ICS202: Data Structure & Algorithms

LAB Project Report

Dictionary Data Structure

Name: Ismael Arqsosi

Section: 56

Date: 9/12/2023

(1) Introduction:

The lab project involves designing and implementing a dictionary data structure for use in a spell checker. The chosen data structure is an AVL tree, a self-balancing binary search tree. The design decision to use AVL trees ensures efficient search, insertion, and deletion operations. The motivation behind this choice is to maintain a balanced tree structure, reducing the risk of performance degradation commonly associated with unbalanced binary search trees and other inefficient data structures.

(2) Problem-solving strategy:

- **Initialization (Loading):**

Input: String, empty dictionary, or a text file.

Pseudo-code:

```
if input is a string:
  create an AVL tree with a single node containing the string.
else if input is an empty dictionary:
  create an empty AVL tree.
else if input is a text file:
  read each line from the file and insert it into the AVL tree.
```

- **Add new word:**

Input: String.

Pseudo-code:

```
try to insert the word into the AVL tree.
if successful:
  word added successfully.
else:
  throw WordAlreadyExistsException
```

- **Search for word:**

Input: String.

Pseudo-code:

```
if AVL tree contains the word:
  return true.
else:
  return false.
```

- **Remove word:**

Input: String.

Pseudo-code:

```
if AVL tree contains the word:
  delete the word from the AVL tree.
  word deleted successfully.
else:
  throw WordNotFoundException
```

- **Search for similar words:**

Input: String.

Pseudo-code:

```
initialize a list to store similar words.
for each word in the AVL tree:
  if the word is similar to the input:
    add it to the list.
return the list of similar words.
```

This also involves the following:

1. **findSimilar Method:**

Create an empty list to store similar words.

Use a helper method to traverse the AVL tree and identify similar words.

Convert the list of similar words to an array for easy handling.

Return the array of similar words.

2. **findSimilarHelper Method:**

Check if the current node is null. If so, stop the search.

If the word at the current node is similar to the input key, add it to the list.

Recursively search the left and right subtrees of the current node.

Continue the process until all nodes have been examined.

3. **isSimilar Method:**

Check if the length difference between the two words is more than 1. If so, they are not similar.

Initialize variables to track the degree of differences in letters.

Compare letters of the two words, accounting for differences in length.

If more than one difference is encountered, the words are not similar.

Return true if the difference is exactly 1, indicating similarity.

- **Save file:**

Input: String (file name).

Pseudo-code:

```
create a new file with the given name.
open the file for writing.
traverse the AVL tree and write each word to the file.
close the file.
print "Dictionary saved successfully."
```

(3) Test results (screenshots):

Searching:

```
Enter file name: mydictionary.txt
Dictionary loaded successfully

 Operations list:
1) Search for Word
2) Add Element to the dictionary
3) Delete Element from the dictionary
4) Search for similar words
5) Save changes to new file
6) Print Options
7) Terminate

Choose an operation you want to perform: 1
Search for:
a
Word has been found successfully

Choose an operation you want to perform: 1
Search for: siuuuuuuu
Word not found!

Choose an operation you want to perform: 1
Search for:
```

Add Word:

```
Enter file name: mydictionary.txt
Dictionary loaded successfully

 Operations list:
1) Search for Word
2) Add Element to the dictionary
3) Delete Element from the dictionary
4) Search for similar words
5) Save changes to new file
6) Print Options
7) Terminate

Choose an operation you want to perform: 2
Add the word: a
Word already exist

Choose an operation you want to perform: 2
Add the word: siuuuuuuuuuuuuuuuuuuuuuuuuu
Word has been added successfully

Choose an operation you want to perform: 2
Add the word:
```

Delete Word:

```
Enter file name: mydictionary.txt
Dictionary loaded successfully

 Operations list:
1) Search for Word
2) Add Element to the dictionary
3) Delete Element from the dictionary
4) Search for similar words
5) Save changes to new file
6) Print Options
7) Terminate

Choose an operation you want to perform: 3
Remove the word: a
Word has been deleted successfully

Choose an operation you want to perform: 3
Remove the word: a
Failed in deleting the word because the word not found

Choose an operation you want to perform: 3
Remove the word:
```

Find Similar:

```
Enter file name: mydictionary.txt
Dictionary loaded successfully

 Operations list:
1) Search for Word
2) Add Element to the dictionary
3) Delete Element from the dictionary
4) Search for similar words
5) Save changes to new file
6) Print Options
7) Terminate

Choose an operation you want to perform: 4
Search for: a
Similar words are:
[ z , y , x , wa , w , va , v , u , ta , t , sa , s , r , q , pa , p , o , n , m , ma , la , l , k , j , ia , i , h , ha , ga , g , f ,
e , d , c , ca , b , az , ax , at , as , ar , an , am , al , ak , ah , ad , ac ]
Choose an operation you want to perform: 4
Search for: siuuuuuuuuuuuuuuuuuuuu
Similar words are:
[  ]
Choose an operation you want to perform: 4
Search for: 1
Similar words are:
[ z , y , x , w , v , u , t , s , r , q , p , o , n , m , l , k , j , i , h , g , f , e , d , c , b , a ]
Choose an operation you want to perform:
```


Save Changes:

```
Enter file name: mydictionary.txt
Dictionary loaded successfully

 Operations list:
1) Search for Word
2) Add Element to the dictionary
3) Delete Element from the dictionary
4) Search for similar words
5) Save changes to new file
6) Print Options
7) Terminate

Choose an operation you want to perform: 5
Save Updated Dictionary (Y/N): N
Choose an operation you want to perform: 5
Save Updated Dictionary (Y/N): Y
Enter file name: UpdatedMyDictionary
Dictionary saved successfully
```

Print Options:

```
Enter file name: mydictionary.txt
Dictionary loaded successfully

 Operations list:
1) Search for Word
2) Add Element to the dictionary
3) Delete Element from the dictionary
4) Search for similar words
5) Save changes to new file
6) Print Options
7) Terminate

Choose an operation you want to perform: 6

 Operations list:
1) Search for Word
2) Add Element to the dictionary
3) Delete Element from the dictionary
4) Search for similar words
5) Save changes to new file
6) Print Options
7) Terminate

Choose an operation you want to perform:
```

End Program:

```
 Operations list:
1) Search for Word
2) Add Element to the dictionary
3) Delete Element from the dictionary
4) Search for similar words
5) Save changes to new file
6) Print Options
7) Terminate

Choose an operation you want to perform: 7
Ends of the program
PS C:\Users\Lenovo\Desktop\ICS202_Project>
```

(4) Challenges faced and how they were overcome:

Challenge1: **Implementing similarity checking efficiently.**

Solution: The “isSimilar” was the biggest challenge in the project. After trying to solve it for 2 days with different strategies, it did not work. Once we started the “String Matching” chapter in the lecture, it gave me a small idea to implement it using a similar way, which finally worked. The way is explained in the code as comments. The method was carefully designed to handle length differences and compare letters, ensuring accuracy and efficiency in finding similar words.

Challenge: **Error handling for invalid input or file-related issues.**

Solution: Try-catch blocks were used to catch exceptions and provide meaningful error messages to the user. For example, catching “IllegalArgumentException” for duplicate word insertion and “IOException” for file-related issues during initialization.

Challenge: **Class Integration.**

Solution**:** an issue of how to create an object of AVL inside Dictionary. It is easy but with multiple classes together it was a little bit confusing at the beginning, but after trying multiple times and asking questions it worked.

Challenge: **“findSimilar” Implementation.**

Solution: Designing an algorithm for the “findSimilar” method, particularly the isSimilar helper method, involves handling various cases efficiently. Rigorous testing and refinement are crucial to ensure accurate results for different scenarios, such as varying word lengths and subtle character differences.