Assignment 4

Team Members

Meet Nitinbhai Patel

(B00899516)

Subject:

Software Development Concepts

Professor:

Mike McAllister

Problem 1: Boggle

Overview

You're given a words and m*n board where every cell in that board has one character. Your task is to find as many words as possible from set of adjacent character. A path begins at one point in the grid and leads to a adjacent neighbors by going to the next letter in any direction (left, right, up, down, or diagonally).

The letters/character which we are visiting along the path will form a word and that word must be present in the know dictionary. Note: you cannot use the cell more than once for forming any word.

Files and external data

Java Files

1. **Boggle.java**

Methods

- **boolean getDictionary(BufferedReader stream)**
  - This method read the set of words from the file.
  - It will read the words till it encounter end of file
  - If while reading it encounter a word which is less than 2 characters, then it will return false as those words are not accepted.
  - If the file is empty, then these methods will return false.
  - It will return true when all the words are correct and ready to use for puzzle solving.

- **boolean getpuzzle(BufferedReader stream)**
  - This method will read a rectangular grid of letters from the file which will form a Boggle puzzle.
  - It will read the letters till it encounter end of file.
  - This method will return false if every row doesn't have same number of letters.
  - It will also return false if the file is empty or if puzzle is not found

- **public List<String> solve ()**
  - This method will return a list of string in the following format:

    <word> <starting X> <staring Y> <navigation sequence>

- It will return an empty list whenever getDictionary() or getpuzzle() methods returns false.

- **private void findWord(TrieNode rootNode, char[][] boggle, int i,int j, boolean[][] visitedNode, String nodeCharacter, String direction)**
  - This is recursive method which is used to find the word from the give m*n puzzle board
  - This method will accept the node of TrieNode type, Boggle puzzle Board, exact position of the character in the boggle puzzle board using I and j, boolean variable to check whether we have visited that character or not, a character and a direction.
  - The directions are as follow:
    - L – go left
    - R – go right
    - U – go up
    - D – go down
    - N – go diagonally up and to the left
    - E – go diagonally up and to the right
    - S – go diagonally down and to the right
    - W – go diagonally down and to the left

- **public String print ()**
  - This method will print the current puzzle.

- **private int getX()**
  - this will return the value of x

- **private void setX(int x)**
  - this method is used to set the value of x

- **private int getY()**
  - this will return the value of y

- **private void setY(int y)**
  - this method is used to set the value ofy

2. **TrieNode.java**

- **boolean isLastNode()**
    - It will return true or false

- **void setLastNode(boolean lastNode)**
    - It will used to set the value of lastNode

Text Files

words.txt

- This will contain list of words which we are going to find from the puzzle board

Puzzle.txt

- This will contain grid of letters from which we will find the possible words

Data structures and their relations to each other

Data Structure Used in the Code

1. **LinkedList**
    - This data structure is used to store the stream of words for the dictionary which we will read from the file.
    - This data structure is also used for storing the puzzle grid which we will read from the file.

    Advantage of using LinkedList over other data structure

    - The main advantages of using LinkedList is that element can be inserted or removed without reorganization of entire structure. i e Insertion and deletion is efficient in LinkedList as compared to other data structure.
    - And another advantage is that we don't need to define size at the time of initialization.
    - We can easily implement stack and queue using LinkedList.

For storing the stream of words, I have used LinkedList because LinkedList is dynamic data structure. It can grow and shrink at runtime. So, it doesn't require any initial size unlike array which requires initial size. As we don't know how many words

we are going to read from the file, LinkedList would be the great option. Another advantage is that there is no memory wastage.

2. **2 D Array**
    - This data structure is used to create the m*n 2d puzzle for boggle game.
    - The reason for using the 2d array is that it allows us to create a matrix which has row and column.
    - For boggle problem we new a puzzle and creating a puzzle using 2d array is a great choice.

Classes used in the code

1. Boggle → Manages the overall functionality of the boggle problem
2. TrieNode → For creating an empty trie

Assumptions

- The word of dictionary is provided in sorted order
- The word is skipped when its length is less than 2 characters

Key algorithms and design elements

- I have implemented the boggle problem using Trie data structure for better efficiency.
- We can also implement boggle problem using depth first search. But dfs is not efficient as trie. There may be reptation of words if we use dfs.
- A Trie is a data structure for storing strings that may be seen as a graph. Nodes and edges make up this graph. Trie supports find and insert operations. The insert operation inserts a key (string) and value in the trie while find operation return a key string. The insert and find operation run in O(n) time, where n =length of key. Hence the time complexity of my algorithm is O(n) where n length of word

This is how in have implemented Trie in my code

- Firstly, I will create an empty trie and insert words from the dictionary into trie.

- After insertion the trie will look line this

- Now we I have to select those character/letters from boggle [][] which are child of the root of trie
    - For example, I picked ‘M’ which is at boggle [0][0] and ‘E’ at boggle [1][0]
- Then I must search a word in the trie which will starts with the letters that we have picked in previous step.
- Now we have create a boolean[][] visitedNode (visitedNode[P][Q] = false)
- Now we have to call solve() method.
    - The solve method call recursively Find word method
        - FindWord(findNode.childNode[(puzzleMatrix[i][j]) - '!' puzzleMatrix, i, j, visitedNode, str.toString(), "");
- In findWord() Method I have done following things
    - If the node is the last node, then I will store that word in linked list
    - If we have seen the element/node for the first time, then we will make that node as visited
    - Then we will traverse all the child node of the current node. We will find all the nodes which are adjacent to boggle[i][j].
    - And finally, e will return the list of words that we have found in following format
        - <word> <starting X> <staring Y> <navigation sequence>

Limitation/Drawbacks

- As I have used Trie data structure, the only limitation is that it needs lots of memory for storing strings. For every node we have to many pointers.
    - To solve this problem, we can use Ternary Search Tree.
    - The time complexity of Ternary Search Tree is O(h) where h is the height of the tree
    - However, the trie data structure is much faster than Ternary Search Tree, but it requires more memory

Test Cases

Input Validation (Generally, tests on bad input data for which you shouldn't crash)

Method 1: boolean getDictionary(BufferedReader stream)

- Stream is null
- Stream is empty
- Length of each word/line in stream must be greater than or equal to 2.

Method 2: boolean getpuzzle(BufferedReader stream)

- Stream is null
- Stream is empty
- Length of each word/line in the stream must be same

Boundary Cases (Tests at the edge of inputs or problem structures)

Method 1: boolean getDictionary(BufferedReader stream)

- Stream contains one word/line
- Word in the stream contains two characters

Method 2: boolean getpuzzle(BufferedReader stream)

- Stream contains one word/line
- Word in the stream contains one character
- Each word/line in the stream contains one character

Control Flow Test Cases (Tests of the core operations)

Method 1: **boolean getDictionary(BufferedReader stream)**

- Duplicate words/lines in the stream
- Stream contains more than one word/line

Method 2: **boolean getpuzzle(BufferedReader stream)**

- Duplicate words/lines in the stream
- Stream contains more than one word/line

Method 3: List<String> solve ()

- Invoke solve() when getDictionary() returns false
- Invoke solve() when getpuzzle () returns false
- Invoke solve() when getpuzzle () and getDictionary() returns false
- Invoke solve() when no word is found from the puzzle
- Invoke solve () when one word is found from the puzzle
- Invoke solve () when multiple words are found from the puzzle
- Words overlap with already found word
- Find list when words have same starting coordinates

Method 4: String print()

- Print string when getpuzzle() is false

Data Flow Test Cases (Tests around the order in which things are done)

Normal execution order:

getDictionary() → getpuzzle() → solve() → print()

Data Flow

1. boolean getDictionary(BufferedReader stream) → *read stream of words such as rain, tail, silk, satin, nail, yeti from the file*

2. boolean getpuzzle(BufferedReader stream) → *read stream of words such as rwkb, tasl, ytik, eaan from the file*

3. List<String> solve() → *It will return*

[nail 4 1 LUE,
rain 1 4 SSS,
satin 3 3 LDRS,
silk 3 3 DED,
tail 1 3 RSE,
yeti 1 2 DER]

4. String print() → it will print

rwkb
tasl
ytik
eaan