Felicitas Pelita, Ace Reyes
Author: Ace Reyes
ICS 311

Saying Tree Documentation

Introduction

Data structures and algorithms play an integral role in many different kinds of applications. If a given program processes and stores data in some way, it is certain that within that said program lies an algorithm or a data structure.

In this assignment, we were tasked with developing and implementing a program designed to store the data of various sayings: a string of words in English, its translation in another language of choice, and the explanation of its meaning (in both English and in the second language). This documentation aims to explain our choice of data structure and algorithm - notably the Binary Search Tree (BST) and recursive algorithms - and why they are effective and efficient.

Binary Search Tree Database

In order to store the sayings, we went with a BST based database storing tree nodes, each of which storing a unique “Saying” data field. Each Saying itself is a class object having one method, a constructor, and holding four different fields of string data: the specific saying in Japanese (Romaji), the saying translated to English, an explanation of the saying’s meaning in English, and an explanation in Japanese.

The choice of a BST for this specific application is effective due to the nature of the BST. In our application, there exists methods that require certain sayings to be searched for. With each saying being alphabetically sorted within the BST, said tree allows us to search for the target word or saying efficiently in O(logn) time, as well as efficiently insert new sayings into the tree in O(logn) time, with a worst case scenario of O(n) in both cases (Sharma, 2023).

These are both possible due to the nature of each tree node having two children - the left and right. When searching for a specific saying, if a node’s left child has a saying which alphabetically precedes the target saying (or if it does not exist at all), we are able to ignore it and traverse through the node’s right child instead, allowing us to ignore an entire part of the database which would have taken unnecessary time to search through, for example.

Divide and Conquer Recursive Algorithms

Another chosen feature of this program is the extensive use of recursive algorithms, from search algorithms to insert algorithms. The recursive nature of these algorithms allows us to traverse and process data to and from the BST in a divide and conquer approach.

This has two major advantages. The first being that this allows us to very efficiently and effectively search through every node in the tree by recursively dividing sections of the tree into smaller subtrees to search through, which can then be further recursively divided. Essentially dividing the big problem into smaller subproblems recursively, as is the definition of divide and conquer (Cormen et al., 2022, pg 76).

The second major advantage of this is that this allows the code needed to traverse through the tree to be simpler, more elegant, and easier to implement and understand (Ferguson, 2022). This allows us to more easily develop the program and makes it less likely to fail due to errors in coding.

Conclusion

In this assignment we were tasked with designing and implementing a “dictionary” program to store specific sayings, as well as to search through them. Our choices of the use of a Binary Search Tree combined with recursive functions allows us to efficiently and effectively store data, and search through the tree for said data.

Citations

Cormen, T H., Leiserson, C E., Rivest, R L., Stein, C. (2022) *Introduction to Algorithms* (4th ed.). The MIT Press.

Ferguson, N. (2022, December 7) *Understanding Recursion Through Practical Examples.* Medium. https://medium.com/@teamtechsis/understanding-recursion-through-practical-examples-a3c586011f9d

Sharma, V. (2023, April 14) *Understanding Binary Search Trees in Python.* Medium. https://medium.com/the-modern-scientist/understanding-binary-search-trees-in-python-f49a5b7901e0