CSC 221 Spring '20 Complete Project due Tuesday 3/24/20 – 9:30am Project 3

Trie Trie Again

Overview

You are given a program to find all words (from a large word list) that can be found on a Boggle board. It works correctly but is very inefficient in the way it checks to see if a sequence of letters from the Boggle board is a legal word (or a word prefix). Your task will be to modify the program so that it stores the word list in a Trie and then uses the Trie to efficiently look up letter sequences to determine if they are words (or a word prefix). Pay close attention to the explanation of the working program that we will go over in class.

Getting Started

1. Go to https://classroom.github.com/a/79F4XmDH and accept the assignment to get a URL for your private repository. This URL will be something *like* https://github.com/WFU-CSC221/project3-*username*. Copy it. You'll need it in the next step.
2. Close all your IntelliJ windows.
3. Create a new IntelliJ project by selecting New Project and "Get from Version Control". On the next screen choose "Repository URL' and NOT "GitHub". Paste in the URL obtained in Step 1 and Clone. Accept all the defaults.
4. The project should compile and run and print solutions to Boggle puzzles of size 3x3, 4x4, 5x5, 6x6, and 7x7. You may want to record the solutions so you can check your results after you *improve* the code.

Overview of Files Provided

twl06.txt – a list of 178,691 Scrabble words that are used in Scrabble tournaments; one word per line

grid.txt – sample Boggle boards for debugging and testing; each board is preceded by a single integer that is the number of rows (same as number of columns); then the letters in the board, in a 2-d grid

Grid.java – a class for representing a Boggle board; you do not need to modify this class

Main.java – the main class that
(1) reads the word list from a file and creates *lexicon*, an ArrayList of Strings – in your modified code *lexicon* should be a Trie, not an ArrayList,
(2) reads the *grid.txt* file to create a 2-D representation of a Boggle board to be solved,
(3) recursively searches the board for all potential words using an algorithm called *backtracking*, as complete words are found they are added to a Set named *foundWords*, and
(4) computes and reports a "score" based on the content of *foundWords*.

Trie.java – a starting point for you to create your Trie class to complete the project

Next Steps

In this project you are going to use a Trie to "try" to improve the performance of the Boggle solver you are given as a starting point. The code you are given stores the 178,691 words from the word list in an ArrayList of Strings. Each time a sequence of characters from the Boggle board has to be checked to see if it's a word (or a prefix) the code does a sequential search through the word list. This gets very slow. Your task is to store the words in a Trie so they can be searched for efficiently. In addition to the running code I have provided a Trie class copied from *Geeks to Geeks*. You'll need to understand it and modify it to complete this project. Here are some details. You'll need to figure out the rest.

- A basic Trie class with an inner class named TrieNode is provided. You'll need to modify it.
- I strongly suggest enabling assertions while you are working on your code and adding some meaningful assert statements as you develop.
- Here are some of the methods / behavior you will need to complete the project:
    - Trie must have a *default constructor* – a constructor with no parameters
    - Trie must have a *public void* method named *insert* that receives a String (a word) and inserts it into the trie.
    - Trie must have a *public Boolean* method named *isWord* that receives a String (a word?) and returns True if the word is present in the Trie, false otherwise.
    - Trie must have a *public int* method named *inTrie* that receives a String (a word?) and returns one of three values as follows:
        - returns 1 if the argument string is a complete word in the Trie; for example, *inTrie("deacon")* should return 1
        - returns 0 if the argument string is not a complete word but it could be the prefix of a longer word, for example *inTrie("deaco")* should return 0 (unless it's a word I'm not familiar with)
        - returns -1 if the argument string is not a complete word and not a prefix of any complete word; for example *inTrie("xdeaco")* should return -1, in fact *inTrie("xd")* alone should have returned -1 but *inTrie("xr")* should return 0 if "xray" is the trie.

Reminders

Remember to occasionally Commit your work and then Push it to GitHub.

A grading rubric will be added later.