Athens University of Economics and Business, CS Department
Course: Data Structures
Academic Year: 2019-2020
Instructor: Vangelis Markakis

1st Project

Queues: Abstract Data Types implementations and applications

The purpose of this project is to familiarize you with basic abstract data types such as stacks and FIFO queues. This assignment consists of ADT implementations (Parts A and C) as well as 1 application (Part B).

Part A [30 points]. StringStack and StringQueue interfaces are given, which state the basic Stack and FIFO Queue methods, with String elements. Create an implementation of the StringStack and StringQueue ADTs, ie write 2 classes that implement the 2 interfaces.

Implementation Hints:

- Your classes must **be named** StringStackImpl και StringQueueImpl.
- Implementation for both interfaces should be done with the use of a singly linked list.
- Each item insertion or deletion (ie each execution of the push and pop methods in the stack, and put and get in the FIFO queue) must be completed in time O(1), meaning that the time of those operations will be independent of the number of items in the queue. Similarly, the size method should be executed in O(1).
- When the stack or queue is empty, the methods that read from these structures should throw a NosuchElementException type exception. This exception belongs to the Java core library. Import it from the java.util package. Do not create your own exception.
- You can use the singly linked list that was shown in the lab or write your own list, or use only Node objects within the stack /queue class. To get better acquainted, we suggest you start from the beginning and write your own classes (you will certainly not lose points by using the lab code though).
- **Optional:** You can use generics to be able to handle stacks and queues on any type of object. There is a 10% bonus to those who use generics
- **You are not allowed to use built in implementations of list, stack, queue structures from the Java library (eg Vector, ArrayList, etc.)**

Part B [30 points]. Using the implementation of the stack in Part A, write a client program that will cross a maze in order to find the exit. Your program should get as input a .txt file, which will contain the maze in the form of a character table, whose dimensions will be n x m, where n,m are integers. The array can only contain the characters 0, 1 and E, where E will only be at one point of the array and will indicate the entrance to the maze. When someone enters the maze, they can move horizontally or vertically (but not diagonally) in any direction that contains 0 (see example below). If you reach the table border (first or last line and first or last column), and you find a 0 value, then you have reached an Exit of the maze. It is possible for multiple exits to exist. Your program should print the coordinates of the exit it found, and if there is no way to exit the maze, it should print an appropriate message.

Example: Input should be a txt file that looks like the example shown below.

```
9  7
0  3
1  1  1  E  1  1  1
1  1  1  0  1  1  1
1  0  0  0  1  0  1
1  0  1  0  1  0  0
1  1  1  0  1  1  1
1  0  0  0  0  0  1
1  0  1  1  1  0  1
1  0  1  1  0  0  1
0  1  1  1  0  1  1
```

In the first line, the dimensions of the maze are given (here n =9, m = 7). In the second line the coordinates of the entry point are given, in this example it is in line 0 and column 3 (assuming we start from 0 to n-1 and 0 to m-1). In this example, the exploration will start by moving down. If you turn left (as you look at the table), you will see that you will soon reach a dead end and you will have to go back. Continuing the exploration, we can see that a possible exit that can be reached is the one with coordinates (8,4).

Implementation Hints:

- Your program must be called Thiseas.java.
- You must make use of the implementation of the stack from Part A. The use of the stack will help you to implement the Exit search by **backtracking**. Think about what you need to do when you reach a dead end, and how you will be able to keep searching.
- To check if your program operates as expected, it is good to make some different mazes with different characteristics (eg multiple exit, no exits, larger dimensions, etc) and run your code with these inputs.
- If you save the maze to a character table, you are allowed to make changes in the table (eg if you want to use another character to indicate that you have already visited a location while running the program). Besides that, however, you should make good use of the stack.
- Your program will be examined with .txt files that look like the one above. If there is an error in the input data, the program must end by printing an appropriate message (eg if the rows and columns that are read in the first entry row do not match the table that is read after or if there is no E in the maze, etc.
- You must specify the entire path for the .txt input file, e.g. if you run it from the command line, the execution of the program will be as follows:

> java Thiseas path_to_file/filename.txt

Part C [30 points]..The implementation of the StringQueue interface with a singly linked list, must use 2 variables as pointers to the head and tail (named head and tail), so that you can properly perform insertions and deletions in the queue. What is required in Part C is to create a new implementation of the FIFO queue, using only 1 of those pointers.
Hint: Use a circular linked list instead of a singly linked list.

Implementation Hints:

- Your class must be called *StringQueueWithOnePointer.java*.
- Insert and delete operations should be done in O(1) and in general all instructions listed for Part A apply here as well.

Part D – Project Report [10 points]. Prepare a short report in a pdf file (Do not submit Word or txt files) that is named project1-report.pdf. In this report you will:

a. Briefly explain how you implemented the interfaces in Parts A and C. Especially for Part C, explain how you can avoid using 2 pointers for the beginning and the end of the queue (3 pages at most).
b. Explain how you used the implementation from Part A to build the required program in Part B (3 pages at most).

Additional Notes

Your program should not have any syntax errors so it can be compiled. Programs that can't be compiled lose 50%.

Project will consist of:

1. The source code. Place the java files you have created in a folder named **src.** Name all classes as it was requested. Also, be sure to include any other source code files needed for a successful compilation. Be sure to add explanatory comments where you think it is necessary.
2. The report.