Sushruti Bansod

CS 1501: Algorithm Implementation

10/28/2020

[SDB88@pitt.edu](mailto:SDB88@pitt.edu)

Assignment 1: Crossword and DLB

This assignment required me to develop a program that generates solutions to crossword puzzles of different sizes. There were multiple files provided to test and check these permutations against words in a provided in a dictionary file. The file that provides an explanation made it a lot easier to tackle the project that first seemed extremely difficult. The additional help that was provided in class such as the notes, class questions and specific pseudo-code was vital in being able to do this Assignment. However, I spent a lot of time doing each task and I even had to start from scratch a few times in order to organize my thoughts and my code. Although there are incomplete aspects of the project, my final solution is pretty good but I believe that it can be implemented better. Through this paper, I will attempt to deliver some understanding to my approach to the problem and the implementation choices I made in building the algorithm.

When getting started with Crossword.java, I started by implementing the code that read in the file name. Then I added all the words from that to a Dict object of type DictInterface. I then attacked the StringBuilders. I had used them before but not in terns of an array. When making this array of StringBuilders, it was really helpful in drawing out my thoughts and writing pseudocode. One issue I faced was a Null Pointer exception which I fixed by appending empty strings into each StringBuilder. The solve method is where most of the logic would lie so I was extremely wary. The path I took was to check for each different case and then address it. The first case is the plus sign which signifies that the coordinate in available. To make sure it was correct, I called the isValid method, which I will address later in this essay. If the coordinate was valid, I appended a letter to the string and checked if it’s a possible addition. Within this I also had to make sure that I was not stepping off the board. If this combination is impossible, I remembered to delete the char from the StringBuilders. Overall, the runtime for just the “+” case looks like O(n2), due to the two for loops.

This is when I started thinking about the hardest case. The program must be able to detect a minus sign and use that to make sure the words around it are valid. This was the hardest part of this assignment and I was not able to achieve it correctly. The code that I thought was correct is commented out for checking. My approach was divided between the solve method and the isValid method. This includes calling the search prefix method with the next row/column in order to make sure that the word after the minus sign is valid. The question arose as to how to check before the minus sign and what to do in case of multiple minus signs in a row/column. Even with the hint of using two arrays to keep track of the coordinates of the minus sign, I was not able to complete this section of the project. My main method includes populating the array with locations of the minus signs, but I was not sure how to implement this in my solve or isValid method.

At a point, I moved on to the last case. I checked for a preset character in the board, which had a relatively simple implementation that required the isValid method once again. Speaking of the isValid method, it was vital to get an accurate algorithm. The purpose was to make sure the coordinate was not illegal which was done using a series of if statements. Within all these if statements, the searchPrefix method was called which returned an integer indicating wither the suggested string was a prefix or a word. The runtime for this method is relatively constant as there are no loops or recursive calls. Generally, this implementation had a longer runtime and was not very practical.

The next task was to create a DLB for a more practical traversal and better runtimes. I used a lot of a previous lab to implement this. The add method was a simple algorithm that included checking if a letter existed or adding it as a sibling or a child. I was able to implement this with a relatively small runtime since there are not loops but a few recursive calls. There were 2 StringBuilder methods that needed to be implemented according to the interface. I decided to view one of the searchPrefix methods a ‘helper’ method for the other one in order to keep the code short. I did this by iterating over the given string and checking the DLB for each char. This included checking all the siblings. At one time, I contemplated writing another helper method called findSibling to make it easier. However, it wasn’t completely necessary, and I just spaced my code so I could stay organized.

Aside from the minus case from task 1, I was able to create an algorithm that successfully solved the problem. If I had more time to work on it, I would attempt to find the solution to the minus case as well as improve the general logic and runtime used in the second task. Below is a table of approximate runtimes from the 2 different implementations.

|  |  |  |
| --- | --- | --- |
|  | Task 1 | Task 2 |
| Public add(String s) | O(n) | O(n) |
| Public searchPrefix(StringBuilder s) | O(n2) | O(n) |
| Public searchPrefix(StringBuilders, int start, int end) | O(n2) | O(n) |
| Public solve(int row, int col) | O(n2) | O(n2) |
| Private isValid(int row, int col, char c) | O(n2) | O(n) |