CSCI 3901 Assignment 4

Due date: 11:59pm Monday, October 21, 2019 in git.cs.dal.ca at
https://git.cs.dal.ca/courses/2019-fall/csci-3901/assignment-4/xxxx.git
where xxxx is your CSID (this repository already exists, so clone it and then add your code to it).

Problem 1

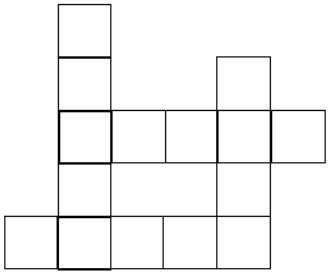
Goal

Work with exploring state space.

Background

Games are a domain in which the player searches through a set of possible local solutions to find an answer that satisfies all of the puzzle's constraints.

In this assignment, you will be solving a fill-in puzzle like the ones at https://api.razzlepuzzles.com/fillin In short, you are given positions on a grid where a word could fit. The positions for these words can cross one another; the two words must have a common letter at the crossing point. You are also given a set of words that should all fit into the puzzle (Figure 1). Your task is to map the words to their positions in the final puzzle (Figure 2).



bash, array, frail, plush

Figure 1 Sample puzzle

Each word provided will have a place in the solved puzzle and each word spot in the puzzle will end up with a word in it. No word is used twice.

Problem

Write a class called "FillInPuzzle" that accepts accepts a puzzle and a set of words and ultimately solves the puzzle.

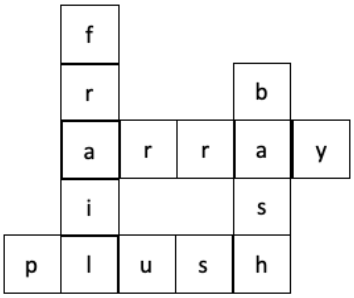


Figure 2 Solved puzzle

The class has at least 3 methods:

- Boolean loadPuzzle(BufferedReader stream) – Read a puzzle in from the given stream of data. Further description on the puzzle input structure appears below. Return true if the puzzle is read and ready to be solved. Return false if some other error happened.
- Boolean solve( ) -- Do whatever you need to do to find a solution to the puzzle. The solution is stored within the class, ready to be retrieved. Return true if you solved the puzzle and false if you could not solve the puzzle with the given set of words.

- Void print(PrintWriter outstream) – print the solution of the puzzle to the output stream Print the solution as an actual puzzle (like in Figure 2).
- Int choices( ) – return the number of guesses that your program had to make and later undo while solving the puzzle.

You get to choose how you will represent the puzzle in your program and how you will proceed to solve the puzzle..

Inputs

The loadPuzzle( ) method will accept a description of the puzzle as an input stream. The input stream will have 3 parts to it.

- Part 1: Consists of 3 integers on the same line, separated by spaces. The first integer is the number of columns in the puzzle grid. The second number is the number of rows in the puzzle grid. The third integer is the number of words in the puzzle; let's call that "n" in this assignment.
- Part2: Consists of n rows, each with 3 integers and one letter. The first two integers in the line are the column and row number where the word in the puzzle starts. Column and row numbers start at 0. The bottom left corner is column 0, row 0. The third integer is the number of letters in the word. The letter at the end tells you if the word appears horizontally (h) or vertically (v) in the puzzle.
- Part 3: Consists of n rows, each with a single word for the puzzle.

Example: The puzzle in Figure 1 would be represented as the following input:

```
6 5 4
0 0 5 h
1 2 5 h
1 4 5 v
0 3 4 v
bash
array
frail
plush
```

Outputs

The print( ) method produces output to the given output stream. Empty spaces between words are filled with spaces. The output will be designed for a fixed-width font. It prints the grid with column 0, row 0 in the bottom left corner (so prints the maximum row first).

Assumptions

You may assume that

- No word is repeated in the set of input words.
- The puzzle is case invariant.

- The words in the puzzle all fill from left to right or from top to bottom.

Constraints

- You may use any data structures from the Java Collection Framework.
- You may not use an existing library that already solves this puzzle
- If in doubt for testing, I will be running your program on bluenose.cs.dal.ca. Correct operation of your program shouldn't rely on any packages that aren't available on that system.

Notes

- Develop a strategy on how you will solve the puzzle before you finalize and start coding your data structure(s) for the puzzle
- Work incrementally. First write the code to read in a puzzle. Test that. Next, write the code to print a solution, whatever solution you have. Test that. Then, write your code to solve the puzzle.

Marking scheme

- Documentation (internal and external), program organization, clarity, modularity, style – 4 marks
- List of test cases for the problem – 3 marks
- Explanation of how you are doing your solution (strategy and algorithm) and what steps you have taken to provide some degree of efficiency (include in your external documentation) – 3 marks
- Ability to solve puzzles – 12 marks
- Degree to which you manage to keep the number of choices you have to undo to a small number – 3 marks