CSCI-P 538 Fall'16 Project 1: Warm-up Programming

A Deadline

September 6 2016 23:59:59 EDT. This is a hard deadline and no extension will be given. Any clarification queries should be sent to `p538fall16-l@list.indiana.edu`.

B Problem Description

The purpose of this project is giving you warm-up of your favourite programming language (C/C++/Java), as well as the basic knowledge of algorithm and data structure. They are prerequisites of the remaining projects of this course.

You are asked to find a solution to a maze (labyrinth) given to you. A maze is "a puzzle in the form of a complex branching passage" (from Wikipedia) through which you will try to find a solution, or report that a solution does not exist. In this project, a maze is a character matrix consisting of four types of blocks:

- The starting point (source) is denoted by "`S`". There is one and only one source in a maze.
- The end point (destination) is denoted by "`D`". There is one and only one destination.
- A wall is denoted by "`#`". A route cannot pass a wall.
- An empty block is denoted by a dot ("`.`"). A solution always begins at the source, passes zero or more empty blocks, and ends at the destination.

Below shows an example of a 13 by 13 maze. The size of a maze is at most 5,000 by 5,000 *i.e.,* all inputs in this project contain no more than 5000 lines, and each line contains no more than 5001 characters (including the line separator \n).

```
#############
S.#.........#
#.#.#######.#
#...#.....#.#
#####.###.#.#
#.....#.#...#
#.#####.#####
#.#...#.....#
#.#.#.#####.#
#...#.#...#.#
#####.#.#.#.#
#.......#...D
#############
```

B.1 Task 1

Read in a maze and determine whether the maze itself is legal. Below are examples of an illegal maze:

- Contains characters not belonging to the following: "S", "D", "#", ".", and newline ("\n").
- Contains no source or destination.
- Contains multiple sources or destinations.
- Is not an $m * n$ character matrix where $n > 0$ and $m > 0$.

Note a legal maze may or may not have a solution. Also a legal $m * n$ maze must contain exactly m lines, and each line (including the last line) must contain exactly n characters followed by a line separator, which is \n (`0x0A` in Linux). There must not be any empty lines in the input.

Your program will read from the standard input (`stdin`) a maze that is either legal (example shown above) or illegal. Your program should write to the standard output (`stdout`) a single line of either "YES" (without quotes), if the matrix is legal, or "NO", if the matrix is illegal. All outputs in this project are case sensitive. More details of handling input and output are given in Section C.

B.2 Task 2

Read in a legal maze to determine if a solution exists. A solution is a route that always begins at the source, passes zero or more empty blocks (and never passes walls), and ends at the destination. An empty block can be stepped on more than once. There are four possible moving directions: up, down, left, and right. Moving diagonally in one step is prohibitive, and moving off the maze (*e.g.*, moving upwards or leftwards from the top-left block) is also impossible. The example above has a solution, but the one below does not.

```
##############
#............#
#..###.......#
#..#S#...D...#
#..###.......#
#............#
##############
```

Your program will read from the standard input (`stdin`) a legal maze. Your program should write to the standard output (`stdout`) a single line of either "YES", if the maze has a solution, or "NO", if the maze does not have a solution.

B.3 Task 3

This task is largely similar to Task 2, except that you are further required to find the optimal solution *i.e.,* the route with the shortest length, for a maze.

Your program will read from the standard input (`stdin`) a legal maze. If the maze does not have a solution, your program should write to the standard output (`stdout`) a single line of "NO" (same to Task 2).

If the maze does have a solution, your output needs to contain a single line of the best solution. A solution is a sequence containing the following characters: "U" (up), "D" (down), "L" (left), and "R" (right), each corresponding to the move direction of one step. The whole sequence thus represents a possible route from the source to the destination. You need to find the shortest route *i.e.,* the sequence with the minimum length. If multiple such routes exist, output any one of them (but only one).

In the first example, there is only one shortest route whose sequence is:

```
RDDRRUURRRRRRRRDDDDLLUULLLLDDLLLLDDDDRRUURRDDDDRRUURRDDRRR
```

B.4 Task 4

This task is similar to Task 3. Your program takes a legal maze as input, and outputs "NO" if solution does not exist, or a single line of the best solution (a move sequence with the shortest length) whose format is the same as that of Task 3. In other words, your code needs to handle everything in Task 3, plus a new feature described below.

In this task, we introduce a new type of block called *teleporter*, which instantaneously transports the player from one location to another. Teleporters always appear in pairs, and each pair is denoted by a number from 0 to 9. In other words, there can be at most 10 pairs of teleporters in a maze. Note teleporters do not appear in Task 1, 2, and 3.

Consider the maze shown below. There are two pairs of teleporters: 1 and 9. Whenever the player steps on one of the "1" blocks, she immediately moves to the other "1" block. The same happens to the "9" blocks. The player can thus leverage either teleporter pair to reach the destination. The move sequence can be `LD` (if Teleporter pair 1 is used) or `RUULLL` (if Teleporter pair 9 is used). Your program must output `LD` since it is the optimal solution with the shortest sequence length.

```
##############
#............#
#.#####..1...#
#.#1S9#..D...#
#.#####......#
#...........9#
##############
```

You can assume the input maze is always legal, and teleporters always appear in pairs.

Hint: Since Task 2, 3, and 4 are similar, you may want to make them share the same code base. Doing so helps reduce your implementation overhead.

C Submit Your Program

You must submit a single C/C++/Java source code file called `project1.c`, `project1.cpp`, or `project1.java`. Take C++ as an example. To compile your program in Linux, type:

```
g++ ./project1.cpp
```

It will generate an executable file named `a.out`. Do not upload the executable.

Your program takes a single argument of the task number (1 to 4). For example, the following command runs Task 3 (assuming the compiled executable is `a.out`):

```
./a.out 3
```

The parameter format for C and Java programs is the same as the above. If you use C/C++, we strongly recommend you work on Linux (*e.g.,* Ubuntu or Fedora) with an update-to-date GNU C/C++ compiler (gcc/g++ version 4.3 and above). Do not use other C/C++ compilers that might be incompatible to gcc/g++. For Java, please use JDK version 7.0 or above.

D Test Your Program

We use an automated system to test your program, so you need to make sure your output strictly conforms to the required format (*e.g.,* no additional white spaces, no extra empty lines, output "YES" instead of "Yes" or "yes"). The output is case-sensitive.

The test system uses input/output redirection for automation. For example, it can issue the following command to force your program to read from `testcase01.in`, and write to `testcase01.out`. You can also use this to test your code.

```
./a.out 3 < testcase01.in > testcase01.out
```

However, in your program, you must read only from standard input (`stdin`) and write only to standard output (`stdout`). Failure to do so will force us to manually test your program, and you will lose half of the points because of that.

To help you ensure the correct output format, we provide several sample test cases. In the `testdata` folder, `taskx.in.y` is the y-th sample input for task x, and its corresponding sample output is `taskx.out.y`. You can compare your output and the sample output using `diff`:

```
diff [your_output_file] [sample_output_file]
```

If they are identical, `diff` outputs nothing, otherwise it will display the difference. In the latter case, your implementation might be wrong – fix it before submitting your code. Note that for Task 3 and 4, when a maze has multiple optimal solutions, your code might produce an output that is different from the sample output. This is acceptable, and can be handled by our test system.

The running time of your program will be measured. You have 3 second for each test case.

E Grading

	# test cases	Points per test case	Total points
Task 1	5	4	20
Task 2	5	4	20
Task 3	6	5	30
Task 4	6	5	30
Total	100 Points		

F Honor Code

Students must follow the IU honor code (`http://www.iu.edu/~code/code/responsibilities/academic/index.shtml`). This project is an individual assignment, and no collaboration among students is allowed. In no case may your code be copied from another student or a third-party source on Internet. We will use an anti-plagiarism software to detect code "shared" among students. Any violations of the honor code will be dealt with strictly, including but not limited to receiving no credit for the entire project.