Adhyaksa Pribadi

Professor Ramirez

CS 1501

4 June 2015

Crossword Algorithm

A crossword is a puzzle consisting of a grid of squares and blanks into which words crossing vertically and horizontally are written according to clues. Just like how solving the crossword puzzle can be difficult, creating a crossword puzzle can be challenging as well. This is because the words both across and down must be legal, and the choice of a word in one direction restricts the possibilities of words in the other direction. The word choices that are made from the start restricts the number of legal words that would fit later on the board. The solution described below takes in a text file that contains the crossword board as the input. A board filled with letters that made legal words is the output. Therefore, the main objective is to find legal words that would fit an N by N board.

The crossword solution algorithm I attempted to implement was based on a recursive backtracking strategy. To implement the backtracking strategy, I created six separate methods. One of the methods that I wrote was “place”. I make recursive calls to this method. Mainly, this method is used to place letters on the board where it is legal to do so which means that the element indexed is neither a letter nor a minus sign. It takes in a column position as a parameter, and with each recursion, it will move the column position by one to the right. I have set the base case so that the puzzle is complete when the column position is at the most right and the row position is at the most bottom. Basically, my aim with this method was to fill the board from left to right and from top to bottom. Before each recursive call, it would check if the series of letters created a word or a prefix of a word or both. This is a crucial step in making the code more efficient. By verifying whether each string constructed so far is a prefix to a valid word, it will greatly prune the size of the exhaustive search tree. Used as a complimentary method, hasroom would check for the availability of the index by seeing if the spot on the board is a plus sign. Additionally, the method addletter adds a letter to the output board and appends the letter to the StringBuilder variable. Doing this will allow me to check the validity of the added letters. Removeletter will remove the letter from the StringBuilder variable. Valid uses dictInterface to check if the series of letters is a word, prefix, both, or neither. Finally, the method Solution prints out the resulting board. This was the plan. However, it seems that my indexes keep going out of bounds. Because my program does not work, I was not able to collect program run times. However, I suspect that the times for the De La Briandais implementation would result in faster puzzle completions because of its faster search.

The first step I took was to write code that would allow me to read the file containing the board and the file with all of the valid words. I used a two dimensional character array to store the crossword board that was read. Furthermore, the dictInterface can be implemented in two ways. The first method is the MyDictionary implementation. MyDictionary is based off the data structure ArrayList. The add method: adds a new string to the end of list. If a string is out of order, the list is sorted. The search method: iterates through an ArrayList, until the end or until the key is passed up. When the key is passed up, iterate through the key and the current string in the ArrayList character by character. Then, check if the string had ended in the ArrayList, stop using the string. The benefits of a sorted ArrayList leads to an decrease in the runtime of the *search* method from Theta(N) to Theta(lgN). However, the amount of space taken up by this data structure is equal to the number of words contained in the ArrayList. An improvement on the MyDictionary implementation would be a decrease in space.

The second method is the De La Briandias Tree (dlb) implementation. In a dlb implementation, the data structure is a node is a linked list of nodes. The advantage of a dlb is that it can save much space compared to MyDictionary. The search method: start from the first character of the given string. Sequentially search the linked list of sibling nodes pointed by the root. If the character is found, follow the child link and repeat the same procedure with the next character in the string and the child’s linked list. If the search fails before reaching the end of the string, the string is neither a word nor a prefix. If the last character in the string is reached, continue search its child’s link list. If a string terminator is found, the search string is contained in the DLB as a word. If some other characters are also found, it is contained in the DLB and also it is a prefix of some strings. If only other characters are found and the string terminator is not found, it is a prefix of some strings contained in the DLB. In the worst case, the search would take the program through every level of the tree. Therefore, the worst case is bound by the tree level. In conclusion, the various implementations of how a word is searched and verified had their advantages and disadvantages.