Assignment 1

Given the task of creating a DLB for assignment one, I decided to set up my DLB using a linked list inside of a class called “Nodelet”. Inside of this class, I initiated 4 variables. I needed a char to hold the letter of a given word, a Nodelet called child to indicate that letter formed a word or was a prefix to another word, a Nodelet called sibling for words containing different letters than the letters that are currently being used, and finally a Boolean called “isWord” to tell whether that Nodelet formed a complete and full word. I also made a global variable called root from the Nodelet class to initiate my DLB. Starting my program off, I knew when adding letters to my DLB that I had to handle three cases. First of which was if the DLB was completely empty, in which I would need to set the start, or root, of my DLB to the first letter of a passed in word. The second case was that if the new word being passed in started with the same letter as the root Nodelet char, in which I would need to check the letters in the word after to see how many of them match with the letters previously incorporated into the DLB, and when I get to a letter that isn’t there I would need to branch the DLB out to compensate by making a sibling Nodelet. The final case was if the root did not contain the letter of the newly passed in word, in which I would have to check the siblings of the root, or if there were no siblings, make a sibling. After every insertion of a letter, I would check in each of these cases if the variable in my loop adding the letters was equal to that of the length of the string minus one, because if it was that would indicate that I had a full word and would need to set my Boolean value to true. During this process, I had to initialize a new Nodelet called “parent” to keep track of the node at which my current node was a child of, because my initial case one checked if curr was equal to null, which means I could re-use this case for my other cases if I used this reference to parent. One major issue I came into contact with was when I was setting the Boolean value to true if the variable in the loop equaled the string’s length minus one. My problem was that I was always setting the “curr” Node to true, and in case three I actually had to set the current Nodelet’s sibling to true, because the third case is adding a sibling if the letter didn’t previously exist, and if I reach the end of the string that means that the current Nodelet’s sibling is a word, and not the current Nodelet. I resolved this by testing my DLB with the given test file in the code folder called “DictTest” and realized that my DLB wasn’t correctly assigning some of the ending letters to words, rather they were being set to only prefixes of words. A way I debugged and tested this even further was by putting a print statement inside my DLB class, that printed my DLB by starting at the root and printing that and all of its children, then moving to the siblings and printing their children. The run time differences between using MyDictionary and the DLB were similar, which is surprising considering MyDictionary is only finding one single solution, while the DLB is finding anywhere from zero to over a million solutions. Some solutions from MyDictionary were faster, such as test3a. I believe this to be because it is only finding one solution and has all spaces available. In other tests, such as test4e, the DLB was extraordinarily faster, and I believe this to be because of how the letters are being stored versus how they are being stored in MyDictionary. To elaborate, whenever MyDictionary determines whether it is a valid word, it uses a method called searchPrefix. Inside of this method, we take the string and determine whether it is a word, prefix, both, or none. To do so, we have to search through the array list, and inside of this array list we stored every single word from the dictionary. The reason this is slower is because in the searchPrefix in the DLB implementation, the words may share a same “stem”, or in better words, there is zero unnecessary repeating of letters. Traversal through the DLB will be faster for this reason, as we can easily navigate to the correct letter then search through the children and siblings to find the correct word.

|  |  |  |
| --- | --- | --- |
| Test # | MyDictionary (seconds) | DLB (seconds) |
| 3a | 0.94 | 5.79 |
| 3b | 1.87 | 0.46 |
| 4a | 4.80 |  |
| 4b | 4.28 |  |
| 4c | 0.66 | 6.37 |
| 4d | 0.63 | 0.50 |
| 4e |  | 5.75 |
| 4f | 5.33 | 0.65 |
| 5a | 3.69 |  |
| 6a |  |  |
| 6b |  |  |
| 6c |  | 44.54 |
| 7a |  |  |
| 8a |  |  |
| 8b |  |  |
| 8c |  | 5.53 |

