Project 1. Searching

Delivery System

1 Description

Thanks to the development of e-commerce, consumers can easily shop for products with just a smartphone from their homes. In this context, the logistics industry plays an important role in ensuring efficient transportation of goods from sellers to buyers.

The most crucial strategy for logistics providers to compete effectively is to develop a system that finds the shortest paths. This system optimizes delivery routes, minimizes travel time, and saves fuel. Your team will be responsible for developing this system for HCMUS Logistic Co. LTD, utilizing search algorithms learned in the course CSC14003 - Introduction to Artificial Intelligence, to **find the path from the delivery vehicle to the customer**.

The city map has been modeled in 2D with n rows and m columns. The map consists of polygons to depict structures such as buildings, walls, etc., and these objects are unchanged on the map. Figure 1 is an example of a city map modeled in the 2D world.

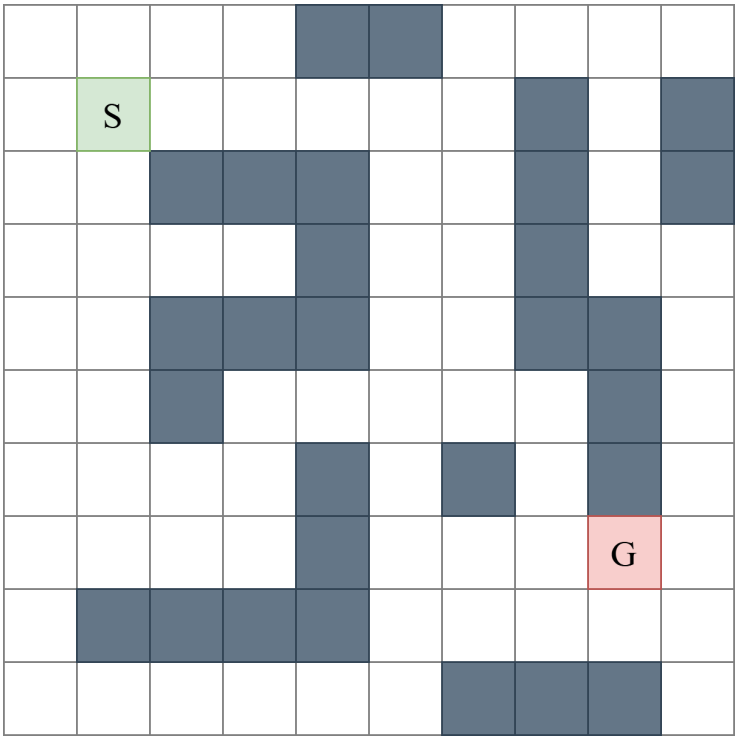


Figure 1: 2D City map.

The map is technically represented as a 2D array, each cell contains a value encoding some meaning. At a basic level, some values are predefined as follows:

- 0: Space that the delivery vehicle can be moved into.
- -1: Impassable space, representing buildings, walls, and objects in the city.
- S: The **Starting location** of the delivery vehicle.
- G: The customer, also the **Goal location** of the delivery vehicle.

The delivery vehicle can move in four directions: **Up**, **Down**, **Left**, and **Right**, and no diagonally. Figure 2 describes the move directions of the delivery vehicle.

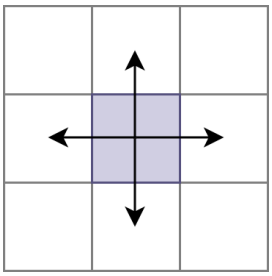


Figure 2: Move directions of the delivery vehicle.

2 Specifications

The project is divided into different levels, each of which offers increasing levels of difficulty to suit the diverse expectations of student groups. Detailed information will be provided below.

2.1 Level 1 (Basic level)

In this level, find the path from the delivery vehicle (S) to the customer (G) with the information provided in section 1 and from the input file.
Students are required to, at a minimum, implement the following search algorithms:

- Blind search methods: Breadth-First Search, Depth-First Search, Uniform-Cost Search.
- Heuristic search method: Greedy Best First Search, A*.

Note that there may be cases where a path does not exist. Please test these cases as well.

2.2 Level 2 (Time limitation)

The time to move between two adjacent cells is **1 minute**. However, there are cells where a vehicle is required to stop for $a[i, j]$ minutes to pay a road toll (a positive integer which is calculated according to the average waiting time of vehicles), called Toll Booths - illustrated by blue squares as shown in Figure 3.

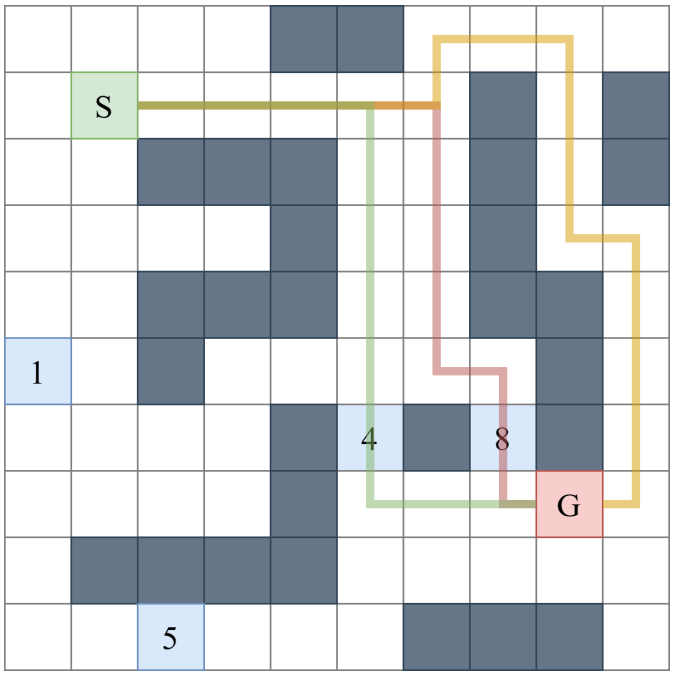


Figure 3: The blue squares are Toll Booths, and the lines are paths.

The presence of toll booths increases the delivery time. To compete with other delivery services, your company has decided to commit to fast delivery within t minutes (provided in the input file). Please program a solution to find the shortest path that satisfies the condition of not exceeding the committed delivery time.
As illustrated in Figure 3, with $t = 20$, it can be seen that:

- The red path is the shortest path (passing 13 cells), but it is not chosen because of overtime committed (21 minutes of delivery).
- The green path and orange path have the same delivery time (17 minutes of delivery), but the green path is chosen because it is shorter (passing through 13 cells).

Note that (at least) an algorithm is required to be implemented at this level.

2.3 Level 3 (Fuel limitation)

Any vehicle has a fuel tank capacity limit, and the company's delivery vehicle is no different, with a capacity of f liters (provided in input). Each move to an adjacent cell consumes 1 liter of fuel. When the vehicle runs out of fuel, the delivery will be stopped. Therefore, gas stations are added to the map (illustrated by yellow squares in Figure 4) so that the delivery vehicle can refuel.

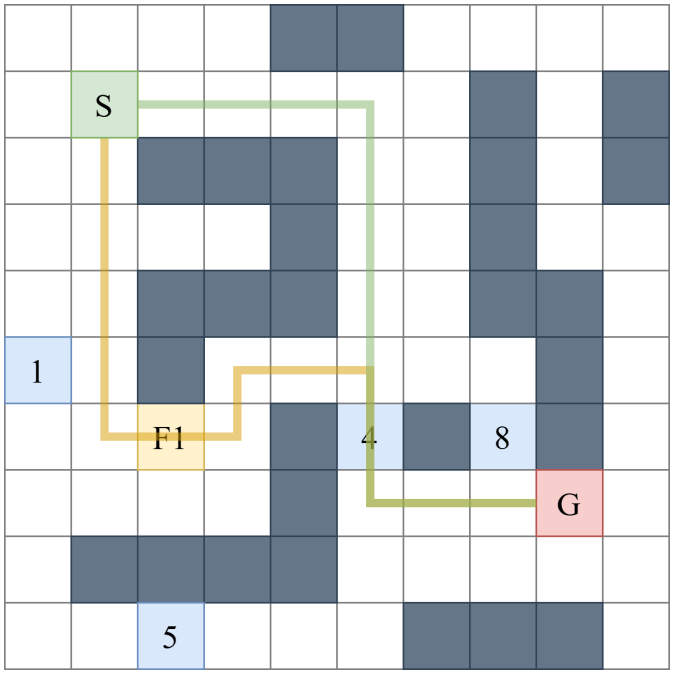


Figure 4: Yellow square with F1 text: F denotes gas station, 1 means it takes 1 minute to refuel.

As illustrated in Figure 4, with $f = 10$, it can be seen that:

- The green path is the shortest path (passing through 13 cells, with 17 minutes of delivery), but it is not feasible because there is not enough gasoline right after leaving the toll booth.
- The yellow path is longer (passing through 15 cells, with 20 minutes of delivery), but it is feasible because the vehicle has refueled at the fuel station F (included 1 minute to refill).

Note that:

- Initially the delivery vehicle was always full of fuel.
- When passing through the fuel station, the delivery vehicle will be refueled to full capacity.
- When stationary at toll booths, the vehicle does not consume fuel.
- An algorithm is required to be implemented at this level.

2.4 Level 4 (Multiple agents)

Besides the vehicles from S to G, there are other vehicles in the city that are on their own trips. Assume that their interactions are limited to competing for movement space, without any form of elimination of each other. The movements are **turn-based**.

There is a maximum limit of 9 other vehicles on the map, with the Starting location numbered from S_1 to S_9, and the Goal location numbered from G_1 to G_9. At each turn, if a vehicle does not have a destination, it will randomly generate a valid destination on the map (i.e., not on obstacles, or other vehicles); and then, the vehicle will make its move. Once a vehicle completes its trip, it immediately generates a new trip to continue its operation. Two vehicles are not allowed to occupy the same cell at the same time. If a vehicle encounters an obstacle, like another vehicle, it can choose to wait or move in a different direction. All vehicles aim to optimize their paths effectively. Figure 5 demonstrates an example of the world at this level.

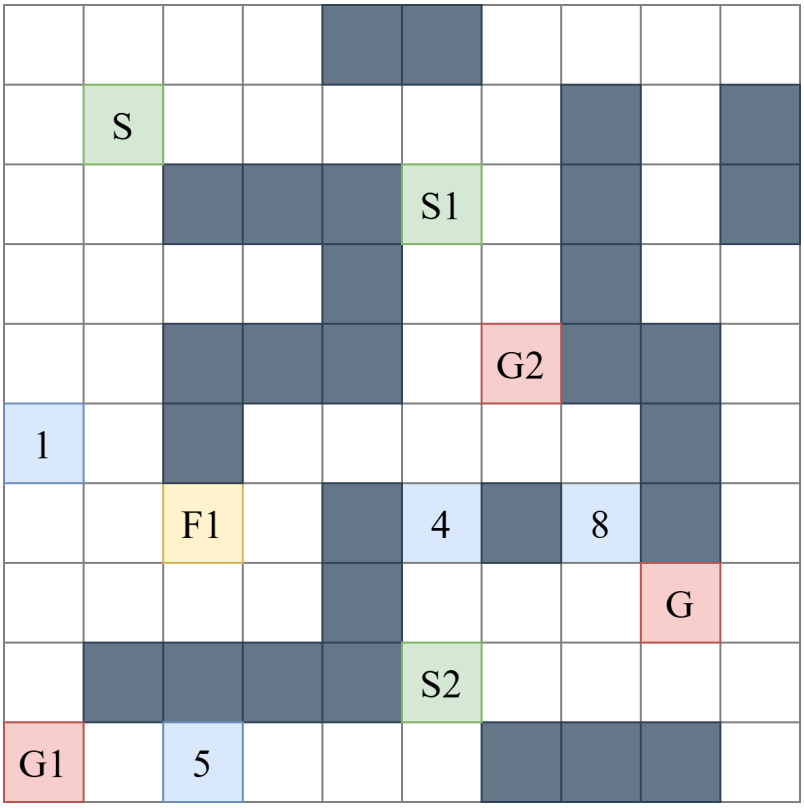


Figure 5: The world at Level 4.

Note that the process is done when the delivery from S to G is completed, or it is determined that the task cannot be completed.

3 Requirements

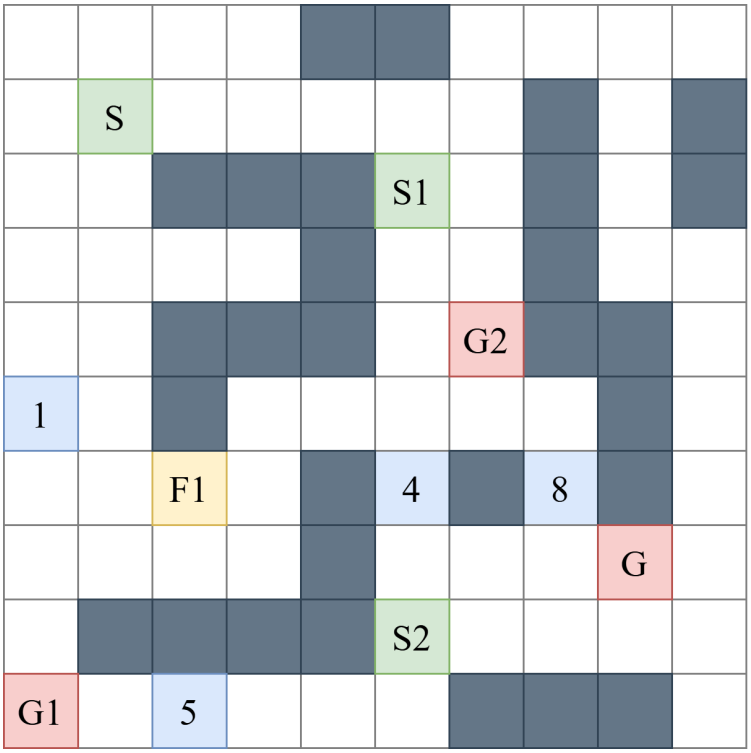
3.1 Programming language

- The source code must be written in Python.
- You can use supporting any libraries, but the main algorithms directly related to the search process must be implemented on your own.

3.2 Input

The input file is numbered according to the convention `input1_level1.txt`, `input1_level2.txt`, etc. The input file format is described as follows:

- The first line contains 4 positive integers n m t f, corresponding to the number of rows and columns of the map, committed delivery time, and fuel tank capacity.
- The following n lines represent the information on the map. Encoding conventions are as in the description, with all letter characters in uppercase.



input.txt

```
10 10 20 10
0 0 0 -1 -1 0 0 0 0 0
0 S 0 0 0 0 0 -1 0 -1
0 0 -1 -1 -1 S1 0 -1 0 -1
0 0 0 0 -1 0 0 -1 0 0
0 0 -1 -1 -1 0 G2 -1 -1 0
1 0 -1 0 0 0 0 0 -1 0
0 0 F1 0 -1 4 -1 8 -1 0
0 0 0 0 -1 0 0 0 G 0
0 -1 -1 -1 -1 S2 0 0 0 0
G1 0 5 0 0 0 -1 -1 -1 0
```

3.3 Output

The output results needs to be visualized using a graphical interface to display the search process, or if not, must be saved as an output file. Below are the detailed output requirements:

Output file (if no GUI):
The result path should be saved with the format `output1_level1.txt`, `output1_level2.txt`, etc. The content of the result file should include:

- The first line contains name of the vehicle.
- The next line list the coordinates of each cell on the path, and each coordinate consists of two integers separated by a comma, representing the row and column of the cell.

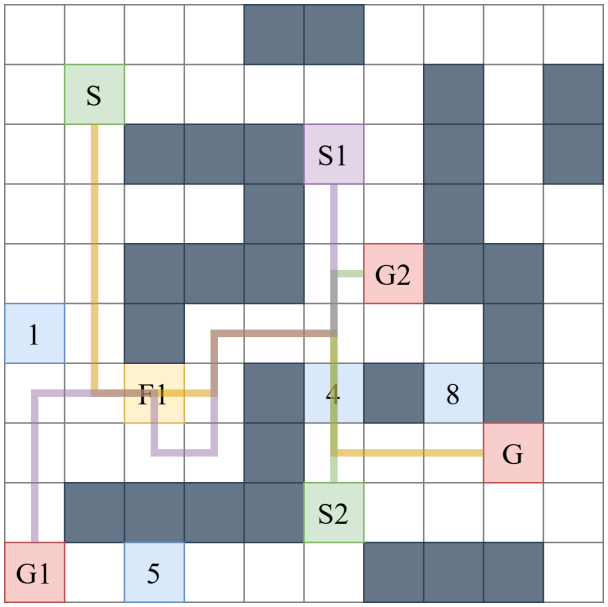


Figure 6: Example paths for Multiple agents level.

For example, with the sample paths in Figure 6, the output file could be:

```
S
(1, 1) (2, 1) (3, 1) (4, 1) (5, 1) (6, 1) (6, 2) (6, 2) (6, 3) ... (7, 8)
S1
(2, 5) (3, 5) (4, 5) (5, 5) (5, 4) (5, 3) (6, 3) (7, 3) (7, 2) ... (9, 2)
S2
(8, 5) (7, 5) (6, 5) (6, 5) (6, 5) (6, 5) (6, 5) (5, 5) (4, 5) ... (3, 9)
```

Graphical User Interface (GUI)

- The program is recommended to have a graphical interface to show the process step by step.
- Each vehicle should use a different color and be printed separately for each vehicle.

3.4 Report

The report must fully give the following sections:

- Member information (student ID, full name, ...)
- Work assignment table, which includes detailed information on each task assigned to team members, along with the completion rate of each member compared to the assigned tasks
- Self-evaluation of the completion rate at each level of the project and other requirements.
- Detailed algorithm description for each level. Although a higher level may encompass the previous ones, presenting the information by levels will enhance clarity for the assessment and commentary process. Illustrative images are encouraged.
- Describe the test cases and the results when run on each of those test cases.
- The report needs to be well-formatted and exported to PDF. Note that for editors like Jupyter Notebook, the team needs to find a way to format it well before exporting it to PDF.
- If there are figures cut off by the page break, etc., points will be deducted.
- References (if any).

3.5 Submission

- Your report, source code, and test cases must be contributed in the form of a compressed file (.zip, .rar, .7z) and named according to the format **StudentID1_StudentID2_...**
- Demo videos (recording the running process of your program for each test case) should be uploaded to YouTube or Google Drive, and the public URLs are included in the report.
- If the compressed file is larger than 25MB, prioritize compressing the report and source code. Test cases may be uploaded to Google Drive and shared via a link.

Details of the directory organization:

```
StudentID1_StudentID2_...
├── Source
│   ├── main.py
│   ├── input1_level1.txt
│   ├── input1_level2.txt
│   ├── output1_level1.txt
│   ├── output1_level2.txt
│   ├── requirements.txt (libraries to install)
│   ├── README.txt (how to run source code)
│   └── ...
└── Report.pdf (included video URLs)
```

4 Assessment

No.	Details	Score
1	Finish Level 1 successfully.	15%
2	Finish Level 2 successfully.	15%
3	Finish Level 3 successfully.	15%
4	Finish Level 4 successfully.	10%
5	Graphical User Interface (GUI)	10%
6	Generate at least 5 test cases for each level with different attributes. Describe them in the experiment section of your report. Videos to demonstrate each test case.	15%
7	Report your algorithm, and experiment with some reflection or comments.	20%
	Total	**100%**

5 Notices

Please pay attention to the following notices:

- This is a **GROUP** assignment. Each group has about 3 to 4 members.
- Any plagiarism, any tricks, or any lie will have a 0 point for the course grade.

The End.