COSC 320 – 001
Analysis of Algorithms
2022/2023 Winter Term 2

Project Topic Number: 1

Keyword Replacement Analysis

Group Lead:

Erin Hiebert

Group Members:

Erin Hiebert

Louis Lascelles-Palys

Patrick Ma

Abstract

Created and understood our chosen algorithm that deals with the problem of replacing abbreviations in a chosen text. Translated the problem to closer resemble an algorithmic problem and supplied pseudocode that will be used in implementation of our algorithm. Analysed our programs running time and proved the correctness of the algorithm. Identified any weaknesses in the algorithm or possible problems that the algorithm might run into and supplied the possible solutions for these problems. A summary of group member tasks is included at the end.

Problem Formulation

Our problem will include a list of strings obtained from multiple CSV files and a **Hash Table** that stores the full word equivalent to abbreviations; these abbreviations will be a static list that we will collect. We will use a **Tokenizer** on the text that is being processed, delimited with spaces. From there the algorithm will go through the text, reading each word and comparing each to the keys in the hash table. Once the algorithm has completed it will return the text with the replaced abbreviations of the words.

Given the list of strings with a length of n: checking each string in the list for abbreviations will run in O(n) time, checking and returning values from the hash table will run in O(1) expected time.

Pseudo-Code

```
retrieve(key, value):
        index = 0
        while array[index] != null:
                index = hashFn(key, index)
                if array[index].key == key:
                        return array[index].value
                else:
                        index++;

readTokens(combinedFile.csv):
        StringTokenizer st = new StringTokenizer(combinedFile.csv);
        while (st.hasMoreTokens()):
                st.NextToken(retrieve(key, token));
```

Algorithm Analysis

Hash Table: O(n) Worst Case.

Whether using open addressing or chaining in a hash tables the worst case will be where accessing a key will have to go through all values in the hash table before returning the correct value that we were searching for. In this case, “n” is the size of the hash table. To combat this we will be using a hash table with a large enough table (larger than “n”) that collisions are very low if not even possible as well as a good hashing function. With these we can expect that a retrieval from the hash table will be in O(1) time.

Tokenizer: O(n) Worst Case

The Tokenizer needs to go through each and every word in the review to search for any keys. At this point, the CSV files have already been combined into one large file, and “n” would therefore be the number of words for all reviews in total. This would amount to a very large number of words, but would still be considered O(n). Upon going through each token, they would be compared directly to the hash table, and flagged for any abbreviations.

Proof of Correctness

Loop Invariant: Abbreviations to the left of the pointer of the StringTokenizer are replaced with their proper phrases/word.

Proof.

Inductive Hypothesis: The loop invariant holds after all tokens of a review are to the left of the StringTokenizer’s pointer..

Base Case: After the initialization of the StringTokenizer, all abbreviated tokens to the left of the pointer are, by nature of the initialization, in their proper forms as there are no tokens to the left yet.

Inductive Step:

- Assuming the loop invariant is true for the previous position of the pointer, proving that the potentially abbreviated token immediately before the current position of the pointer has been replaced with its proper form proves the loop invariant true for the current position of the counter as well.
- By nature of the StringTokenizer, all tokens will be retrieved which means that the potentially abbreviated token immediately before the current position of the pointer will not be missed by our algorithm.
- Using a hash table, all tokens will be used as keys and hashed to an numerical index that corresponds to each abbreviation's proper form. Keys that do not match to an included abbreviation’s hash will result in the token not being replaced in the resulting text. Keys that do match an included abbreviation will map to a proper form in the hash table and will thus be replaced in the resulting text.
- As the tokenizer moves over from its previous position to the current position, it will read in the token and hash it to see if it matches the hashes of included abbreviations. Assuming that the token read is an abbreviation, its hash will match the hash of an included abbreviation and our algorithm will replace it with its proper form. If the token is not an abbreviation, the algorithm will take it as is.
- Thus, the loop invariant holds for the current position of the StringTokenizer.

Conclusion: Therefore, once the StringTokenizer has finished processing all tokens in a review, all abbreviated tokens are replaced with their proper forms and the inductive hypothesis is held true. It is clear how this then

translates to all abbreviated tokens being replaced with their proper forms within a set of reviews in a CSV file, and thus, within multiple CSV files.

Proof of Running Time

The total running time of this algorithm is **O(n)** expected time, and **O(nm)** worst-case time, where n represents the number of tokens in a review and m represents the number of abbreviations our algorithm looks to replace.

As certain elements of our algorithm such as our hash function are not yet decided upon, certain assumptions must be made about our algorithm:

- Our hash table will take O(1) expected time to search using each token as the key
    - For this to be the case, our hash function and the size of our hash table must be optimal in that they reduces collisions as much as possible
    - As collisions are still possible even in optimal situations, searching may take O(m) worst-case time
- Our StringTokenizer takes O(1) time to read in an individual token

Proof.

Let ***A*** represent the set of tokens in any given review to be read by the StringTokenizer.

Let ***a*** represent any given token within A.

We will prove by generalizing from a generic particular that the running time of our algorithm is O(n) expected time and O(nm) worst-case time.

As tokenizing any ***a*** takes constant time, and ***A*** has n tokens, tokenizing the whole review will take O(n) time. Within each tokenization step, the token is hashed in O(1) time and searched for in the hash table of included abbreviations which takes O(1) expected time or O(m) worst-case time. Replacing the abbreviated words in the review will take O(1) time since the number of operations for this step does not depend on n or m.

Conclusion: Therefore, the whole algorithm runs in O(n) expected time and O(nm) worst-case time.

Unexpected Cases/Difficulties

Difficulties would arise in two scenarios of the hashing part of the algorithm:

1. Going through the entire list of values to find a match to the key (O(n))
2. Going through the entire list of values to notice there's no equivalent key in the table

Otherwise, there's special cases to take care of:

- Empty review (just a rating)
- No abbreviations in the review
- An abbreviation is located at the end of a sentence (token includes a period)
- Multi-line reviews — when combining them, surround them by quotes to avoid confusion

- Be careful if a word is ALSO an abbreviation (ex: “so” being like “significant other”)

Combining the CSV files into one single consolidated file will be done outside the algorithm.

Tasks Separation and Responsibilities

Tasks were separated evenly between group members. Erin wrote the abstract and formulated the problem into an algorithmic process and analysed the running time of the algorithm. Louis supplied the pseudocode for the algorithm and added unexpected cases or difficulties that our algorithm would run into and analysed the running time of the algorithm. Patrick proved the algorithm's correctness and running time and supplied some unexpected cases and difficulties that the algorithm would run into. Editing of the report was done by all members.