CS/COE 0445: Data Structures (Fall 2019)

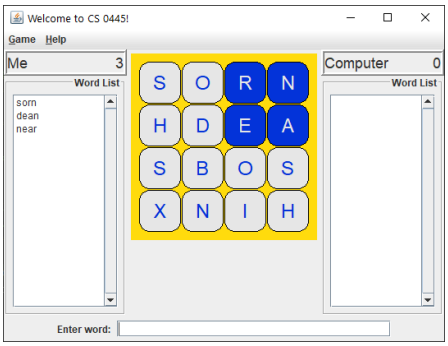
Department of Computer Science, University of Pittsburgh

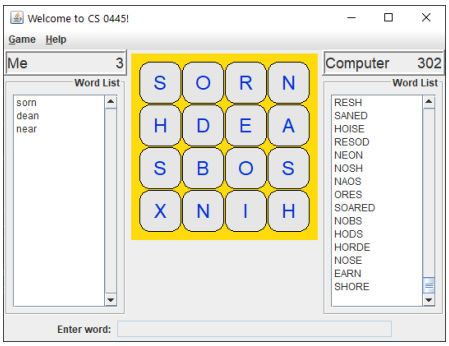
Assignment #5: Boggle

Released: Nov 21, 2019 **Due:** 8:00 PM, Friday, Dec 06, 2019

Goal

The purpose of this work is to implement a program that will find all possible words that can be formed in a Boggle game. Allowed words will be stored in a dictionaries given to you. If you hit enter (return) without any input the computer will start playing and it is going to find all the possible solutions.





Problem Description

First, carefully read the provided code.

A Boggle game (http://en.wikipedia.org/wiki/Boggle) is essentially a 4x4 table, with each cell displaying a letter or letter 'Q' that can be calculated as a pair of letters 'Qu'. In order for a word to be found in the 4x4 table to be valid, you must move from one cell to one or more cells so that the number of letters in the word is equal to the number of cells you visited.

When you are in a cell, you are allowed to move to its next neighboring cell vertically, horizontally or diagonally. However, you are not allowed to visit a cell you have previously visited (In other words, in a path using the cells you are not allowed to re-visit a cell twice).

- Note 1: You should use a **trie**, which will represent the dictionary.
- Note 2: You will need to think carefully how you can find all the paths, depending on the words in the dictionary. The general idea is, when designing the algorithm, as it is executing mark all the paths found so far that are either:
  - words, or
  - could be words if they had more letters.
- Note 3: In each path you should mark:
  - The cells you've already visited, and
  - The node of the trie representing the word (path)

Your program should take as an argument (at the command line) the name of a file containing a dictionary. Each line in the file will contain one word and will be separated from the next by using the character of the new line ('$\backslash n$').

There is a graphical user interface (GUI) implemented through the BoggleGUI class as shown above. You will **NOT** do anything about the graphical interface.

Example of execution:

```
java edu.pitt.cs.as5.BoggleGUI words1.txt
```

Example of dictionary:

```
FLUORITES
JUJITSU
CROOKEDER
LEUCINE
SPALPEEN
---
SNAKINESS
STENOTOPIC
OSMIUM
UNDERFUNDED
```

The algorithm initially uses one letter paths (the beginning of the path) corresponding to each cell. These paths represent the possible initial letters of the words that may appear in the table. At each step of the algorithm, the segment path formed is evaluated as follows:

- If the path is a word then it should be listed in the possible solutions.
- Try to expand the path by moving to a neighboring cell that you have **NOT** visited. This can create new paths that are at least one letter larger than the current path.

Boggle.java: Here are the main functions of the program. This class contains all the methods that will be called from the graphical interface.

BoggleGUI.java: Is responsible for the graphical user interface (GUI).

You must complete **ONLY** the following code:

1. **SortedDictionary.java**: The class must have all the functions of a dictionary. The dictionary is used by the class Boggle. An intelligent algorithm for classification (eg, quicksort) should be implemented in this class. The sorting is necessary for the binary search you need in the basic part of the program.
2. **TreeDictionary.java**: The class must have all the functions of a dictionary. The dictionary is used by the class Boggle. You need to create a Trie and the private class Node is already given.

Hints:

1. To test your program you need to replace the BoggleDictionary in line 39 in the Boggle.java with the SortedDictionary or the TreeDictionary.
2. It's great opportunity to check if your data structures are faster than the brute force approach (BoggleDictionary). Hit enter (return) on the input on the GUI and wait for the computer to find all the possible words :). Do the same for all 3 dictionaries.

Academic Honesty

The work in this assignment is to be done *independently*. Discussions with other students on the assignment should be limited to understanding the statement of the problem. Cheating in any way, including giving your work to someone else will result in an **F** for the course and a report to the appropriate University authority.

What to submit

Upload all your java files (preferably is a .zip) using the website. If you overwrite the provided interfaces, remember to restore them to their original versions. All programs will be tested on the command line, so if you use an IDE to develop your program, you must export the java files from the IDE and ensure that they compile and run on the command line. Do not submit the IDE's project files. Your TA should be able to compile and run your code as discussed in Lab 1. For instance, javac edu/pitt/cs/as5/BoggleGUI.java and java edu.pitt.cs.as5.BoggleGUI must compile and run InfixExpressionEvaluator.

In addition to your code, you may wish to include a README.txt file that describes features of your program that are not working as expected, to assist the TA in grading the portions that do work as expected.

How to submit your assignment

We will use a Web-based assignment submission interface. To submit your assignment:

- Go to the class web page `http://db.cs.pitt.edu/courses/cs0445/current.term/` and click the `Submit` button.
- Use your pittID/username as the username and your PeopleSoft ID as the password for authentication. There is a reminder service via email if you forgot your PeopleSoft ID.
- Upload your assignment file(s) to the appropriate assignment (from the drop-down list).
- Check (through the web interface) to verify what is the file size that has been uploaded and make sure it has been submitted in full. **It is your responsibility to make sure the assignment was properly submitted.**

You must submit your assignment before the due date (**8:00 PM, Friday, Dec 6, 2019**). There are no late submissions.