

## Program 3: Maze problem

**Due Date:** April 19<sup>th</sup> 2024 (11:59PM)

### 1- **Assignment Learning Objectives:**

- Graph algorithm implementation.
- Mapping problems into data structures.
- Solving problems in an object-oriented manner.

### 2- **Associated Files:**

- Assignment description (this document)
- Test input files (e.g., maze.txt, coordinates.txt)

Both files will be on Moodle under (Programming 3 Assignment)

### 3- **Description:**

In this program, you will be given two input files, a maze file and a coordinate file. In the maze file, it contains a  $m \times n$  maze structure with values 0 and 1 where 0 represents walls and 1 represents empty spaces (see the sample file). The coordinates file includes the coordinates (x, y) of the starting point and destination. The first line contains the coordinates of the starting point, while the second line contains the coordinates of the destination (see the sample file). The goal for this program is to find a valid path in this maze from the starting point to the destination. Specifically, the path must only traverse locations marked as 1 in the maze file and cannot pass through locations marked as 0. Additionally, at each valid location (i.e. locations marked as 1 in the maze file), you can only move one step at a time in the directions of up, down, left or right to reach the next valid location. You are asked to implement this based on three different searching algorithms: Breadth First search (BFS), Depth First Search (DFS) and Best-First Search (GBFS)(see the next section for Best-First Search). Then, Print out the result in the following format as your output.

- Print “#” at every location with value 0.
- Print “s” and “e” as the starting point and endpoint, respectively, in the maze.
- Print the path using “+” and leave the rest as “-”.

Finally, includes all the results in your report.

Here is some reminders:

- The heuristic value should be computed as the Manhattan distance, which is the sum of the absolute differences in each coordinate. For example, the Manhattan distance of (1,1) to (8,8) should be  $|1-8| + |1-8| = 14$ .
- When implementing GBFS, you don't have to create your own priority queue class; You can use it from the library.
- It is acceptable to implement the movement to the next location in any order you prefer (up, down, right, left)
- To prevent the search algorithm from getting stuck in an infinite loop, avoiding revisiting the same location in your algorithm.

To test your implementation, your program should expect three input parameters, the name of the input file including the maze structure, the file with coordinates of the starting point and the destination, and the choice of the searching method. You can assume both coordinates are valid and there always exists a path from starting point to destination. Thus, to execute your program it takes the following form:

***./program <maze filename> <coordinate files> <searching method>***

For example:

***./program maze.txt coordinates.txt BFS***

The program that you will implement shall do the following:

1. Parse the input parameters to determine the maze to be used, the coordinates of starting point and destination, and the searching method to be used.
2. Write a Maze class that stores the structure in maze.txt.
3. Under this class, write a method called BFS\_searchpath that takes the coordinates of start and destination as inputs, and uses BFS to find the path.
4. Print out the path found using BFS\_searchpath in the specific format.
5. Similar to 3. Write a method called DFS\_searchpath that takes the same inputs as BFS\_searchpath.
6. Print out the path found using DFS\_searchpath in the specific format.
7. Similar to 3. Write a method called GBFS\_searchpath that takes the same inputs as BFS\_searchpath.
8. Print out the path found using GBFS\_searchpath in the specific format.

#### **4- Best-First Search:**

In BFS and DFS, we don't rely on any information about the graph. However, in reality, when we are looking for a path from one place to another, we have some prior information about the graph, which we can leverage to develop other search algorithms. For example, when looking for

a path from Raleigh to New York. Since the path is likely to head north, we might prefer like to choose a city A in the north of Raleigh rather than a city B in the south of Raleigh. The reason behind this is that the distance between New York to city A is smaller than the distance to city B.

With the same philosophy, we introduce a heuristic search algorithm called Best-First search algorithm. Unlike BFS and DFS, it selects the “best” node according to the heuristic value. For example, in Figure 1, we have a graph with heuristic values assigned to each node. Assuming the starting node is A and the goal is to reach node D. From A, there are direct paths to 6 other nodes, we choose the path with lowest heuristic value at each step. Since node C has the lowest value among 6 nodes reachable from A, we proceed from node A to node C. In the next step, with 5 nodes remaining and 1 additional node D, we choose node D due to its lowest heuristic value, leading us to our goal.

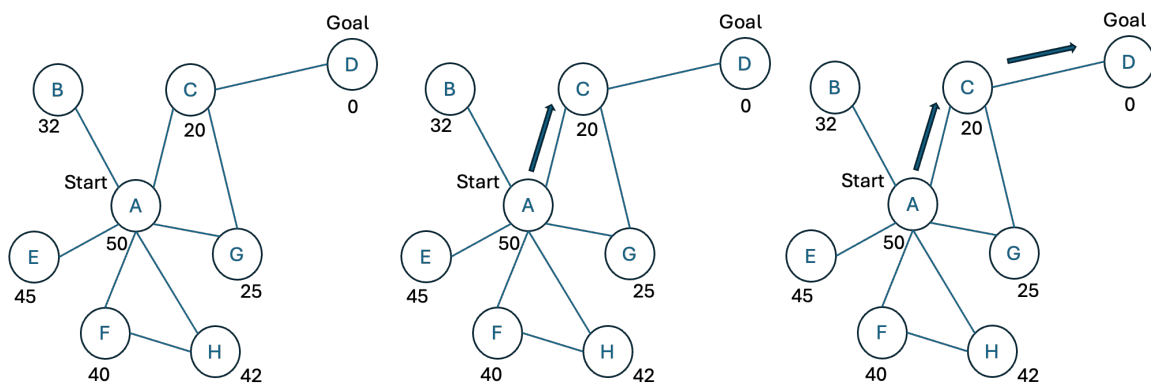


Figure 1 Demonstration for Best-First search algorithm.

## 5- Report Format:

The report should be in the following format:

The output of BFS:

<print your result in here>

The output of DFS:

<print your result in here>

The output of GBFS:

<print your result in here>

## 6- Submission Format:

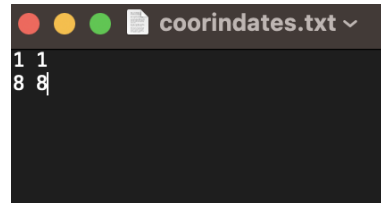
main.cpp and report.pdf

## 7- Sample file for input:



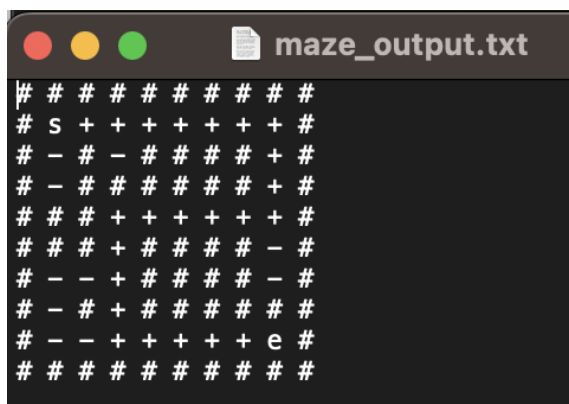
```
0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 0
0 1 0 1 0 0 0 0 1 0
0 1 0 0 0 0 0 0 1 0
0 0 0 1 1 1 1 1 1 0
0 0 0 1 0 0 0 0 1 0
0 1 1 1 0 0 0 0 1 0
0 1 0 1 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 0
0 0 0 0 0 0 0 0 0 0
```

*Sample file for maze.txt.*



```
1 1
8 8
```

*Sample file for coordinates.txt.*



```
# # # # # # # # # #
# s + + + + + + #
# - # - # # # # + #
# - # # # # # # + #
# # # + + + + + + #
# # # + # # # # - #
# - - + # # # # - #
# - # + # # # # # #
# - - + + + + + e #
# # # # # # # # # #
```

*Demonstration for output format.*

## 8- Grading:

- 10 points: Files submitted with proper name.
- 10 points: Program is complete. The program successfully takes 3 inputs and print out the result in the right format as output.
- 10 points: Proper coding style and comments.
- 60 points: the program performs all functions correctly:
  - (15 pts) Create a class named Maze that correct store the data from maze.txt.
  - (15 pts) Find-path method based on BFS.
  - (15 pts) Find-path method based on DFS.
  - (15 pts) Find-path method based on GFS.
- 10 points: Report.

***Good luck***