COSC 320 – 001

Analysis of Algorithms

2022/2023 Winter Term 2

Project Topic Number: 1

Milestone 3

Keyword Replacement for Text

Group Lead: Brandon Mack

Group Members: Aidan Elliott, Eddy Zhang

Abstract. One paragraph about your achievements in this milestone. This should be included for all milestones.

In this milestone, we explore another way of implementing the acronym replacement algorithm using the trie data structure. Our goal is to achieve a similar time complexity as our first hashmap-based algorithm. We discuss the reason for using a trie data structure and how it enables an O(nm) time, where n is the number of words in the tweet and m is the maximum length of a keyword that is in the list. We use proof by induction and the pseudocode to demonstrate the validity of our algorithm and its time complexity. We also state the possible difficulties and unexpected cases we might encounter when implementing the algorithm and provide possible solutions to those problems.

Analysis of the Algorithm

Algorithm Analysis: Compared to a naive approach of searching for each keyword in the tweet the use of a trie data structure will greatly reduce the search time. In the worst case, the time complexity will be O(nm) where n is the number of words in the tweet and m is the maximum length of a keyword that is in the list. For the most part, the total amount of words in a tweet will outway the number of keywords therefore the number of potential keywords that need to be checked in each tweet is smaller than the number of all words. This means that the algorithm only needs to check words that have the potential of being keywords. This being said the actual runtime of the algorithm is highly likely to be faster than the worst-case scenario.

Proof of Correctness: Inductive Hypothesis

- Let P(n) represent the assumption that the algorithm will correctly replace all abbreviations in a tweet of length n
- Base Case:

  For n = 1, a tweet will only contain one word, if this word is a keyword that matches in the valid keyword list, the algorithm will correctly replace it with the full meaning. Therefore P(1) is true.

- Inductive Step:

  We can assume that P(n) is true for some n > 1. Let's consider a tweet of length n + 1, the algorithm will check each prefix of a tweet to determine if there is a potential keyword that needs to be replaced. For each of these, the algorithm will traverse the trie to match if a keyword is in the keyword list.

Since we assumed that P(n) is true, the algorithm will correctly replace all keywords in a tweet of length n. This means that all prefixes of the tweet up to length n have been checked for potential keywords and any keywords have been appropriately replaced.

Let's consider a suffix at the end of the tweet of length 1. If this suffix is a keyword that does appear within the keyword list, the algorithm will correctly replace it with the full meaning. Otherwise, the suffix will be a keyword that does not need to be replaced and the algorithm will not change it.

Thus, the algorithm will correctly replace all keywords in a tweet that are of length n + 1, therefore we can say P(n+1) is true.

Through the use of mathematical induction, P(n) is true for all n > 0 and the algorithm is correct for any tweet of length n as long as n > 0.

Pseudocode:

```
for each tweet in the corpus
    words = split tweet into individual words
    i = 1
    while i < length(words) do
        j = i
        node = root of trie
        while j < length(words) and node has a child with label words[j] do
            node = child node with label words[j]
            j = j + 1
        done
        if node has a value then
            replace words[i] to words [j-1] with nodes value
            i = j
        else
            i = i + 1
        done
    done
done
```

Runtime: O(nm)

Data Structure. Choice of your data structure and the rationale behind it.

Our choice of data structure is to use a Trie. We can build a Trie with characters as nodes that contain every abbreviation. At each branch where an abbreviation can be found, the phrase

that the abbreviation is short for will be stored. By searching each word in a sentence through this Trie and replacing it with its full phrase, our algorithm will take O(m) time where m is the number of characters in the sentence. We choose this structure because like the hashmap, using a trie has linear runtime and also does not involve any collisions so this runtime is not just expected.

Unexpected Cases/Difficulties. If you encountered any issues with the design and implementation of your work, include them here. Also, mention the solutions you have considered to alleviate the issue.

- Unlike our hashMap, there is not a standardized Trie data structure implemented in Python so we may have to design our own or find a library to download.
- We will likely have a scenario where an abbreviation can be made but is also a prefix of other abbreviations so we must ensure our Trie can handle these scenarios. (Ex: H=Hydrogen but HD=High Definition)

Task Separation and Responsibilities. Who did what for this milestone? Explicitly mention the name of group members and their responsibilities.

Aidan Elliott - Data Structure and Unexpected Cases/Difficulties

Brandon Mack - Algorithm Analysis

Eddy Zhang - Abstract