Middle East Technical University
Department of Computer Engineering

# CENG 140

Spring 2023

Take Home Exam 2

## REGULATIONS

**Due date:** 24 May 2023, Wednesday, 23:55 *(Not subject to postpone)*

**Submission:** You will use OdtuClass system for the homework just like Lab Exams. You can use the system as editor or work locally and upload the source files. Late submission **is not** allowed.

**Team:** There is **no** teaming up. The homework has to be done/turned in individually.

**Cheating:** All parts involved (source(s) and receiver(s)) get zero and face disciplinary action. ChatGPT and solutions from the WEB (if any) will be considered cheating.

## INTRODUCTION

A labyrinth is a web of passages. These passages might be organized in various geometries. Good examples of real life are corridors of buildings, the railroad system or the roads of a city. One of a very common problem on labyrinths is to find a path, if exists, from a starting point to an end point. usually the end point of which is defined as the *exit*.

## PROBLEM

The labyrinths that we will consider are rectangular areas of grid structure. Each grid cell is either of type *wall* or of type *passage*. The north-west corner is the origin (0,0) of the coordinate system, where increasing $x$ is in the west→east direction and increasing $y$ is in the north→south direction.

An *exit* is a *passage cell* on the outermost part of the labyrinth. There may be more than one such exists.

There is no regulation about the organization of the wall/passage cells.

The labyrinth and the starting point coordinate is defined in the first line read from the standard input. Here is an example of a possible input file:

```
5 9
XXXXXXXXXXXXXXXXXXXX XXXX
X XXXX          X XX  XXXX
X      X XXXXXX   XX    XX
X XXXXXX   XXXXXXXXX XX XX
X XX   XXXX X     XX X  XX
X XX X  XXX XX XXXXX XX XX
X X  XX   X    X        XX
X X   XXX XXX XXXXXX XXXXX
X XX   XX     X   XX XX XX
X XX     XXXX X X XX XX XX
X XX XX  XX   XXX XX    XX
X XX  X XXX       XX  X XX
X XX    XXX XXXXXXXXX   XX
X XXXX   XX           XXXX
XXXXXXXXXXXXXXXXXXXXXXXXX
```

The first line contains the $(x, y)$ coordinates of the start point. The next lines are the definition the labyrinth. As is obvious, '`X`' is marking a wall cell, where blank is representing a passage cell.

The program that you will write will find and mark a path that will lead to the exit. A solution to the above given input is:

```
XXXXXXXXXXXXXXXXXXXX*XXXX
X XXXX          X XX**XXXX
X      X XXXXXX   XX*   XX
X XXXXXX   XXXXXXXXX*XX XX
X XX***XXXX X     XX*X  XX
X XX*X**XXX XX XXXXX*XX XX
X X *XX***X    X    *   XX
X X **XXX*XXX XXXXXX*XXXXX
X XX * XX*****X   XX*XX XX
X XX *   XXXX*X X XX*XX XX
X XX XX  XX***XXX XX**  XX
X XX  X XXX*      XX *X XX
X XX    XXX*XXXXXXXXX*  XX
X XXXX   XX***********XXXX
XXXXXXXXXXXXXXXXXXXXXXXXX
```

The path will be marked by '`*`'. If the start point is in a closed vicinity and hence there is no way to any exit then you are supposed to mark all the interior of this vicinity by '`.`' characters. Let us assume the first line of the input would read as

```
3 2
```

Then the output printed to the standard output should look like:

```
XXXXXXXXXXXXXXXXXXXX XXXX
X.XXXX.........X.XX  XXXX
X......X.XXXXXX...XX    XX
X.XXXXXX...XXXXXXXXX XX XX
X.XX   XXXX X     XX X  XX
X.XX X  XXX XX XXXXX XX XX
X.X  XX   X    X        XX
X.X   XXX XXX XXXXXX XXXXX
X.XX   XX     X   XX XX XX
X.XX     XXXX X X XX XX XX
X.XX XX  XX   XXX XX    XX
X.XX  X XXX       XX  X XX
X.XX    XXX XXXXXXXXX   XX
X.XXXX   XX           XXXX
XXXXXXXXXXXXXXXXXXXXXXXXX
```

# SPECIFICATIONS

- The labyrinth is a rectangular.
- The labyrinth will contain at most 100000 cells. All row/column counts $\geq$3 that satisfy this restriction are possible.
- Walls are persistent. It is not possible to move to a wall cell.
- The start point is a passage cell.
- A valid path is made up of adjacent moves marked by '`*`' and made in one of the west, north, east or south directions. Diagonal moves are <u>not</u> accepted.
- Any valid path leading to any exit will be accepted. No need to search for shortest path.
- If no path exists then mark the interior of the vicinity (that you are stucked in) with '`.`' characters.
- Each test case will be tested with our own solution program not to cause a stack overflow. Our solution program will not be using an external stack (a stack structure implemented in the program).
- A maximal run time of 1 sec will granted. Normally your program shall require not more than 0.1 sec.
- You are strongly advised to hold the labyrinth information in a pointer array, where each element of the array is pointing to a `char` array that holds a row of the labyrinth. Certainly do the space allocation dynamically (by `malloc`).

# GRADING

- No partial grading for a test case. Also no eye inspection+grading of non-working code.
- Any modification of the walls will lead to a 0 (zero) grading, independent of the correctness of the solution path (be very careful about this).