Full name: Ngô Quốc Quý

ID: 21127679

Class: 21CLC08

CSC140003 – ARTIFICIAL INTELLIGENT

LAB01: SEARCH STRATEGIES

REPORT

I. Checklist

No.	Task	Progress
1	Implement and provide results for the following search strategies: - Breadth-first search - Depth-first search - Uniform-cost search - Greedy Best First Search using the Manhattan distance as heuristic - Graph-search A* using the Manhattan distance as heuristic	100%
2	For each search strategy, print to the console the following information: - The time to escape the maze - The list of explored nodes (in correct order) - The list of nodes on the path found (in correct order).	100%
3	Create some input as text file with: - The first line must contain a positive integer N, representing the size of the maze. - The second line contains 2 integers representing the entrance and exit. - $N \times N$ next line contains an adjacency list or adjacency matrix.	50%

In task 3: just create one maze for the input test file.

II. Brief description of main functions

1. *main.py*

In *'main.py'*, creating 5 graphs and implementing for each search strategy. Calling the functions: input_maze(), write_output(arg), print_output(arg) which are imported from IDHandle.py.

2. *IOHandle.py*

- input_maze(): create an array to save adjacency list which is read from the input file (input.txt, path "../input/input.txt")
- write_output(arg): get the results from arguments (arg) then write the output into file .txt (output.txt, path "../output/output.txt")
- print_output(arg): get the results from argument (arg) then print the output to the console

3. *graph.py*

- class Graph: define the graph with defaultdict(list) data type, in graph class, there are some variables which are named: **explored (list)**, **visited (dict)**, **parent (dict).** Because it's defined in a class, so that, each function have to passed in the **self** argument.
    - **self.explored = []:** use to get the list of nodes explored in the correct order.
    - **self.visited = {}:** use to identify which nodes have been traversal.
    - **self.parent = {}:** use to save the parents of each node
- function **create_graph(self, adj_list):** create a graph
- function **find_correct_path(self, goal):** to find the correct path after each search strategy
- function **manhattan_distance(self, node, goal)**: calculate Manhattan distance, use to implement the Graph-search A* and Greedy Best First Search
- main function:
    - **breadth_first_search(self, start, goal):** implement the idea of Breadth-first search, this function is passed in 3 argument (self, start, goal), return 3 values (time escape, explored, path_found). If there's no solution, it will return None for all.
    - **depth_first_search(self, start, goal):** implement the idea of Depth-first search, this function is passed in 3 argument (self, start, goal), return 3 values (time escape, explored, path_found). If there's no solution, it will return None for all.

- **uniform_cost_search(self, start, goal):** implement the idea of Uniform-cost search, this function is passed in 3 argument (self, start, goal), return 3 values (time escape, explored, path_found). If there's no solution, it will return None for all.
- **Greedy_best_first_search:** implement the idea of Greedy Best First Search, this function is passed in 3 argument (self, start, goal), use the function **manhattan_distance** to calculate the heuristic value, return 3 values (time escape, explored, path_found). If there's no solution, it will return None for all.
- **A_star_search:** implement the idea of Graph-search A*, this function is passed in 3 argument (self, start, goal), use the function **manhattan_distance** to calculate the heuristic value, return 3 values (time escape, explored, path_found). If there's no solution, it will return None for all.

THE END