CS 251 Summer 2020
Project 5: Word Search
Due date:

General Guidelines

The APIs given below describe the required methods in each class that will be tested. You may need additional methods or classes that will not be directly tested but may be necessary to complete the assignment. Keep in mind that anything that does not *need* to be public should generally be kept private (instance variables, helper methods, etc.).

Unless otherwise stated in this handout, you are welcome (and encouraged) to add to/alter the provided java files as well as create new java files as needed.

Note on academic dishonesty: Please note that it is considered academic dishonesty to read anyone else's solution code to this problem, whether it is another student's code, code from a textbook, or something you found online. You MUST do your own work! You are allowed to use resources to help you, but those resources should not be code, so be careful what you look up online!

Note on implementation details: Note that even if your solution passes all test cases, if a visual inspection of your code shows that you did not follow the guidelines in the project, you will receive a 0.

Note on grading and provided tests: The provided tests are to help you as you implement the project. The Vocareum tests will be similar but not exactly the same. The final grade will be determined by your results on Vocareum.

Project Overview

A word search is a puzzle containing a bunch of letters in a grid in which you search for specific words that are hidden in the grid. There is an example below.

D	R	L	B	V	W	B
P	E	E	K	P	E	O
O	U	D	K	P	S	I
U	A	R	O	A	T	L
L	E	Z	D	N	M	E
Y	D	N	I	U	E	R
I	O	P	H	H	E	Y

This is a 7×7 grid of letters. It is up to you how you want to represent the puzzle, but you have to be able to construct the puzzle from an input file.
The input files are in the following format: The first line is the size of the grid (7 for a 7×7 grid). All the grids are squares. After that, each line is one line of the grid.

You should implement your solution in *Puzzle.java*. The following API describes the required methods. Do not change these method signatures.

Puzzle.java API

Puzzle(String fn)	This is the constructor. Construct the Puzzle from the given file. How you construct it is up to you.
int[] search(String word)	This is the main method you are tested on. It searches the puzzle for the given word and returns a 4-element array of integers: {a, b, c, d} where (a,b) are the row and column of where the word starts and (c, d) are the row and column of where the word ends. If the word cannot be found, return null. Indexing should start in the upper left hand corner of the grid at (0, 0) like a 2-D

	`array.`

Examples

D	**R**	L	B	V	**W**	**R**
P	E	**E**	K	P	**E**	**E**
O	**U**	D	**K**	P	**S**	**L**
U	A	**R**	O	**A**	**T**	**I**
L	E	Z	**D**	N	**M**	**O**
Y	D	N	I	**U**	E	**B**
I	O	P	H	H	**E**	Y

search("PURDUE"): `{1, 0, 6, 5}`
search("WEST"): `{0, 5, 3, 5}`
search("BOILER"): `{5, 6, 0, 6}`
search("MAKER"): `{4, 5, 0, 1}`

A few more things…

- Words can be found diagonally (either way), bottom to top, top to bottom, left to right, or right to left.
- Words are always in a line (horizontally, vertically, or diagonally).
- You will be tested on three grids:
  - a 15×15 grid with 15 words
  - a 25×25 grid with 25 words
  - a 50×50 grid with 50 words
- You will get 1 point per word for a total of 90 points. The remaining 10 points will be for the report described below.
- You should think about performance as a slow algorithm would not work well on the larger inputs.
- You are allowed to use most data structures and/or algorithms *covered in this course,* as well as Java implementations of said data structures. If you are unsure of what is allowed, just ask!
- You may **not** use the following String functions: *contains, contentEquals, copyValueOf, indexOf, lastIndexOf, matches, offsetByCodePoints, regionMatches, toCharArray*

- Sometimes Java has implementations using some of the data structures we discussed, but they are not as obvious as they might be. Some examples of what you may use are: *Queue, PriorityQueue, Stack, ArrayList, Hashmap,* and *Treemap* (which is implemented with a red-black tree).
- You should make a plan before starting to code, as your algorithm may be affected by how you implement the Puzzle itself, so think it through!
- You may want to consider doing some pre-processing on the grid itself to help the searches go faster.

Report

The final 10 points will be for your report. Type answers to the following questions and submit the PDF to Gradescope. Be sure to cite any sources that you use in this report.

1. Briefly describe your solution to this problem. What data structures did you use and how did you use them? Please note that we do not want a full detailed description of your code. If we wanted that, we could read your code. What we want is a high-level description of how you solved the problem.
2. Given an $N \times N$ grid and a word of length M, what are the best and worst-case runtimes for your *search* algorithm? Explain your reasoning. Do you have a sense of what the average runtime would be? Explain your reasoning. Keep in mind that there are different implementations of these data structures. If you use a Java class, you should look in to the runtimes of the operations you use.
3. How much extra space did your solution use. By extra space, we mean space that is in addition the $N \times N$ grid. Explain your reasoning.