COSC 320 – 001

Analysis of Algorithms

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 brainstormed ideas for different algorithms that can be used. Each of us came up with one idea and chose two out of those for the project. Once we deepened our understanding of the algorithm that we chose, we worked on the pseudocode and algorithm analysis for the same. We met via discord to do all of the above-mentioned activities.

Problem Formulation:

We can save our words in a tree, T, that will sort words alphabetically, and each node, N, of this tree will contain two parts. The first part is the abbreviation of the word, and the algorithm will compare the word with the text. If both seem to match, the elaboration (expansion) part will replace the abbreviation part. Alternatively, we can also implement this algorithm without sorting; instead, we could search branches parallel and find the word and replace it.

Pseudo-code:

```
class Node<T> {
    T abbreviations;
    T expansion;
    Node<T> next;
    Node<T> tail;
    Node(T abbreviations, T expansion) {
        this.abbreviations = abbreviations;
        this.expansion = expansion;
    }
}
Class AbbTree {
    Node root;
    Node tail;
    int maxBranchSize = n;

    AbbTree(root) {
        root.abbreviations = null;
        root.expansion = null;
        root.next = null;
    }
    Insert(String abbreviations, String expansion) {
        If(branchSize < maxBranchSize) {
            Node newNode = new Node(abbreviations,expansion);
            tail.next = newNode;
            tail = newNode;
        }
        Else {
            Node newNode = new Node(abbreviations,expansion);
            root.next = newNode;
            tail = newNode;
        }
    }
    //all getters and setters
}
//each branch is going to send from cpu to gpu for parallel computing
Function parallelTraverse(String word,Branch b1) {
    Node current = b1.root;
    For(i; i < branch.len; i++) {
        If(word.equals(node.abbreviations)) {
            Replace(node.expansion);
        }
        Else {
            current = current.next;
        }
    }
}
```

Algorithm Analysis:

Problem statement: We will be discussing the use of text analytics in replacing a set of keywords with a given set in documents to create an algorithm that, despite the possibly long list of keywords and acronyms, can quickly and accurately locate and replace these terms in a high volume of tweets.

Algorithm definition: We can store our words in a tree that will arrange them alphabetically, with two parts at each node. The algorithm will compare the term with the text after analyzing the word's abbreviation in the first portion. If both appear to be accurate, the expansion (elaboration) component will take the place of the abbreviation section. This approach can also be used without sorting; instead, we could scan parallel branches for the word and replace it.

Input and output specifications: The input is a string representing an abbreviation and a string representing its expansion. The output is the word after it has been replaced with its expansion, if applicable.

Time and space complexity: Due to the parallelTraverse function's time complexity of O(n), where n is the number of nodes in the branch, the overall time complexity of the code is O(n). This is due to the fact that the function utilizes a for loop to explore the branch and, in the worst scenario, will do so across all n nodes. The time complexity of the code as a whole is unaffected by the O(1) time complexity of the Insert method of the AbbTree class.

The AbbTree class employs a linked list data structure, and each node in the list occupies a fixed amount of space; hence the overall space complexity of the code is O(n).

Loop invariants: The function's invariant in this situation might be that it always begins at the root node and iterates through the following nodes until it either finds the word or reaches the branch's end. The parallelTraverse function for loop invariant can be expressed more formally as follows-

The current node is set to the following node in the branch from the previous iteration or to the root node if it is the first iteration at the start of each iteration of the for loop. The loop runs until the branch's end is reached, in which case the word is left unmodified or until the word is found and substituted with its expansion.

Initialization: The creation of an AbbTree object starts the algorithm. The root node's expansion and abbreviation are set to null by the constructor, along with the next field of the root node to null and the tail field of the AbbTree object to the root node. The value of the maxBranchSize variable is n. The tail field is used to monitor the end of the linked list, while the root node acts as a placeholder for the tree structure.

Maintenance: In order to iterate through the linked list of nodes in the branch, the parallelTraverse function makes use of a loop. Using a variable current to track the current node being examined, the loop invariant is maintained. The function determines whether the input word matches the current node's abbreviations value. The expansion value takes the place of the word in this case, and the loop ends. If not, the loop continues, and the current variable is moved to the next node. By iterating through the linked list iteratively until the word is found or the branch ends, the algorithm moves closer to a solution.

Termination: The parallelTraverse function ends when either the end of the branch is reached without finding a match or the word is found and replaced with its expansion. By searching the linked list for a matching abbreviation and replacing it with its expansion if one is found, the algorithm produces the desired result. The word remains unchanged if it is not found. The algorithm is able to expand abbreviations in a text because of this.

Correctness argument: The algorithm comes to a successful conclusion for the following reasons-

Loop Invariant: In each iteration, the variable current is updated to the next node, advancing the linked list.

Termination Condition: When the word is found, or the branch's end is reached, the loop stops.

Correctness: If a matching abbreviation is found, the algorithm replaces it with its expansion.

Linear Search: The matching abbreviation is found using the algorithm's linear search, resulting in a time complexity that is proportional to the number of nodes in the branch.

The parallelTraverse function uses a loop invariant, termination conditions, a linear search, and checks for a matching abbreviation in each iteration to reach the correct conclusion.

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.

Task Separation and Responsibilities:

Anna, Erfan, Sahil - Problem formulation

Anna - Analysis algorithm, abstract

Erfan - Pseudo code

Sahil - Unexpected Cases/Difficulties