C++ Data Structure Programming - CS 20
Honors Project - Dictionary Tree

Researcher: Shahaf Dan
Institution: Las Positas College
Level: Lower Division Undergraduate
Title: Design and Development of a Dictionary Binary Tree Data Structure.

Research Focus: Use the learned materials about Binary Trees and apply them on a broader level to design a memory-efficient dictionary.

Abstract: Using the concept of a binary tree, in which each item [node] at every level, points to two (or more) items at a lower level, each item in the dictionary tree, points to 26 more nodes, each one with the capability to hold a different letter. Effectively, each node would point to an array of 26 additional nodes, all initially set to null. The project will test the idea that by sharing repetitive information (such as initial letters of words starting with the same sequence of letters) between items, in a tree data structure memory can be saved effectively and efficiently.

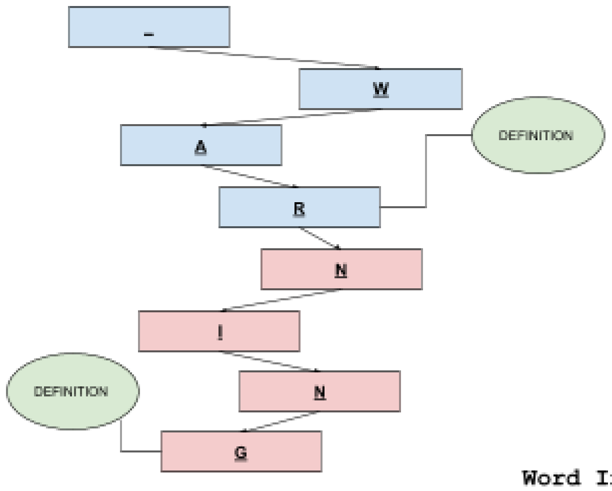
In the designed data structure, the first letter will be a neutral one, such that no word contains ('_'). The root (the neutral, uppermost node in the tree), will points to an array of 26 additional nodes, each holding a letter and pointing to another array of 26 letters [nodes], and so on. The end of each inserted word will be marked by the letter at which the word's definition is stored. In other words, each node can hold a letter and a definition. If the node holds a definition that is not null, hence it is the end of the word, and the path of letters pointing to each other until that node, create an inserted word.

At the beginning of the runtime, the user will be prompted to choose either to insert a word, display a word's definition, or to remove a word, where each method is done recursively. Inserting a word using an initial sequence of letters already used will take advantage of this data structure, saving the space and memory of storing additional nodes. For example, if the word 'war' is in the dictionary and the user inserts the word 'warning' the three first letters [nodes]

and their memories will be the same memory as the 'war' item, allowing the system to save the memory of three nodes for the new word. Each new word's end is marked with a definition.

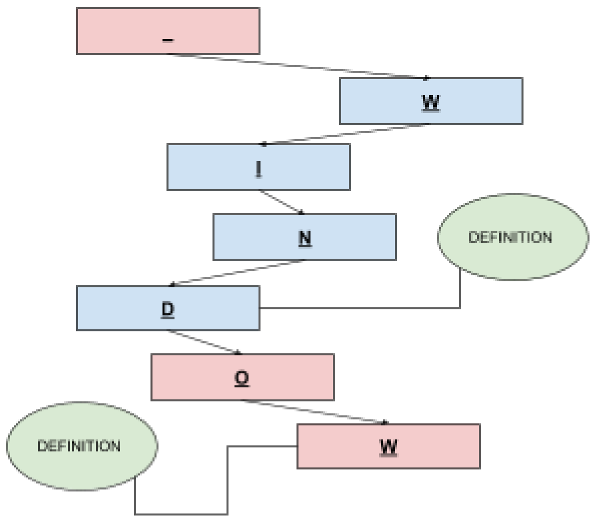
In a case where the long word with the repeated sequence is inserted first, the program will not create any new memory, rather will just store the definition of the new word ('war' in this case) in the last letter's node of that word. The concept of removing a word acts upon the same concept pf removing letters only if necessary, and to save space and action, removing only the necessary definition at specific places at the right scenarios. Each one method uses the 'check' function to validate and disprove the existence of a word or sequence of letters.

`Word Inserted: _War, _Warning`



Short word inserted before the long word.

`Word Inserted: _Window, _Wind`



Long word inserted before the short word