Robotique
CRC
Robotics

CRC

PROGRAMMING COMPETITION

PRELIMINARY PROBLEM 3

A FEW NOTES

- The complete rules are in section 5 of the rulebook.
- You have until **Sunday, January 22, 11:59 pm** to submit your code
- Feel free to use the programming forum on the CRC discord to ask questions and discuss the problem with other teams. That's what it's there for!
- **We are giving you quick and easy-to-use template files for all languages allowed. Please use them.**

USING THE TEMPLATE FILE

- Basically, the template file uses your solve() code to transform a test case into its output and allows you to quickly check if it did what was expected or not. **All your code (except additional functions and classes, which should go right above it in the file) should be written in the solve() function.**
- **Do not touch anything lower than the solve().** There will be a list of input and a list of expected outputs, and a showing of how they would look in the problem statement for your convenience. There will also be a function that checks your answers by looping over all test cases of that input list. **Your solve() takes only one test case (arrays/lists in this case) at a time.**
- You can swap the **console's language to French** at the top of the file.

STRUCTURE

Every problem contains a small introduction like this about the basics of the problem and what is required to solve it. Points distribution is also given here for the preliminary problems.

Input and output specification:

In these two sections, we specify what the inputs will be and what form they will take, and we also say what outputs are required for the code to produce and in what format they shall be.

Sample input and output:

In these two sections, you will find a sample input (that often has multiple entries itself) in the sample input and what your program should produce for such an input in the sample output.

Explanation of the first output:

Sometimes, the problem might still be hard to understand after those sections, which is why there will also be a usually brief explanation of the logic that was used to reach the first output from the first input.

A-MAZE-D

As a traveling merchant, you're always taking the fastest shortcuts possible, traveling through tall grass and hills in order to get to your destination quicker. Unluckily for you, as you are innocently going through a cornfield today, you realize that it is in fact a maze. You will have to get out of this to resume your profit-making. However, after a bit of research on the terrain, you also realize that you lost a package on your way through. Your task will be to recover the lost package and to extract to the exit of the maze.

The package you are trying to find will be represented by a dollar sign ($). The corn rows will be hashtags (#) that cannot be crossed. You will be starting at the top left corner of the maze (0, 1) and will be needing to reach the bottom right corner (y,x-1). The spaces you can move to are occupied by blank spaces ( ). To indicate the path you will be taking through the maze, you must print the sequence of movement actions to get to the package, and then to get to the exit. Each movement action moves you one space in the direction specified by a letter as follows: “U” for up, “D” for down, “L” for left and “R” for right. These actions will constitute your answer.

Scoring:

You will get **5 points for getting to the package** (regardless of getting to the exit) and **5 points for reaching the exit** (with or without the package). Five additional points will be awarded according to the number of moves your solution has compared to the other teams' solutions. For your number of moves, it will be similar to the *Prelim2* problem with 5 different mazes. Your number will be compared to the best number of moves found by other teams for that same maze, The maximum of points awarded for the problem is 15.

For comparison with the other teams, your number moves will only be taken into account if you recover the package AND reach the exit. Every single one of the 5 tests will reward 1 point for its best solution found among the teams having completed it. This reduces to 0 in an inversely proportional fashion. So, if the best team solution for a given maze is 40, all teams having that number of moves will be awarded 1 point. If a team finds a solution with 60 moves, this team will receive 0.67 points out of 1 (40/60 = 0.67). If another team finds a solution with 80 moves, it will be awarded 0.5 points (40/80 = 0.5). If another team finds 50 moves, they will get 0.8 points (40/50 = 0.8) and so on. Note that tests used to determine scoring will be different than those contained in the template files but will be of the same format.

$$points = 5(package\ recovered) + 5(exit\ reached) + sum\ 5\ tests(min_{found}/nb_{moves})$$

Input specification:

Your code will receive two integers WIDTH, corresponding to the width of the labyrinth, and HEIGHT, corresponding to its height. Border walls are included in those dimensions. The labyrinth will be given to you as a string array called maze. Every string is a row. Row maze[0] contains the first line with the entrance to the maze and its superior wall, for example. Each character (char) is a square of the maze. Squares on which you can move are blank spaces. Corn walls are hashtags (#) and the package is a dollar sign ($). Given labyrinths are strictly rectangular.

Output specification:

Your solve() method will have to return an action sequence in the form of a string, allowing us to test the sequence's validity. Moves are indicated by capital letters and the answer must only contain either "U" for upwards, "D" for downwards, "L" for going to the left or "R", for the right.

First input of template file (Python example) :

```
WIDTH = 25
HEIGHT = 21
maze = ["# ########################",
        "#   #   #                #",
        "# # # # # # # ### # ### #",
        "# # # # #       # #   # #",
        "# # # # ####### # ### ###",
        "#           # # #  $    #",
        "# ### ### # # # # ### # #",
        "# #   #   #         # # #",
        "# ### ### ######### # ###",
        "#   # # #   #   #   #   #",
        "### # # # # ### # # # # #",
        "#     # # #     # # # # #",
        "# ### # # # ### # # # # #",
        "#           # # #       #",
        "# ### # # # # # # #######",
        "# #   #           # # # #",
        "# ### ########### # # # #",
        "#     # #   #   #   #   #",
        "# ### # # # # # # # # ###",
        "#                       #",
        "######################## #"]
```

Output example for the first input:

This is only one of the many possible answers (and not necessarily the most efficient)
answer =
“DDDDDRRRRRRRRRRDDRRRRRRUURRRRDDDDDDDDLLLLDDDDDDRRRRRRD”

Example of a first output:

A visualization of the path taken is just below.
Starting from the opening at the top of the maze (0,1), we go downwards by 5 squares (“DDDDD”), then go to the right by 10 squares. We continue the journey by going downwards 2 times, to the right 6 times before going back up twice (“UU”). We then proceed to go twice to the right to recover the package! Going twice to the right, 8 times down, 4 times to the left, 6 times down, 6 to the right and, finally, once down to reach the exit.

```
Your number of moves is: 54
This is the maze with your path indicated as * and the conflicts
indicated as @
#*########################
#*  #   #                #
#*# # # # # # ### # ### #
#*# # # #       # #   # #
#*# # # ####### # ### ###
#***********# # #**$**  #
# ### ### #*# # #*####*# #
# #   #   #*******  #*# #
# ### ### ######### #*###
#   # # #   #   #   #*  #
### # # # # ### # # #*# #
#     # # #     # # #*# #
# ### # # # ### # # #*# #
#           # # #*****  #
# ### # # # # # #*#######
# #   #          *# # # #
# ### ###########*# # # #
#     # #   #   #*  #   #
# ### # # # # # #*# # ###
#                *******#
########################*#
Congratulations you have picked up the package
Congratulations you have reached the end of the maze
```