1. Members (Group 12)

Hasan Basri UZUN	20058071
Onur TURAN	19058006
Zeynep YILDIZ	19058909
Buse Sena VARDAR	20052024

2. The definition of the problem

We designed two mazes with sizes of 10 x 10 and 26 x 26, and allowed the user to choose which maze and the starting and ending points to use. We tested the shortest path between these two coordinates using 7 search algorithms and the Manhattan Distance heuristic function. To move within the maze, we defined four actions: go up, go down, go right, and go left.

Data Types

The program defines several data types that are specific to the problem.

typedef struct COORDINATE

It has been created in the form of a binary coordinate system in the shape of x and y, and its definition has been made in this struct.

enum ACTIONS

The enum ACTIONS data type defines six possible actions that can be taken by an agent in the maze: GO_east, GO_west, GO_northeast, GO_northwest, GO_southeast and GO_southwest. These actions are used by the search algorithms to generate new states and explore the maze.

struct State

The struct State data type represents a cell in the maze. It contains three fields: h_n, x, and y.

h_n is the heuristic function value of the cell.

x is the x-coordinate of the cell and the columns for the array.

y is the y-coordinate of the cell and the rows for the array.

The enum MAZES field specifies which maze the cell belongs to.

enum METHODS

The enum METHODS data type lists the different search algorithms that can be used by the program. It currently lists seven algorithms: BreastFirstSearch, UniformCostSearch, DepthFirstSearch, DepthLimitedSearch, IterativeDeepeningSearch, GreedySearch, and AStarSearch.

struct Transition_Model

The struct Transition_Model data type represents a new state that can be reached by taking a specific action from a current state. It contains two fields: new_state and step_cost.

new_state is a struct State object representing the new state.

step_cost is the cost of taking the action that leads to the new state.

struct Node

The data type “Node” defines a single node in a search tree, which has four fields: “state”, “path cost”, “action”, and “parent”. The “state” field is a struct object representing the state of the node, “path_cost” denotes the cost of reaching the node from the initial state, “action” specifies the action taken to reach the node from its parent, and “parent” is a pointer to the parent node.

Additionally, the “Number_of_Child” field is used specifically for depth-first search algorithms to keep track of the number of child nodes.

struct Queue

The data type “queue” defines a queue used for storing nodes in search algorithms, twho fields: node and next. The node field is a pointer to a “node” object, representing the node to be stored in the queue. The “next” field is a pointer to the next “queue” object in the queue, allowing for the implementation of a linked list data structure to maintain the order of nodes in the queue.

Search Algorithms

The program implements seven search algorithms using the data types defined above.

Breadth-First Search

The search algorithm known as Breadth-First Search (BFS) is executed by utilizing a queue data structure. Starting from the initial node, the algorithm explores all nodes that are directly adjacent to the initial node, before proceeding to nodes at a distance of 1 from the initial node, and continuing to explore nodes at increasing distances from the initial node in a breadth-first manner.

Uniform Cost Search

To implement the Uniform Cost Search algorithm, a prioriyy queueu data structure is utilized. The algorithm proceeds by expanding the node with the lowest path cost, and then explores all neighboring nodes that have not yet been explored. In this manner, the algorithm prioritizes finding the lowest-cost path to the goal node.

Depth-First Search

The Depth-First Search algorithm traverses a search tree by employing a stack data structure and exploring the nodes at increasing depths. In other words, the algorithm begins at the root node and explores as far as possible along each branch before backtracking to explore the next branch at the next lower depth level.

Depth-Limited Search

The Depth-Limited Search algorithm is a modification of the Depth-First Search algorithm, which restricts the macimum depth of exploration. Similar to Depth-First Search, algorithm employs a stack data structure.

Iterative Deepening Search

The Iterative Deepening Search algorithm is a type of Depth-First Search algorithm that gradually increases the depth of exploration until the goal state is reached. This search algorithm also utilizes a stack data structure, similar to Depth-Firsh Search, in order to traverse the search tree.

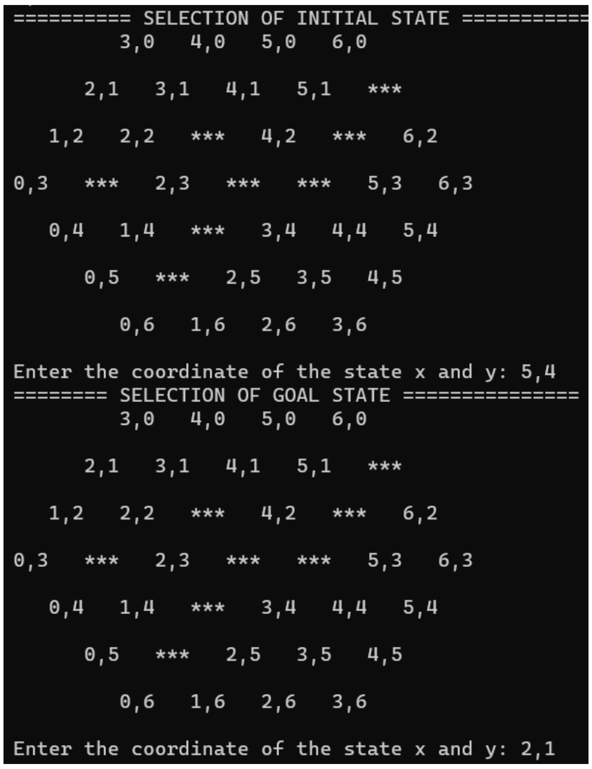
Greedy Best-First Search

The Greedy Best-First Search algorithm utilizes a priority queue data structure to prioritize the node closest to the goal state for expansion. The algorithm determines the proximity of a node to the goal state based on a heuristic function that approximates the distance between the node and the goal. In this manner, the algorithm attempts to find the best possible path to the goal state by prioritizing nodes that appear to be closest to it.

A* Search

The A* search algorithm is an informed search algorithm that draws on the strenghts of both Breadth-First and Best-First search algorithms. It coniders both the actual path cost from the start node and an estimated cost to reach the goal node, determined using a heuristic function. A priority queue data structure is employed to implement the algorithm, enabling the nodes to be ordered according to their estimated cost to the goal node. By considering both the actual path cost and estimated cost, A* search aims to find the optimal path to the goal node in the most efficient manner possible.

First Map:

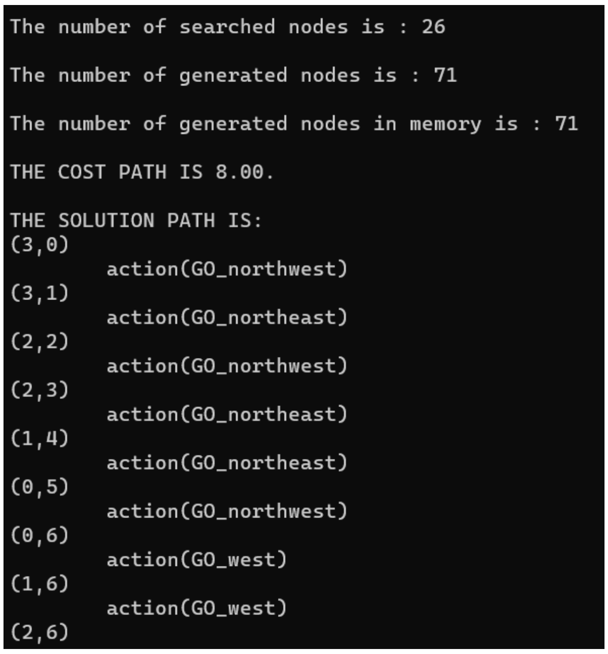


```
========== SELECTION OF INITIAL STATE ===========
         3,0   4,0   5,0   6,0

      2,1   3,1   4,1   5,1   ***

   1,2   2,2   ***   4,2   ***   6,2

0,3   ***   2,3   ***   ***   5,3   6,3

   0,4   1,4   ***   3,4   4,4   5,4

      0,5   ***   2,5   3,5   4,5

         0,6   1,6   2,6   3,6

Enter the coordinate of the state x and y: 5,4
======== SELECTION OF GOAL STATE ===============
         3,0   4,0   5,0   6,0

      2,1   3,1   4,1   5,1   ***

   1,2   2,2   ***   4,2   ***   6,2

0,3   ***   2,3   ***   ***   5,3   6,3

   0,4   1,4   ***   3,4   4,4   5,4

      0,5   ***   2,5   3,5   4,5

         0,6   1,6   2,6   3,6

Enter the coordinate of the state x and y: 2,1
```

1 --> Breast-First Search

Initial State: 2,6

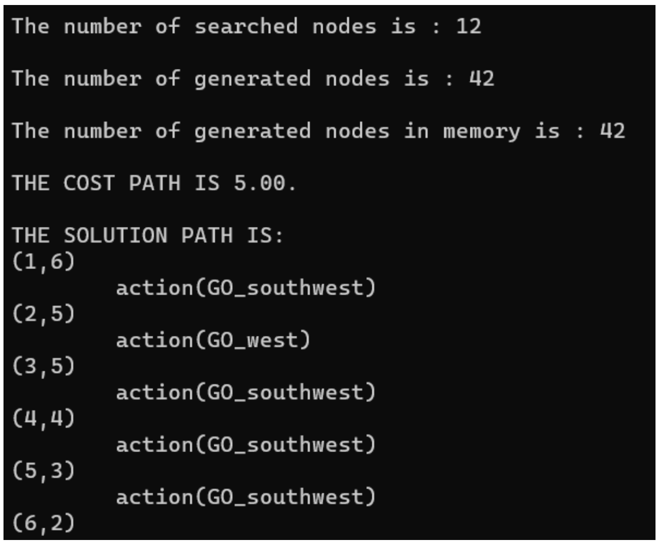
Goal State: 3,0



```
The number of searched nodes is : 26

The number of generated nodes is : 71

The number of generated nodes in memory is : 71

THE COST PATH IS 8.00.

THE SOLUTION PATH IS:
(3,0)
        action(GO_northwest)
(3,1)
        action(GO_northeast)
(2,2)
        action(GO_northwest)
(2,3)
        action(GO_northeast)
(1,4)
        action(GO_northeast)
(0,5)
        action(GO_northwest)
(0,6)
        action(GO_west)
(1,6)
        action(GO_west)
(2,6)
```

2 --> Uniform-Cost Search
Initial State: 1,5

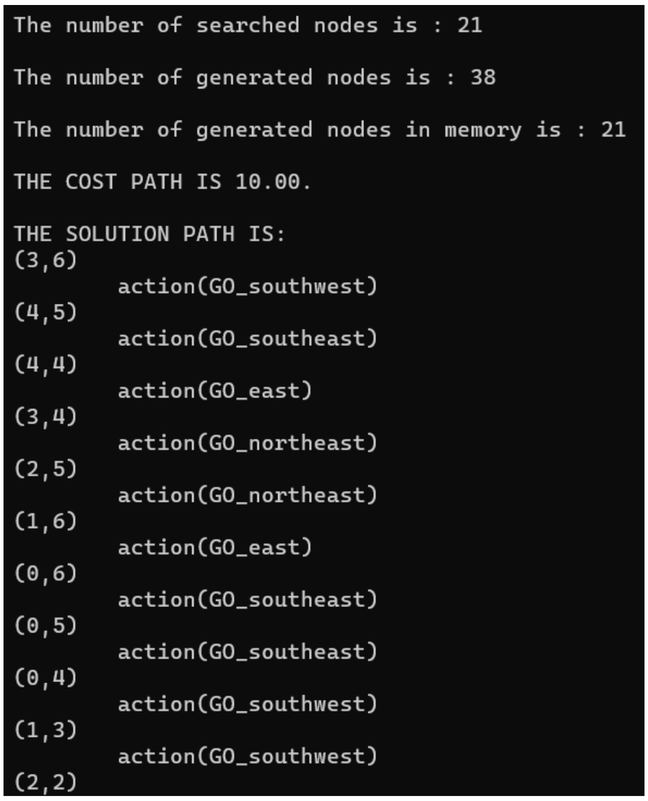
Goal State: 6,2



```
The number of searched nodes is : 12

The number of generated nodes is : 42

The number of generated nodes in memory is : 42

THE COST PATH IS 5.00.

THE SOLUTION PATH IS:
(1,6)
        action(GO_southwest)
(2,5)
        action(GO_west)
(3,5)
        action(GO_southwest)
(4,4)
        action(GO_southwest)
(5,3)
        action(GO_southwest)
(6,2)
```

3 --> Depth-First Search
Initial State: 2,2

Goal State: 3,6

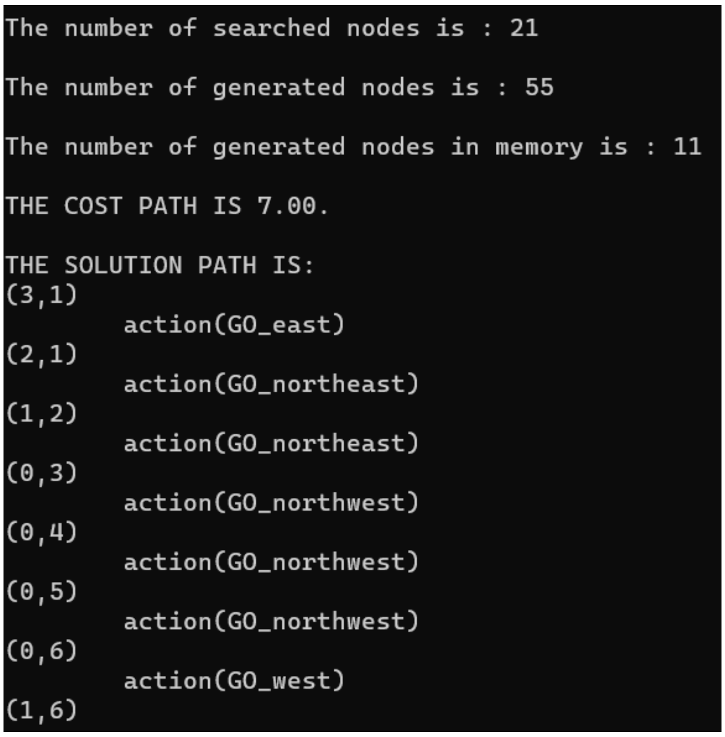


```
The number of searched nodes is : 21

The number of generated nodes is : 38

The number of generated nodes in memory is : 21

THE COST PATH IS 10.00.

THE SOLUTION PATH IS:
(3,6)
        action(GO_southwest)
(4,5)
        action(GO_southeast)
(4,4)
        action(GO_east)
(3,4)
        action(GO_northeast)
(2,5)
        action(GO_northeast)
(1,6)
        action(GO_east)
(0,6)
        action(GO_southeast)
(0,5)
        action(GO_southeast)
(0,4)
        action(GO_southwest)
(1,3)
        action(GO_southwest)
(2,2)
```

4 --> Depth-Limited Search

maximum level for depth-limited search : 7
Initial State: 1,6

Goal State: 3,1

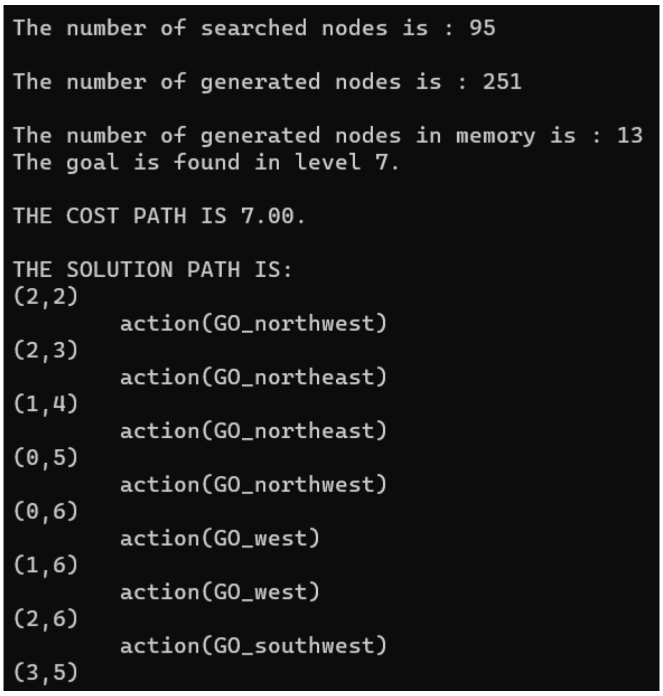


```
The number of searched nodes is : 21

The number of generated nodes is : 55

The number of generated nodes in memory is : 11

THE COST PATH IS 7.00.

THE SOLUTION PATH IS:
(3,1)
        action(GO_east)
(2,1)
        action(GO_northeast)
(1,2)
        action(GO_northeast)
(0,3)
        action(GO_northwest)
(0,4)
        action(GO_northwest)
(0,5)
        action(GO_northwest)
(0,6)
        action(GO_west)
(1,6)
```

5 --> Iterative Deepening Search

Initial State: 3,5

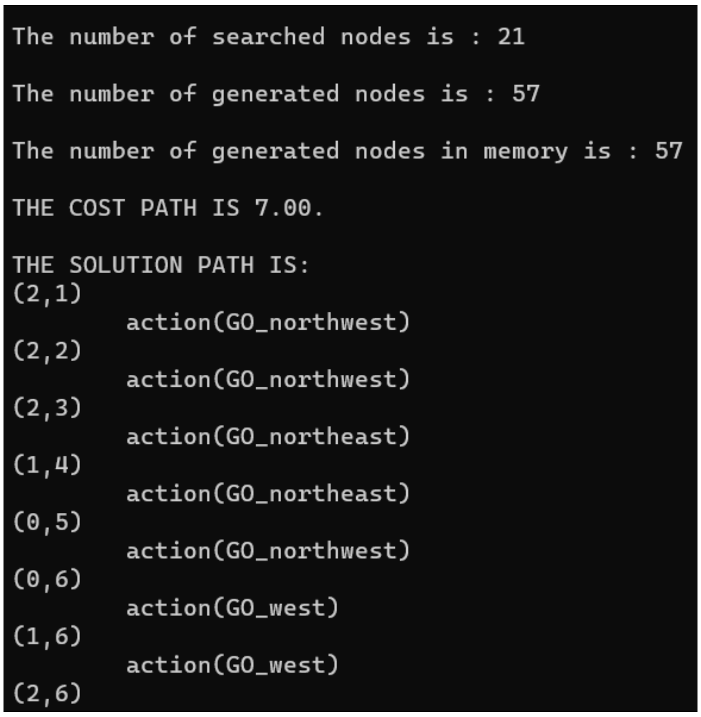
Goal State: 2,2



```
The number of searched nodes is : 95

The number of generated nodes is : 251

The number of generated nodes in memory is : 13
The goal is found in level 7.

THE COST PATH IS 7.00.

THE SOLUTION PATH IS:
(2,2)
        action(GO_northwest)
(2,3)
        action(GO_northeast)
(1,4)
        action(GO_northeast)
(0,5)
        action(GO_northwest)
(0,6)
        action(GO_west)
(1,6)
        action(GO_west)
(2,6)
        action(GO_southwest)
(3,5)
```

6 --> Greedy Search
Initial State: 2,6

Goal State: 2,1

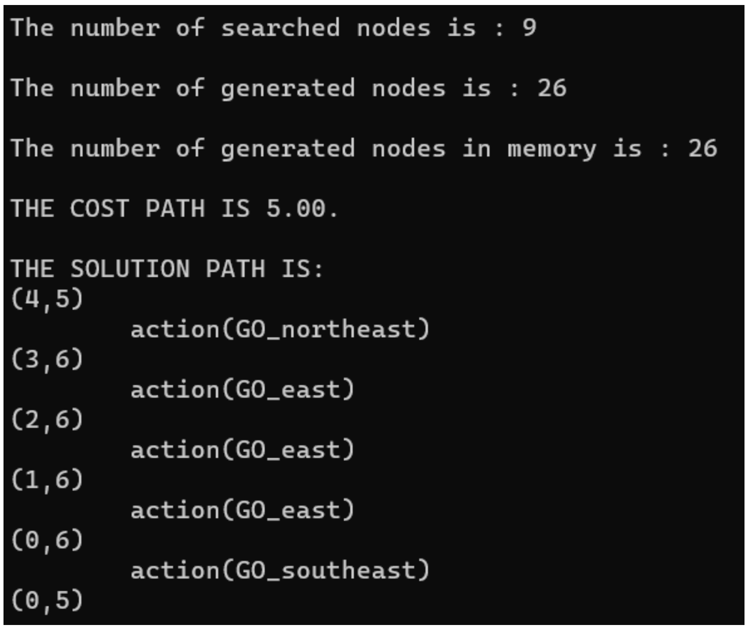


```
The number of searched nodes is : 21

The number of generated nodes is : 57

The number of generated nodes in memory is : 57

THE COST PATH IS 7.00.

THE SOLUTION PATH IS:
(2,1)
        action(GO_northwest)
(2,2)
        action(GO_northwest)
(2,3)
        action(GO_northeast)
(1,4)
        action(GO_northeast)
(0,5)
        action(GO_northwest)
(0,6)
        action(GO_west)
(1,6)
        action(GO_west)
(2,6)
```

7 --> A* Search

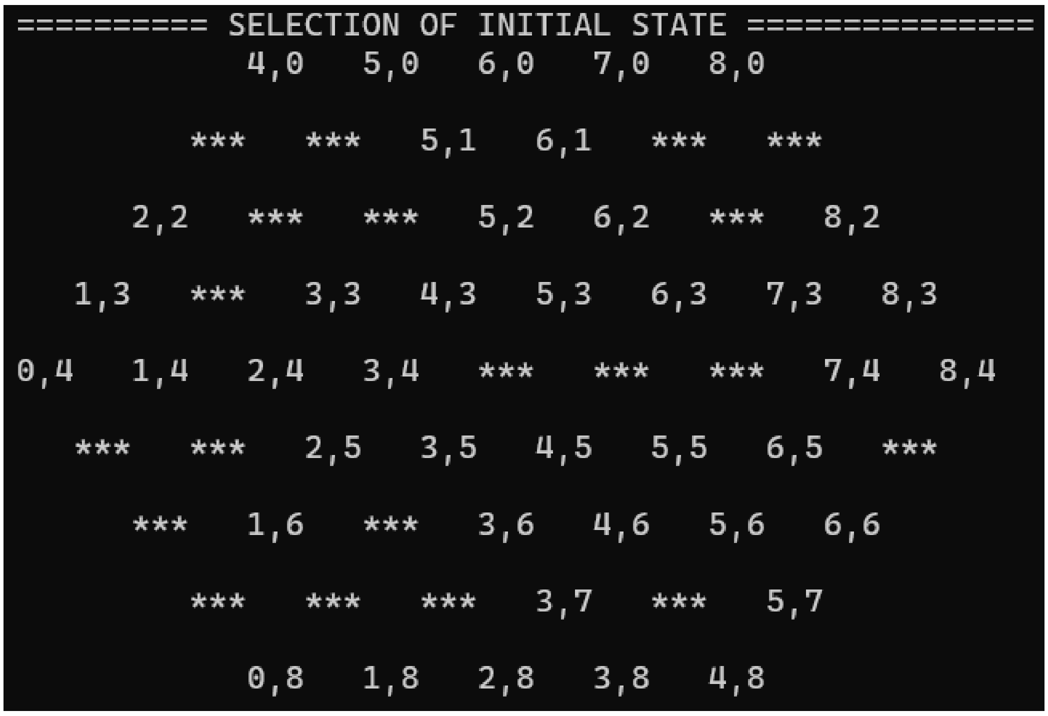
Initial State: 0,5

Goal State: 4,5



```
The number of searched nodes is : 9

The number of generated nodes is : 26

The number of generated nodes in memory is : 26

THE COST PATH IS 5.00.

THE SOLUTION PATH IS:
(4,5)
        action(GO_northeast)
(3,6)
        action(GO_east)
(2,6)
        action(GO_east)
(1,6)
        action(GO_east)
(0,6)
        action(GO_southeast)
(0,5)
```

Second Map:

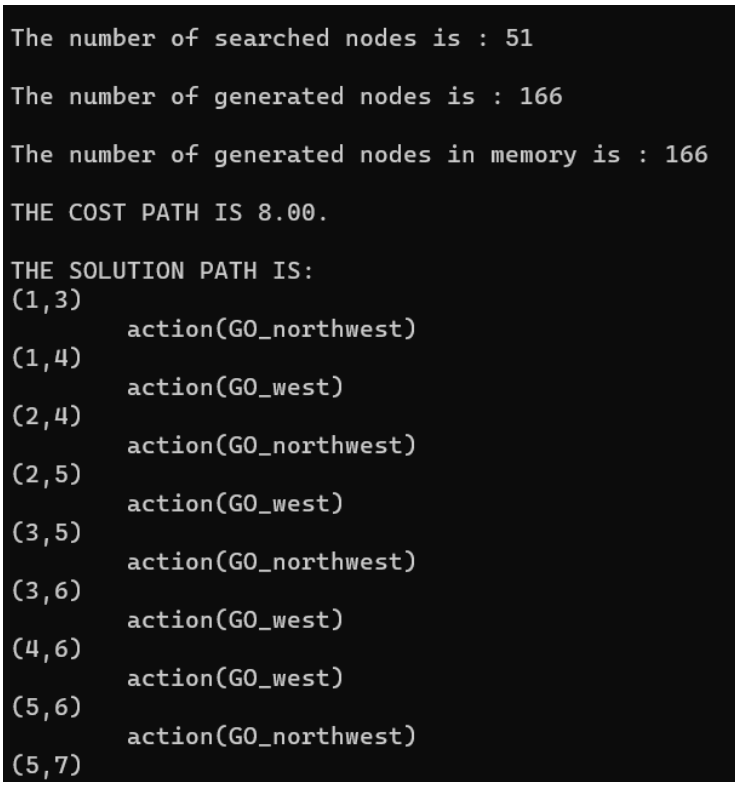


```
========== SELECTION OF INITIAL STATE ===============
            4,0   5,0   6,0   7,0   8,0

         ***   ***   5,1   6,1   ***   ***

      2,2   ***   ***   5,2   6,2   ***   8,2

   1,3   ***   3,3   4,3   5,3   6,3   7,3   8,3

0,4   1,4   2,4   3,4   ***   ***   ***   7,4   8,4

   ***   ***   2,5   3,5   4,5   5,5   6,5   ***

      ***   1,6   ***   3,6   4,6   5,6   6,6

         ***   ***   ***   3,7   ***   5,7

            0,8   1,8   2,8   3,8   4,8
```

1 --> Breast-First Search

Initial State: 5,7

Goal State: 1,3

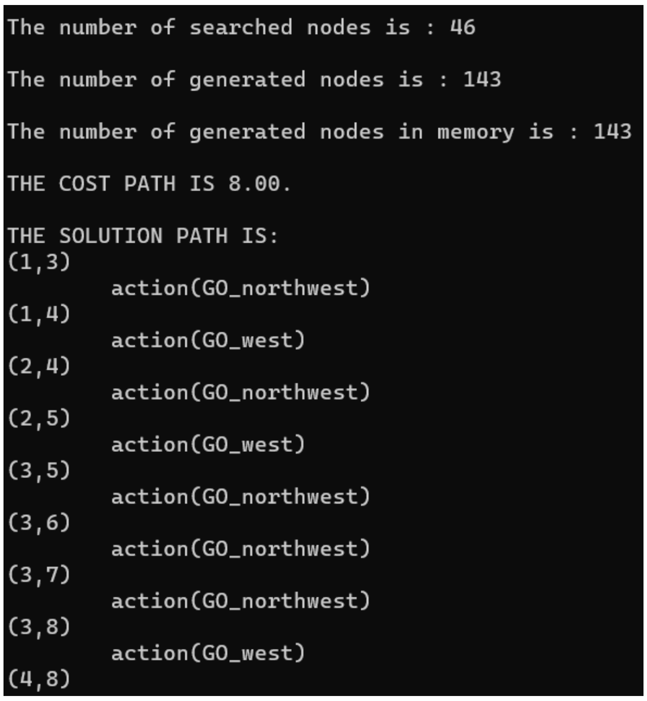


```
The number of searched nodes is : 51

The number of generated nodes is : 166

The number of generated nodes in memory is : 166

THE COST PATH IS 8.00.

THE SOLUTION PATH IS:
(1,3)
        action(GO_northwest)
(1,4)
        action(GO_west)
(2,4)
        action(GO_northwest)
(2,5)
        action(GO_west)
(3,5)
        action(GO_northwest)
(3,6)
        action(GO_west)
(4,6)
        action(GO_west)
(5,6)
        action(GO_northwest)
(5,7)
```

2 --> Uniform-Cost Search

Initial State: 4,8

Goal State: 1,3

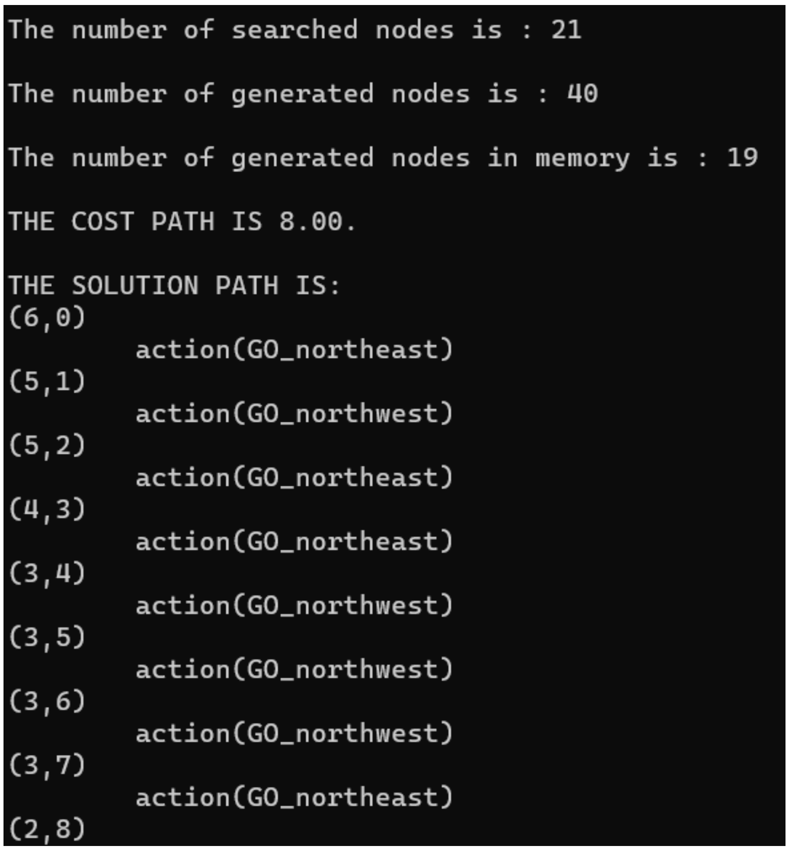


```
The number of searched nodes is : 46

The number of generated nodes is : 143

The number of generated nodes in memory is : 143

THE COST PATH IS 8.00.

THE SOLUTION PATH IS:
(1,3)
        action(GO_northwest)
(1,4)
        action(GO_west)
(2,4)
        action(GO_northwest)
(2,5)
        action(GO_west)
(3,5)
        action(GO_northwest)
(3,6)
        action(GO_northwest)
(3,7)
        action(GO_northwest)
(3,8)
        action(GO_west)
(4,8)
```

3 --> Depth-First Search

Initial State: 2,8

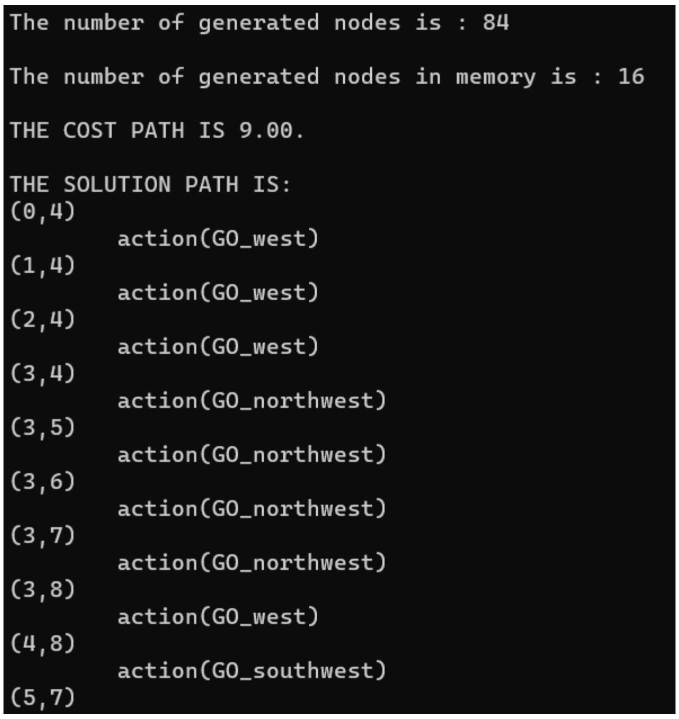
Goal State: 6,0



```
The number of searched nodes is : 21

The number of generated nodes is : 40

The number of generated nodes in memory is : 19

THE COST PATH IS 8.00.

THE SOLUTION PATH IS:
(6,0)
        action(GO_northeast)
(5,1)
        action(GO_northwest)
(5,2)
        action(GO_northeast)
(4,3)
        action(GO_northeast)
(3,4)
        action(GO_northwest)
(3,5)
        action(GO_northwest)
(3,6)
        action(GO_northwest)
(3,7)
        action(GO_northeast)
(2,8)
```

4 --> Depth-Limited Search
Initial State: 5,7

Goal State: 0,4

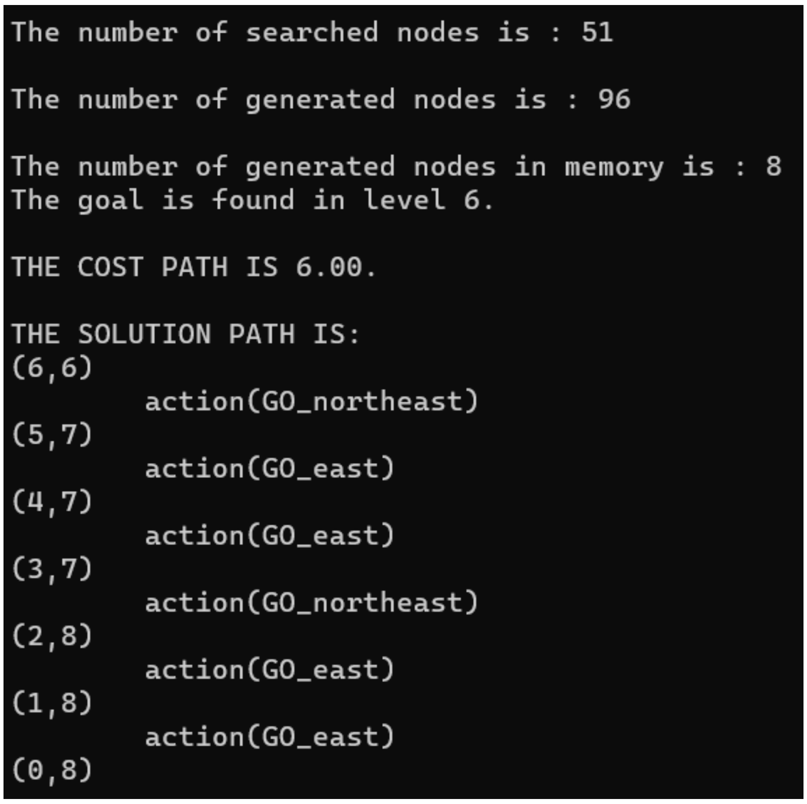


```
The number of generated nodes is : 84

The number of generated nodes in memory is : 16

THE COST PATH IS 9.00.

THE SOLUTION PATH IS:
(0,4)
        action(GO_west)
(1,4)
        action(GO_west)
(2,4)
        action(GO_west)
(3,4)
        action(GO_northwest)
(3,5)
        action(GO_northwest)
(3,6)
        action(GO_northwest)
(3,7)
        action(GO_northwest)
(3,8)
        action(GO_west)
(4,8)
        action(GO_southwest)
(5,7)
```

5 --> Iterative Deepening Search

Initial State: 0,8

Goal State: 6,6

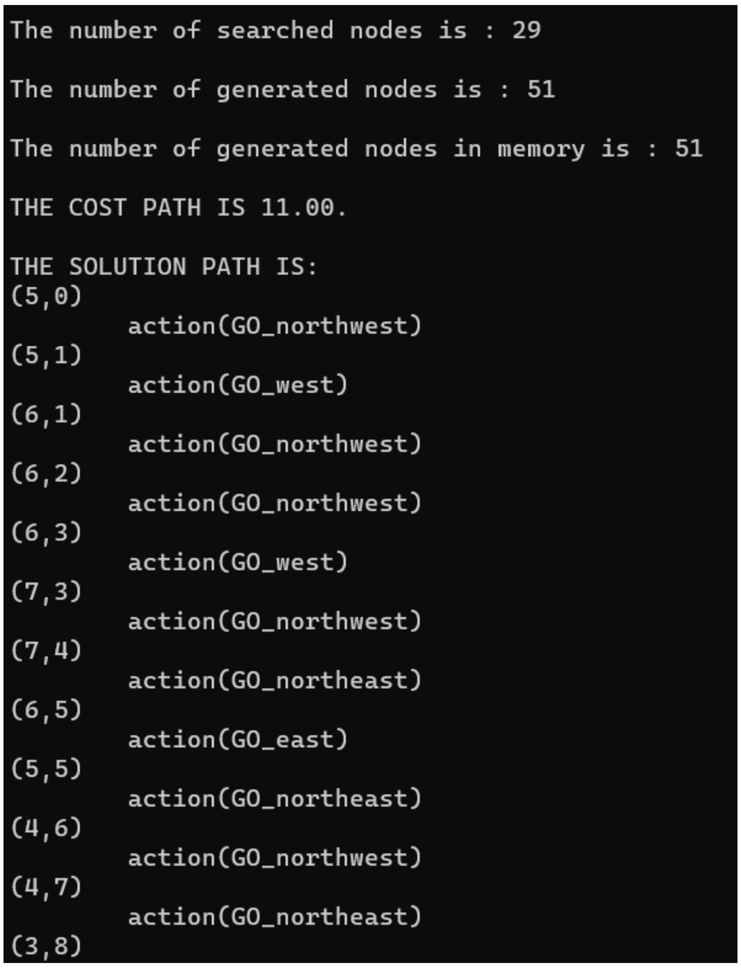


```
The number of searched nodes is : 51

The number of generated nodes is : 96

The number of generated nodes in memory is : 8
The goal is found in level 6.

THE COST PATH IS 6.00.

THE SOLUTION PATH IS:
(6,6)
        action(GO_northeast)
(5,7)
        action(GO_east)
(4,7)
        action(GO_east)
(3,7)
        action(GO_northeast)
(2,8)
        action(GO_east)
(1,8)
        action(GO_east)
(0,8)
```

6 --> Greedy Search

Initial State: 3,8

Goal State: 5,0

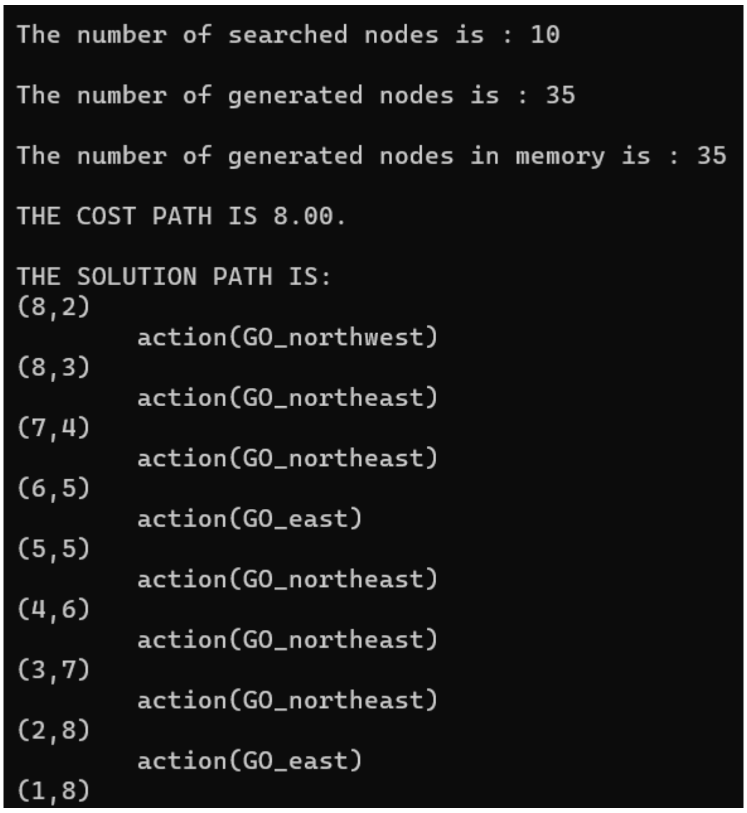


```
The number of searched nodes is : 29

The number of generated nodes is : 51

The number of generated nodes in memory is : 51

THE COST PATH IS 11.00.

THE SOLUTION PATH IS:
(5,0)
        action(GO_northwest)
(5,1)
        action(GO_west)
(6,1)
        action(GO_northwest)
(6,2)
        action(GO_northwest)
(6,3)
        action(GO_west)
(7,3)
        action(GO_northwest)
(7,4)
        action(GO_northeast)
(6,5)
        action(GO_east)
(5,5)
        action(GO_northeast)
(4,6)
        action(GO_northwest)
(4,7)
        action(GO_northeast)
(3,8)
```

7 --> A* Search

Initial State: 1,8

Goal State: 8,2



```
The number of searched nodes is : 10

The number of generated nodes is : 35

The number of generated nodes in memory is : 35

THE COST PATH IS 8.00.

THE SOLUTION PATH IS:
(8,2)
        action(GO_northwest)
(8,3)
        action(GO_northeast)
(7,4)
        action(GO_northeast)
(6,5)
        action(GO_east)
(5,5)
        action(GO_northeast)
(4,6)
        action(GO_northeast)
(3,7)
        action(GO_northeast)
(2,8)
        action(GO_east)
(1,8)
```