Morris–Pratt algorithm**. Its a linear time algorithm for string matching problem. It avoids comparisons with the element in the *Source* that have previously been involved with the comparison with some element(word) of our *Target* string to be matched. Backtracking of our *Source* string never occurs.

0.2 Methodology

This program will give an overview of how KMP algorithm can be using into finding similar phrases in two different strings, while using string primitive functions and manipulating strings in a way to derive our desired results.

0.3 Implementation

Out Program starts of by asking user to input the both file names. The program makes sure the file extension is *.txt* and makes sure the file **exist** is the directory. Once that's all cleared, our string from both files are stored in *Source* and *Target* array according to which the user deemed was the source and which was the target.
Next comes our ***KMP*** logic which finds the matched phrases. So how this logic is implemented is. *Firstly*, tokenization of each word one at a time from *Source* and *Target* string. Once we receive the Tokenized word, These words are compared to make sure if they are equal or not (all uppercase characters are converted into lowercase to avoid any conflicts). If the are equal, the next set of words are Tokenized and compared and it continues until we reach the end of either string. Mostly, its the *Source* string kept in mind to see if it ended or not.
If there is a **Mismatch** between our Tokenized words, then we traverse the *Target* string until the word that was **initially mismatched** is found(Word in Source) in the *Target* string. If that word is found then we start traversing from the found spot in the *Target*, and start matching it with the *Source* String until the next mismatch is met.
Then the Logic is implemented, that if there are 3 consecutive words that were matched and so on, then display user the Matched phrase (Plagiarized phrase).
The *'Pointer'* pointing to the *Target* string words, is brought back to the place where initially the mismatched occur-ed and from there it starts comparing the next words. The pointer of *Target* string and *Source* string is both incremented to point to the next word.
EAX, EBX registers acts as indexing of **words** is both *Target* and *Source* respectively. You could say they act as our "Pointers". Bare in mind that the **Source String is never backtracked**.

0.3.1 Overview of the program

Program Takes input of the Filename.
The strings in both the files are stored in Source and Target variables.
KMP algorithm is implemented to find the matched phrase(s).
User is displayed the Matched phrases and the percentage.

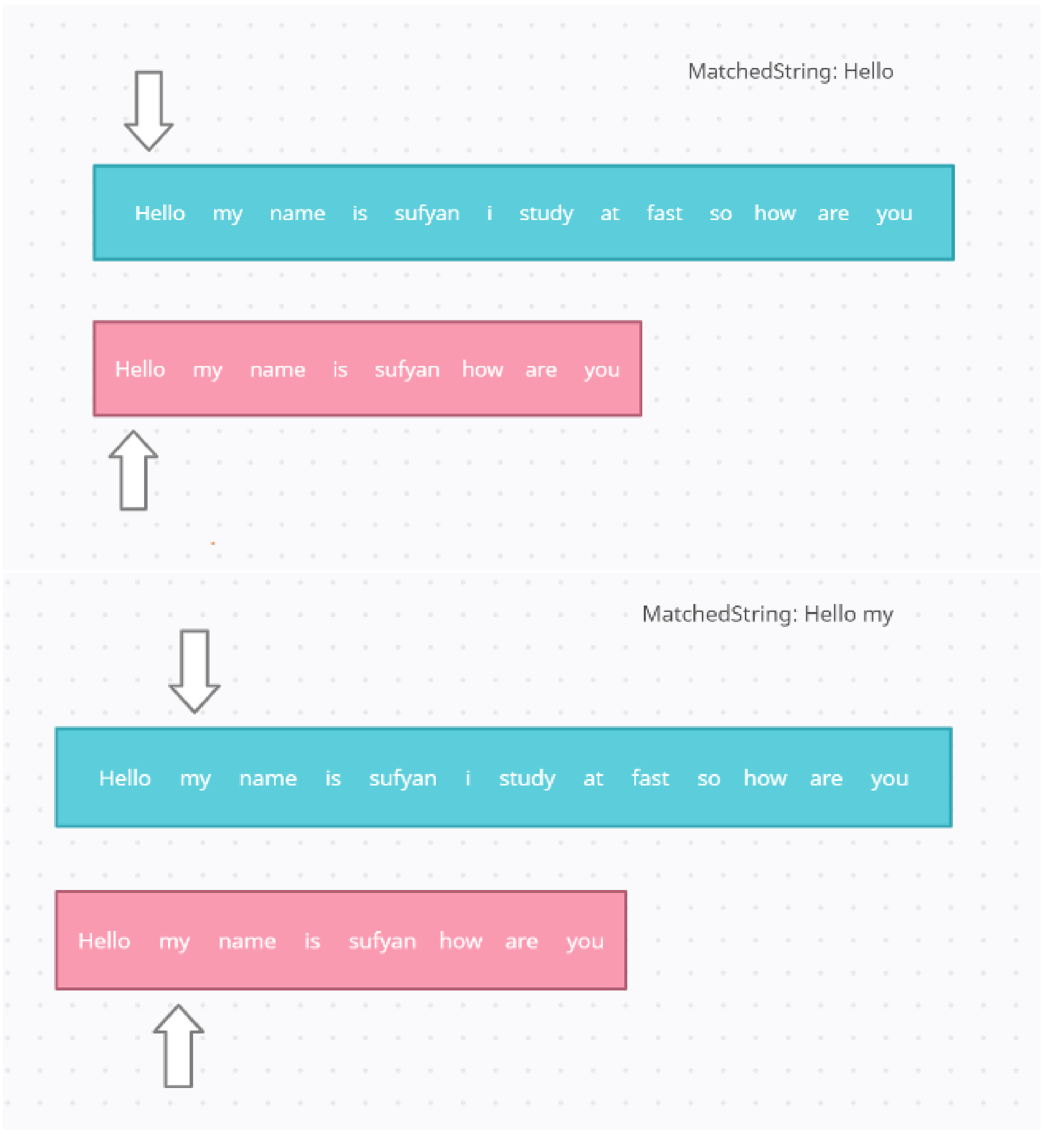
0.4 Results and Conclusions

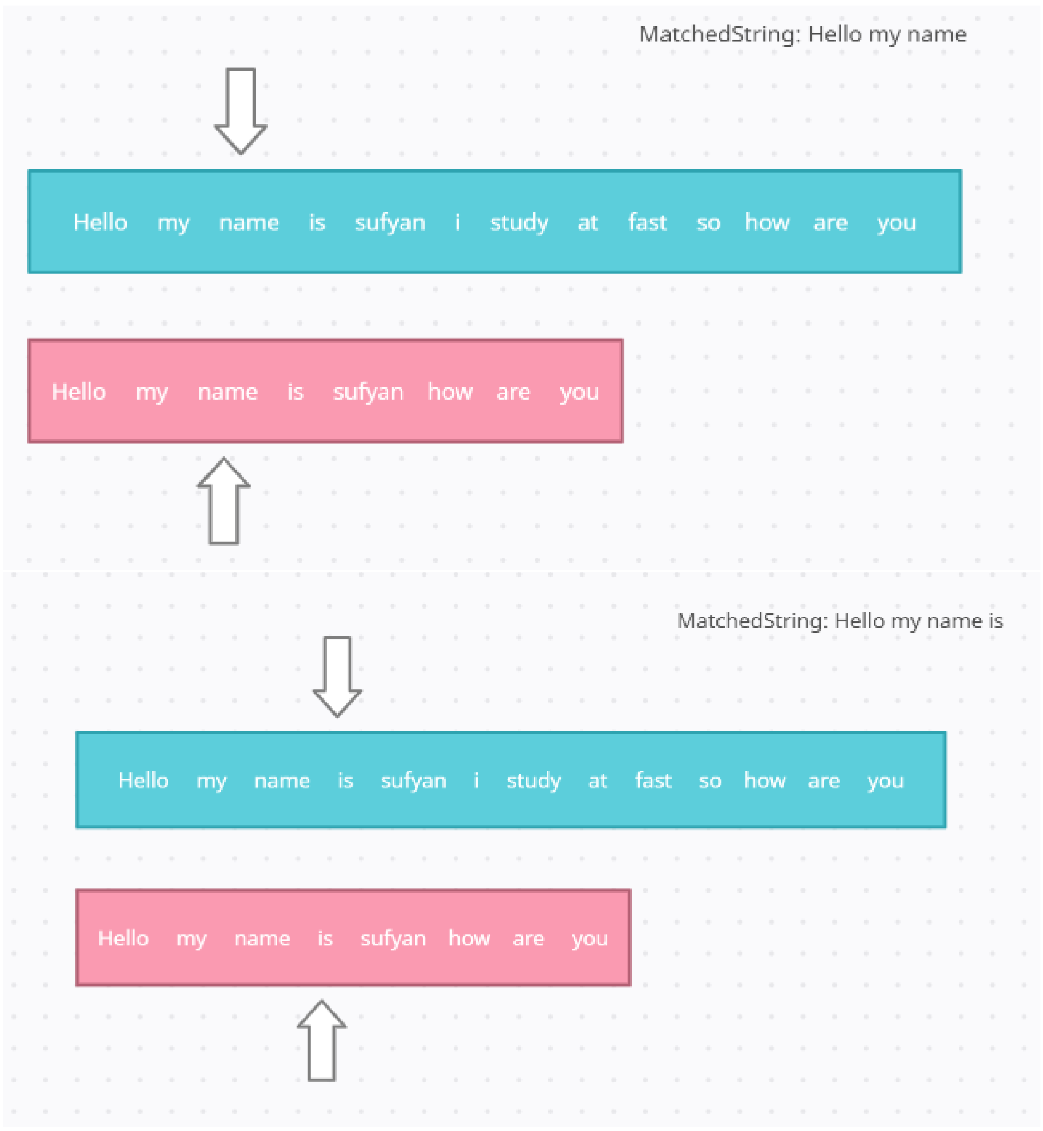
End Result was to use KMP algorithm to display the matched phrases that has been detected in the Target string.

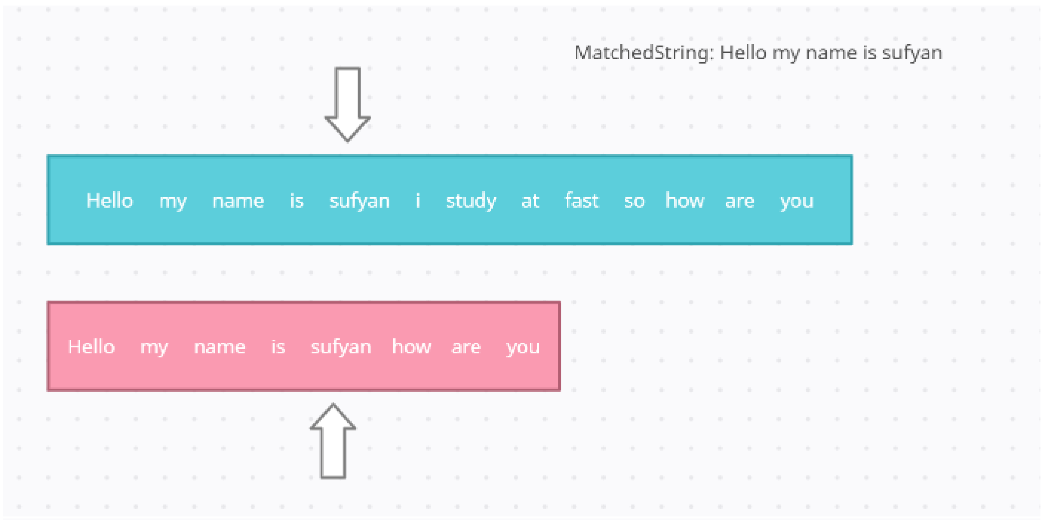
0.4.1 Improvements

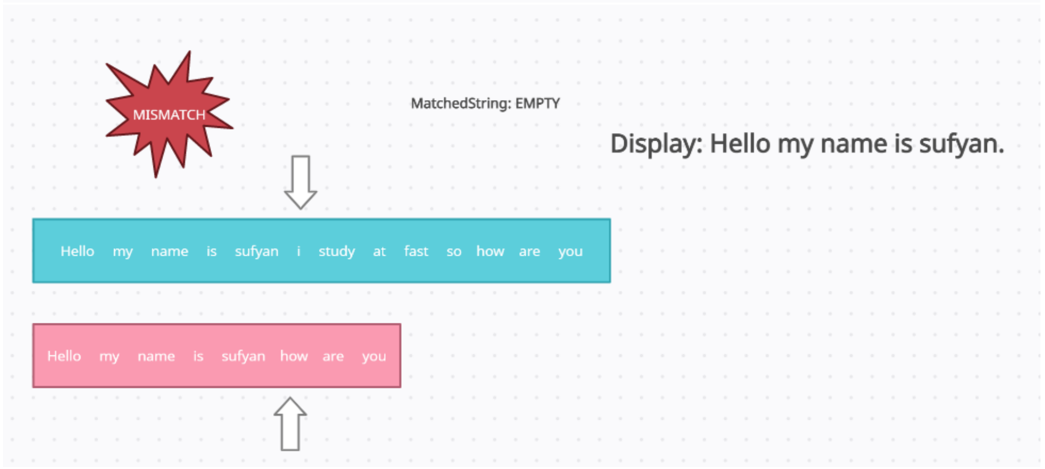
There could have been a system in place which alerts the student who has been caught of plagiarism and is emailed about the phrases which have been copied from the Source file. The source is sent to him from where the plagiarism is detected.

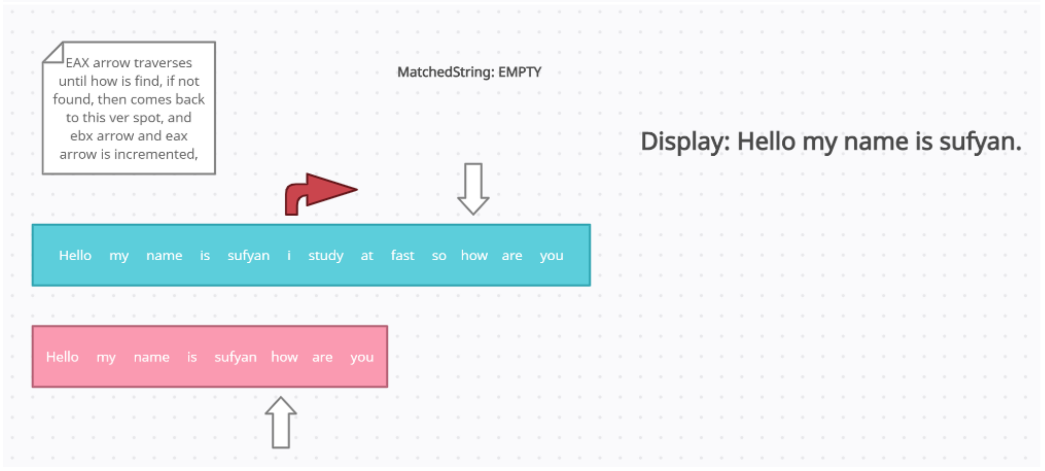
0.5 Algorithm Visualised.

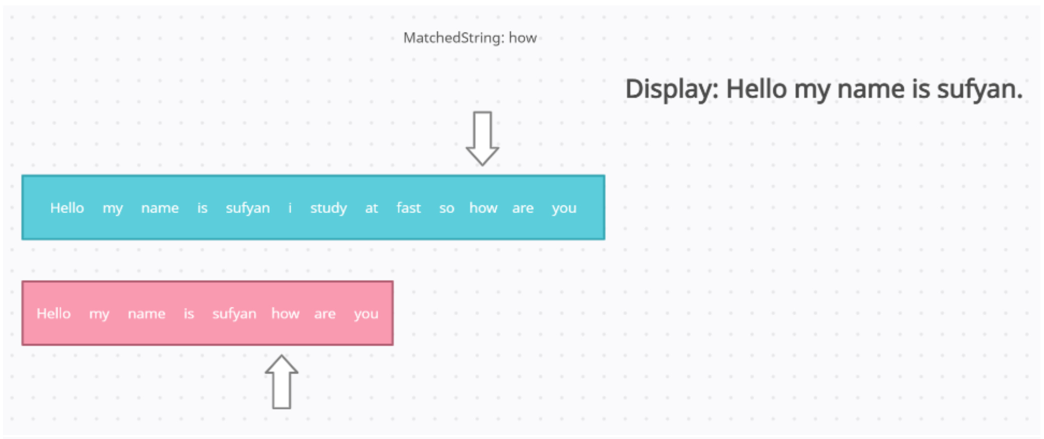

MatchedString: Hello
Hello my name is sufyan i study at fast so how are you
Hello my name is sufyan how are you
MatchedString: Hello my
Hello my name is sufyan i study at fast so how are you
Hello my name is sufyan how are you

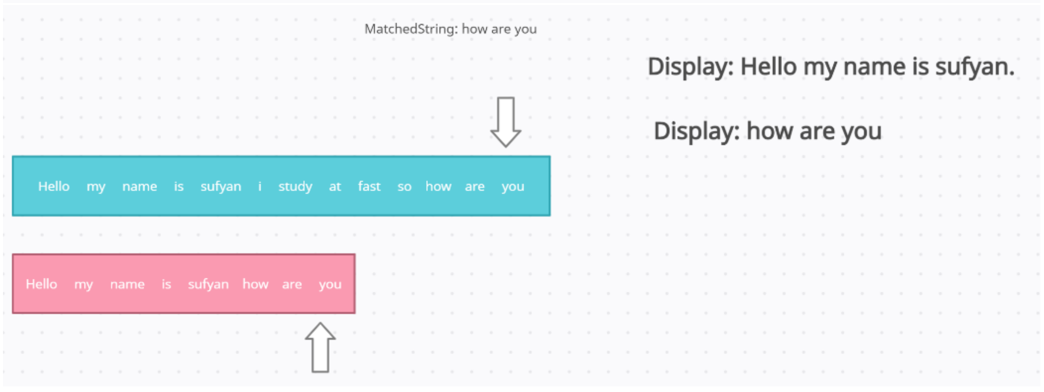

MatchedString: Hello my name
Hello my name is sufyan i study at fast so how are you
Hello my name is sufyan how are you
MatchedString: Hello my name is
Hello my name is sufyan i study at fast so how are you
Hello my name is sufyan how are you


MatchedString: Hello my name is sufyan
Hello my name is sufyan i study at fast so how are you
Hello my name is sufyan how are you


MISMATCH
MatchedString: EMPTY
Display: Hello my name is sufyan.
Hello my name is sufyan i study at fast so how are you
Hello my name is sufyan how are you


EAX arrow traverses until how is find, if not found, then comes back to this ver spot, and ebx arrow and eax arrow is incremented,
MatchedString: EMPTY
Display: Hello my name is sufyan.
Hello my name is sufyan i study at fast so how are you
Hello my name is sufyan how are you


MatchedString: how
Display: Hello my name is sufyan.
Hello my name is sufyan i study at fast so how are you
Hello my name is sufyan how are you


MatchedString: how are you
Display: Hello my name is sufyan.
Display: how are you
Hello my name is sufyan i study at fast so how are you
Hello my name is sufyan how are you

0.6 Sources

- https://www.javatpoint.com/daa-knuth-morris-pratt-algorithm
- https://www.youtube.com/watch?v=V5-7GzOfADQ
- https://www.ics.uci.edu/ eppstein/161/960227.html