Report for LAB 3

OVINDU RANDUNU CHAKRAWARTHIGE

Deadline: 6th of December, 2023 at 23:59

Task description

The program is designed to function as a spell checker. Its primary task is to process a stream of text and identify words that are not present in a pre-loaded dictionary. The program reads two sets of inputs: first, the dictionary words, and then the text to be spell-checked. It efficiently identifies and reports the non-dictionary words while handling various input formats and ignoring case differences. The main challenge is in the efficient handling of large dictionaries and accurate detection of word boundaries in varied text formats.

Structures and Algorithms

Data Structures Used:

Trie: The trie is a specialized tree used to store associative data structures.This facilitates efficient retrieval, often used for dictionary lookups. In this program, it replaces an array-based dictionary to optimize the storage and retrieval of words. This choice addresses the inefficiencies of linear search in an array, especially as the size of the dictionary increases.

Program Structure:

Trie Node Structure (trieNode): Represents each node in the trie. It includes a boolean flag to indicate the end of a word and an array of pointers to child nodes, representing each letter of the alphabet.

Dictionary Structure (dict): Encapsulates the trie by holding a pointer to its root node.

Utility Functions:

newTrieNode: Allocates and initializes a new trie node.

addWord: Inserts a word into the trie.

check: Searches for a word in the trie.

freeTrieNode and freeDict: Recursively frees allocated memory used by the trie.

Main Function: Manages input processing. It first reads the dictionary words, adding them to the trie. Then, it processes the text input character-by-character, identifying and reporting words not found in the dictionary.

Evaluation of the program

Testing Strategy:

Functional Testing: Using predefined input sets (dictionary and text) to verify that the program correctly identifies unknown words.

Boundary Case Testing: Inputs like long words, special characters, and varying word separators to ensure robust handling of word boundaries.

Performance Testing: Utilizing large dictionaries to evaluate the efficiency gains from using a trie over an array-based approach.

Notes: All tests were performed using the makefile or Codegrade.

Identified Problems and Resolutions:

Word Boundary Detection: Initially, the program couldn't distinguish concatenated words separated by non-alphabetical characters. This was resolved by modifying the input processing loop in main to read one character at a time and identify word boundaries more accurately.

Array-Based Dictionary Limitations: The initial version used an array to store dictionary words, leading to inefficiencies in word lookup (linear search complexity). By implementing a trie, the search complexity was reduced, significantly improving the program's performance, especially with large dictionaries.

Conclusion and Self-Reflection

Key Learnings: This assignment emphasized the importance of choosing the right data structure for specific problems, particularly how a trie can vastly improve performance for tasks like dictionary lookups.

Challenges: Implementing and integrating the trie data structure was the most challenging aspect, requiring careful consideration of memory allocation and pointer manipulation.

Further Interests: Exploring further optimizations, like compressed tries or hash tables for different scenarios would be an interesting learning path.