COSC 320 – 001

Analysis of Algorithms

Milestone 2

2022/2023 Winter Term 2

Project Topic Number: 1

Title of project:

Keyword replacement in a corpus

Group Number 12:

Anna Ciji Panakkal

Erfan Kazemi

Sahil Chawla

Abstract: In this milestone, we used one of the other algorithms we came up with. After gaining a better comprehension of the algorithm we selected, we proceeded to develop the pseudocode and analyze the algorithm accordingly. To accomplish these tasks, we convened on Discord.

Problem Formulation:

The algorithm for this approach would involve using regular expressions to search for patterns in the text and using a string-matching algorithm, such as the Levenshtein distance algorithm, to find the closest matches to an abbreviation. The algorithm would begin by preprocessing the list of abbreviations by sorting them in order of length and then creating an index of the corpus of tweets. Once the index is created, the algorithm would search through the index for each abbreviation in the list and then use a string-matching algorithm to find the closest matches. The algorithm would then replace the abbreviations in the text with the appropriate word or phrase.

Input: A list of abbreviations, a corpus of text
Output: The corpus of text with the abbreviations replaced with the correct words or phrases

Algorithm:

1. Preprocess the list of abbreviations by sorting them in order of length

2. Create an index of the corpus of text

3. For each abbreviation in the list:
    a. Search the index for the abbreviation
    b. If the abbreviation is found, use a string-matching algorithm to find the closest match
    c. Replace the abbreviation in the text with the appropriate word or phrase

4. Output the corpus of text with the abbreviations replaced with the correct words or phrases

Pseudo-code:

```
// Preprocess abbreviations
List<String> abbreviations = sortByLength(listOfAbbreviations);

// Create index of corpus
Map<String, List<Integer>> index = createIndex(corpusOfText);
```

```
// Iterate through abbreviations
for (String abbreviation : abbreviations){
    // Search index for abbreviation List<Integer> matches =
searchIndex(abbreviation, index);

    // If abbreviation found, use string matching algorithm

    if (matches.size() > 0)
    {
    String closestMatch = stringMatchingAlgorithm(abbreviation,
matches);

    // Replace abbreviation with closestMatch

    corpusOfText = replaceAbbreviation(abbreviation, closestMatch,
corpusOfText);
    }
}

// Output corpus with abbreviations replaced
outputCorpus(corpusOfText);
```

Algorithm Analysis:

The correctness of the algorithm can be proven by showing that the algorithm produces the correct output for all possible inputs. For example, given a list of abbreviations and a corpus of text, the algorithm should produce the corpus of text with the abbreviations replaced with the correct words or phrases.

The running time of the algorithm can be proven by analyzing the time complexity and showing that it is proportional to the number of abbreviations in the list and the length of the corpus. This can be done by counting the number of operations performed by the algorithm and showing that it is proportional to the number of abbreviations and the length of the corpus.

This algorithm is a fast and efficient way to replace abbreviations in a corpus of text.

The algorithm begins by preprocessing the list of abbreviations and creating an index of the corpus of text. Then, it iterates through the abbreviations, searching for each one in the index. If an abbreviation is found, a string-matching algorithm is used to find the closest match. Finally, the abbreviation is replaced in the text with the appropriate word or phrase.

The time complexity of this algorithm is O(n*m) where n is the number of abbreviations in the list and m is the length of the corpus.

The space complexity is O(n+m) as the algorithm requires space to store the list of abbreviations, the index, and the corpus of text.

Unexpected Cases/Difficulties:

One of the difficulties with this algorithm is dealing with abbreviations that have multiple meanings. For example, the abbreviation "ASAP" could mean "As Soon As Possible" or "Always Seek Actionable Plans". In this case, the algorithm would need to use some kind of context-aware technique to determine the correct meaning.

Another difficulty with this algorithm is dealing with homophones, which are words that have the same pronunciation but different meanings. For example, the words "their" and "there" have the same pronunciation but different meanings. In this case, the algorithm would need to use a dictionary or other source of linguistic data to determine the correct word.

How would this second difficulty even affect us?

This second difficulty could affect the accuracy of the algorithm since it would be unable to determine the correct meaning of homophones. In some cases, this could lead to incorrect words or phrases being used in the output. For example, if the algorithm replaced the abbreviation "thr" with the homophone "their", the output would not be accurate.

Task Separation and Responsibilities:

Anna, Erfan, Sahil - Problem formulation

Erfan - Analysis algorithm, abstract

Sahil, Erfan - Pseudo code

Anna - Unexpected Cases/Difficulties