

## Bonus Assignment: Maze Problems

**Due:** Sun, 12 Dec at 5pm.

In this assignment, we are going to solve two problems by using the knowledge of stack, set/file, and Exception. Study the following problem descriptions and prepare your solutions for the two questions. Submit the solutions on the Blackboard website by Sunday, 12 December 2021.

For each program add sufficient java doc comments to explain the logic of the codes throughout the program. Use exception handler whenever appropriate. In addition, add screenshots that will show the output of each of the programs.

### The Maze Background:

By using a 2D array, it is possible to represent a maze. For example, in the following, we have lines of text that represents a maze. These values can be read from a file and loaded in a 2D array of characters. In this case the maze has 8 rows and the exploration will start at the cell 3, 4 as indicated by the character @ in the lines of text. You can assume that there will be only one start position in the given lines of text.

```
#####  
#       #  
####  ###  
      @  #  
####  ###  
#      ###  
####  ###  
#####
```

As explained during the lecture, the maze that has been provided does not have a loop. In order to conduct the exploration to determine the exit from a given position we need to use stack data structure. As you explore possible paths from the given position, we will eventually come to an intersection. From here, we can attempt to explore one of the paths. However, we need to save the other paths from this position. In case, the current path does not lead to the exit, we can go back and try one of the other choices.

Therefore, we need to use a stack to remember the paths that still need to be tried.

Let us understand the technique by using an example. In the following, the exploration begins at position (3, 4). From here, there are four possible paths. We push them all on a stack (as object of the pair class). Afterwards, we pop off the topmost one, traveling north from (3, 4).

Now, following this path leads to position (1, 4). Here, we now push two choices on the stack, going west or east. Both of them lead to dead ends.

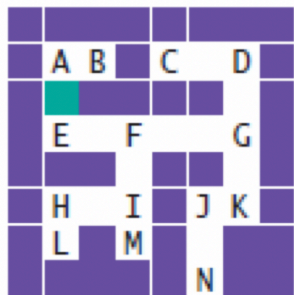
At this point, we pop off the path from (3,4) going east from the stack.



Please note, this algorithm will work if there are no cycles in the maze. You never circle back to a previously visited intersection.

### Sample Input & Output

The input to the program would be lines of text from a file as explained earlier. However, the paths will be labelled as the following.

	##### #AB#C D# #^#### # #E F G# ### ## # #H I#JK# #L#M# ## #####N##
---	--

The start position in this case is (2,1). Assuming that we start at the marked point and push paths in the order West, South, East, North then the output of the program would be: A B E F G D C K J N

### Question : Backtracking Maze [BasicMaze.java]

The program will read the lines of text from a file named "maze.txt". The program will load these lines of text in a 2D array of characters. By analyzing the text the program will further determine the start position. By employing the algorithm explained above, the program will then output the labelled path that leads to the exit. You can assume that there will be (a) only one start position, (b) no loop in the maze, (c) one exit from the maze, (d) maze will be labelled. Unlabelled path entries do not have to be displayed. The program must use the algorithm demonstrated above.

#### Backtracking Maze | Total points: 10

- [2] The program reads lines of texts from a text file named maze.txt
- [6] The program employs the algorithm by using a stack and calculates the path to the exit
- [2] The program uses exception handler throughout the code

### Question : Maze Loop [MazeLoop.java]

Now, in this iteration of the problem, let us consider the scenario where there are loops in the maze. A maze with loop is shown in the following figure.

	A	B		C	Y
				X	P
	E	V	F	Z	O
			U		
	H	R	I	S	J
	L		M		Q
				N	

```

#####
#AB#CYD#
#^##X#P#
#EVFZOG#
###U###
#HRISJK#
#L#M#Q##
#####N##

```

In this maze problem, there can be one or more loops. For example, when we reach at Z, we push X and O to the stack. We finally reach to Z via G path again. In order to avoid looping through the maze, we can create a set variable in our program. By using a set variable, we are going to keep track of the visited path items, in order to avoid loop.

When we visit a path item and also when we pop a path item from the stack, we would add those path items in the set variable. Now, let us try to explore the maze by using this new version of the algorithm.

To start, the set variable V is set to empty. From the start position, we would add {E, A} to the stack since the set variable is empty. We now pop A and add it to the set variable  $V = \{^, A\}$ . In addition, the path elements that we visit are also added in the set variable. This path leads to dead end. We pop E and add it the set  $V = \{^, A, E, B, V, F\}$ . When we reach at F, we add U, and Z in the stack as it does not exist in the set. We pop Z and add it to the set variable,  $V = \{^, A, E, B, V, F, Z\}$  and stack  $Stk = \{U\}$ . From, Z, we add O, X to the stack they do not exist in the set variable V. We pop, X and add it to the set variable. Here,  $V = \{^, A, E, B, V, F, Z, X\}$  and stack,  $Stk = \{U, O\}$ .

When we arrive at Z again, we can no longer add F and X to the stack as they exist in the set variable,  $V = \{^, A, E, B, V, F, Z, X, C, Y, D, P, G, O\}$ . Hence, we are going to pop O; O can not be added to the set variable, hence we do not explore this path again.

We now pop U, add it to the set variable and arrive at the intersection I. Here,  $V = \{^, A, E, B, V, F, Z, X, U, I\}$ . We add R, S in the stack,  $Stk = \{R, S\}$ . We pop, S and arrive at the intersection J. Here the set variable is,  $V = \{^, A, E, B, V, F, Z, X, U, I, S, J\}$ . Now, we add Q, and K to the stack,  $Stk = \{R, Q, K\}$ . K leads to dead end. We pop Q, now the set variable is,  $V = \{^, A, E, B, V, F, Z, X, U, I, S, J, K, Q\}$  and stack,  $Stk = \{R\}$ . We now reach to the exit N.

Therefore, the output of the program will be, A B E V F Z X C Y D P G O U I S J K Q N.

The program will read the lines of text from a file named "maze.txt". The program will load these lines of text in a 2D array of characters. By analyzing the text the program will further determine the start position. By employing the algorithm explained above, the program will then output the labelled path that leads to the exit.

### Maze with Loops | Total points: 10

- [2] The program reads lines of texts from a text file named maze.txt
- [4] The program employs the algorithm by using a stack and calculates the path to the exit
- [2] The program avoids a loop by using a set variable that keeps track of visited path
- [2] The program uses exception handler throughout the code

Note, the instructor is going to use different input files to test the programs. In the test input file, the length of the each lines of text can be more than 20.

Criterion	Details	Deductions
Classes	At minimum, we need the BasicMaze.java, MazeLoop.java files.	-10 x: The required Java files not submitted -5 x: The program does not provide the desired output -3 x: screenshot of the program run was not attached
Code quality	Identifier names, class names, proper use of public/private, ample comments in main, etc.	-15: incoherent, inconsistent coding style -10: comments were not used throughout the code - 5 x: exceptions were not handled throughout the code
Test	Note, whether the program compile and run	<b>-30: The portion of the code of the assignment is a copy. The midterm score would be 0.</b> -10 x: The program does not compile