In assignment two, our tasks were to implement algorithms for DLB insertion and word score maintenance, and to implement algorithms for retrieving word suggestions in order to create a simple automatic word-completion system. Autocomplete is a common feature that is present in many of our devices such as phones and search engines. When the user starts to type a few letters, the autocomplete feature displays a list of word suggestions to help the user complete the word they are typing. In the assignment, we used a DLB trie to create the autocomplete system. In the first part, I created a private helper recursive method to add words to the trie. The add method passes in a DLBNode, the stringbuilder object, word, and the position. In the add method, if the node is null, then I created a new DLBNode at that position, and if the position is less than the word’s length minus one, then I recursed on the child node because it is not a word yet. The position is pos + 1 because the child is placed at the next spot down the trie. If the position is equal or greater than the word’s length minus one, then it is a word. Furthermore, I checked if the letter is equal to the letter at that position, then I recursed on the child node if the position is less that the word’s length minus one, and if it equal to or greater than then it is a word. Lastly, if the letter does not equal the letter at the position, then I recursed on the sibling node. In the next method, notifyWordSelected, I obtained the last node by calling the getNode method and checked if it is not null. If it is not null, then I incremented the score. The score keeps track of the number of times a word has been selected as a suggestion, which shows the words that are more commonly used first in the list. The next method, getScore, retrieves the score. For that, I obtained the last node and returned zero if the node is null, otherwise I returned the score. For the last method, retreiveWords, we had to retrieve all the words that have the same prefix from the DLB trie. I created a helper method, collect, that finds all the words that have the given stringbuilder as a prefix. The method checks if the current node is not null, and if is a word, then it adds the word to the queue. It checks the same for the child node and sibling node, and it recursively calls collect on each. In the main retrieveWords method, I got the last node and checked if it is not null and a word, which means it is a prefix, then added it to the list. Since the list had to be in descending order based off the scores, in the helper suggestions class, I created a compareTo method that compared each of the scores, and listed the autocomplete words in that order. A certain issue I faced was that when I ran the program, I kept receiving a list of only “a” in the output. This was occurring because when I was checking if it is a word in the collect method, I was not creating a new Stringbuilder inside new Suggestion. Since there was no new Stringbuilder, it was only printing the old word over and over again. The worst-case run-time for the insertion, score incrementing and score retrieval is theta(w\*R). W is the number of characters in a word and/or prefix, and R is the alphabet size. To insert the nodes into the tree, and to increment the score and retrieve the score of the word, it is required that you go through each and every element. The runtime for suggestions retrieval is nw +nlogn. It’s plus nlogn because it places the words in random and then collection.sort is called to put them in order and then into a new trie.