

MTRN4110 20T2 Project Phase B Task Descriptions (week 4-6)

(Updated 20/6/2020)

1. Overview of the Course Project:

The main project of MTRN4110 20T2 is a simulation-based project adapted from the [Micromouse](#) competition. [Webots](#) will be used as the simulation platform throughout the course. You will design a mobile robot and implement a controller and a vision program to negotiate a maze autonomously in Webots. The project will contribute **55%** to your final mark of this course.

The project consists of four sequential phases:

- Phase A: Driving and Perception (week 1-3, 15%, individual)
- Phase B: Path Planning (week 4-6, 15%, individual)
- Phase C: Vision (week 7-9, 15%, individual)
- Phase D: Integration and Improvement (week 10-11, 10%, group)

This document will describe the tasks of **Phase B**.

2. Overview of Phase B – Path Planning:

The purpose of Phase B is to develop the path planning module for the maze-solving robot. You are required to complete the tasks of this phase by the end of week 6.

2.1. Expectations:

By the end of Phase B, you are expected to have been able to:

- read a map containing a maze layout and the initial location and heading of a robot from a text file;
- understand how to represent such a map in the program;
- implement a path planning algorithm to find an optimal path for the maze-solving robot;
- generate a sequence of motion commands and write it into a text file.

2.2. Learning Outcomes Associated with this Assignment:

- **LO1:** Apply relevant theoretical knowledge pertaining to mobile robots including locomotion, perception and localisation using onboard sensors, navigation and path planning, for practical problem-solving
- **LO3:** Demonstrate practical skills in mechatronics design, fabrication, and implementation

3. Phase B Task Descriptions:

In this phase, you are supposed to implement a **new** controller for the robot that has been used in the previous phase. The controller should be able to plan an optimal path for the robot given a map of the maze and the initial state of the robot.

Technically, the implementation can be done independently of Webots as no simulations are involved in this phase. However, for the sake of integration at the final stage and convenience of assessment, you are **REQUIRED** to implement the controller in Webots (refer to section 4).

The controller should complete the following tasks once started.

3.1. Read a map from a text file and display it in the console

You will be given a text file named "Map.txt" that contains a map of the maze layout and the initial location and heading of the robot.

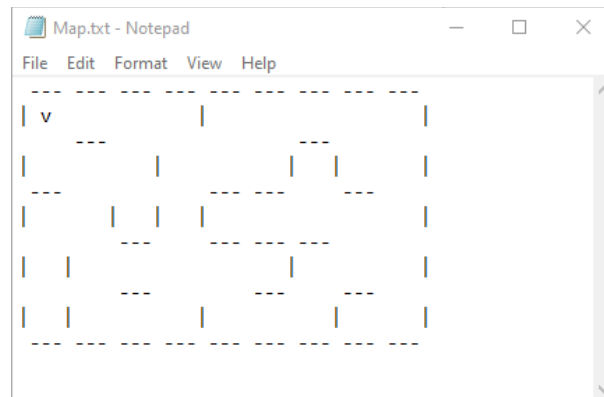


Fig. 1. An example map in the text file containing the maze layout and the initial location and heading of the robot

As shown in Fig. 1, each horizontal wall of the maze is described by three dashes "---", each vertical wall of the maze is represented by a vertical bar "|", and the location and heading of the robot is indicated by a sign ("^" - north, ">" - east, "v" - south, "<" - west). Note that throughout this course we refer to the top, bottom, left, and right of the maze as North, South, West, and East, respectively. All the characters that could appear in the text file are illustrated as below, except for spaces and line breaks (only lowercase v is used).

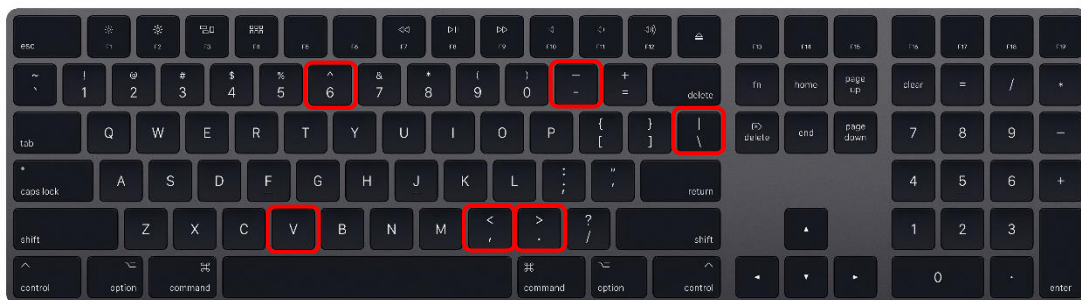


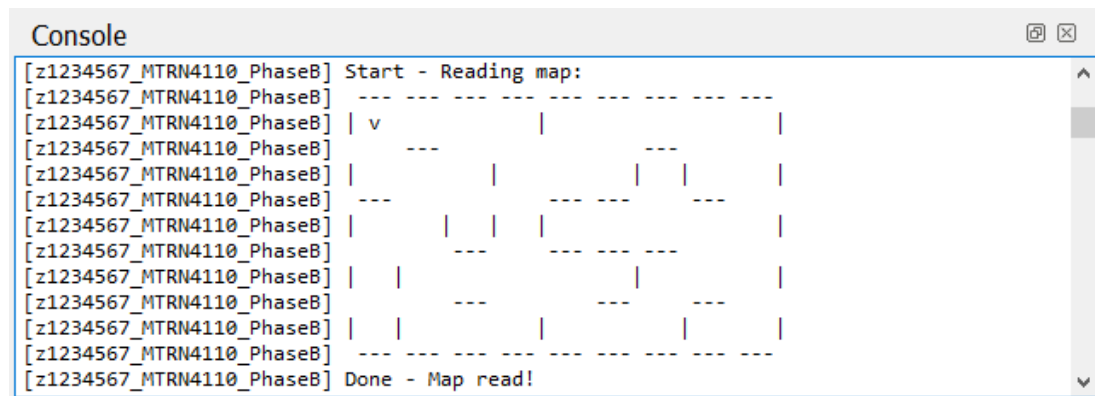
Fig. 2. All the characters that could appear except for spaces and line breaks (only lowercase v is used)

All the empty cells are represented by 3 spaces " ". For the cell that has a robot in it, it is represented by 1 space plus 1 sign ("^" - north, ">" - east, "v" – south, "<" - west) followed by 1 space, making a total of 3 characters as well.

All the vacant walls are represented by spaces and the number of characters always match when there is a wall in place.

All the blocks between adjacent walls are depicted by spaces.

You should read the information from the text file to your controller and display the map in the console. For example,



```

Console
[z1234567_MTRN4110_PhaseB] Start - Reading map:
[z1234567_MTRN4110_PhaseB]  ---  ---  ---  ---  ---  ---  ---  ---  ---  ---
[z1234567_MTRN4110_PhaseB] | v      |      |      |      |      |
[z1234567_MTRN4110_PhaseB] |  ---  |      |      |      |      |
[z1234567_MTRN4110_PhaseB] |      |      |      |      |      |
[z1234567_MTRN4110_PhaseB] |  ---  |      |      |      |      |
[z1234567_MTRN4110_PhaseB] |      |      |      |      |      |
[z1234567_MTRN4110_PhaseB] |      |      |      |      |      |
[z1234567_MTRN4110_PhaseB] |      |      |      |      |      |
[z1234567_MTRN4110_PhaseB] |      |      |      |      |      |
[z1234567_MTRN4110_PhaseB] |      |      |      |      |      |
[z1234567_MTRN4110_PhaseB] |  ---  |      |      |      |      |
[z1234567_MTRN4110_PhaseB] |      |      |      |      |      |
[z1234567_MTRN4110_PhaseB] |  ---  |      |      |      |      |
[z1234567_MTRN4110_PhaseB] Done - Map read!

```

Fig. 3. Reading map into program from text file

You can choose to **hard-code** the map into your program and **forfeit** the marks associated. To do so, you need to **define a variable to store the map** at the beginning of your program (so that we can replace it when assessing your solution). You should also **EXPLICITLY** indicate hard-coding the map in the comment block at the top of your program. Failing to do so would affect the assessment of your submission.

```

/*
 * File:          z1234567_MTRN4110_PhaseB.c
 * Date:
 * Description:
 * Author:
 * Modifications:
 * Hard-coding: The map is hard-coded.
 */

```

Fig. 4. EXPLICITLY indicate hard-coding in the top comment block of your program

3.2. Find ALL the shortest paths for the robot from its initial location to the centre of the maze

After reading the map, you should run a path planning algorithm to find **ALL** the **shortest** paths from the initial location of the robot to the centre of the maze.

Here the shortest paths mean the paths in which **the number of cells** that the robot will traverse is the smallest among all the possible paths.

For the example above, there are 6 such shortest paths with which the number of cells traversed is 17 (and no other paths can achieve a number smaller than 17). Display each of the path in the console (this is the recommended format; other formats may also be acceptable if they **clearly** indicate the path in the map):

```

Console
[z1234567_MTRN4110_PhaseB] Done - Map read!
[z1234567_MTRN4110_PhaseB] Start - Finding shortest paths:
[z1234567_MTRN4110_PhaseB] path - 1:
[z1234567_MTRN4110_PhaseB] ---
[z1234567_MTRN4110_PhaseB] | v  15  14  13| 10  9  8  7  6 |
[z1234567_MTRN4110_PhaseB] |      ---
[z1234567_MTRN4110_PhaseB] |      | 12  11 |  |  | 5 |
[z1234567_MTRN4110_PhaseB] |      ---
[z1234567_MTRN4110_PhaseB] |      |  |  | 0  1  2  3  4 |
[z1234567_MTRN4110_PhaseB] |      ---
[z1234567_MTRN4110_PhaseB] |  |      |      |      |
[z1234567_MTRN4110_PhaseB] |      ---
[z1234567_MTRN4110_PhaseB] |  |      |      |      |
[z1234567_MTRN4110_PhaseB] |      ---
[z1234567_MTRN4110_PhaseB] path - 2:
[z1234567_MTRN4110_PhaseB] ---
[z1234567_MTRN4110_PhaseB] | v      |      |
[z1234567_MTRN4110_PhaseB] | 15  14 |      |  |  |
[z1234567_MTRN4110_PhaseB] |      ---
[z1234567_MTRN4110_PhaseB] |      13|  |  | 0  1  2  3 |
[z1234567_MTRN4110_PhaseB] |      ---
[z1234567_MTRN4110_PhaseB] |  | 12  11  10  9 | 5  4 |
[z1234567_MTRN4110_PhaseB] |      ---
[z1234567_MTRN4110_PhaseB] |  |      | 8  7  6 |  |
[z1234567_MTRN4110_PhaseB] |      ---
[z1234567_MTRN4110_PhaseB] path - 3:
[z1234567_MTRN4110_PhaseB] ---
[z1234567_MTRN4110_PhaseB] | v  15  14  13|      |
[z1234567_MTRN4110_PhaseB] |      ---
[z1234567_MTRN4110_PhaseB] |      | 12      |  |  |
[z1234567_MTRN4110_PhaseB] |      ---
[z1234567_MTRN4110_PhaseB] |      | 11| 0  1  2  3 |
[z1234567_MTRN4110_PhaseB] |      ---
[z1234567_MTRN4110_PhaseB] |  |      10  9 | 5  4 |
[z1234567_MTRN4110_PhaseB] |      ---
[z1234567_MTRN4110_PhaseB] |  |      | 8  7  6 |  |
[z1234567_MTRN4110_PhaseB] |      ---
[z1234567_MTRN4110_PhaseB] path - 4:

```

```

Console
[z1234567_MTRN4110_PhaseB] path - 4:
[z1234567_MTRN4110_PhaseB] --- --- --- --- ---
[z1234567_MTRN4110_PhaseB] | v 15 14 13| 9 8 7 6 |
[z1234567_MTRN4110_PhaseB] --- --- --- --- ---
[z1234567_MTRN4110_PhaseB] | | | 12 11 10| | 5 |
[z1234567_MTRN4110_PhaseB] --- --- --- --- ---
[z1234567_MTRN4110_PhaseB] | | | | 0 1 2 3 4 |
[z1234567_MTRN4110_PhaseB] --- --- --- --- ---
[z1234567_MTRN4110_PhaseB] | | | | | | | |
[z1234567_MTRN4110_PhaseB] | | | | | | | |
[z1234567_MTRN4110_PhaseB] | | | | | | | |
[z1234567_MTRN4110_PhaseB] --- --- --- --- ---
[z1234567_MTRN4110_PhaseB] path - 5:
[z1234567_MTRN4110_PhaseB] --- --- --- --- ---
[z1234567_MTRN4110_PhaseB] | v 15 14 13| 10 9 8 7 |
[z1234567_MTRN4110_PhaseB] --- --- --- --- ---
[z1234567_MTRN4110_PhaseB] | | | 12 11 | | 6 5 |
[z1234567_MTRN4110_PhaseB] --- --- --- --- ---
[z1234567_MTRN4110_PhaseB] | | | | 0 1 2 3 4 |
[z1234567_MTRN4110_PhaseB] --- --- --- --- ---
[z1234567_MTRN4110_PhaseB] | | | | | | | |
[z1234567_MTRN4110_PhaseB] | | | | | | | |
[z1234567_MTRN4110_PhaseB] | | | | | | | |
[z1234567_MTRN4110_PhaseB] --- --- --- --- ---
[z1234567_MTRN4110_PhaseB] path - 6:
[z1234567_MTRN4110_PhaseB] --- --- --- --- ---
[z1234567_MTRN4110_PhaseB] | v 15 14 13| 9 8 7 |
[z1234567_MTRN4110_PhaseB] --- --- --- --- ---
[z1234567_MTRN4110_PhaseB] | | | 12 11 10| | 6 5 |
[z1234567_MTRN4110_PhaseB] --- --- --- --- ---
[z1234567_MTRN4110_PhaseB] | | | | 0 1 2 3 4 |
[z1234567_MTRN4110_PhaseB] --- --- --- --- ---
[z1234567_MTRN4110_PhaseB] | | | | | | | |
[z1234567_MTRN4110_PhaseB] | | | | | | | |
[z1234567_MTRN4110_PhaseB] | | | | | | | |
[z1234567_MTRN4110_PhaseB] --- --- --- --- ---
[z1234567_MTRN4110_PhaseB] Done - 6 shortest paths found!

```

Fig. 5. Find All the shortest paths

3.3. Find the shortest path with the least turns and generate a sequence of motion commands

Compare the found shortest paths and find the one with the least turns, i.e., the one with which the robot needs the minimum number of turns.

In case there are **more than one** such paths, you can **randomly** select one as the found path.

Generate a sequence of motion commands as defined in Phase A and calculate the total number of steps.

Display the found path, the total number of steps, and the sequence of motion commands in the console.

For the example above, both path – 1 and path – 4 have 7 turns, while the other paths have 9 turns. We randomly select path – 1 and generate the motion sequence:

```

Console
[z1234567_MTRN4110_PhaseB] Done - 6 shortest paths found!
[z1234567_MTRN4110_PhaseB] Start - Finding shortest path with least turns:
[z1234567_MTRN4110_PhaseB] --- --- --- --- --- --- ---
[z1234567_MTRN4110_PhaseB] | v  15  14  13| 10  9   8   7   6 |
[z1234567_MTRN4110_PhaseB] |      ---      ---      ---
[z1234567_MTRN4110_PhaseB] |      | 12  11 |   |   | 5 |
[z1234567_MTRN4110_PhaseB] |      |      |   |   |   |
[z1234567_MTRN4110_PhaseB] |      | 0   1  2  3  4 |
[z1234567_MTRN4110_PhaseB] |      ---      ---      ---
[z1234567_MTRN4110_PhaseB] |      |      |   |   |   |
[z1234567_MTRN4110_PhaseB] |      |      |   |   |   |
[z1234567_MTRN4110_PhaseB] |      |      |   |   |   |
[z1234567_MTRN4110_PhaseB] |      ---      ---      ---
[z1234567_MTRN4110_PhaseB] steps: 23 - 00SLFFFRFLFLFRFFFFRFRFFFF
[z1234567_MTRN4110_PhaseB] Done - Shortest path with least turns found!

```

Fig. 6. Find the shortest path with the least turns and generate the motion sequence

3.4. Write the found path to a text file

Write the generated motion sequence to a text file named “PathPlanFound.txt”. Note the generated motion sequence **MUST** follow the conventions defined in Phase A. A motion sequence using a different convention (e.g., using “L” to represent a motion “turn left” followed by a motion “forward”), even if it is essentially correct, will **not** get the marks associated with this task.

Display a message in the console showing this step is done (optional).

```

Console
[z1234567_MTRN4110_PhaseB] steps: 23 - 00SLFFFRFLFLFRFFFFRFRFFFF
[z1234567_MTRN4110_PhaseB] Done - Shortest path with least turns found!
[z1234567_MTRN4110_PhaseB] Start - Writing path plan to PathPlanFound.txt
[z1234567_MTRN4110_PhaseB] Done - Path plan written to PathPlanFound.txt

```



```

PathPlanFound.txt - Notepad
File Edit Format View Help
00SLFFFRFLFLFRFFFFRFRFFFF

```

3.5. Task summary:

| Task | Description |
|------|---|
| 1 | Read a map from a text file and display it in the console |
| 2 | Find ALL the shortest paths for the robot from its initial location to the centre of the maze |
| 3 | Find the shortest path with the least turns and generate a sequence of motion commands |
| 4 | Write the found path to a text file |

4. Specifications and Hints:

4.1. Specifications:

Maze:

1. At the beginning of Phase B, you will be given a text file named “**Map.txt**” in which the maze layout is the same as shown in the example.
2. This map is for your practice. For assessment, you may be tested with a **different** map in which the maze layout and the initial location and heading of the robot may be **different**.
3. The maze will always be the same size (**5** rows by **9** columns) with **closed** borders.
4. The initial position of the robot will always be one of the four **corners**.
5. The initial heading of the robot will always be towards an **opening**.
6. The target position will always be the **centre** of the maze.
7. In the map text file given to you, each **horizontal wall** of the maze is represented by **3** dashes “**---**”.
8. Each **vertical wall** of the maze is represented by **1** vertical bar “**|**”.
9. Each **empty cell** is represented by **3** spaces “”.
10. Each **vacant horizontal wall** is represented by **3** spaces “”.
11. Each **vacant vertical wall** is represented by **1** space “”.
12. Each **block** between adjacent walls is represented by **1** space “”.

Robot:

1. The location and heading of the robot are indicated by a sign (“**^**” - north, “**>**” - east, “**v**” – south, “**<**” - west). Note that throughout this course we refer to the top, bottom, left, and right of the maze as North, South, West, and East, respectively. Only **lowercase v** is used.
2. On the left and right sides of this sign are 2 spaces so that **3** characters, e.g., “**^**”, are used to represent a cell with a robot in it.
3. A turn is accounted when the robot needs to rotate at any cell along the path, including the starting cell.

Implementation:

4. You **MUST** implement the controller in Webots using the same language as in Phase A (either C or C++).
5. For portability, you **MUST** use the pre-installed MinGW C/C++ compiler for Windows or Xcode for macOS.
6. The text file storing the map should be named “**Map.txt**”. You should define a path variable at the beginning of your controller program indicating the path of this text file.

`#define MAP_FILE_NAME “**/Map.txt”`

where ****** is the **RELATIVE** path of the text file stored on your computer.

- To make the assessment of your submission easier, you should follow the instructions of folder structure in 5.1 and then here you should define the path as

`#define MAP_FILE_NAME “../Map.txt”`

7. The text file storing the found path plan should be named “**PathPlanFound.txt**”. You should define a path variable at the beginning of your controller file indicating the path of this text file.

```
#define PATH_PLAN_FILE_NAME "**/PathPlanFound.txt"
```

where ****** is the **RELATIVE** path of the text file stored on your computer.

- To make the assessment of your submission easier, you should follow the instructions of folder structure in 5.1 and then here you should define the path as

```
#define PATH_PLAN_FILE_NAME "../PathPlanFound.txt"
```

8. The generated motion sequence **MUST** follow the conventions defined in Phase A. A motion sequence using a different convention (e.g., using "L" to represent a motion "turn left" followed by a motion "forward"), even if it is essentially correct, will **not** get the marks associated with this task.

4.2. Hints:

1. Consult the lecturer/demonstrators if you are unclear about anything.
2. A tutorial on the flood fill algorithm will be provided. However, you are free to use any path planning algorithms for the tasks.
3. The numbers of feasible paths/shortest paths/shortest paths with least turns in the assessment are not necessarily the same as in the example. You should accommodate more possibilities in your program.
4. There is no simulation involved in this phase so you don't need to change the world file in Webots. However, you will need to add the controller you developed to the robot so that you can run the path planning algorithm and see the results in the console.
5. You are suggested to plan the entire algorithm carefully before writing the code.
6. You should use forward slash / or double backward slash \\ instead of single backward slash \ to define the path of the file.

5. Assessment:

5.1. Submission of your work

You should zip your project folder and rename it as “z*****_MTRN4110_PhaseB.zip” where ***** is your zID. Submit this zip file to Moodle.

In the folder, you should include both the <worlds> folder and the <controllers> folder.

In the <worlds> folder, you should include your world file, named as “z*****_MTRN4110_PhaseB.wbt” where ***** is your zID.

In the <controllers> folder, you should include your controller folder, named as “z*****_MTRN4110_PhaseB” where ***** is your zID.

You should have the following folder structure in your submission:

```
z*****_MTRN4110_PhaseB
|--controllers
|   |--z*****_MTRN4110_PhaseB
|       |--build
|       |--Makefile
|       |-- z*****_MTRN4110_PhaseB.c(pp)
|       |-- z*****_MTRN4110_PhaseB.exe
|--worlds
|   |--z*****_MTRN4110_PhaseB.wbt
|-- Map.txt
|-- PathPlanFound.txt
```

These are the essential folders and files you should include. You can also include other folders/files if needed but you should not change the name/location of these essential folders and files. It is your responsibility to make sure your submission is self-contained.

5.2. Marking criteria:

This assignment will contribute 15% to your final mark.

You will be assessed twice with different setups. Your final mark will be the average of the marks of the two attempts., i.e.,

$$\text{mark}_{\text{final}} = 50\% * \text{mark}_{\text{attempt}_1} + 50\% * \text{mark}_{\text{attempt}_2}$$

Each attempt will be assessed by using the following criteria.

| Task | Description | Marking (out of 100%) |
|------|---|--|
| 1 | Read a map from a text file and display it in the console | +10% if all correct, otherwise <ul style="list-style-type: none">• (8% maximum)<ul style="list-style-type: none">○ +2% each for correctly displaying a line of the maze (excluding the outer walls)• (no mark)<ul style="list-style-type: none">○ If hard-coding the map into program |

| | | |
|---|---|---|
| 2 | Find ALL the shortest paths for the robot from its initial location to the centre of the maze | +50% if all correct, otherwise <ul style="list-style-type: none"> • (40% maximum) <ul style="list-style-type: none"> ○ +15% for finding a valid path ○ +15% if the found path is a shortest one ○ +5% each for finding an additional valid path ○ +5% each if the found additional path is a shortest one |
| 3 | Find the shortest path with the least turns and generate a sequence of motion commands | +30% if all correct, otherwise <ul style="list-style-type: none"> • +10% for finding a shortest path with the least turns • +20% for generating correct motion commands¹, otherwise <ul style="list-style-type: none"> ○ (15% maximum) +5% each for correctly generating 3 motion commands². |
| 4 | Write the found path to a text file | +10% if all correct, otherwise <ul style="list-style-type: none"> • (8% maximum) <ul style="list-style-type: none"> ○ +2% each for correctly writing 3 characters³. |

¹If the found path was not correct but the motion generated was correctly corresponding to the found path, 20% can be granted.

²A motion command is considered correctly generated if and only if this motion command and all the preceding motion commands are correctly generated.

³A character is considered correctly written if and only if this character and all the preceding characters are correctly written.

5.3. Deadline

The submission deadline is **17:00 AEST 12 July 2020 (Sunday Week 6)**.

If your assignment is submitted after this date, each 1 hour it is late reduces the maximum mark it can achieve by 1%. For example if an assignment worth 74% were submitted 10 hours late, the late submission would have no effect. If the same assignment were submitted 30 hours late, it would be awarded 70%, the maximum mark it can achieve at that time.

5.4. Progress Check

You will have your progress checked with your demonstrator in a 5 min meeting on the afternoon of Thursday Week 5 (or another time on weekdays Week 5 by appointment).

To pass the progress check, you must demonstrate that you are able to **generate at least one shortest path** from the given map.

6. Additional Resources:

- C File I/O: https://www.tutorialspoint.com/cprogramming/c_file_io.htm
- C++ File I/O: <http://www.cplusplus.com/doc/tutorial/files/>
- Tutorials on the Flood Fill algorithm (attend the tutorial sessions)
- <https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/>
- <https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/>
- https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-using-priority_queue-stl/
- <https://www.geeksforgeeks.org/a-search-algorithm/>
- <https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/>