1. **Implementation**

**With reading file**, read line by line, then add new word has been read into tree, the tree balanced by itself.

**Rotate left and right:** implemented following rotating algorithms.

**In case of look up a word**, recursively traversing to left tree if the keyword is smaller than root, to the right tree if the keyword is bigger than the root, else, print out the meaning of this key. If reach the leaf node and found nothing print the notification that not found keyword in tree.

**In case of adding new word**, if new word is existed, do nothing. If the tree is empty, create new tree and the new word as the root. If the keyword is smaller than current root, recursively add into the left tree, if adding makes tree unbalanced, rotate current tree to the right. Else, recursively add into right tree, if unbalanced, rotate left.

**In case of edit meaning of a word**, just search the position of the given word, if existed then I replace the old mean with new mean. Then, editing costs no time but searching for its position.

**In case of deleting a word**, recursively traverse the tree and look for the keyword, if existed, set the value to “” .

**In case of saving**, traverse the the tree and output all *not “” keyword (cause of deleting word)* following LNR order.

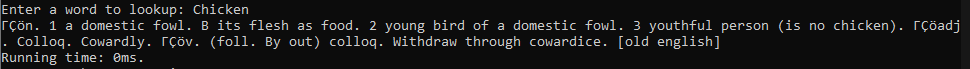
1. **Experiments**

**File reading:**

****

****

**Look up a word:**



Text

Description automatically generated

**Editing word:**

Graphical user interface, text

Description automatically generated

Then, new mean:

Graphical user interface, text

Description automatically generated

**Adding word:**

**Text

Description automatically generated**

Text

Description automatically generated

**Deleting word:**



Text

Description automatically generated

**Saving to file:**  
Text

Description automatically generated

1. **Time complexities**

I don’t put Edit bellow because Edit just Search and replace values, nothing different in time complexities.

Bellow are average time complexities:

|  |  |  |  |
| --- | --- | --- | --- |
| **Data stucture** | **Search** | **Insert** | **Delete** |
| Array | O(n) | O(n) | O(n) |
| Hash table | O(1) | O(1) | O(1) |
| BST | O(log n) | O(log n) | O(log n) |
| AVL tree | O(log n) | O(log n) | O(log n) |

1. **Analyze**

Strong points:

* The height of tree is always balanced.
* Better search time.
* Self-balancing capabilities.

Weakness:

* Hard to implement.
* High cost due to frequently rotating.

1. **Table and graph**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Time | Array | Hash Table | BST | AVL Tree |
| Load | 200 | 400 | 350 | 330 |
| Sort | 160 | - | - | - |
| Save | 400 | 212 | 311 | 260 |
| Look up | 0 | 0 | 0 | 0 |
| Insert | 14 | 1 | 7 | 0 |
| Remove | 11 | 0 | 3 | 0 |
| Edit | 0 | 0 | 0 | 0 |

1. **Guide to use**

I create a menu that allows user to use my program easier. But there is something I must remind:

File name inputted must has .txt at the end.

While using menu, the inputted choice value must an integer.