Final Project for CS 372

***Erase this*** *before you submit: You are more than welcome to bullet point or tabulate everything like in the first heading and built-in tests. I’m going to be “checking” off what I can from the report. The easier it is to find, the less likely a requirement will be missed. Team only parts are in green.*

*Ryan Driscoll and Jacob B. West*

# Algorithm, Application, Language Choice

* *Knuth-Morris-Pratt Algorithm (KMP)*
* *Searching for subreddit patterns on Reddit*
* *Javascript*

# Program usage or README

[OPTIONAL, and only used if needed]

# Where It Is Used

*[Brief description of the class of problems your algorithm(s) solves]*

Knuth-Morris-Pratt (KMP) is a pattern matching algorithm used in a variety of settings. KMP will traverse input from a user (commonly in the form of a string) from left to right. The class of problems this algorithm solves in commonly referred to as the “Needle in the Haystack” problems due to the fact that it is common to be searching for a small pattern or string within a large dataset. Because this algorithm will find matching patterns in a string it is commonly used in DNA sequencing. This is only one of many applications that can utilize KMP since it can be applied to any type of data set that involves finding a pattern of characters.

## Other applications

* *Finding text in a document based on a pattern input by a user*
* *Finding the occurrence of a substring within a string (which is what our application will be conducting.)*
* *Finding plagiarism by comparing documents. (The input string would be a part of another document and if that pattern exists in the document under review then plagiarism would exist)*

## Alternative algorithms

* *Boyer Moore Pattern Matching Algorithm*
* *Rabin-Karp*
* *Aho-Corasick Algorithm (where the set of patterns would be just one pattern)*

## Reason for choice

*[Explain why you picked the algorithm you did. If I picked, explain what areas it “wins” over the alternative algorithms]*

*[1/2 page max]*

*KMP is an efficient pattern matching algorithm with the worse case run time being O(n). This along with the elegant simplicity of the algorithm is what attracted us to use it for our application. Aho-Corasick has a run time of O(N + L + Z) which will depend on the length of the text, the number of patterns searching (which in our case would just be one pattern so it would be negligible) and the number of matches found. With KMP the N in O(N) represents the length of the string we are matching against, therefore resulting in a bit faster algorithm. Boyer Moore Algorithm has a much greater need in the set up and in the end involved more code than necessary compared to KMP and our needs for the application. We wanted to choose an algorithm that was easier to write due to other aspects of the project that would involve time developing such as the frontend GUI and the API calls to Reddit. Lastly, KMP was chosen over Rabin-Karp due to time complexity as Rabin-Karp runs in O(nm) for worst case whereas KMP worst case will always be O(n), which is a much greater run time.*

# How Your Project Works

*[Pretend you are explaining how your project works to a Programming II freshman. 1-3 paragraphs will likely be sufficient, but ask if you are unsure This is mostly to make sure that you understand what you did and didn’t just copy and paste code from somewhere.]*

*(Team projects are expected to have more detail. You must “chunk” the algorithm and explain each part.)*

## Correctness (team only)

*[Formally, justify why your algorithm is correct or why it works. You must formally use loop invariant, pre-and post conditions, etc. Any and all properties for a given algorithm, if applicable, are required. Assume you are speaking to someone who has completed this course]*

# Run time

*[Name and explain why the project has this big-O run time as the theoretical run time* **for your implementation***]*

*[A* ***fully*** *labeled graph* runtime graph for varying n with a minimum of 10 points goes here. It **must clearly show** the run time. If you have more than 1 value that affects input, you may just vary one for the graph, but be clear which one you used. Two graphs would be ideal, and you may use a minimum of 5 points for each in this case)

(team only): Formally prove the run time through instruction counting, probability, or recursion analysis depending on your problem. You **will need** to use psudeocode to prove this.

# Code Correctness Tests

(team only, you must have a minimum of 6 rather than 3 built-in tests)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Test Case*** | ***Description*** | ***Input*** | ***Expected Output*** | ***Actual output*** |
| *[test 1]* |  |  |  |  |
| *[test 2]* |  |  |  |  |
| … |  |  |  |  |

Alternatively, you may have this in the following format (copy paste from code if in a similar format is also approved):

## Test 1; name

Description

### Input

The input

### Expected Output

The output

### Actual Output

The output

# References

[Where did you find the explanation of your code]

[Tutorials]

[Etc.]