ALGORITHMICS - Bachelor of Software Engineering, 2020

LAB GUIDE. SESSION 7

GOALS:

- **Branch and bound algorithms**

1. Playing Boggle

In the previous sessions, the Boggle® game has been solved with the Backtracking technique, resulting in all the possible valid words in a board, as well as the sum of all the scores related to those words. In this session, we will continue "playing" Boggle.

To make the game more entertaining we are going to change some of the rules:

Concatenated words: In the traditional Boggle game, it is not possible to concatenate valid words multiple times. From now on, a maximum of concatenated words can be presented (value entered as a parameter) if the game cells allow it.

Cells repeated more than once: in the traditional game of Boggle, it is not possible to use the same cell/position more than once. From now on, from an adjacent cell it will be possible to return to a previously visited cell (you cannot go from one cell to itself). Each cell can be visited a maximum number of times (value entered as a parameter).

Game score: from now on the score obtained when playing will be calculated with the following formula: $\mathbf{p = x^k + x * k}$.

Being **p** the score, **x** the length of the word and **k** the number of times that a word has been concatenated. Words of length 1 will score 0 points.

2. Examples of the game

Assuming the following board:

```
3

H I O

H I O

R E S
```

And the following dictionary:

```
hi

her

sos
```

If we want to find all the solutions with the following configuration:

- Maximum number of concatenated words = 1
- Maximum number of repetitions of the same cell = 1

The result should be like the following:

```
hi   repetitions: 1 score: 4.0

her  repetitions: 1 score: 6.0
```

With the following configuration:

- Maximum number of concatenated words = 2
- Maximum number of repetitions of the same cell = 1

The result should be like the following (as you can see, each word only appears once, but with the maximum number of repetitions – "hihi" is "hi" twice):

```
her  repetitions: 1 score: 6.0

hihi repetitions: 2 score: 8.0
```

With the following configuration:

- Maximum number of concatenated words = 1
- Maximum number of repetitions of the same cell = 2

The result should be like the following:

```
hi   repetitions: 1 score: 4.0

her  repetitions: 1 score: 6.0

sos  repetitions: 1 score: 6.0
```

With the following configuration:

- Maximum number of concatenated words = 2
- Maximum number of repetitions of the same cell = 2

The result should be like the following:

```
sos       repetitions: 1 score: 6.0

hihi      repetitions: 2 score: 8.0

herher    repetitions: 2 score: 15.0
```

With the following configuration:

- Maximum number of concatenated words = 3
- Maximum number of repetitions of the same cell = 3

The result should be like the following:

```
sos       repetitions: 1 score: 6.0
hihihi    repetitions: 3 score: 14.0
herherher repetitions: 3 score: 36.0
```

Note that if we add the word `hihi` to the dictionary:

```
hi
hihi
her
sos
```

With the following configuration:

- Maximum number of concatenated words = 1
- Maximum number of repetitions of the same cell = 1

The result should be like the following (it is observed that, regardless of the score, the appearance of a word from the dictionary -`hihi`- takes precedence over the repetition of a word `-hi-`):

```
hi        repetitions: 1 score: 4.0
her       repetitions: 1 score: 6.0
hihi      repetitions: 1 score: 8.0
```

What happened is that we found “hi” and then “hihi”. Since “hihi” is also a word in the dictionary, instead of considering that “hihi” is a repetition of “hi”, we must consider that “hihi” is just a new word.

If, for the same dictionary, we have the following configuration:

- Maximum number of concatenated words = 2
- Maximum number of repetitions of the same cell = 2

The result should be similar to the following:

```
hi        repetitions: 1 score: 4.0
sos       repetitions: 1 score: 6.0
herher    repetitions: 2 score: 15.0
hihihihi  repetitions: 2 score: 24.0
```

What happened is that we found “hi” and then “hihi”. Since “hihi” is also a word in the dictionary, instead of considering that “hihi” is a repetition of “hi”, we must consider that “hihi” is just a new

word. However, we were lucky, and we found a repetition of "hihi", so we return the longest version of "hihi", that is "hihihihi".

3. Files you are provided with

In addition to the dictionaries and the boards delivered with the previous session, some files with Java code are attached in this session:

- `BoggleAllBacktrackingRepetitionsTest.java`. It contains JUNIT test cases that must be passed without changing anything in the code (except for the path where the text files are located).
- `BoggleBestPruningBacktrackingRepetitionsTest.java`. It contains JUNIT test cases that must be passed without changing anything in the code (except for the path where the text files are located).

TO DO:

A. Work to be done

- An Eclipse `session7` **package** in your course project. The content of the package should be:
  - You are asked to design the algorithm to solve the problem described by **Backtracking, finding all possible solutions**. The class must be called `BoggleAllBacktrackingRepetitions.java` to pass the unit tests contained in `BoggleAllBacktrackingRepetitionsTest.java`.
  - You are asked to design the algorithm to solve the problem described by **Backtracking, finding the sequence (or one of them) that scores the most (the best one).** The class must be called `BoggleBestPruningBacktrackingRepetitions.java` to pass the unit tests contained in `BoggleBestPruningBacktrackingRepetitionsTest.java`. Try to prune as much as possible.
  - You are asked to design the algorithm to solve the problem described by **Branch and Bound, finding the sequence (or one of them) that scores the most (the best one)**. The goal will be to try to improve the implementation carried out by Backtracking in the previous point by passing the code through the same tests as for the exercise with Backtracking.
- A **PDF document** using the course template. The activities of the document should be the following:
  - **Activity 1. Test cases**
    - Screenshots are requested to see the successful tests, as well as the time necessary to execute those tests (one capture for each set of tests).
  - **Activity 2. Times for different executions**

- You are asked to create a class to measure and compare the execution times of the two implementations for different sizes of **n**. To do this, use the 80368-word dictionary. Also use the following settings:
  - Maximum number of concatenated words = 1
  - Maximum number of repetitions of the same cell = 1

n	*BacktrackingTime*	*BranchAndBoundTime*
10	…..	
20	…..	
30	…..	
40	…..	
50	…..	
60	…..	
70	…..	
80	…..	
…	…..	
If the executions are very slow, change the size of n		

Explain the times obtained and compare the execution of both algorithms.

- **Activity 3. Times for different executions (II)**
  - You are asked to create a class to measure and compare the execution times of the two implementations for different sizes of **n**. To do this, use the 80368-word dictionary. Also use the following settings:
    - Maximum number of concatenated words = 100
    - Maximum number of repetitions of the same cell = 1

n	*BacktrackingTime*	*BranchAndBoundTime*
10	…..	
20	…..	
30	…..	
40	…..	
50	…..	
60	…..	
70	…..	
80	…..	
…	…..	
If the executions are very slow, change the size of n		

Explain the times obtained and compare the execution of both algorithms.

- **Activity 4. Times for different executions (III)**
  - You are asked to create a class to measure and compare the execution times of the two implementations for different sizes of **n**. To do this, use the 80368-word dictionary. Also use the following settings:
    - Maximum number of concatenated words = 1
    - Maximum number of repetitions of the same cell = 3

n	*BacktrackingTime*	*BranchAndBoundTime*
10	…..	
20	…..	
30	…..	
40	…..	
50	…..	
60	…..	
70	…..	
80	…..	
…	…..	
If the executions are very slow, change the size of n		

Explain the times obtained and compare the execution of both algorithms.

- **Activity 5. Times for different executions (IV)**
  - You are asked to create a class to measure and compare the execution times of the two implementations for different sizes of **n**. To do this, use the 80368-word dictionary. Also use the following settings:
    - Maximum number of concatenated words = 15
    - Maximum number of repetitions of the same cell = 5

n	*BacktrackingTime*	*BranchAndBoundTime*
10	…..	
20	…..	
30	…..	
40	…..	
50	…..	
60	…..	
70	…..	
80	…..	
…	…..	
If the executions are very slow, change the size of n		

Explain the times obtained and compare the execution of both algorithms.

You should **commit and push** your project with your new `session7` package in your Github repository with the following content inside it:

- All the requested source files.
- The requested PDF document called `session7.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 29, 2020 at 11:55pm.
- Group L.I-02 (Wednesday): April 28, 2020 at 11:55pm.
- Group L.I-03 (Tuesday): April 27, 2020 at 11:55pm.
- Group L.I-04 (Monday): May 3, 2020 at 11:55pm.