Our focus this time is on deletion of ALV tree and some operations of trie. For deletion of trie, during our discussion , two possible methods may be feasible .

The first one is lazy deletion , which are used by us this time. As we can see, it has the advantage of quick deletion with only a find operation and setting the variable edw value 0, and it can make further insertion easier if the first few letters of the new word have been inserted into the trie.

However, the space wasted by this method can be horrible because we can never delete a node of no use that has been inserted. One way to solve the problem is using a variable count to store the times of a node passed by words totally. Compare the space of the variable count, which is 2 BYTE , to the space of a node, it can save space theoretically. However , in practical use , only a few nodes may need to be deleted completely. Each time we renew the trie, we need to renew the count of the nodes and it can cost O(N). And the cost of some further insertion may be large, so it’s hard to determine whether this method works or not. So finally, we decided to use lazy deletion in consideration that it is easy .

Deletion of AVL tree consists of deletion of a leaf node and rotation, which saves the space while the time complexity is not very large.

Finally , during testing, we found that some strings generated randomly are the same. Most of them has the length of 1 or 2. It can hardly influence the testing results so we ignored this problem.

Our testing algorithm can test the running time of insertion, search, and deletion of both trie and AVL tree so we can find the relationship between time complexity and each factor.

Some other possible improvements . For example, we used a lot of arrays in our structure , so it can be improved by using list to save space with more time complexity. In practice works, we need to use array and list appropriately to balance time complexity and space complexity.

During our testing, we only insert all strings and delete them, so the result may be of much difference with that in practical use. Due to our bad programming skill, the routine can , of course , be improved by balance the complexity of algorithm and time complexity.