Author: Shivangi Ajaykumar Bhatt
B00 # : B00863408
Subject Code: CSCI 3901
Date: Nov 16th, 2021
Assignment #: 4

Boggle

Overview:
The program called Boggle uses state space traversal to find all the possible words in any direction present in the dictionary and it uses the following methods:

1. getDictionary(): gets the dictionary from the Buffered Stream.
2. getPuzzle(): gets and stores the puzzle from the Buffered Stream.
3. solve(): Solves the boggle Puzzle.
4. print(): Print the puzzle
5. isWordInDictionary(String word): Supporting method to check if a word exists in the dictionary
6. wordStartsWithExists(String word): Supporting method to check if a word exists in the dictionary that starts with word given in the argument
7. getDirection(): Supporting method to get the direction traversed from current cells to adjacent cells.

The list of words that are reported must be sorted alphabetically, and in case of a tie X coordinate of the starting character is compared. If there is a tie in X coordinate Y character must be compared

Files:
The program uses only one java file for the execution of the program:

1. Boggle: It is the only class that performs actions like getting the dictionary, getting the puzzle, solving the puzzle and printing the puzzle

Data structures and their relations to each other
The program uses the following data structures:

1. List<String> dictionary:
   A list of string that stores all the words present in the dictionary.
2. <List<List<Character>> puzzle:
   The puzzle is stored as a two dimensional list, where each each row is a list of character and hence every cell has a character stored in it.
3. List<String> detectedWords:
   The list of string that reports all the detected words found from the boggle puzzle that exists in the dictionary.

Assumptions

Following assumptions have been made for the program:

- If two paths are found for the same word the word that is encountered first while traversing the adjacent paths in the clockwise direction from North
- If there is an empty line encountered in the dictionary all the words after that line would not be a part of the dictionary.
- If there is an empty line encountered in the puzzle all the words after that line would not be a part of the puzzle.
- There would be no white spaces in the puzzle or dictionary between words
- All the words in the dictionary and puzzle would be in lower case

Strategy

- The strategy that I followed to get the solution of a boggle is to traverse through all the characters in the puzzle, in order to find all the words starting with that character. For Instance, we are currently looking for all the words starting with the character at coordinates (2,3), say ‘r’.
- Each character then checks all the 8 adjacent cells around it, i.e., all the characters in the cell around (2,3) are traversed. For instance, we are currently looking at the adjacent cell of ‘r’, say, (1,3) which has character ‘a’. It would mark (1,3) as visited and hence the found word will be ‘ra’
- The program would check if there exists any word starting with ‘ra’, if exists, the program would check all the adjacent cells of ‘a’, say (1,4) which has the character ‘i’. So now, the found word would be ‘rai’ and in a similar way the program continues to find the adjacent cells of the current cell until all the words are found.
- If there does not exist any word starting with ‘ra’, the program would stop checking the characters of the adjacent cell. It would remove the last character from the found word, and therefore the found word would now be ‘r’, and the program searches for the next adjacent word of ‘r’ except ‘a’ and continue finding words in the same way.

Why my strategy works and the steps taken to provide some degree of efficiency

My strategy works because it traverses through every possible word in the state space and matches it with the word present in the dictionary and hence it could solve the boggle puzzle. Besides, the step to stop checking the characters of the adjacent cell when there does not exist any word starting with a given word found optimizes the problem. It works because the need for recursively calling the method when it is not required is eliminated.

Key algorithms

The class Boggle uses four methods and their algorithms are explained as follows:

getDictionary()

1. Loop through the bufferedReader stream, line by line until the end of line or blank line is encountered
2. Check if length of line is less than 2 and if true return false and empty the dictionary
3. Add the line to the List<String> dictionary

getPuzzle():

1. Loop through the bufferedReader stream line by line until the end of line or blank line is encountered
2. Split each line to the list of characters.
3. Check if the line is the first line in the stream,
4. If true store the width of the puzzle and add the list of characters to the List<List<Character>> puzzle
5. Else check if length of line is not equal to the puzzle width
6. If step 5 returns true, throw IllegalArgumentException and set the puzzle as not ready
7. Else add the list of characters to the List<List<Character>> puzzle
8. Set the puzzle as ready
9. Store the puzzleHeight as the size of the puzzle
10. Return true

solve():

1. Check if puzzle is not ready
2. If step 1 is true, throw an Exception
3. Create a boolean array “visited” of size puzzleWidth X puzzleHeight and mark all the cells as not visited
4. Initialize foundWord and path strings as empty
5. Loop through each cell in the puzzle to find all the words starting with the character in that cell.
    - Inside the loop call solveRecursive() method to recursively find all the words

SolveRecursive(boolean[][] visited, int row, int col, String wordFound, String path, String newDirection):

1. Mark the current cell as visited
2. Get the character at the current cell and append it to the words found
3. If the length of the word found is 1, store the starting X and Y coordinate of the character
4. Append the current newDirection to the path string
5. Check if the word is present in the dictionary:

If true:

- Loop through the detected words
- Check if the word is already present in the list
- If present, check if the X coordinate of the current word found is less than the X coordinate of the word already present in the list of detected words.
  If true, replace the word in the detectedWords list with the current word, its coordinates, and the path followed
- Check if the X coordinate of the current word found is equal to the X coordinate of the word already present in the list of detected words.

  If true, check if the Y coordinate of the current word found is less than the Y coordinate of the word already present in the list of detected words.
  If true, replace the word in the detectedWords list with the current word, its coordinates, and the path followed
- If the word is not present, add it to the detectedWords list

6. Check if a word in dictionary exists that starts with the found word
7. If true, traverse all 8 the adjacent cells of current cell
8. Get the direction of adjacent cell being traversed from the current cell
9. Call solveRecursive()
10. Remove the current character from the found word
11. Set the current cell's visited array state to false.

print():

1. Define an empty output string
2. Loop through all characters in the puzzle
3. Append all the characters encountered to the output string
4. Return output string

Test cases

For the given Boggle class following would be the test cases for its given methods:

Input Validation

getDictionary()

- Stream is null
- Stream is empty
- Stream has only one line
- Stream has more than one line
- If one or more words exist in the stream with less than 2 length

getPuzzle()

- Stream is null
- Stream is empty
- Stream has only one line
- Stream has more than one line
- If one or more line exists in the stream that does not match the puzzle width

solve()

- Solve the puzzle when it is not ready
- Solve the puzzle when it is empty
- Solve a 0X0 puzzle
- Solve a 1X1 puzzle
- Solve the puzzle for which paths can be found for all the words in the dictionary
- Solve the puzzle for which no path can be found for all the words in the dictionary
- Solve the puzzle for which paths can be found for some of the words in the dictionary
- Solve the puzzle for which more than 1 path exists for words that start with different coordinates
- Solve the puzzle for which more than 1 path exists for words that start with the same X and Y coordinates

print()

- Puzzle is null
- Puzzle is empty
- The puzzle has a width of 1
- The puzzle has a height of 1
- Puzzle has both height and width of 1