External Documentation

- Assignment 4

(Version 1.0)

Table of Contents

1. Overview

The goal of this assignment is to design a fill-in puzzle that takes the dimensions, words and their placeholder as an input and outputs a valid solution (if the given puzzle has one) else returns false. It includes the following operation to produce the desired result:

- **Load a Puzzle (BufferReader reader):**
It read the parameter reader, process, and load all the data from the reader into the puzzle. If the data provided is correct and the framework of the puzzle can be formed, it returns true, otherwise, it returns false.

- **Solve ():**
After loading the data of the puzzle, this method is called to solve the puzzle. If with loaded data, the puzzle can be solved successfully, it returns true; otherwise false.

- **Print (PrintWriter outStream):**
It prints the solution of the puzzle as an actual puzzle to the output stream.

- **Choices ():**
It returns the Number of Choices which were made that later were undone as they were wrong. This count will suggest the efficiency of the algorithm used. Lower the count, more efficient the algorithm would be.

The solution delivered is based on several important factors as the implementation of data the structure used for storing the Place Holders, Intersection, and Crossword puzzle on overall.

2. Files and external data

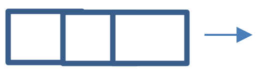
Below is the java files for the Implementation of the required operation:

- **FillInPuzzle.ava**: The program that has methods to load puzzle “loadPuzzle”, solve Puzzle “solve”, print the puzzle “print” and getting the number of wrong choices made which were later undone “choices”.

Three java files are implementing the GridPoints, Place Holder, Intersection, and crossword object:

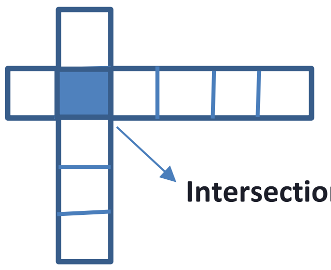
- **GridPoint.java**: Implements a blueprint of the Index of the multidimensional array in the form of a grid with X and Y values. Index [0][0] can be represented in form GridPoint (X=0, Y=0) in form of Object.

- **PlaceHolder.java**: a Placeholder is a place for inserting words in a crossword puzzle. For instance, below given is the place holder starting at grid point (x=0, y=0), with size ‘0’ and oriented horizontally. The crossword puzzle consists of the number of such placeholders. The same Implementation is provided in the class file.



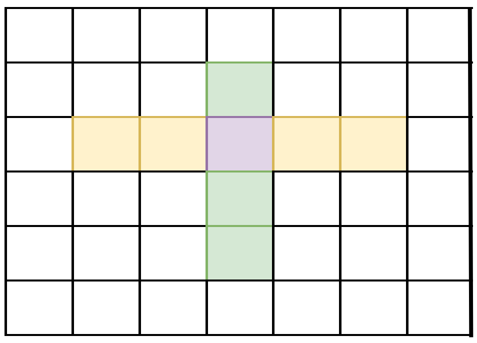
PlaceHolder (x = 0, y =0, size = 3, orientation = horizontal, Set<String> possible word of length 3)

- **Intersection.java**: Each Crossword puzzle will have multiple Horizontal and vertical place holders which will intersect with one another at some point, those points are called as an intersection. We are treating the intersection as an object formed by two placeholders at point.
For instance, below are the two Placeholder pl1 and pl2 meeting at some point say GridPoint (x=3, y =1) or index [3][4]. So we treat this object made up of placeholder pl1, placeholder pl2 and the point of intersection (x=3, y=1).



Intersection [Placeholder P1, Placeholder p2, Grid Point(X=3,Y=1)]

- **CrossWordPuzzle.java:** Provides blueprints of the puzzle made up of the GridPoint(indexes), List of all the Placeholders and the set of all the intersections on a defined matrix of given row and column size.



3. Data structures and their relationship with each other

The program treats GridPoint, PlaceHolder, Intersections as an object which together represents Crossword puzzle as an Object.

The Crossword the object uses below data structure to store and manage the data:

- **HashSet of Intersection**: We are storing all the Intersection into a HashSet, which takes care of any duplicate (already existing) the intersection is being added.

- **List of PlaceHolders**: We are storing all the Placeholder (provided in input) into an array list, which takes care of insertion order.

- **Character[] []** : Represent Matrix of the provided row and column size which is eventually used by the placeholder to fill in and represent graphically as a crossword puzzle.

The above forms Crossword object, remaining data structured used in “FillInPuzzle” class areas shown below and are used while solving the puzzle.

- **Set of a visited word:** We use this while solving the puzzle to keep the track of the words which are already used in filling any of the placeholders, so that they can not be considered by another placeholder for filling.

- **Set of visited Placeholders:** We use this while solving the puzzle to keep track of the already visited placeholders, so that those Placeholders are not visited and considered again which can eventually, lead to a never-ending.

4. Assumptions

Following are the assumptions made while the implementation of the fill-in puzzle:

- 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.

5. Key algorithms and design elements

1. **Loading the puzzle**: This Method validates the input provided and load the valid data to form the puzzle.

   i. The first line data is considered to get the row and column size of the puzzle to be formed. Also, the word count is taken.
   ii. Then the further data is taken and every line a placeholder object is created with start point as Grid Point(x,y) and size , and then placed into a list of placeholders.
   iii. Once the Place Holder object are created, then the words provided in input are processed.

      - Based on the length of the word, list of placeholders is iterated and comparing the size of the placeholder and length of the word, the words are placed into a set called “setofPotentialCandidateWords” and allocated to the particular placeholder.
      - This process is repeated for every word.

   iv. At the End a crossword puzzle object is created where the row size, column size and list of Placeholder’s are allocated.
   v. Once the Object of the Crossword puzzle is created, it internally takes care of finding the intersection between the placeholder and store all the intersection object in a set as a part of crossword puzzle object.

2. **Solving the Puzzle:** We take the Crossword puzzle object and fetch the Matrix representation of the puzzle in form of 2d character [][] array. We also fetch the list of placeholders stored in crossword object. Then we follow below steps to solve the Puzzle.

   - Start a method SolvePuzzle() which accept the character array and the list of placeholders.

     I. We take out the last element(placeholder) say PL1 from the list of placeholders.
     II. We add PL1 in the visited set of the placeholder
     III. We find out the intersection formed by the placeholder PL2.

        - Intersection (PL1, PL2, (x,y))

     IV. With each intersection we find out the other related placeholder PL2.
     V. We check if the PL2 is already visited or not, if not we shuffle the position of the PL2 in the list of placeholders to make it to last element.
     VI. We then try filling out the placeholder PL1 with all the possible candidate words.
     VII. If the Word fits, we call the SolvePuzzle() method Again(Recursively) with new filed character array[][] and list of placeholder(PL1 removed and PL2(forming Intersection with PL1) moved to end).

   - Each Call to the Method Solve Puzzle () returns filed character array[][] if the puzzle is successfully solve for those placeholder, otherwise it return false.
   - Each return from recursive SolvePuzzle() method contributes to end result.
   - If the return is null, we backtrack and try with another possible candidate Word till all the placeholder are visited and all the words are tried for each place holders.

6. Strategies and Degree of Efficiency

- With use of the Placeholder and intersection, we only have to concentrate on the placeholder and iterate through them, instead of going through each cell of the matrix.

- While solving we go through each placeholder to find out all the intersection and through those intersection we traverse to another Placeholder (forming the intersection) and work on filing/solving them.

- Through this way, it is easier to find out the wrong guesses in early stage and efficiently having the correct guess for the next placeholder as intersection would already be suggesting one of the characters required for the next word.