Fall 13

Assignment 7 – Crossword Puzzle Solution Generator

Richard Fox

Notes:
I did the Extra Credit GUI application.

Introduction

Below is an explanation and analysis how I chose to implement my CWSolution class. It is crucial to first understand what exactly the problem is and what we are trying to solve before approaching the question: how should we solve it?

Problem

1. We are given a list of words that will constitute as our dictionary and we need to load these words into a data structure as quickly as possible.

2. We are given a pattern consisting of *'s and letters. From that pattern we need to return a list of words that match the pattern in the quickest time possible.

Comments:

From a technical standpoint this means we are looking for a structure that will allow us to add elements the quickest possible and then iterate over the elements as quickly as possible.

Solution

When we load the words into our data structure, we need to consider the fact that we will be searching based around the length of the pattern given, so we will create around 12 different "bins" that correspond to a length of the words within the bins. For example the word "APPLE" would go in bin 5 because it has 5 letters. The reason for this is that when we are given a pattern of "**S" we know we can immediately disregard any words that are not of length 3 and we should only search in the bin that has 3 letter words.

To further explain the searching logic, once we access the correct bin based on the length of the pattern given, we will then need to iterate through every single item of that bin. Because the *s represent wild card characters, we do not need to consider them when we are trying to find matches against the words in the dictionary. So, for the pattern "**S" , we are only checking if the character at index 2 (remember String index start at 0), if it is equal to "S".

In this way we can drastically reduce the time of finding a match by not iterating through every letter in a word.

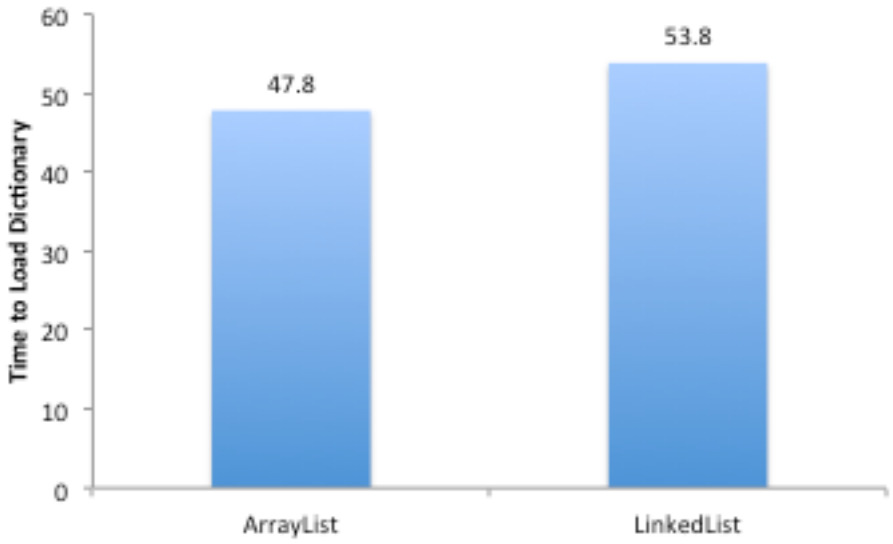
Now that we know the way in which we will approach this problem from a logic standpoint, the question is: What structure should we use when adding and iterating through the words in our dictionary?

Analysis of Data Structures

In my opinion there are really only two data structures that should be considered when approaching such a problem: LinkedLists or ArrayLists. The reason for that is we only care about two functions, which are insertion and iteration, and these are the only two data structures that are even in the realm of possibilities of performing the best

Insertion

With respect to insertion we found the average time it takes to load the entire dictionary into bins that were backed by LinkedLists and bins that were backed by ArrayLists.



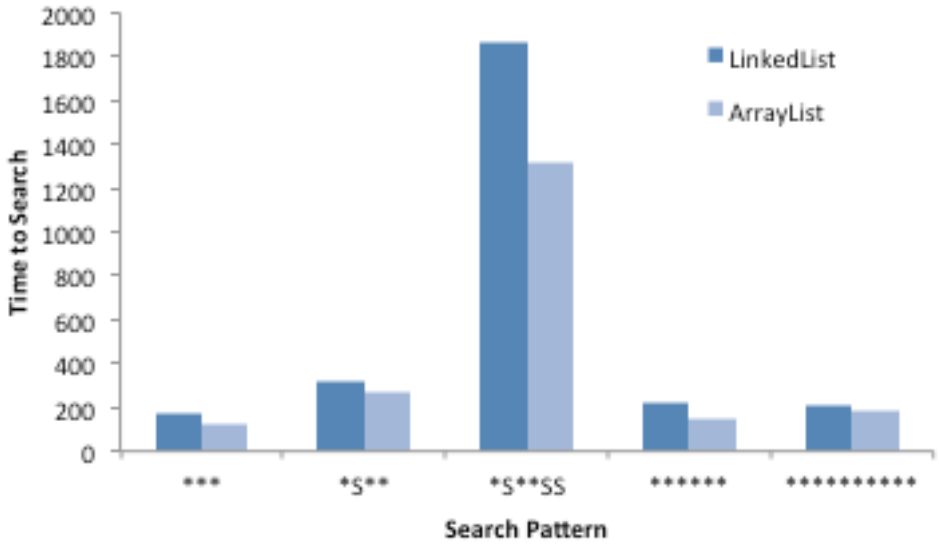
As you can see from the chart above, the average time it took to load the dictionary into ArrayList backed bins, was only slightly faster than the LinkedList backed bins. I believe the reason that they are essentially the same, is

due to the fact that both have a insertion time of O(1) and that the ArrayList slightly outperforms the LinkedList simply because actual implementation speed is faster than that of the Linkedlist regardless of the theoretical Big-O complexity.

So for insertion, it is basically a toss-up, although slightly leaning towards the ArrayList. However, we will see that the iteration speed, is by far the most important and the most differential metric between the two data structures.

Iteration

We took 5 different search patterns and compared the aggregate speed of finding the solution 1000 times with ArayList backed bins and LinkedList backed bins.



As you can see the ArrayList outperforms the LinkedList in iteration, which is expected. The ArrayList is backed by a contiguous array in memory while the LinkedList is backed by references in memory, which on a lower level means it takes a longer to go from one element to the next.

Conclusion

From the above analysis it became clear that the ArrayList was by far the best data structure to use in this case since it outperformed the LinkedList in

every respect that we were concerned with. The final statistics as compared to the benchmark metrics are below.