FACULTY OF BUSINESS, DESIGN AND ARTS

COS30019 INTRODUCTION TO ARTIFICIAL INTELLIGENCE

Lecturer: **Dr Joel Than**

Student Name	Student ID
Kelvin Zhong Wei WONG	104390158

Contents

Instructions

In order to execute the robot navigation program, you must have the programming language Python3 installed into your system. This program is a command-line application that can help you read your map input files and applies a selected search algorithm to navigate the robot from a start state to one or more goal state as well as avoid obstacles.

```
print("Usage: python search.py <filename> <method>")
```

Figure 1 Instruction 1

As you can see from the figure above, this code allows you to run and see the output of the robot navigation code. The <filename> as seen above is a text file that describes the map layout, starting position, goal position and wall replacement, and the <method> are the search strategy being used, in this assignment we will use BFS, DFS, GBFS, AS, CUS1, and CUS2. After running the code, the program will print the goal reached and the nodes explored, the full action path to reach the goal and finally the list of nodes visited in order.

Introduction

Robot Navigation Problem is the challenge of enabling a robot to autonomously move from a starting state to a goal state in any environment, the robot must navigate itself around the obstacles and make a sequence of legal moves to reach the goal while minimizing cost time, or steps. This problem is an example of pathfinding which can be commonly found in artificial intelligence , robotics or game development. You can say this problem effectively becomes a bit more complex as the environment can grow larger or become more cluttered. So to address this situation, lots of search algorithms are employed, each with different strengths and weaknesses depending on the situation.

To solve the navigation problem computationally, the environment can be represented as a graph where the nodes represent a valid positions, the edge

represents the legal move set between the adjacent cells and finally the cost being assigned to movements to simulate the environment distances and difficulty. A search tree is a tree like data structure that is designed to locate specific keys within a set of data by the search algorithm during execution. For example the root node is the start state, and each child node can be represented as a possible move to be explored, and the paths represent the sequences of action. In addition, we also have tree based search like Breadth-First Search where we explore all nodes one by one at the current depth before moving in deeper, the Depth-First search, explores as deep as possible along a brand but if it hit a dead end it will backtrack, and finally the A* search and Greedy Best-Free Search which use heuristic to guide their search.

Robot navigation is said to be a foundational topic in the world of artificial intelligence because it combines problem solving, decision-making and optimization. Search algorithms form the foundation for many AI system in the real world, for example like robotic vacuum cleaner or autonomous self-driving cars. So you can say studying this problem can help us understand how AI agent adapts to the world and make the best decisions for the outcome.

Search Algorithms

In this assignment, we are task to implement several known and custom used search algorithms to solve the robot navigation problem. Each algorithm explores the environment differently, which leads to the variations of their performance, optimality, completeness and execution speed. Therefore let's look into the algorithm we are task to use.

The first is the Breadth-First Search, BFS explores all neighbour one by one at the current level before going into a deeper level, does it complete the search? Yes. Is it Optimal? Yes, but only if step cost is uniformed. In summary, BFS can guarantee the shortest path to completion, but it can be memory-intensive especially in a larger environment. This search is best suited for scenarios where the goal state is typically closer to the start state.

Next is the Depth-First Search, DFS explores as deep as possible along each brands but as stated before, once it hits a dead end it will backtrack. So does it complete the search? No. Is it optimal? No. In summary, DFS can be efficient in memory but it fail to find the shortest path and even the valid ones, especially in environments with many loops and dead ends.

Next is the Greedy Best-First Search(GBFS), this search uses a heuristic function h(n) to estimate the cost from nodes to goal state; it prioritize nodes with lower h(n) So does it complete the search? No. Is it optimal? No. In summary, GBFS can be fast and efficient in many cases but it cannot guarantee us it will find the shortest path. Although it performs well on maps where the heuristic can closely estimate the actual cost, it can still be misled by an inaccurate heuristic.

Next is the A* Search, AS for short, is a pathfinding algorithm that find the shortest path between two points. It uses the formula f(n) = g(n) + h(n), meaning g(n) is the cost from the start state to the current node and h(n) is the heuristic estimate to goal. So does it complete the search? Yes. Is it optimal? Yes. In summary, AS is said to consistently find optimal paths, and it can be considered one of the best general-purpose search algorithms.

Next is the Custom strategy 1, for my custom strategy 1 I have implemented Bidirectional BFS, where two simultaneous BFS searches are used, it can have one BFS forward from the start node and one backward from the goal node, in addition the search terminates when the two frontiers intersect. Just like BFS it can complete the search and is optimal. In summary, the CUS1 Bidirectional BFS can easily reduced the number of nodes explored compared to the traditional BFS. Thus, making it more efficient for larger maps.

Finally, for my custom strategy 2, I decided to use Bi directional GBFS, where two heuristics searches are performed, where one start at the initial post and another at the goal state position and proceed to move towards each other using heuristic to guide them. In summary, it has the speed advantage of GBFS but with bidirectional logic, this greatly help the search by reducing the explored search environment compared to the traditional GBFS.

Therefore, based on the information, I have I believe that A*Search is the best overall search Algorithm, because it is more optimal and reliable than the other search algorithms to reach its goals. But on second thoughts GBFS can be another choice because it is the fastest but only when the heuristics are accurate.

Implementation

Breadth-First Search

BFS was implemented using a queue to explore the nodes one by one. This ensures that the upper nodes are all explored first before going deeper. This algorithm is guaranteed to find the shortest path on an unweighted grid. One of the key components in the implementation are the queue which is used to maintain the frontier of the nodes being explored, the visited set which ensures that nodes are not revisited, next the parent dictionary that tracks the movement of each nodes and how they reached the environment allowing it to do path reconstruction, and finally the visited order list which is used to records the movement of the node in order for the next expansion.

```
def bfs(grid, start, goals, rows, cols):
    queue = deque([start])
    visited = set([start])
    visited_order = [start]
    parent = {start: None}
    nodes_explored = 0

    while queue:
        current = queue.popleft()
        nodes_explored += 1

        if current in goals:
            path = reconstruct_path(parent, current)
            return current, nodes_explored, path, visited_order

        x, y = current
        for dx, dy, move in DIRECTIONS:
            neighbor = (x + dx, y + dy)
            if 0 <= neighbor[0] < rows and 0 <= neighbor[1] < cols and neighbor not in visited and grid[neighbor[0]][neighbor[1]] != '#':
                queue.append(neighbor)
                visited.add(neighbor)
                visited_order.append(neighbor)
                parent[neighbor] = (current, move)

    return None, nodes_explored, None, visited_order
```

Figure 2 BFS code

As seen in the figure above this is the code implemented to run the BFS search.

```
def dfs(grid, start, goals, rows, cols):
    stack = [start]
    visited = set()
    visited_order = []
    parent = {start: None}
    nodes_explored = 0
```

Figure 3 BFS code 2

It starts with creating the BFS function which consists of the map grid, start state, goal state and the rows and columns. Then it initializes the queue with the starting node. This queue controls the order of how the nodes are being explored. Then the visited = set ([start]) is created to keep track of all nodes that have been visited to avoid processing them again. For visited _order = [], it is a list that records the order of the nodes we have visited, which we can use it for visualization or debugging. Then there is the parent = {start: none} : which stores each node to help reconstruct the path once the goal is found. Finally, there is the nodes explored which counter for how many nodes have been explored during the search.

```
    while queue:
        current = queue.popleft()
        nodes_explored += 1

        if current in goals:
            path = reconstruct_path(parent, current)
            return current, nodes_explored, path, visited_order
```

Figure 4 BFS code 3

In this part, the while queue keep exploring in the queue as long as there are nodes to explore, then the current = queue.popleft() is created to retrieves and eliminates the front node from the queue, ensuring BFS explore step by step. Next it will increment the count of how many nodes have been explored so far. So if the current node is a goal. Then it will track back the path from the goal to the start using the parent map.

```
        x, y = current
        for dx, dy, move in DIRECTIONS:
            neighbor = (x + dx, y + dy)
            if 0 <= neighbor[0] < rows and 0 <= neighbor[1] < cols and neighbor not in visited and grid[neighbor[0]][neighbor[1]] != '#':
                queue.append(neighbor)
                visited.add(neighbor)
                visited_order.append(neighbor)
                parent[neighbor] = (current, move)

    return None, nodes_explored, None, visited_order
```

Figure 5 BFS code 4

In this part, the code generates all valid neighbouring cells of the current nodes, if there are any nodes that are visited or out of bound it will filter it out and add in a valid one to the queue for future exploration. In addition, it also record each neighbour parent for path reconstruction purposes.

Depth-First Search

DFS algorithm was implemented using a stack-based approach to traverse the grid in a depth-prioritized manner. DFS explores as deep as possible along each branch before backtracking when it hit a dead end. This implementation ensures minimal usage compared to BFS, but it does not always find the shortest path. Just like BFS, DFS Implementation includes visited tracking where they keep track of explored nodes to avoid revisiting, and parent tracking where it maps each node to the previous explored node and the moves taken to reach it, this data is used to reconstruct the final path. What is new is the exploration loop.

```python
def dfs(grid, start, goals, rows, cols):
    stack = [start]
    visited = set()
    visited_order = []
    parent = {start: None}
    nodes_explored = 0

    while stack:
        current = stack.pop()
        if current in visited:
            continue

        visited.add(current)
        visited_order.append(current)
        nodes_explored += 1

        if current in goals:
            path = reconstruct_path(parent, current)
            return current, nodes_explored, path, visited_order

        x, y = current
        for dx, dy, move in reversed(DIRECTIONS):
            nx, ny = x + dx, y + dy
            if 0 <= ny < rows and 0 <= nx < cols and (nx, ny) not in visited and grid[ny][nx] != '#':
                stack.append((nx, ny))
                parent[(nx, ny)] = (current, move)

    return None, nodes_explored, None, visited_order
```

Figure 6 DFS code

As seen in the figure above, this is the code implemented to run the DFS search. The DFS function implements the Depth First-search program to navigate a grid from a start state to one or more Goal state. DFS uses stack to explore deeply before backtracking. Its function tracks down visited nodes, records the order of visit then reconstructs the path to the goal state if found. Just like BFS it also avoids revisiting nodes that it visits and avoid the walls. once the goal is reached, the program will show the output which are the goal position, numbers of explored nodes, the path taken and the order of the visited nodes but if the path is not found, it is program to return none.

Greedy Best-First Search

GBFS algorithm implements a priority queue where its priority is determined mostly on the heuristic estimate of cost from the given node to the goal state. This search prioritizes nodes that are estimated to be closer to the goal state, its aim is to reach the goal state quickly but it does not guarantee us that it will be the shortest path. The data structure of this search have heap which stores nodes that have heuristic values, like other search it had visited which keep tracks of the nodes that are already

explored. A parent dictionary that records how each node is reached so later on it can reconstruct a new path to reach the goal state. And the visited order which tracks and shows the sequence of node exploration. It is said that this search algorithm can repeatedly explore the nodes with the lowest heuristic values from the heap, and it reconstructs the path using the parent dictionary when the goal state is reached, otherwise it will continue to examine all the valid neighbouring position.

```
def gbfs(grid, start, goals, rows, cols):
    heap = [(heuristic(start, goals), start)]
    visited = set([start])
    visited_order = [start]
    parent = {start: None}
    nodes_explored = 0

    while heap:
        _, current = heapq.heappop(heap)
        nodes_explored += 1

        if current in goals:
            path = reconstruct_path(parent, current)
            return current, nodes_explored, path, visited_order

        x, y = current
        for dx, dy, move in DIRECTIONS:
            neighbor = (x + dx, y + dy)
            if 0 <= neighbor[0] < rows and 0 <= neighbor[1] < cols and neighbor not in visited and grid[neighbor[0]][neighbor[1]] != '#':
                heapq.heappush(heap, (heuristic(neighbor, goals), neighbor))
                visited.add(neighbor)
                visited_order.append(neighbor)
                parent[neighbor] = (current, move)

    return None, nodes_explored, None, visited_order
```

Figure 7 GBFS code

As seen in the figure above, this is the code implemented to run the GBFS search. GBFS function, implements the search for navigating the grid. It starts at the given position which is the start state and begins to explores toward the goal state by choosing the node that appears closest to the goal state based on the heuristic function. In the code, a priority queue (heap) is used to order which node to explore next based on the estimated cost. This function keeps track of visited nodes, the order of the exploration and how the nodes are explored. It will reconstruct and return the path if the goal state is found. Therefore, GBFS can be fast but it does not mean you will find the shortest path.

A* Search

A*search algorithms finds an optimal path from a start state to a goal state in a grid. Just like GBFS, it uses a priority queue to explore the paths that minimize the total cost based on this formula f(n) = g(n) + h(n), g(n) is the actual cost from the start node to the current node, and h(n) is the heuristic estimate cost from the current node

to the nearest Goal state. This algorithm keeps track of the visited nodes and their cost to avoid revisiting path that are less efficient.

```python
def asearch(grid, start, goals, rows, cols):
    heap = [(heuristic(start, goals), 0, start)]
    visited = {start: 0}
    visited_order = [start]
    parent = {start: None}
    nodes_explored = 0

    while heap:
        f, g, current = heapq.heappop(heap)
        nodes_explored += 1

        if current in goals:
            path = reconstruct_path(parent, current)
            return current, nodes_explored, path, visited_order

        x, y = current
        for dx, dy, move in DIRECTIONS:
            neighbor = (x + dx, y + dy)
            if 0 <= neighbor[0] < rows and 0 <= neighbor[1] < cols and grid[neighbor[0]][neighbor[1]] != '#':
                new_g = g + 1
                if neighbor not in visited or new_g < visited[neighbor]:
                    visited[neighbor] = new_g
                    visited_order.append(neighbor)
                    heapq.heappush(heap, (new_g + heuristic(neighbor, goals), new_g, neighbor))
                    parent[neighbor] = (current, move)

    return None, nodes_explored, None, visited_order
```

Figure 8 A Search*

As seen in the figure above, this this is the code implemented to run the A* search. The asearch function implements the A* search algorithm to find the best and most efficient path from the start state to one of the goal state in the grid. This program uses the both the actual cost to reach a node (g) and the heuristic estimate (h) to prioritize which node should we explore next. This algorithm allows us to track visited nodes and only explore nodes we already visited if the new path constructed has a lower cost. And once the goal is reached it will reconstruct and give the output of the path taken, the numbers of nodes explored and the order in which nodes were visited.

Bidirectional BFS

```python
def BID_BFS(grid, start, goals, rows, cols):

    if not goals:
        return None, 0, None, []

    goal = next(iter(goals))

    forward_queue = deque([start])
    forward_visited = {start: None}
    forward_path = []

    backward_queue = deque([goal])
    backward_visited = {goal: None}
    backward_path = []

    visited_order = [start, goal]
    nodes_created = 2

    while forward_queue and backward_queue:
        current_forward = forward_queue.popleft()
        fx, fy = current_forward
        for dx, dy, move in DIRECTIONS:
            nx, ny = fx + dx, fy + dy
            neighbor = (nx, ny)
            if (0 <= nx < rows and 0 <= ny < cols and grid[nx][ny] != '#' and neighbor not in forward_visited):
                forward_queue.append(neighbor)
                forward_visited[neighbor] = (current_forward, move)
                visited_order.append(neighbor)
                nodes_created += 1
                if neighbor in backward_visited:
                    meet_point = neighbor
                    fpath = []
                    n = meet_point
                    while forward_visited[n] is not None:
                        n, m = forward_visited[n]
                        fpath.append(m)
                    fpath.reverse()
                    bpath = []
                    n = meet_point
                    while backward_visited[n] is not None:
                        n, m = backward_visited[n]
                        bpath.append(m)
                    direction_map = {'UP': 'DOWN', 'DOWN': 'UP', 'LEFT': 'RIGHT', 'RIGHT': 'LEFT'}
                    bpath = [direction_map[m] for m in bpath]
                    return meet_point, nodes_created, fpath + bpath, visited_order
```

```
        current_backward = backward_queue.popleft()
        bx, by = current_backward
        for dx, dy, move in DIRECTIONS:
            nx, ny = bx + dx, by + dy
            neighbor = (nx, ny)
            if (0 <= nx < rows and 0 <= ny < cols and grid[nx][ny] != '#' and neighbor not in backward_visited):
                backward_queue.append(neighbor)
                backward_visited[neighbor] = (current_backward, move)
                visited_order.append(neighbor)
                nodes_created += 1
                if neighbor in forward_visited:
                    meet_point = neighbor
                    fpath = []
                    n = meet_point
                    while forward_visited[n] is not None:
                        n, m = forward_visited[n]
                        fpath.append(m)
                    fpath.reverse()
                    bpath = []
                    n = meet_point
                    while backward_visited[n] is not None:
                        n, m = backward_visited[n]
                        bpath.append(m)
                    direction_map = {'UP': 'DOWN', 'DOWN': 'UP', 'LEFT': 'RIGHT', 'RIGHT': 'LEFT'}
                    bpath = [direction_map[m] for m in bpath]
                    return meet_point, nodes_created, fpath + bpath, visited_order

    return None, nodes_created, None, visited_order
```

Figure 9 Bidirectional BFS

For this custom search strategy 1, I have chosen to use Bidirectional BFS, as seen in the figure above, this is the code implemented to run the Bidirectional BFS search. The BID_BFS function searches for both the start and goal state simultaneously using two BFS frontier. It attempts to meet in the middle after it tracks visited nodes in every direction. Once both searches encounter a common node, it will reconstruct the path by combining the forward and reversed backward paths. You can say this way often finds path much faster than the regular BFS only because it reduces the search space by expanding both ends.

Bidirectional GBFS

```
def BID_GBFS(grid, start, goals, rows, cols, epsilon=2):
    goal = list(goals)[0]
    open_start = [(0, 0, start)]
    open_goal = [(0, 0, goal)]
    g_start = {start: 0}
    g_goal = {goal: 0}
    parent_start = {start: None}
    parent_goal = {goal: None}
    visited_start = set()
    visited_goal = set()
    visited_order = []
    nodes_created = 2
    count = 0

    while open_start and open_goal:
        _, _, current_s = heapq.heappop(open_start)
        visited_start.add(current_s)
        visited_order.append(current_s)

        if current_s in visited_goal:
            path = reconstruct_path_bidir(parent_start, parent_goal, current_s)
            return current_s, nodes_created, path, visited_order

        x, y = current_s
        for dx, dy, move in DIRECTIONS:
            nx, ny = x + dx, y + dy
            neighbor = (nx, ny)
            if (0 <= nx < rows and 0 <= ny < cols and grid[nx][ny] != '#' and neighbor not in visited_start):
                tentative_g = g_start[current_s] + 1
                if neighbor not in g_start or tentative_g < g_start[neighbor]:
                    g_start[neighbor] = tentative_g
                    f = tentative_g + epsilon * heuristic(neighbor, [goal])
                    count += 1
                    heapq.heappush(open_start, (f, count, neighbor))
                    parent_start[neighbor] = (current_s, move)
                    nodes_created += 1

        _, _, current_g = heapq.heappop(open_goal)
        visited_goal.add(current_g)
        visited_order.append(current_g)

        if current_g in visited_start:
            path = reconstruct_path_bidir(parent_start, parent_goal, current_g)
            return current_g, nodes_created, path, visited_order
```

```
        x, y = current_g
        for dx, dy, move in DIRECTIONS:
            nx, ny = x + dx, y + dy
            neighbor = (nx, ny)
            if (0 <= nx < rows and 0 <= ny < cols and grid[nx][ny] != '#' and neighbor not in visited_goal):
                tentative_g = g_goal[current_g] + 1
                if neighbor not in g_goal or tentative_g < g_goal[neighbor]:
                    g_goal[neighbor] = tentative_g
                    f = tentative_g + epsilon * heuristic(neighbor, [start])
                    count += 1
                    heapq.heappush(open_goal, (f, count, neighbor))
                    parent_goal[neighbor] = (current_g, move)
                    nodes_created += 1

    return None, nodes_created, None, visited_order
```

Figure 10 Bidirectional GBFS

For this custom strategy 2, I have chosen to use Bidirectional GBFS, as seen in the figure above, this is the code implemented to run the Bidirectional GBFS search. The BID_GBFS function implements a Bidirectional GBFS with heuristic guidance from both start and goal state. In this program two priority queues are used, one expanding forward from the start and another one expanding backward from the goal state. At each step, the node with the lowest estimated cost is explored from both directions using the equation heuristic + optional epsilon-weighted path cost. The search halts when two of the frontiers meet, reconstructing the complete path using a bidirectional parent map. Thus, this approach is said to be generally faster and more efficient in large spaces than the traditional GBFS.

Testing

Case 1

```
1   [6,12]
2   (0,0)
3   (7,2) | (10,5)
4   (1,2,2,2)
5   (4,1,1,3)
6   (6,1,1,1)
7   (2,4,2,1)
8   (5,5,1,2)
9   (8,3,2,1)
10  (9,5,1,1)
11
```

Figure 11 CaseTest 1

Result:

BFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case1.txt BFS
case1.txt BFS
(7, 2) 44
down,right,right,right,down,right,right,right,right
Visited nodes:
visited: [(0, 0), (0, 1), (1, 0), (0, 2), (1, 1), (2, 0), (0, 3), (2, 1), (3, 0), (0, 4), (3, 1), (4, 0), (0, 5), (1, 4), (3, 2), (5, 0), (0, 6),
(1, 5), (3, 3), (4, 2), (6, 0), (0, 7), (1, 6), (3, 4), (4, 3), (5, 2), (7, 0), (0, 8), (1, 7), (2, 6), (3, 5), (4, 4), (5, 3), (6, 2), (7, 1), (8
, 0), (0, 9), (1, 8), (2, 7), (3, 6), (4, 5), (5, 4), (6, 3), (7, 2), (8, 1), (9, 0), (0, 10), (1, 9), (2, 8), (3, 7), (4, 6), (6, 4), (7, 3)]
PS C:\xampp\htdocs\AI-individual-Assignment>
```

Figure 12 Case1 BFS

DFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case1.txt DFS
case1.txt DFS
(10, 5) 72
down,down,down,down,down,down,down,down,down,right,right,right,right,right,right,right,right,right,right,right,up,left,left,left,left,left,up
,right,right,right,right,right,up,left,left,left,left,left,left,left,left,left,up,right,right,up,left,left,up,right,right,right,right,down,down,ri
ght,right,right,right,right,up,left
Visited nodes:
visited: [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 7), (0, 8), (0, 9), (0, 10), (1, 10), (2, 10), (3, 10), (4, 10), (5, 10), (6
, 10), (7, 10), (8, 10), (9, 10), (10, 10), (11, 10), (11, 9), (10, 9), (9, 9), (8, 9), (7, 9), (6, 9), (6, 8), (7, 8), (8, 8), (9, 8), (10, 8), (
11, 8), (11, 7), (10, 7), (9, 7), (8, 7), (7, 7), (6, 7), (5, 7), (5, 8), (4, 7), (3, 7), (2, 7), (2, 8), (2, 9), (3, 9), (4, 9), (1, 9), (1, 8),
(1, 7), (2, 6), (3, 6), (4, 6), (4, 5), (3, 5), (2, 5), (2, 4), (3, 4), (4, 4), (5, 4), (6, 4), (6, 5), (6, 6), (7, 6), (8, 6), (9, 6), (10, 6), (
11, 6), (11, 5), (10, 5)]
```

Figure 13 Case1 DFS

GBFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case1.txt GBFS
case1.txt GBFS
(7, 2) 11
down,right,right,right,down,right,right,right,right
Visited nodes:
visited: [(0, 0), (0, 1), (1, 0), (0, 2), (1, 1), (0, 3), (2, 1), (3, 1), (2, 0), (3, 2), (3, 0), (3, 3), (4, 2), (4, 3), (5, 2), (5, 3), (6, 2),
(6, 3), (7, 2)]
```

A*Search:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case1.txt AS
case1.txt AS
(7, 2) 19
down,right,right,right,down,right,right,right,right
Visited nodes:
visited: [(0, 0), (0, 1), (1, 0), (0, 2), (1, 1), (2, 0), (0, 3), (2, 1), (3, 0), (3, 1), (4, 0), (3, 2), (5, 0), (3, 3), (4, 2), (6, 0), (4, 3),
(5, 2), (7, 0), (5, 3), (6, 2), (7, 1), (8, 0), (6, 3), (7, 2), (8, 1)]
```

Bidirectional BFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case1.txt BID_BFS
case1.txt BID_BFS
(7, 2) 87
down,right,right,right,down,right,right,right,right,right,right,down,down,right,down
Visited nodes:
visited: [(0, 0), (10, 5), (0, 1), (1, 0), (10, 6), (10, 4), (0, 2), (1, 1), (10, 7), (9, 6), (2, 0), (9, 4), (10, 3), (0, 3), (10, 8), (9, 7), (2
, 1), (8, 6), (3, 0), (9, 3), (0, 4), (10, 2), (3, 1), (10, 9), (9, 8), (4, 0), (8, 7), (0, 5), (1, 4), (7, 6), (8, 5), (3, 2), (9, 2), (5, 0), (1
0, 1), (0, 6), (1, 5), (10, 10), (9, 9), (8, 8), (3, 3), (4, 2), (7, 7), (6, 0), (6, 6), (7, 5), (0, 7), (1, 6), (8, 2), (9, 1), (3, 4), (4, 3), (
10, 0), (5, 2), (10, 11), (9, 10), (7, 0), (8, 9), (0, 8), (1, 7), (7, 8), (2, 6), (6, 7), (3, 5), (4, 4), (5, 6), (5, 3), (7, 4), (6, 2), (7, 2),
 (8, 1), (7, 1), (8, 0), (9, 0), (0, 9), (1, 8), (2, 7), (9, 11), (3, 6), (8, 10), (4, 5), (7, 9), (5, 4), (6, 8), (6, 3), (5, 7), (7, 2)]
```

Bidirectional GBFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case1.txt BID_GBFS
case1.txt BID_GBFS
(0, 1) 58
down,right,right,right,down,right,right,right,right,right,right,down,down,right,down
Visited nodes:
visited: [(0, 0), (10, 5), (0, 1), (10, 4), (0, 2), (9, 4), (0, 3), (9, 3), (0, 4), (9, 2), (0, 5), (8, 2), (1, 5), (7, 2), (1, 4), (6, 2), (1, 6)
, (5, 2), (2, 6), (4, 2), (3, 6), (3, 2), (4, 6), (3, 1), (5, 6), (2, 1), (6, 6), (1, 1), (7, 6), (0, 1)]
```

Case 2

case2.txt

```
[6,10]
(0,1)
(6,5) | (8,2)
(2,1,1,3)
(4,3,1,2)
(6,1,1,2)
(7,4,1,2)
```

Result:

BFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case2.txt BFS
case2.txt BFS
(8, 2) 53
down,right,right,right,right,right,right,right,right
Visited nodes:
visited: [(0, 1), (0, 2), (1, 1), (0, 0), (0, 3), (1, 2), (1, 0), (0, 4), (1, 3), (2, 2), (2, 0), (0, 5), (1, 4), (2, 3), (3, 2), (3, 0), (0, 6),
(1, 5), (2, 4), (3, 3), (4, 2), (4, 0), (0, 7), (1, 6), (2, 5), (3, 4), (5, 2), (5, 0), (0, 8), (1, 7), (2, 6), (3, 5), (4, 4), (6, 2), (5, 1), (6
, 0), (0, 9), (1, 8), (2, 7), (3, 6), (4, 5), (5, 4), (6, 3), (7, 2), (7, 0), (1, 9), (2, 8), (3, 7), (4, 6), (5, 5), (6, 4), (7, 3), (8, 2), (8,
0), (2, 9), (3, 8), (4, 7), (5, 6), (6, 5), (8, 3)]
```

DFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case2.txt DFS
case2.txt DFS
(6, 5) 35
down,down,down,down,down,down,down,right,right,right,up,left,up,right,right,right,down,down,right,right,right,right,up,left,left,left,up,right,rig
ht,right,up,left,left,left
Visited nodes:
visited: [(0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 7), (0, 8), (1, 8), (2, 8), (3, 8), (3, 7), (2, 7), (2, 6), (3, 6), (4, 6), (5, 6),
(5, 7), (5, 8), (6, 8), (7, 8), (8, 8), (9, 8), (9, 7), (8, 7), (7, 7), (6, 7), (6, 6), (7, 6), (8, 6), (9, 6), (9, 5), (8, 5), (7, 5), (6, 5)]
```

GBFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case2.txt GBFS
case2.txt GBFS
(8, 2) 10
down,right,right,right,right,right,right,right,right
Visited nodes:
visited: [(0, 1), (0, 2), (1, 1), (0, 0), (0, 3), (1, 2), (1, 3), (2, 2), (2, 3), (3, 2), (3, 3), (4, 2), (5, 2), (6, 2), (5, 1), (6, 3), (7, 2),
(7, 3), (8, 2)]
```

A*Search:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case2.txt AS
case2.txt AS
(8, 2) 11
down,right,right,right,right,right,right,right,right
Visited nodes:
visited: [(0, 1), (0, 2), (1, 1), (0, 0), (0, 3), (1, 2), (1, 0), (1, 3), (2, 2), (2, 3), (3, 2), (3, 3), (4, 2), (5, 2), (6, 2), (5, 1), (6, 3),
(7, 2), (7, 3), (8, 2)]
```

Bidirectional BFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case2.txt BID_BFS
case2.txt BID_BFS
(3, 2) 35
down,right,right,right,right,right,right,right,right
Visited nodes:
visited: [(0, 1), (8, 2), (0, 2), (1, 1), (0, 0), (8, 3), (7, 2), (8, 1), (0, 3), (1, 2), (7, 3), (1, 0), (6, 2), (8, 0), (0, 4), (1, 3), (6, 3),
(2, 2), (5, 2), (2, 0), (7, 0), (0, 5), (1, 4), (6, 4), (2, 3), (4, 2), (5, 1), (3, 2), (6, 0), (3, 0), (6, 5), (5, 4), (0, 6), (1, 5), (3, 2)]
```

Bidirectional GBFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case2.txt BID_GBFS
case2.txt BID_GBFS
(4, 2) 23
down,right,right,right,right,right,right,right,right
Visited nodes:
visited: [(0, 1), (8, 2), (0, 2), (7, 2), (1, 2), (6, 2), (2, 2), (5, 2), (3, 2), (4, 2), (4, 2)]
```

Case 3

case3.txt

```
[5,12]
(1,0)
(9,4) | (10,7)
(3,2,2,1)
(5,3,1,3)
(7,0,1,2)
(8,5,2,1)
(10,6,1,1)
```

Result:

BFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case3.txt BFS
case3.txt BFS
(9, 4) 87
down,down,down,down,right,right,right,right,right,right,right,right
Visited nodes:
visited: [(1, 0), (1, 1), (2, 0), (0, 0), (1, 2), (2, 1), (0, 1), (3, 0), (1, 3), (2, 2), (0, 2), (3, 1), (4, 0), (1, 4), (2, 3), (0, 3), (4, 1),
(5, 0), (1, 5), (2, 4), (0, 4), (4, 2), (5, 1), (6, 0), (1, 6), (2, 5), (0, 5), (3, 4), (4, 3), (5, 2), (6, 1), (1, 7), (2, 6), (0, 6), (3, 5), (4
, 4), (6, 2), (7, 1), (1, 8), (2, 7), (0, 7), (3, 6), (4, 5), (5, 4), (7, 2), (8, 1), (1, 9), (2, 8), (0, 8), (3, 7), (4, 6), (5, 5), (6, 4), (8,
2), (9, 1), (1, 10), (2, 9), (0, 9), (3, 8), (4, 7), (5, 6), (6, 5), (7, 4), (8, 3), (9, 2), (10, 1), (9, 0), (1, 11), (2, 10), (0, 10), (3, 9), (
4, 8), (5, 7), (6, 6), (7, 5), (8, 4), (9, 3), (10, 2), (10, 0), (2, 11), (0, 11), (3, 10), (4, 9), (5, 8), (6, 7), (7, 6), (9, 4), (10, 3), (3, 1
1), (4, 10), (5, 9), (6, 8), (7, 7)]
```

DFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case3.txt DFS
case3.txt DFS
(10, 7) 33
down,down,down,down,down,down,down,down,down,down,right,right,right,right,up,right,right,down,right,right,right,right,up,left,left,left,up,right,r
ight,right,up,left
Visited nodes:
visited: [(1, 0), (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (1, 8), (1, 9), (1, 10), (2, 10), (3, 10), (4, 10), (5, 10), (5, 9), (6,
 9), (7, 9), (7, 10), (8, 10), (9, 10), (10, 10), (11, 10), (11, 9), (10, 9), (9, 9), (8, 9), (8, 8), (9, 8), (10, 8), (11, 8), (11, 7), (10, 7)]
```

GBFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case3.txt GBFS
case3.txt GBFS
(9, 4) 13
down,down,down,down,right,right,right,right,right,right,right,right
Visited nodes:
visited: [(1, 0), (1, 1), (2, 0), (0, 0), (1, 2), (2, 1), (0, 1), (1, 3), (2, 2), (0, 2), (1, 4), (2, 3), (0, 3), (1, 5), (2, 4), (0, 4), (2, 5),
(3, 4), (3, 5), (4, 4), (4, 5), (5, 4), (4, 3), (5, 5), (6, 4), (6, 5), (7, 4), (7, 5), (8, 4), (9, 4), (8, 3)]
```

A*Search:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case3.txt As
case3.txt AS
(9, 4) 37
down,down,down,down,right,right,right,right,right,right,right,right
Visited nodes:
visited: [(1, 0), (1, 1), (2, 0), (0, 0), (1, 2), (2, 1), (0, 1), (3, 0), (1, 3), (2, 2), (0, 2), (3, 1), (4, 0), (1, 4), (2, 3), (0, 3), (4, 1),
(5, 0), (1, 5), (2, 4), (0, 4), (4, 2), (5, 1), (6, 0), (2, 5), (3, 4), (4, 3), (5, 2), (6, 1), (3, 5), (4, 4), (6, 2), (7, 1), (4, 5), (5, 4), (7
, 2), (8, 1), (5, 5), (6, 4), (8, 2), (9, 1), (6, 5), (7, 4), (8, 3), (9, 2), (10, 1), (9, 0), (7, 5), (8, 4), (9, 3), (10, 2), (9, 4), (10, 3)]
```

Bidirectional BFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case3.txt BID_BFS
case3.txt BID_BFS
(3, 7) 100
down,down,down,down,down,down,down,right,right,right,right,right,right,right,right,right
Visited nodes:
visited: [(1, 0), (10, 7), (1, 1), (2, 0), (0, 0), (10, 8), (9, 7), (1, 2), (2, 1), (0, 1), (10, 9), (9, 8), (3, 0), (8, 7), (9, 6), (10, 10), (9,
 9), (1, 3), (2, 2), (0, 2), (8, 8), (3, 1), (7, 7), (9, 5), (4, 0), (10, 11), (9, 10), (1, 4), (2, 3), (0, 3), (8, 9), (7, 8), (6, 7), (7, 6), (4
, 1), (10, 5), (9, 4), (5, 0), (9, 11), (1, 5), (2, 4), (0, 4), (8, 10), (7, 9), (6, 8), (4, 2), (5, 1), (5, 7), (6, 6), (6, 0), (7, 5), (1, 6), (
2, 5), (0, 5), (10, 4), (3, 4), (8, 4), (9, 3), (8, 11), (4, 3), (5, 2), (7, 10), (6, 1), (6, 9), (5, 8), (1, 7), (2, 6), (0, 6), (4, 7), (5, 6),
(3, 5), (6, 5), (7, 4), (4, 4), (10, 3), (8, 3), (6, 2), (9, 2), (7, 1), (7, 11), (1, 8), (2, 7), (0, 7), (6, 10), (3, 6), (5, 9), (4, 8), (4, 5),
 (3, 7), (4, 6), (5, 4), (5, 5), (7, 2), (6, 4), (8, 1), (1, 9), (2, 8), (0, 8), (10, 2), (3, 7)]
```

Bidirectional GBFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case3.txt BID_GBFS
case3.txt BID_GBFS
(2, 7) 50
down,down,down,down,down,down,down,right,right,right,right,right,right,right,right,right
Visited nodes:
visited: [(1, 0), (10, 7), (1, 1), (9, 7), (1, 2), (8, 7), (1, 3), (7, 7), (1, 4), (6, 7), (1, 5), (5, 7), (1, 6), (4, 7), (1, 7), (3, 7), (2, 7),
 (2, 7)]
```

Case 4

case4.txt

```
1   [11,5]
2   (1,0)
3   (0,7) | (3,10)
4   (2,2,0,2)
5   (2,1,0,8)
6   (1,1,0,10)
7   (2,1,3,2)
8   (1,3,4,3)
9   (1,1,3,9)
10  (1,2,4,8)
```

Result:

BFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case4.txt BFS
case4.txt BFS
(0, 7) 17
left,down,down,down,down,down,down,down
Visited nodes:
visited: [(1, 0), (2, 0), (0, 0), (3, 0), (0, 1), (4, 0), (0, 2), (5, 0), (0, 3), (6, 0), (0, 4), (7, 0), (0, 5), (8, 0), (0, 6), (9, 0), (0, 7),
(10, 0)]
```

DFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case4.txt DFS
case4.txt DFS
(3, 10) 54
right,right,right,right,right,right,down,down,down,down,down,down,down,down,down,down,left,left,left,left
Visited nodes:
visited: [(1, 0), (2, 0), (3, 0), (4, 0), (5, 0), (6, 0), (7, 0), (7, 1), (7, 2), (7, 3), (7, 4), (7, 5), (7, 6), (7, 7), (7, 8), (7, 9), (7, 10),
 (8, 10), (9, 10), (10, 10), (10, 9), (9, 9), (8, 9), (8, 8), (9, 8), (10, 8), (10, 7), (9, 7), (8, 7), (8, 6), (9, 6), (10, 6), (10, 5), (9, 5),
(8, 5), (8, 4), (9, 4), (10, 4), (10, 3), (9, 3), (8, 3), (8, 2), (9, 2), (10, 2), (10, 1), (9, 1), (8, 1), (8, 0), (9, 0), (10, 0), (6, 10), (5,
10), (4, 10), (3, 10)]
```

GBFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case4.txt GBFS
case4.txt GBFS
(0, 7) 9
left,down,down,down,down,down,down,down
Visited nodes:
visited: [(1, 0), (2, 0), (0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 7)]
```

A*Search:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case4.txt as
case4.txt AS
(0, 7) 9
left,down,down,down,down,down,down,down
Visited nodes:
visited: [(1, 0), (2, 0), (0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 7)]
```

Bidirectional BFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case4.txt bid_bfs
case4.txt BID_BFS
(0, 4) 24
left,down,down,down,down,down,down,down
Visited nodes:
visited: [(1, 0), (0, 7), (2, 0), (0, 0), (0, 8), (1, 7), (0, 6), (3, 0), (0, 9), (1, 8), (0, 1), (2, 7), (4, 0), (0, 5), (0, 2), (0, 10), (1, 9),
 (5, 0), (2, 8), (0, 3), (3, 7), (6, 0), (0, 4), (0, 4)]
```

Bidirectional GBFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case4.txt bid_gbfs
case4.txt BID_GBFS
(0, 4) 16
left,down,down,down,down,down,down,down
Visited nodes:
visited: [(1, 0), (0, 7), (0, 0), (1, 7), (0, 1), (0, 6), (0, 2), (0, 5), (0, 3), (0, 4), (0, 4)]
```

Case 5

case5.txt

```
[6,11]
(0,2)
(7,6) | (9,2)
(2,2,1,2)
(4,4,1,2)
(6,1,1,1)
(7,5,1,1)
(8,1,2,1)
```

Result:

BFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case5.txt BFS
case5.txt BFS
(7, 6) 76
down,down,down,down,right,right,right,right,right,right,right
Visited nodes:
visited: [(0, 2), (0, 3), (1, 2), (0, 1), (0, 4), (1, 3), (1, 1), (0, 0), (0, 5), (1, 4), (2, 3), (2, 1), (1, 0), (0, 6), (1, 5), (2, 4), (3, 3),
(3, 1), (2, 0), (0, 7), (1, 6), (2, 5), (3, 4), (4, 3), (4, 1), (3, 0), (0, 8), (1, 7), (2, 6), (3, 5), (5, 3), (4, 2), (5, 1), (4, 0), (0, 9), (1
, 8), (2, 7), (3, 6), (4, 5), (6, 3), (5, 2), (5, 0), (0, 10), (1, 9), (2, 8), (3, 7), (4, 6), (5, 5), (6, 4), (7, 3), (6, 2), (6, 0), (1, 10), (2
, 9), (3, 8), (4, 7), (5, 6), (6, 5), (7, 4), (8, 3), (7, 2), (7, 0), (2, 10), (3, 9), (4, 8), (5, 7), (6, 6), (8, 4), (9, 3), (7, 1), (8, 0), (3,
 10), (4, 9), (5, 8), (6, 7), (7, 6), (8, 5), (9, 4), (9, 2), (9, 0), (4, 10), (5, 9), (6, 8), (7, 7)]
```

DFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case5.txt dfs
case5.txt DFS
(7, 6) 40
down,down,down,down,down,down,down,right,right,right,right,right,right,right,right,right,right,up,left,left,left,left,left,left,left,up,right,up,r
ight,right,down,right,right,right,right,up,left,left,left
Visited nodes:
visited: [(0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 7), (0, 8), (0, 9), (1, 9), (2, 9), (3, 9), (4, 9), (5, 9), (6, 9), (7, 9), (8, 9), (9, 9),
(10, 9), (10, 8), (9, 8), (8, 8), (7, 8), (6, 8), (5, 8), (4, 8), (3, 8), (3, 7), (4, 7), (4, 6), (5, 6), (6, 6), (6, 7), (7, 7), (8, 7), (9, 7),
(10, 7), (10, 6), (9, 6), (8, 6), (7, 6)]
```

GBFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case5.txt GBFS
case5.txt GBFS
(9, 2) 16
right,up,right,right,right,down,right,right,right,down,right,right,up
Visited nodes:
visited: [(0, 2), (0, 3), (1, 2), (0, 1), (1, 3), (1, 1), (2, 1), (1, 0), (3, 1), (2, 0), (4, 1), (3, 0), (4, 2), (5, 1), (4, 0), (4, 3), (5, 2),
(5, 3), (6, 2), (6, 3), (7, 2), (7, 3), (7, 1), (7, 0), (7, 4), (8, 3), (8, 4), (6, 4), (9, 3), (9, 4), (9, 2)]
```

A*Search:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case5.txt as
case5.txt AS
(7, 6) 43
down,down,down,down,right,right,right,right,right,right,right
Visited nodes:
visited: [(0, 2), (0, 3), (1, 2), (0, 1), (1, 3), (1, 1), (0, 0), (0, 4), (0, 5), (1, 4), (2, 1), (1, 0), (2, 3), (0, 6), (1, 5), (2, 4), (3, 1),
(2, 0), (3, 3), (0, 7), (1, 6), (2, 5), (3, 4), (4, 1), (3, 0), (4, 3), (1, 7), (2, 6), (3, 5), (4, 2), (5, 1), (4, 0), (5, 3), (2, 7), (3, 6), (4
, 5), (5, 2), (5, 0), (6, 3), (3, 7), (4, 6), (5, 5), (6, 2), (6, 4), (7, 3), (4, 7), (5, 6), (6, 5), (7, 2), (7, 4), (8, 3), (5, 7), (6, 6), (7,
1), (8, 4), (9, 3), (6, 7), (7, 6), (9, 4), (9, 2)]
```

Bidirectional BFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case5.txt bid_bfs
case5.txt BID_BFS
(3, 6) 78
down,down,down,down,right,right,right,right,right,right,right
Visited nodes:
visited: [(0, 2), (7, 6), (0, 3), (1, 2), (0, 1), (7, 7), (8, 6), (6, 6), (0, 4), (1, 3), (7, 8), (8, 7), (6, 7), (1, 1), (9, 6), (8, 5), (0, 0),
(5, 6), (6, 5), (0, 5), (1, 4), (7, 9), (8, 8), (6, 8), (2, 3), (9, 7), (2, 1), (1, 0), (5, 7), (9, 5), (0, 6), (1, 5), (8, 4), (2, 4), (4, 6), (5
, 5), (3, 3), (6, 4), (3, 1), (2, 0), (7, 10), (8, 9), (6, 9), (9, 8), (0, 7), (1, 6), (5, 8), (2, 5), (3, 4), (4, 7), (4, 3), (9, 4), (4, 1), (3,
 0), (7, 4), (8, 3), (3, 6), (4, 5), (0, 8), (1, 7), (2, 6), (6, 3), (3, 5), (8, 10), (6, 10), (9, 9), (5, 3), (4, 2), (5, 9), (5, 1), (4, 0), (4,
 8), (0, 9), (1, 8), (3, 7), (2, 7), (9, 3), (3, 6)]
```

Bidirectional GBFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case5.txt bid_gbfs
case5.txt BID_GBFS
(2, 6) 33
down,down,down,down,right,right,right,right,right,right,right
Visited nodes:
visited: [(0, 2), (7, 6), (0, 3), (6, 6), (0, 4), (5, 6), (0, 5), (4, 6), (0, 6), (3, 6), (1, 6), (2, 6), (2, 6)]
```

Case 6

case6.txt

```
1   [6,10]
2   (0,0)
3   (7,3) | (10,2)
4   (2,0,1,2)
5   (4,1,1,2)
6   (6,2,1,1)
7   (8,1,1,2)
8   (9,3,1,1)
```

Result:

BFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case6.txt BFS
case6.txt BFS
(7, 3) 47
down,down,down,right,right,right,right,right,right,right
Visited nodes:
visited: [(0, 0), (0, 1), (1, 0), (0, 2), (1, 1), (0, 3), (1, 2), (2, 1), (0, 4), (1, 3), (2, 2), (3, 1), (0, 5), (1, 4), (2, 3), (3, 2), (0, 6),
(1, 5), (2, 4), (3, 3), (4, 2), (0, 7), (1, 6), (2, 5), (3, 4), (4, 3), (5, 2), (0, 8), (1, 7), (2, 6), (3, 5), (4, 4), (5, 3), (0, 9), (1, 8), (2
, 7), (3, 6), (4, 5), (5, 4), (6, 3), (1, 9), (2, 8), (3, 7), (4, 6), (5, 5), (6, 4), (7, 3), (2, 9), (3, 8), (4, 7), (5, 6), (6, 5), (7, 4)]
```

DFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case6.txt dFS
case6.txt DFS
(7, 3) 57
down,right,down,down,right,down,down,right,down,down,down,right,down,down,right,right,right,right,right,up,left,left,left,left,up,right,right,righ
t,right,up,left,left,left,left,left,up,right,right,right,right,right,up,left,left,left,left,left,up,right,right,right,right,right,up,left,left
Visited nodes:
visited: [(0, 0), (0, 1), (1, 1), (1, 2), (1, 3), (2, 3), (2, 4), (2, 5), (3, 5), (3, 6), (3, 7), (3, 8), (4, 8), (4, 9), (4, 10), (5, 10), (6, 10
), (7, 10), (8, 10), (9, 10), (9, 9), (8, 9), (7, 9), (6, 9), (5, 9), (5, 8), (6, 8), (7, 8), (8, 8), (9, 8), (9, 7), (8, 7), (7, 7), (6, 7), (5,
7), (4, 7), (4, 6), (5, 6), (6, 6), (7, 6), (8, 6), (9, 6), (9, 5), (8, 5), (7, 5), (6, 5), (5, 5), (4, 5), (4, 4), (5, 4), (6, 4), (7, 4), (8, 4)
, (9, 4), (9, 3), (8, 3), (7, 3)]
```

GBFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case6.txt gbfs
case6.txt GBFS
(7, 3) 11
down,down,down,right,right,right,right,right,right,right
Visited nodes:
visited: [(0, 0), (0, 1), (1, 0), (0, 2), (1, 1), (0, 3), (1, 2), (0, 4), (1, 3), (1, 4), (2, 3), (2, 4), (3, 3), (2, 2), (3, 4), (4, 3), (3, 2),
(4, 4), (5, 3), (4, 2), (5, 4), (6, 3), (5, 2), (6, 4), (7, 3)]
```

A*Search:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case6.txt as
case6.txt AS
(7, 3) 20
down,down,down,right,right,right,right,right,right,right
Visited nodes:
visited: [(0, 0), (0, 1), (1, 0), (0, 2), (1, 1), (0, 3), (1, 2), (2, 1), (0, 4), (1, 3), (2, 2), (3, 1), (1, 4), (2, 3), (3, 2), (2, 4), (3, 3),
(4, 2), (3, 4), (4, 3), (5, 2), (4, 4), (5, 3), (5, 4), (6, 3), (6, 4), (7, 3)]
```

Bidirectional BFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case6.txt bid_bfs
case6.txt BID_BFS
(5, 3) 65
down,down,down,right,right,right,right,right,right,right,right,up,right,right
Visited nodes:
visited: [(0, 0), (10, 2), (0, 1), (1, 0), (10, 3), (9, 2), (10, 1), (0, 2), (1, 1), (10, 4), (8, 2), (0, 3), (1, 2), (10, 0), (2, 1), (10, 5), (9
, 4), (0, 4), (1, 3), (8, 3), (7, 2), (2, 2), (9, 0), (3, 1), (10, 6), (9, 5), (0, 5), (1, 4), (8, 4), (2, 3), (7, 3), (3, 2), (7, 1), (8, 0), (0,
 6), (1, 5), (10, 7), (9, 6), (2, 4), (8, 5), (3, 3), (7, 4), (4, 2), (6, 3), (0, 7), (1, 6), (6, 1), (7, 0), (2, 5), (3, 4), (10, 8), (9, 7), (4,
 3), (8, 6), (5, 2), (7, 5), (0, 8), (1, 7), (6, 4), (2, 6), (5, 3), (3, 5), (6, 0), (4, 4), (5, 3)]
```

Bidirectional GBFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case6.txt bid_gbfs
case6.txt BID_GBFS
(8, 2) 44
down,down,right,right,right,right,right,down,right,right,right,up,right,right
Visited nodes:
visited: [(0, 0), (10, 2), (0, 1), (9, 2), (0, 2), (8, 2), (1, 2), (7, 2), (2, 2), (7, 1), (3, 2), (6, 1), (4, 2), (6, 0), (5, 2), (5, 0), (5, 3),
 (4, 0), (6, 3), (7, 0), (7, 3), (8, 0), (8, 3), (10, 1), (8, 2)]
```

Case 7

case7.txt

```
1   [5,14]
2   (0,2)
3   (7,8) | (10,5)
4   (2,4,1,3)
5   (4,5,1,2)
6   (6,6,1,1)
7   (8,4,1,2)
8   (9,6,1,1)
```

Result:

BFS:

```
case7.txt BFS
(7, 8) 107
down,down,down,down,down,down,right,right,right,right,right,right,right
Visited nodes:
visited: [(0, 2), (0, 3), (1, 2), (0, 1), (0, 4), (1, 3), (2, 2), (1, 1), (0, 0), (0, 5), (1, 4), (2, 3), (3, 2), (2, 1), (1, 0), (0, 6), (1, 5),
(3, 3), (4, 2), (3, 1), (2, 0), (0, 7), (1, 6), (2, 5), (4, 3), (5, 2), (4, 1), (3, 0), (0, 8), (1, 7), (2, 6), (3, 5), (5, 3), (6, 2), (5, 1), (4
, 0), (0, 9), (1, 8), (2, 7), (3, 6), (5, 4), (6, 3), (7, 2), (6, 1), (5, 0), (0, 10), (1, 9), (2, 8), (3, 7), (4, 6), (6, 4), (7, 3), (8, 2), (7,
 1), (6, 0), (0, 11), (1, 10), (2, 9), (3, 8), (4, 7), (5, 6), (6, 5), (7, 4), (8, 3), (9, 2), (8, 1), (7, 0), (0, 12), (1, 11), (2, 10), (3, 9),
(4, 8), (5, 7), (7, 5), (9, 3), (10, 2), (9, 1), (8, 0), (0, 13), (1, 12), (2, 11), (3, 10), (4, 9), (5, 8), (6, 7), (7, 6), (8, 5), (10, 3), (10,
 1), (9, 0), (1, 13), (2, 12), (3, 11), (4, 10), (5, 9), (6, 8), (7, 7), (8, 6), (9, 5), (10, 4), (10, 0), (2, 13), (3, 12), (4, 11), (5, 10), (6,
 9), (7, 8), (8, 7), (10, 5), (3, 13), (4, 12), (5, 11), (6, 10), (7, 9)]
```

DFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case7.txt dfs
case7.txt DFS
(7, 8) 30
down,down,down,down,down,down,down,down,right,right,right,right,right,right,right,right,right,right,right,right,right,up,left,left,left,left,left,
left,up
Visited nodes:
visited: [(0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 7), (0, 8), (0, 9), (0, 10), (1, 10), (2, 10), (3, 10), (4, 10), (5, 10), (6, 10), (7, 10),
(8, 10), (9, 10), (10, 10), (11, 10), (12, 10), (13, 10), (13, 9), (12, 9), (11, 9), (10, 9), (9, 9), (8, 9), (7, 9), (7, 8)]
```

GBFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case7.txt gbfs
case7.txt GBFS
(7, 8) 14
down,down,down,down,down,down,right,right,right,right,right,right,right
Visited nodes:
visited: [(0, 2), (0, 3), (1, 2), (0, 1), (0, 4), (1, 3), (0, 5), (1, 4), (0, 6), (1, 5), (0, 7), (1, 6), (0, 8), (1, 7), (0, 9), (1, 8), (1, 9),
(2, 8), (2, 9), (3, 8), (2, 7), (3, 9), (4, 8), (3, 7), (4, 9), (5, 8), (4, 7), (5, 9), (6, 8), (5, 7), (6, 9), (7, 8), (6, 7)]
```

A*Search:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case7.txt as
case7.txt AS
(7, 8) 59
down,down,down,down,down,down,right,right,right,right,right,right,right
Visited nodes:
visited: [(0, 2), (0, 3), (1, 2), (0, 1), (0, 4), (1, 3), (2, 2), (1, 1), (0, 5), (1, 4), (2, 3), (3, 2), (2, 1), (0, 6), (1, 5), (3, 3), (4, 2),
(3, 1), (0, 7), (1, 6), (2, 5), (4, 3), (5, 2), (4, 1), (0, 8), (1, 7), (2, 6), (3, 5), (5, 3), (6, 2), (5, 1), (0, 9), (1, 8), (2, 7), (3, 6), (5
, 4), (6, 3), (7, 2), (6, 1), (1, 9), (2, 8), (3, 7), (4, 6), (6, 4), (7, 3), (8, 2), (7, 1), (2, 9), (3, 8), (4, 7), (5, 6), (6, 5), (7, 4), (8,
3), (9, 2), (8, 1), (3, 9), (4, 8), (5, 7), (7, 5), (9, 3), (10, 2), (9, 1), (4, 9), (5, 8), (6, 7), (7, 6), (8, 5), (10, 3), (10, 1), (5, 9), (6,
 8), (7, 7), (8, 6), (9, 5), (10, 4), (6, 9), (7, 8), (8, 7), (10, 5)]
```

Bidirectional BFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case7.txt bid_bfs
case7.txt BID_BFS
(5, 4) 84
down,right,right,right,right,right,down,right,down,right,right,right,right
Visited nodes:
visited: [(0, 2), (10, 5), (0, 3), (1, 2), (0, 1), (10, 6), (9, 5), (10, 4), (0, 4), (1, 3), (10, 7), (2, 2), (1, 1), (8, 5), (0, 0), (10, 3), (0,
 5), (1, 4), (10, 8), (9, 7), (2, 3), (8, 6), (7, 5), (3, 2), (2, 1), (9, 3), (10, 2), (1, 0), (10, 9), (9, 8), (8, 7), (0, 6), (1, 5), (7, 6), (6
, 5), (7, 4), (3, 3), (8, 3), (9, 2), (4, 2), (3, 1), (10, 1), (2, 0), (10, 10), (9, 9), (8, 8), (0, 7), (1, 6), (7, 7), (2, 5), (4, 3), (6, 4), (
5, 2), (4, 1), (7, 3), (3, 0), (8, 2), (9, 1), (0, 8), (1, 7), (10, 0), (2, 6), (10, 11), (9, 10), (3, 5), (8, 9), (5, 3), (7, 8), (6, 2), (5, 1),
 (6, 7), (4, 0), (5, 4), (6, 3), (7, 2), (0, 9), (1, 8), (8, 1), (2, 7), (9, 0), (3, 6), (10, 12), (9, 11), (5, 4)]
```

Bidirectional GBFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case7.txt bid_gbfs
case7.txt BID_GBFS
(1, 3) 43
down,right,right,right,right,right,down,right,down,right,right,right,right
Visited nodes:
visited: [(0, 2), (10, 5), (0, 3), (9, 5), (0, 4), (8, 5), (0, 5), (7, 5), (1, 5), (6, 5), (2, 5), (6, 4), (3, 5), (5, 4), (1, 4), (5, 3), (3, 6),
 (4, 3), (4, 6), (3, 3), (5, 6), (2, 3), (1, 3), (1, 3)]
```

Case 8

case8.txt

```
[6,12]
(1,1)
(8,3) | (10,4)
(3,2,1,3)
(5,3,1,2)
(6,2,1,1)
(7,4,1,2)
(9,3,1,1)
```

Result:

BFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case8.txt BFS
case8.txt BFS
(8, 3) 64
right,right,right,right,right,right,down,down,right
Visited nodes:
visited: [(1, 1), (1, 2), (2, 1), (0, 1), (1, 0), (1, 3), (2, 2), (0, 2), (3, 1), (2, 0), (0, 0), (1, 4), (2, 3), (0, 3), (4, 1), (3, 0), (1, 5),
(2, 4), (0, 4), (3, 3), (5, 1), (4, 0), (1, 6), (2, 5), (0, 5), (3, 4), (4, 3), (6, 1), (5, 0), (1, 7), (2, 6), (0, 6), (3, 5), (4, 4), (7, 1), (6
, 0), (1, 8), (2, 7), (0, 7), (3, 6), (4, 5), (5, 4), (7, 2), (8, 1), (7, 0), (1, 9), (2, 8), (0, 8), (3, 7), (4, 6), (5, 5), (6, 4), (7, 3), (8,
2), (9, 1), (8, 0), (1, 10), (2, 9), (0, 9), (3, 8), (4, 7), (5, 6), (6, 5), (8, 3), (9, 2), (10, 1), (9, 0), (1, 11), (2, 10), (0, 10), (3, 9), (
4, 8), (5, 7), (6, 6), (7, 5)]
```

DFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case8.txt dfs
case8.txt DFS
(10, 4) 65
down,down,down,down,down,down,down,down,down,down,right,right,right,right,right,right,right,right,right,right,up,left,left,left,left,left,left,up,right
,right,right,right,right,right,up,left,left,left,left,left,left,up,right,right,right,right,right,right,up,left,left,left,left,left,left,left,up,ri
ght,right,right,right,right,right
Visited nodes:
visited: [(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (1, 8), (1, 9), (1, 10), (2, 10), (3, 10), (4, 10), (5, 10), (6, 10), (7, 10), (
8, 10), (9, 10), (10, 10), (11, 10), (11, 9), (10, 9), (9, 9), (8, 9), (7, 9), (6, 9), (5, 9), (4, 9), (5, 8), (6, 8), (7, 8), (8, 8), (9, 8), (10
, 8), (11, 8), (11, 7), (10, 7), (9, 7), (8, 7), (7, 7), (6, 7), (5, 7), (5, 6), (6, 6), (7, 6), (8, 6), (9, 6), (10, 6), (11, 6), (11, 5), (10, 5
), (9, 5), (8, 5), (7, 5), (6, 5), (5, 5), (4, 5), (4, 6), (4, 4), (5, 4), (6, 4), (7, 4), (8, 4), (9, 4), (10, 4)]
```

GBFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case8.txt gbfs
case8.txt GBFS
(10, 4) 15
down,down,right,right,right,down,right,right,down,right,right,right,up,right
Visited nodes:
visited: [(1, 1), (1, 2), (2, 1), (0, 1), (1, 0), (1, 3), (2, 2), (0, 2), (1, 4), (2, 3), (0, 3), (2, 4), (3, 3), (3, 4), (4, 3), (4, 4), (4, 5),
(5, 4), (5, 5), (6, 4), (6, 5), (6, 6), (7, 5), (7, 6), (8, 5), (8, 6), (9, 5), (9, 6), (10, 5), (9, 4), (10, 4)]
```

A*Search:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case8.txt as
case8.txt AS
(8, 3) 18
right,right,right,right,right,right,down,down,right
Visited nodes:
visited: [(1, 1), (1, 2), (2, 1), (0, 1), (1, 0), (1, 3), (2, 2), (0, 2), (3, 1), (2, 0), (1, 4), (2, 3), (0, 3), (4, 1), (3, 0), (2, 4), (3, 3),
(5, 1), (4, 0), (3, 4), (4, 3), (6, 1), (5, 0), (4, 4), (7, 1), (6, 0), (7, 2), (8, 1), (7, 0), (7, 3), (8, 2), (9, 1), (8, 0), (8, 3), (9, 2)]
```

Bidirectional BFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case8.txt bid_bfs
case8.txt BID_BFS
(7, 1) 73
right,right,right,right,right,right,down,right,right,right,down,down
Visited nodes:
visited: [(1, 1), (10, 4), (1, 2), (2, 1), (0, 1), (1, 0), (10, 5), (9, 4), (10, 3), (1, 3), (2, 2), (0, 2), (10, 6), (9, 5), (3, 1), (2, 0), (0,
0), (10, 2), (10, 7), (9, 6), (1, 4), (2, 3), (0, 3), (8, 5), (9, 2), (10, 1), (10, 8), (9, 7), (4, 1), (3, 0), (8, 6), (7, 5), (8, 2), (9, 1), (1
, 5), (2, 4), (0, 4), (10, 0), (3, 3), (10, 9), (9, 8), (8, 7), (5, 1), (4, 0), (7, 6), (6, 5), (1, 6), (2, 5), (0, 5), (8, 3), (7, 2), (8, 1), (3
, 4), (9, 0), (4, 3), (10, 10), (9, 9), (6, 1), (5, 0), (8, 8), (7, 7), (1, 7), (2, 6), (0, 6), (6, 6), (3, 5), (5, 5), (6, 4), (7, 3), (4, 4), (7
, 1), (8, 0), (7, 1)]
```

Bidirectional GBFS:

```
PS C:\xampp\htdocs\AI-individual-Assignment> python search.py case8.txt bid_gbfs
case8.txt BID_GBFS
(1, 1) 64
right,right,right,right,right,right,down,right,right,right,down,down
Visited nodes:
visited: [(1, 1), (10, 4), (1, 2), (9, 4), (1, 3), (10, 3), (1, 4), (10, 2), (2, 4), (9, 2), (3, 4), (8, 2), (4, 4), (7, 2), (5, 4), (7, 1), (6, 4
), (6, 1), (6, 5), (5, 1), (7, 5), (4, 1), (8, 5), (3, 1), (9, 5), (2, 1), (10, 5), (1, 1)]
```

Features/Bugs/Missing

One of the list of features I have implemented would be the multiple search Algorithms like Breadth-First Search, Depth-First Search, Greedy Best-First Search and other, another is the dynamic map parsing where it can reads the input map dimensions, start state, goal state and the wall from the file, another is the flexible goal handling that support multiple goal locations of the first reachable goal with correct detection, another is the visited node tracking where it can help us keep track of visited or explored nodes in the order that they were explored and print them for debugging.

One of the bugs I encountered is the Goal selection in Bidirectional Algorithms, BID_BFS and BID_GBFS only use the first goal instead of the second goal, this can result in limitation when multiple goals are provided. Preferably, it should have dynamically check for intersection with any of the other goals.

In my code if one think is missing it would be the visualizer, although my implementation have the program that can print out the visited nodes and path, I wasn't able to code in visualization for my program like using matplotlib to animate or display the map.

Conclusion

In conclusion, this assignment helps me understand the in-depth exploration into the design and implementation of common and advanced search algorithms for robot navigation test in a grid-based environment. Through the development of the six search algorithms, BFS, DFS, GBFS, A*S, Bidirectional BFS, and Bidirectional GBFS, I gained the understanding of uninformed and informed search methods, their capabilities and their limitations in terms of path optimality, speed and memory usage, Therefore, the project improved my thinking of AI pathfinding techniques and provided me with hands on experience in search algorithms design.

Acknowledgements/ Resources

1. ChatGPT (openAI)
   I have used ChatGPT to help me clarify the design and debugging of search algorithms such as BFS, DFS, A*S and Bidirectional strategies. ChatGPT also assisted in refining my code and generate test case interpretations, and give me assistance in writing this report.

   Prompt used include:
   - "Explain how bidirectional BFS works."
   - "Help me fix this DFS implementation."
   - “Teach and guide me how to code the search Algorithms”
   - “How to fix the issue in my DFS”
   - “Why are the output of my search Algorithms like this”
   - "What are the bugs or missing features in this code?"
   - “Guide me how to write this report”

Reference

Lima, S.D. (2020). A quick explanation of DFS & BFS (Depth First Search & Breadth-First Search). [online] Medium. Available at: https://medium.com/analytics-vidhya/a-quick-explanation-of-dfs-bfs-depth-first-search-breadth-first-search-b9ef4caf952c.

GeeksforGeeks (2022). Greedy Best first search algorithm. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/greedy-best-first-search-algorithm/.

Belwariar, R. (2018). A Search Algorithm - GeeksforGeeks. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/a-search-algorithm.*