ALGORITHMICS - Bachelor of Software Engineering, 2020

LAB GUIDE. SESSION 6

GOALS:

- **Backtracking algorithms**

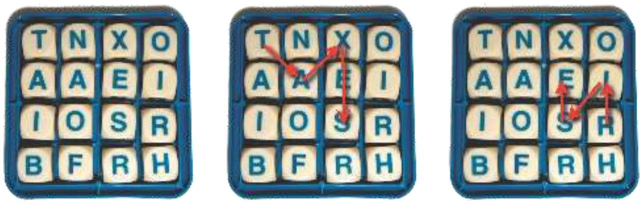
1. Playing Boggle

Boggle® is a classic word game usually played on a 4 by 4 board of letters. The letters are randomly generated by shaking 16 cubes labeled with a distribution of letters like that found in English words. Players try to find words hidden within the board. Words are composed from letters that adjoin horizontally, vertically, or diagonally. Thus, we can move to any of the 8 possible adjacent cells to a given cell. However, no cell may be used more than once within a single word.

The score awarded for a word depends on its length, with longer words being worth more points. A word is only ever scored once, even if it appears multiple times in the table. Exact point values for a word (in our version of the game) are shown in the table below:

Number of letters:	0	1	2	3	4	5	6	7	8	9	10
Points:	0	0	1	2	5	7	9	12	15	19	24

This is an example for a 4 by 4 table in which we can found the words "taxes" and "rise", giving a total of 12 points (7 plus 5 points respectively). Another valid word would, for example, "tax".



2. Files you are provided with

You are provided with several files. For example, some dictionaries to check if the words are correct:

- `dictionary100.txt`. It is a list of 100 of the most used words in English.
- `dictionary80386.txt`. It is a list of 80386 of the most used words in English.

Some boards to play:

- `table01.txt`. It is a board with 1x1 cells to play Boggle. In all files, the first line contains the size `n` of the board.
- `table02.txt`. It is a board with 2x2 cells to play Boggle.

- `table03.txt`. It is a board with 3x3 cells to play Boggle.
- `table04.txt`. It is a board with 4x4 cells to play Boggle.
- `table05.txt`. It is a board with 5x5 cells to play Boggle.
- `table15.txt`. It is a board with 15x15 cells to play Boggle.
- `table46.txt`. It is a board with 46x46 cells to play Boggle.
- `table100.txt`. It is a board with 100x100 cells to play Boggle.
- `table205.txt`. It is a board with 205x205 cells to play Boggle.
- `table500.txt`. It is a board with 500x500 cells to play Boggle.
- `table1000.txt`. It is a board with 1000x1000 cells to play Boggle.

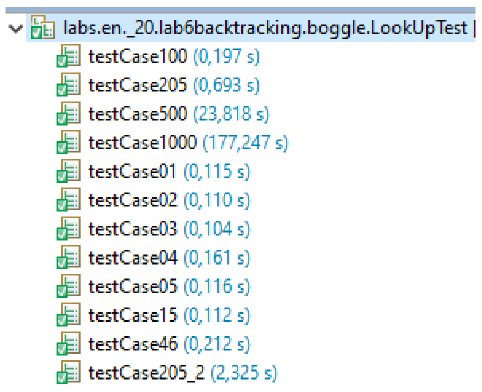
Some Java code files:

- `BoggleTest.java`. It contains JUNIT test cases you must pass without changing anything in the file (except the path of the text files).
- `BoggleTimes.java`. It contains a loop to measure times for different sizes of the board. You cannot change this file (except the path of the text file).

TO DO:

A. Work to be done

- An Eclipse `session6.boggle` **package** in your course project. The content of the package should be:
    - All the files that were given with the instructions for this session together with the implementation of `Boggle.java`. This file should implement the Boggle game using backtracking to make the test pass. Please, consider that a word cannot be repeated in a solution and the list of words in a solution should be sorted alphabetically in ascending order. If not, tests will not pass.
- An Eclipse package with the working implementation of the example seen during the session.
- A **PDF document** using the course template. The activities of the document should be the following:
    - **Activity 1. Test cases**
        - Put a screenshot where it is possible to see the successful test cases with the times taken. For example:



```
labs.en._20.lab6backtracking.boggle.LookUpTest |
    testCase100 (0,197 s)
    testCase205 (0,693 s)
    testCase500 (23,818 s)
    testCase1000 (177,247 s)
    testCase01 (0,115 s)
    testCase02 (0,110 s)
    testCase03 (0,104 s)
    testCase04 (0,161 s)
    testCase05 (0,116 s)
    testCase15 (0,112 s)
    testCase46 (0,212 s)
    testCase205_2 (2,325 s)
```

- **Activity 2. Times for different executions**
  - Fill in the table with the values obtained from `BoggleTimes.java`. You should modify that class in two circumstances:
    - If times are <50 ms you should use `nTimes` to get representative values.
    - If times in the table are good enough, change the way `n` grows: you can increment the size of `n` by 10 in each iteration or even better, you may multiply the size by 2 in each iteration.

n	*time*
1	…..
2	…..
3	…..
4	…..
5	…..
6	…..
7	…..
8	…..
…	…..
Until you get at least 50 representative values	

  - What do you think about the complexity of the algorithm?
- **Activity 3. Code improvements**
  - Depending on factors such as the data structures used and the possibility of doing pruning in the tree, the performance of the algorithm can change dramatically. Have you made any choice that you think improves the performance of your algorithm? Which one?

You should **commit and push** your project with your new `session6` package in your Github repository with the following content inside it:

- All the requested source files.
- The requested PDF document called `session6.pdf` with the corresponding activities.

Important:

- Make sure your Github course project is up to date after the delivery and that the last commit is strictly before the deadline. If not, the mark for this session will be 0 without any exception.
- Your mark will be eventually included in the `Marks.xlsx` file in your repository. Please, do not modify anything in that file to avoid conflicts between versions.

Deadlines:

- Group L.I-01 (Thursday): April 15, 2020 at 11:55pm.
- Group L.I-02 (Wednesday): April 14, 2020 at 11:55pm.
- Group L.I-03 (Tuesday): April 13, 2020 at 11:55pm.
- Group L.I-04 (Monday): March 19, 2020 at 11:55pm.