COSC 320 – 001
Analysis of Algorithms
2020 Winter Term 2

Project Topic Number: #4

String Matching Problem - Milestone 2

Group Lead:

Matthew Borle - 42615435

Group Members:

Barry Feng - 12981156

Guy Kaminsky - 65234775

Ross Morrison - 36963411

Abstract.

In this milestone, we went into further detail about the three algorithms in terms of analyzing them. In particular, we started with understanding how each of the three algorithms differs in matching strings. Recognizing the differences allowed us to decide how to implement all three algorithms and determine whether a document plagiarizes another document. As a result, we concluded to run each algorithm separately and apply an equal weight to them. Following this, we will combine all three weights and get a quantitative number. The resulting number is then checked against a predetermined range to determine if plagiarism exists. With this in mind, we separated each algorithm and analyzed which data structure fits best. In other words, we use these data structures to see efficient run times when running the algorithms independently.

Algorithm Analysis.

To ensure proper pattern matching, each algorithm requires a separate data input. For KMP, start by splitting the document by sentence ending characters, period and semicolon for example, and put each into a string list. Then merge each string list into a master list. Finally run each sentence through the KMP algorithm and sum the total number of matches.

For LCSS, start by splitting the document by paragraph breaks into a string list. Then run each paragraph through the LCSS algorithm. If a paragraph has a matching subsequence over a certain number of characters then sum the total number of matches of paragraphs.

For RBK, start by splitting the document by a single space character into a string list. Run RBK for each word. Sum the total number of matches and divide by the length of the split string list.

In the end, combine each sum found by the algorithms and apply the equal weight factor. This gets the total percentage of plagiarism found in the input document for the selected corpus document. If the percentage is high enough, add the corpus document to the list of potentially plagiarised ones. Continue to run this for a selected amount of corpus documents.

The running time of Python's string.split method is **O(*n*).**
For setting up KMP, our algorithm's running time will be **O(*kn*)** where *k* is the number of sentence ending characters. For setting up LCSS, the running time will be **O(n)** and for setting up RBK the running time will be **O(*n*)** again.
So at most the pre-processing will take **O(*kn*).** Consider that there are *m* corpus documents that will result in a final runtime of **O(*knm*).**

Data Structure.

- Dictionary: RABIN-KARP uses hashes to compare words without wasting time.

  PROS:

  Avoids repeated comparisons and quicker to traverse.

  CONS:

  Takes time to set up O(*n*), Makes code difficult to read.

  Reference:
  https://brilliant.org/wiki/rabin-karp-algorithm/

- List: LCSS and KMP require arrays/lists, python uses lists.

  PROS:

  Lists are supported by default in Python, Python lists are faster than Python arrays continuously create and destroy objects when reading and writing. Lists are also ordered.

  CONS:

  Lists allow duplicates which may result in redundant comparisons.

  Reference:
  https://physics.nyu.edu/pine/pymanual/html/chap3/chap3_arrays.html

Unexpected Cases/Difficulties.

A short-term challenge we faced was deciding how to approach the algorithm analysis section. With the uniqueness of our project being structured, we analyzed all three algorithms in the first milestone. Since this section repeated, we changed our approach by individualizing each algorithm. Given that, we took all these individual analyses and went into further detail. In particular, we also analyzed how all three algorithms will be functioning together.

The next challenge appeared when we had to decide which data structures to use. For example, we had to determine which data structures would make a difference to the run time. As all three algorithms are similar, it was tough to decide when a different data structure would be optimal. After some research, we concluded the difference comes when implementing the RBK algorithm. This algorithm uses a hash map, whereas LCSS and KMP use lists. A hash map compares phrases and will not repeat comparisons as it is hashing everything, thus making the process quicker.

Another interesting point that came up with using hash maps is identifying the differences between lists and arrays in Python. We had to look at the Python API to get more information. In general, both lists and arrays are used to store data in Python. However, lists are ordered and dynamically typed. Meanwhile, arrays are not as straightforward to use. Thus, we will use lists in our final program.

Task Separation and Responsibilities.

In terms of task separation, we implemented peer programming techniques. Specifically, Guy Kaminsky wrote the abstract. Guy Kaminsky and Ross Morrison worked on Algorithm Analysis. They did this task by completing the proof of correctness and running time of the algorithms. Next, Barry Feng and Matt Borle researched the data structures and which data structures to use for each of our algorithms. We also included the pros and cons of each data structure for that particular algorithm. Finally, everyone contributed to the unexpected cases and difficulties.