CS300 – Summer 2020-2021 - Sabancı University

Homework #1 – Maze

Due: 25/07/2020, Sunday, 23:55

Brief Description

In this homework, first, you will implement a program which will generate a random maze of size MxN, where M represents the number of rows, and N represents the number of columns. Then, you will also need to implement a function which will find the path between designated entry and exit points in a given maze.

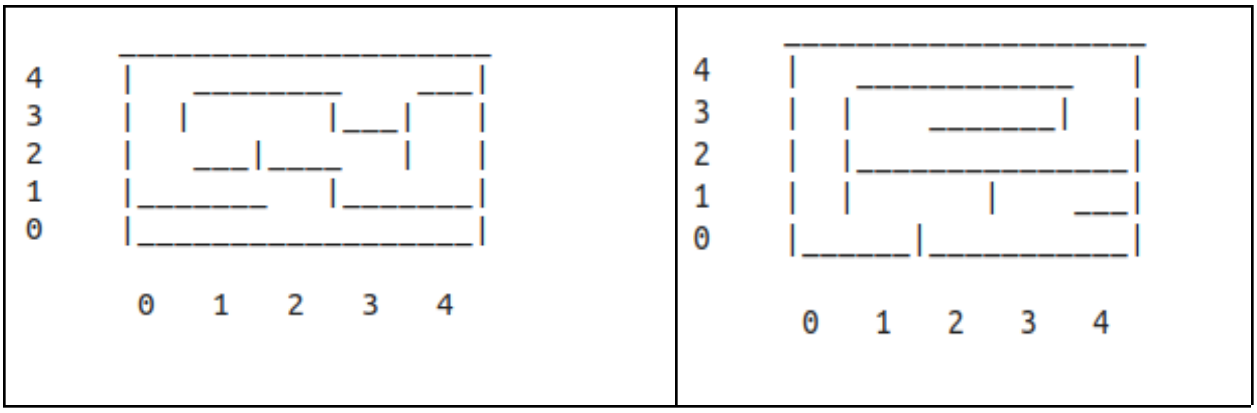
To do this homework, you are required to implement a **Stack** using a **LinkedList** data structure, i.e., you can not use vectors in the stack implementation. The stack class **MUST** be a template-based class. You may want to store basic data types, structs or classes in this stack.

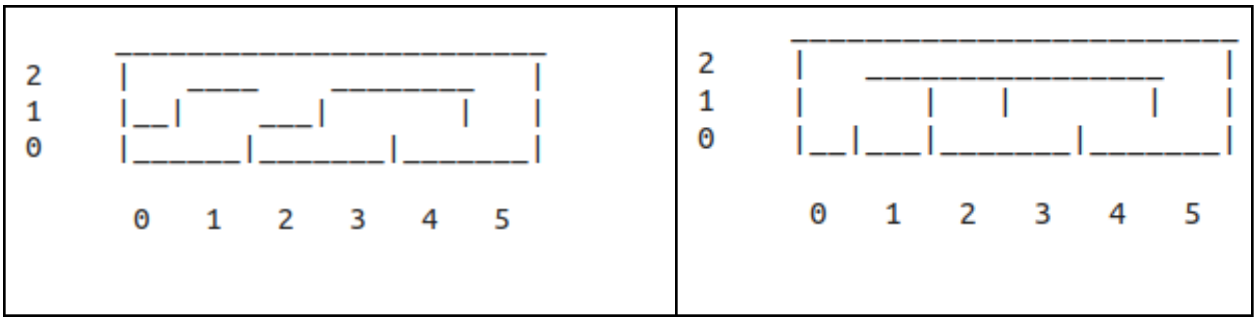
The Maze

The mazes are presented as MxN (M rows and N columns) 2D vectors as given below. The left bottom of the mazes are considered to be the (0,0) coordinate. There are 2 important rules for a given 2D vector to be a maze:

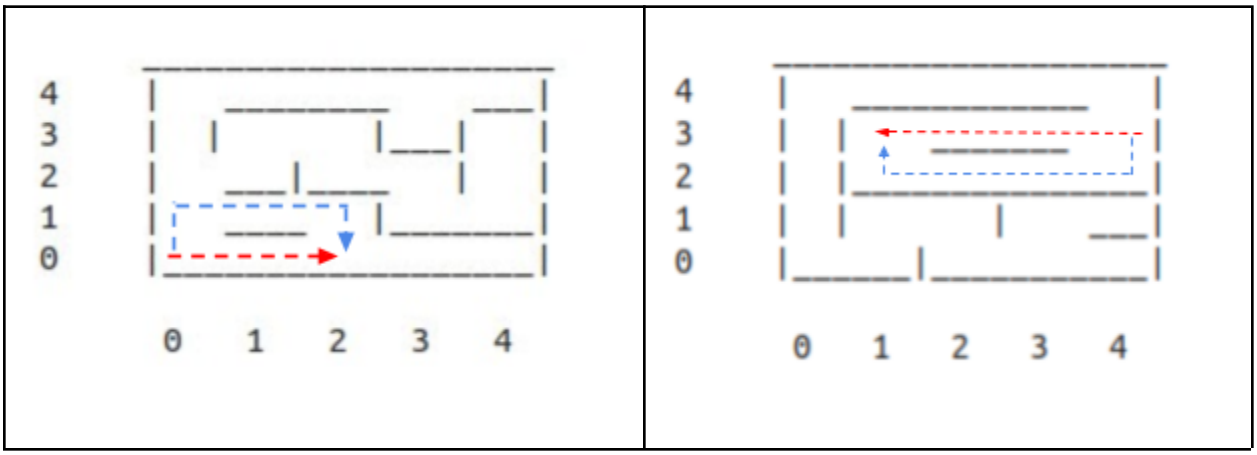
1) Every cell must be reachable from any other cell.
2) The reachability of cells from one to another must be unique, i.e., there must be only **<u>one</u>** path from a cell to another.

Some example mazes of size 5x5 and 3x6 can be found below. Note there are no entry or exit points yet.





We also share some invalid maze examples below. In both mazes, for some cell pairs (c1 ,c2), there are more than one path to reach from c1 to c2.



Program Flow

There are 2 phases of this homework;

1. The maze generation.
2. Finding a path for given entry and exit points.

In both phases, you **MUST** use the stacks to solve the problem!

The maze generation

In this phase, you need to generate K random MXN mazes using stacks for given M, N and K values. The general idea to generate a maze is to knock down walls between 2 cells, iteratively, until no unvisited cell remains. Hence, you need to knock down exactly MxN-1 walls. In each step, the maze rules (given above in the maze definition) must apply all the time.

The basic steps are:

1. Start with an initially empty stack.
2. Push (0,0) cell to your stack.
3. Use the top element (currentCell) in the stack to choose the next cell to be added to your maze.
4. Choose a random wall of the currentCell to add a new unvisited cell to the maze.

5. Knock down the wall and add the new cell to the stack.
6. If no wall exists to knock down for the currentCell, backtrack using the stack until you find a cell which has an unvisited neighbour cell.
7. Continue steps 3-6 until no cell remains unvisited.

Path discovery in a maze

Finding a path is almost the same as the maze generation process. First, the program will ask the user which maze should be chosen among the ones generated in the first phase. Then, the program;

1. Start with an initially empty stack.
2. Push the entry cell to your stack (will be entered by the user).
3. Use the top element in the stack to choose the next cell to visit.
4. If you cannot find any cell to go, backtrack using the stack until you find an unvisited cell to go.
5. Continue steps 3-4 until you reach the exit point.

Note that the path generated in this phase **MUST** be unique!

You can assume that:

- All the inputs entered by the user are valid, you do not need to validate them.
- The entry and exit points of the mazes are on the edges or corners of the mazes.

Input and Output

In the first phase, your program will ask the user to enter 3 inputs, namely, K, M, and N. You can assume that M and N are greater than 1, and K is a positive integer. However, your program should be able to generate huge mazes, too.

You will write your mazes into a file named as maze_mazeID.txt, where mazeID will go from 1 to K such as maze_1.txt and maze_4.txt. Your output format must be **exactly the same format** as we give below. In the first line, sizes of the maze, M and N, must be given. In the following lines, for each cell, the x and y coordinates of the cell, as well as the walls of the cell, must be given. If there is a wall on the left (l), right (r), up (u), or down (d), assign 1 to the wall, if not, assign 0. For example, in the maze_1.txt below, at (0,0) cell, there are walls on the left, right and down side, but not on the up side.

maze_1.txt

```
5 4
x=0 y=0 l=1 r=1 u=0 d=1
x=1 y=0 l=1 r=0 u=1 d=1
x=2 y=0 l=0 r=0 u=1 d=1
…
```

In the second phase (after the mazes are generated), your program will ask the user to enter 5 more inputs; mazeID, entryX, entryY, exitX, and exitY. Maze ID will be a number between 1 and K, i.e., 1 <= mazeID <= K, indicating the specific mazes that are generated in the first phase to be traveled. For example, if mazeID = 3, then, the program needs to find a path for the third maze. entryX and entryY are the coordinates for entry point, and exitX and exitY are the coordinates for the exit point. Note that, y coordinates becomes the row of the maze and x coordinates becomes the column of the maze. For instance, the point (2,4) refers to the 4th row and 2nd column of the maze.

The output will be written to a file named as maze_mazeID_path_entryX_entryY_exitX_exitY.txt. For instance, for the maze with mazeID = 2, entry = (0,0) and exit = (3,4), the file name will be maze_2_path_0_0_3_4.txt.

In the output file, the coordinates of the cells to get the exit point from entry point must be written line by line. See example output file below:

maze_2_path_0_0_3_4.txt

```
0 0
…
…
3 4
```

Debugging your code

We will provide a mazeDrawer program (mazeDrawer.exe for windows, mazeDrawer.linux for Linux and mazeDrawer.mac for macOS machines) for you to draw any given maze. To draw the maze for a given input file described above (maze_1txt), you must apply the following rules:

On windows machine:

1. Put the mazeDrawer.exe and libpwinthread-1.dll along with the input files into the same folder.
    a. Please note that mazeDrawer.exe and libpwinthread-1.dll need to **be in the same folder** for your program to work.
2. Click on mazeDrawer.exe.
3. Follow the instructions as needed.

On macOS or Linux machine:

1. Put the mazeDrawer (mazeDrawer.mac for macOS and mazeDrawer.linux for Linux) program and the input files into the same folder.
2. For macOS;
    i. Open a Terminal.app
    ii. Copy the folder with Command+C (this will copy the path to reach the folder)

iii. In terminal, type **cd**

iv. Use Command+V to paste the folder path and click enter

Example usage (assuming mazeDrawer.mac is in the following path):

cd /Users/oguzozsaygin/Homework1

For Linux;

i. Go to the folder.

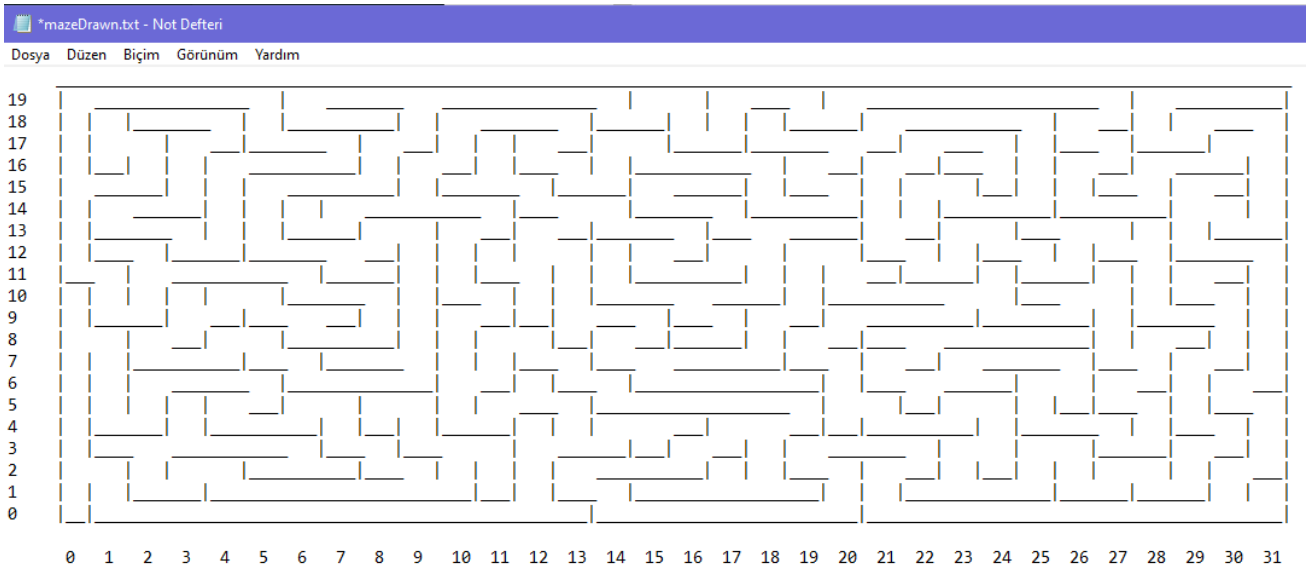
ii. Right click anywhere in the folder.

iii. Click open a terminal **.**

3. Execute the following command on the terminal by typing:
    a. For macOS;
    ./mazeDrawer.mac
    b. For Linux;
    ./mazeDrawer.linux
4. Follow the instructions as needed.

If you face any issues running the mazeDrawer program, please go to an office hour or ask your TA to explain it in the recitation.

You will be given the file maze_example.txt to draw the maze below as an example. Use it as a way to understand how to use the program. This program is for you to better visualize your maze, though you are not obligated, we highly recommend you to use it for debugging your code.



Sample Run

```
Enter the number of mazes: 5

Enter the number of rows and columns (M and N): 20 32

All mazes are generated.


Enter a maze ID between 1 to 5 inclusive to find a path: 1

Enter x and y coordinates of the entry points (x,y) or (column,row): 0 0

Enter x and y coordinates of the exit points (x,y) or (column,row): 31 19
```

General Rules and Guidelines about Homeworks

The following rules and guidelines will be applicable to all homeworks, unless otherwise noted.

How to get help?

You may ask questions to TAs (Teaching Assistants) of CS300. Office hours of TAs can be found here. Recitations will partially be dedicated to clarify the issues related to homework, so it is to your benefit to attend recitations.

What and Where to Submit

Please see the detailed instructions below/in the next page. The submission steps will get natural/easy for later homeworks.

Grading and Objections

Careful about the semi-automatic grading: Your programs will be graded using a semi-automated system. Therefore, you should follow the guidelines about input and output order; moreover, you should also use the exact same prompts as given in the Sample Runs. Otherwise the semi-automated grading process will fail for your homework, and you may get a zero, or in the best scenario you will lose points.

Grading:

- **We will grade your homeworks during the demo session. Each one of you must show that your code is running as expected and explain several parts of your code if necessary. Wait for an announcement for the demo scheduling.**
- Late penalty is 10% off the full grade and only one late day is allowed.
- **Having a correct program is necessary, but not sufficient to get the full grade. Comments, indentation, meaningful and understandable identifier names, informative introduction and prompts, and especially proper use of required functions, unnecessarily long program (which is bad) and unnecessary code duplications will also affect your grade.**
- Please submit your own work only (even if it is not working). It is really easy to find out "similar" programs!
- For detailed rules and course policy on plagiarism, please check out http://myweb.sabanciuniv.edu/gulsend/courses/cs201/plagiarism/

Plagiarism will not be tolerated!

Grade announcements: Grades will be posted in SUCourse, and you will get an Announcement at the same time. You will find the grading policy and test cases in that announcement.

Grade objections: Since we will grade your homeworks with a demo session, there will be very likely no further objection to your grade once determined during the demo.

What and where to submit (IMPORTANT)

Submission guidelines are below. Most parts of the grading process are automatic. Students are expected to strictly follow these guidelines in order to have a smooth grading process. If you do not follow these guidelines, depending on the severity of the problem created during the grading process, 5 or more penalty points are to be deducted from the grade.

Add your name to the program: It is a good practice to write your name and last name somewhere in the beginning program (as a comment line of course).

Name your submission file:

- Use only English alphabet letters, digits or underscore in the file names. Do not use blank, Turkish characters or any other special symbols or characters.
- Name your cpp file that contains your program as follows.
  "SUCourseUserName_yourLastname_yourName_HWnumber.cpp"
- Your SUCourse user name is actually your SUNet username which is used for checking sabanciuniv e-mails. Do NOT use any spaces, non-ASCII and Turkish characters in the file name. For example, if your SUCourse user name is cago, name is Çağlayan, and last name is Özbugsızkodyazaroğlu, then the file name must be:
  cago_ozbugsizkodyazaroglu_caglayan_hw1.cpp
- Do not add any other character or phrase to the file name.
- Make sure that this file is the latest version of your homework program.
- You need to submit ALL .cpp and .h files including the data structure files in addition to your main.cpp in your VS solution.

Submission:

Submit via SUCourse ONLY! You will receive no credits if you submit by other means (e-mail, paper, etc.).

Successful submission is one of the requirements of the homework. If, for some reason, you cannot successfully submit your homework and we cannot grade it, your grade will be 0.

Good Luck!
Gülşen Demiröz