ICS-202 Project Report

Mohammed Busaleh - 202158210

Introduction:

The project simply deals with words from the user to store and traverse on multiple ways and for different purposes.

For this project I decided to go with AVL as the data structure to store the words. The main motivation behind this choice is due to low cost in searching, adding, and deleting. Which are the most used functions in this type of problems.

- Linked list will have a very high cost foreach of these tasks.
- Binary tree will have the problem of randomness in storage.
- BST is good generally but will not function well when the words are in alphabetical order, it will cause an unbalanced skewed tree. Which will result in a complexity similar to that of the linked list.

The need to store the data in some sort of order with the need to balance makes AVL the best choice.

The class has three constructors and five methods, most of them are the usual AVL tree methods such as insert, delete, find, and traverse (used the inorder travers for alphabetical order).

This decision is because we don't need to re-invent the wheel, the typical AVL methods works in very efficient way and with simple edits will fit our project perfectly.

Problem solving strategies:

1. **Constructor Operations (`Dictionary()`):**
   - Initialize a new AVLTree to store words.
   - Print a success message.

2. **Constructor Operations (`Dictionary(String s)`):**
   - Initialize a new AVLTree
   - insert the provided word (converted to lowercase) into the tree.
   - Print a success message.

3. **Constructor Operations (`Dictionary(File infile)`):**
   - Initialize a new AVLTree.
   - Try opening the specified file. If you can't throw an exception.
   - Read each word, convert it to lowercase, and insert it into the AVLTree.
   - Print success messages.

4. **Method: `addWord(String s)`:**
   - Convert the input word to lowercase.
   - Check if the word is already in the AVLTree.
   - If not, insert the word into the AVLTree using the regular insertAVL function and print a success message.
   - If the word already exists, print a message indicating that the word is already in the dictionary.

5. **Method: `findWord(String s)`:**
   - Check if the given word (converted to lowercase) exists in the AVLTree.
   - Return true if found, false otherwise.

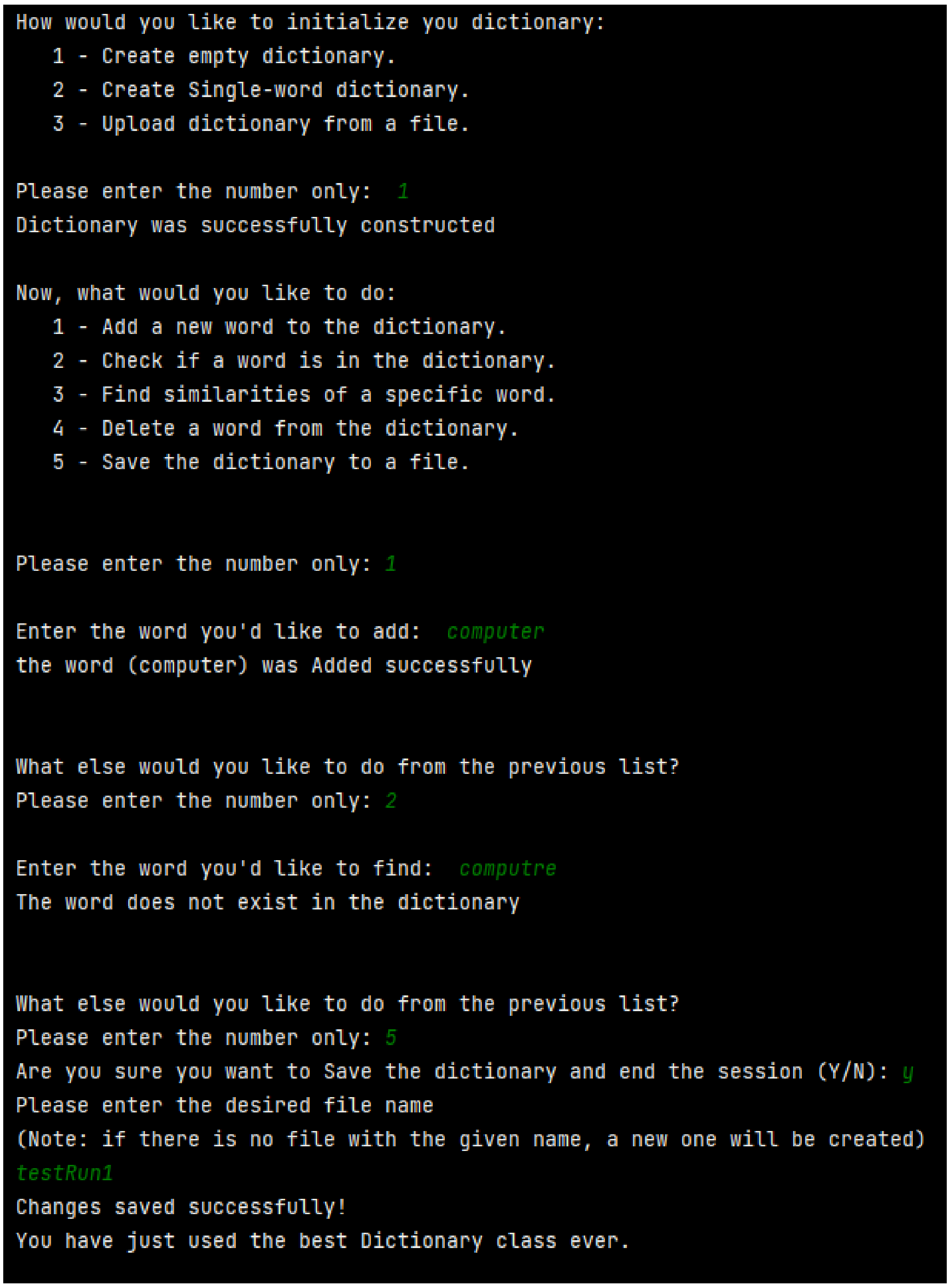
6. **Method: `deleteWord(String s)`:**
   - Delete the specified word (converted to lowercase) from the AVLTree using a regular deleteAVL function.

7. **Method: `findSimilar(String s)`:**
   - Convert the input word to lowercase.
   - Iterate over all words in the AVLTree (inorder traversal).
   - Compare each word with the input word and identify similar words based on a letter difference criterion (<= 1).
   - Return a list of similar words.

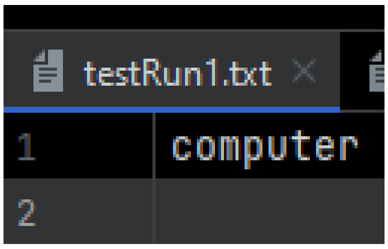
8. **Method: `writeToFile(String s)`:**
   - Check if the given filename contains an extension; if not, add ".txt".
   - Create a PrintWriter to write to the specified file or create a new file.
   - Get all words from the AVLTree (inorder traversal) and write them to the file.
   - Close the PrintWriter.
   - Print a success message.

Test Runs:

Test Run 01:

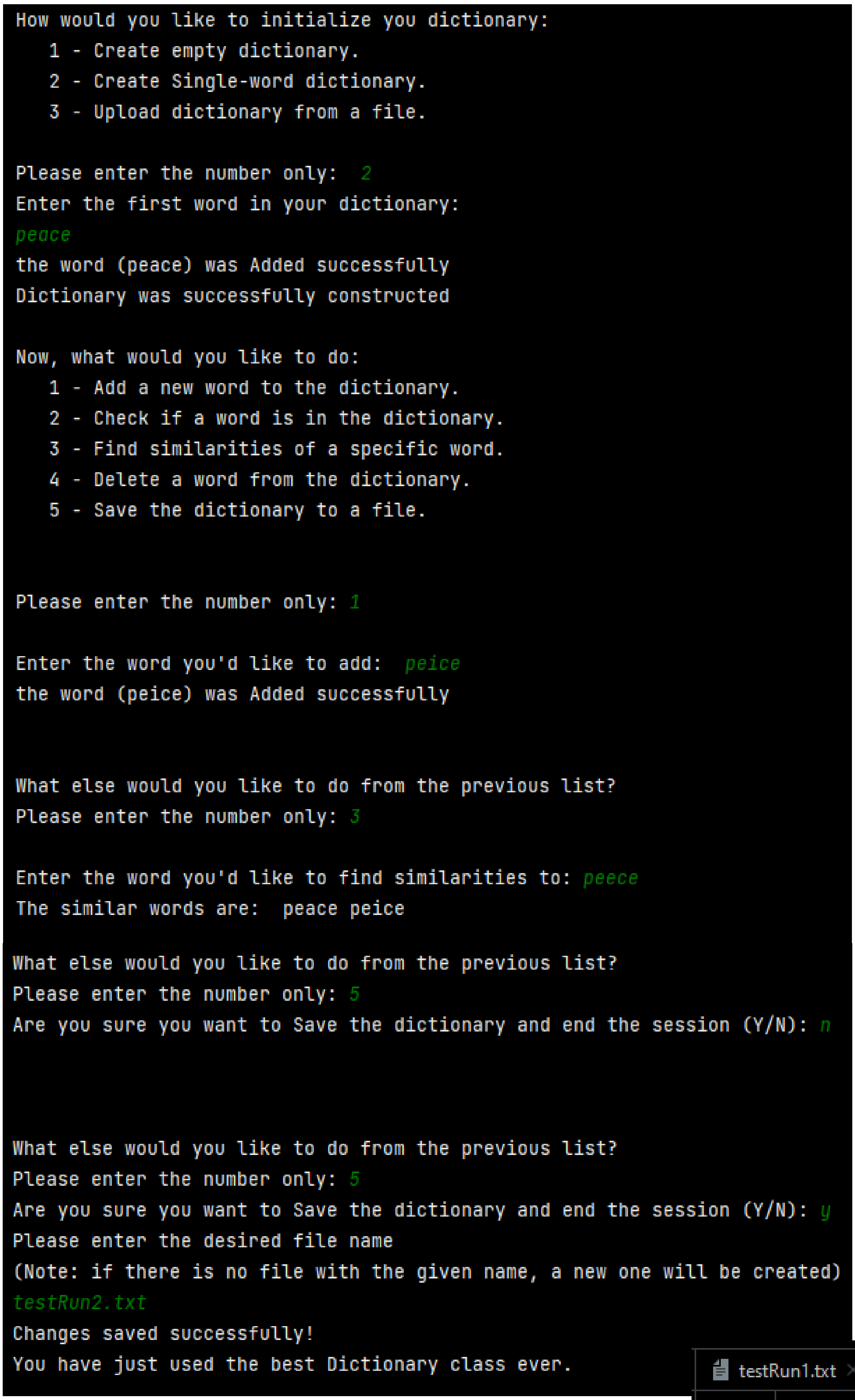


```
How would you like to initialize you dictionary:
   1 - Create empty dictionary.
   2 - Create Single-word dictionary.
   3 - Upload dictionary from a file.

Please enter the number only:  1
Dictionary was successfully constructed

Now, what would you like to do:
   1 - Add a new word to the dictionary.
   2 - Check if a word is in the dictionary.
   3 - Find similarities of a specific word.
   4 - Delete a word from the dictionary.
   5 - Save the dictionary to a file.


Please enter the number only: 1

Enter the word you'd like to add:  computer
the word (computer) was Added successfully


What else would you like to do from the previous list?
Please enter the number only: 2

Enter the word you'd like to find:  computre
The word does not exist in the dictionary


What else would you like to do from the previous list?
Please enter the number only: 5
Are you sure you want to Save the dictionary and end the session (Y/N): y
Please enter the desired file name
(Note: if there is no file with the given name, a new one will be created)
testRun1
Changes saved successfully!
You have just used the best Dictionary class ever.
```

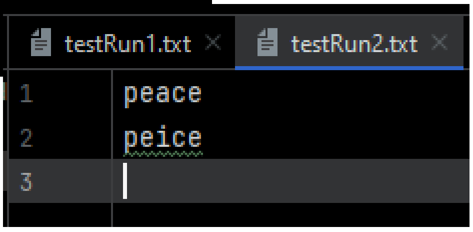


```
testRun1.txt

computer

```

Test Run 02:

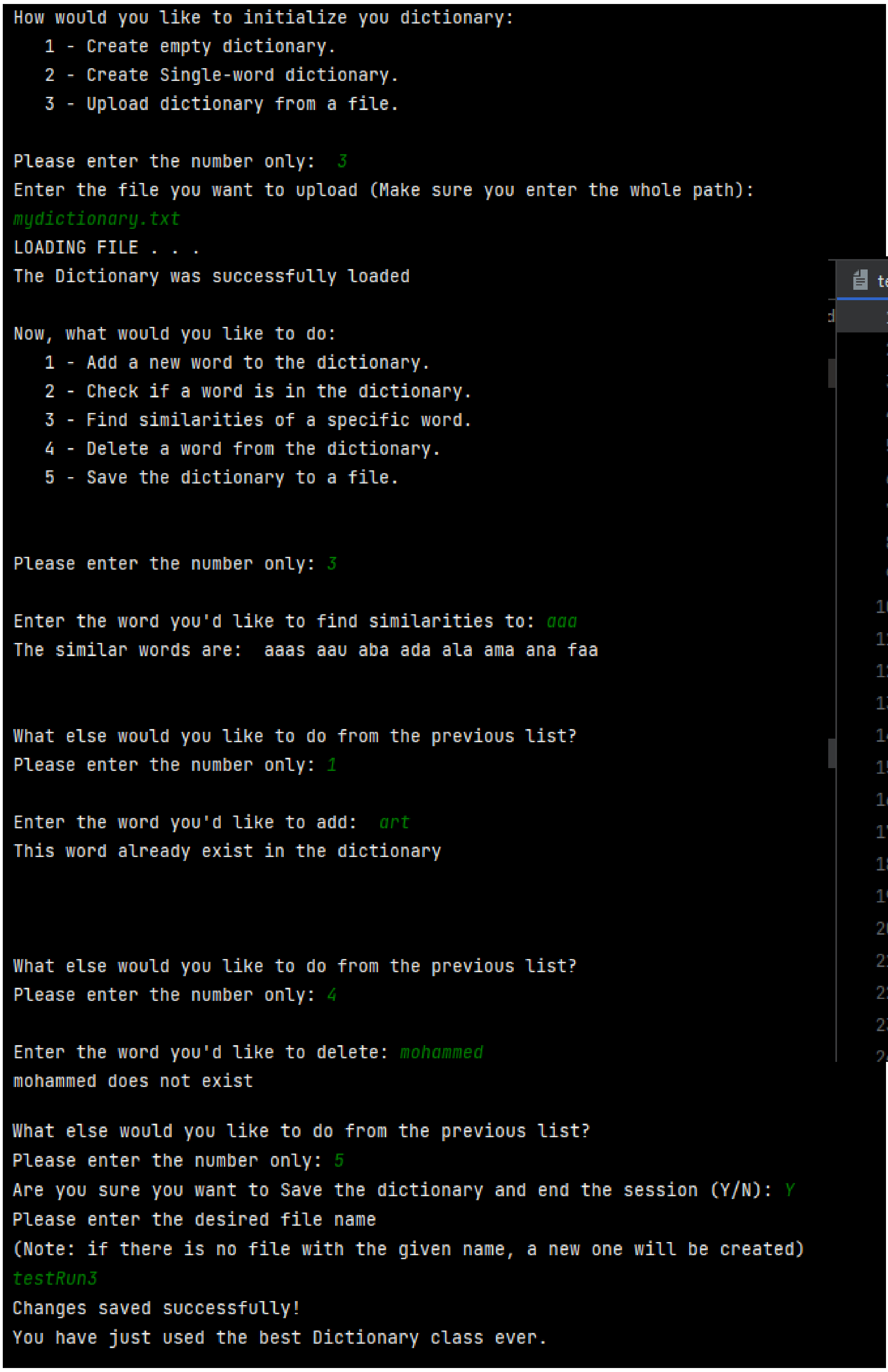


```
How would you like to initialize you dictionary:
   1 - Create empty dictionary.
   2 - Create Single-word dictionary.
   3 - Upload dictionary from a file.

Please enter the number only:  2
Enter the first word in your dictionary:
peace
the word (peace) was Added successfully
Dictionary was successfully constructed

Now, what would you like to do:
   1 - Add a new word to the dictionary.
   2 - Check if a word is in the dictionary.
   3 - Find similarities of a specific word.
   4 - Delete a word from the dictionary.
   5 - Save the dictionary to a file.


Please enter the number only: 1

Enter the word you'd like to add:  peice
the word (peice) was Added successfully


What else would you like to do from the previous list?
Please enter the number only: 3

Enter the word you'd like to find similarities to: peece
The similar words are:  peace peice

What else would you like to do from the previous list?
Please enter the number only: 5
Are you sure you want to Save the dictionary and end the session (Y/N): n



What else would you like to do from the previous list?
Please enter the number only: 5
Are you sure you want to Save the dictionary and end the session (Y/N): y
Please enter the desired file name
(Note: if there is no file with the given name, a new one will be created)
testRun2.txt
Changes saved successfully!
You have just used the best Dictionary class ever.
```

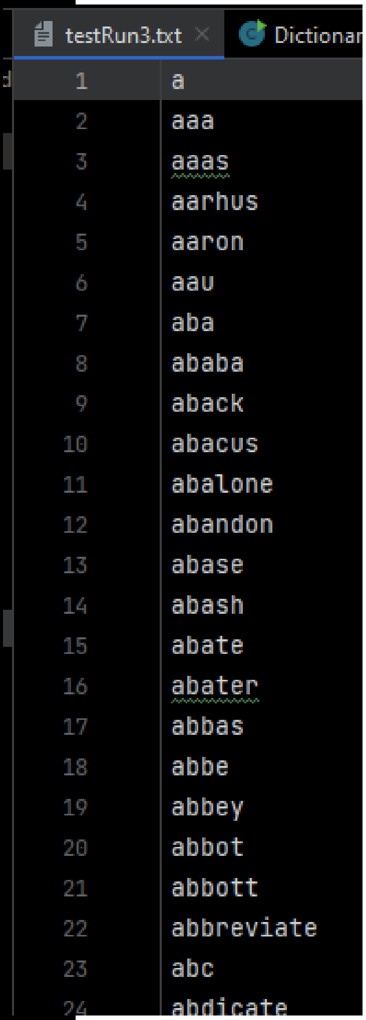


```
peace
peice
```

Test Run 03:



```
How would you like to initialize you dictionary:
   1 - Create empty dictionary.
   2 - Create Single-word dictionary.
   3 - Upload dictionary from a file.

Please enter the number only:  3
Enter the file you want to upload (Make sure you enter the whole path):
mydictionary.txt
LOADING FILE . . .
The Dictionary was successfully loaded

Now, what would you like to do:
   1 - Add a new word to the dictionary.
   2 - Check if a word is in the dictionary.
   3 - Find similarities of a specific word.
   4 - Delete a word from the dictionary.
   5 - Save the dictionary to a file.


Please enter the number only: 3

Enter the word you'd like to find similarities to: aaa
The similar words are:  aaas aau aba ada ala ama ana faa


What else would you like to do from the previous list?
Please enter the number only: 1

Enter the word you'd like to add:  art
This word already exist in the dictionary




What else would you like to do from the previous list?
Please enter the number only: 4

Enter the word you'd like to delete: mohammed
mohammed does not exist

What else would you like to do from the previous list?
Please enter the number only: 5
Are you sure you want to Save the dictionary and end the session (Y/N): Y
Please enter the desired file name
(Note: if there is no file with the given name, a new one will be created)
testRun3
Changes saved successfully!
You have just used the best Dictionary class ever.
```



```
a
aaa
aaas
aarhus
aaron
aau
aba
ababa
aback
abacus
abalone
abandon
abase
abash
abate
abater
abbas
abbe
abbey
abbot
abbott
abbreviate
abc
abdicate
```

Challenges:

The program is straightforward, no complicated functions were needed.

The hard part was finding the similar words. However, once the idea came to me to use a variable to keep track of the characters difference, it became easy to implement and do.

Another issue here was with the words shorter than each other, which was a logic mistake from my side. The mistake was that I used the length of the key as the cap. However, there are words similar to the key but shorter than it, doing that would cause a *IndexOutOfBounds* exception. I fixed it by taking the minimum of the lengths of the two words as my stopping point rather than always taking the length of the key.

Other than these two issues, the implementation and design was easy, or probably I was too good.