Figure 1: University of Arkansas Logo

CSCE 46103/56103: Introduction to Artificial Intelligence

Homework #1: Uniform and Informed Search

Submission Deadline: 11:59 PM, September 2^{nd}, 2025

Instructions

- **Written Format & Template:** Students can use either Google Doc or LaTeXfor writing the report.
- Write your full name, email address, and student ID in the report.
- Submission through BlackBoard.
- **Submissions:** Your submission should be a zip file containing your report with the screenshots of your outputs, and the source code including your implementation.
- **Name the zip file as `lastname_studentID.zip`**
- submission should be made via black board.
- **Policy:** Review the late days policy and include the total number of "Late Days" used in your report.

Question 1: BFS, DFS [10 pts]

You are given an $m \times n$ matrix (zero-based indices). Each cell contains 0 or 1; 0 means blocked, 1 means accessible. Starting at $(0, 0)$, find a path to (m, n). If you intend the lower-right cell in zero-based indexing, this is usually $(m-1, n-1)$. You may move up, down, left, or right to an in-bounds cell with value 1, and you may visit each cell at most once.

1	1	1	0	1
0	0	1	0	0
1	1	1	1	1
1	1	0	1	1
1	1	0	1	1

Output format:

$$(0,0) \to (0,1) \to (0,2) \to (1,2) \to (2,2) \to (2,3) \to (3,3) \to (4,3) \to (4,4)$$

Notes: If multiple paths exist, print any one. If no path exists, print -1.

(a) [5 pts] Implement DFS to solve the problem.

```
def DFS(A):
    # A is an m x n matrix
    pass
```

(b) [5 pts] Implement BFS to solve the problem.

```
def BFS(A):
    # A is an m x n matrix
    pass
```

Question 2: DFS, BFS, UCS [20 pts]

Use the provided graph consisting of **seven** states with their respective connection costs. Work through the problem manually. Do not use programming or code implementation in your solution.

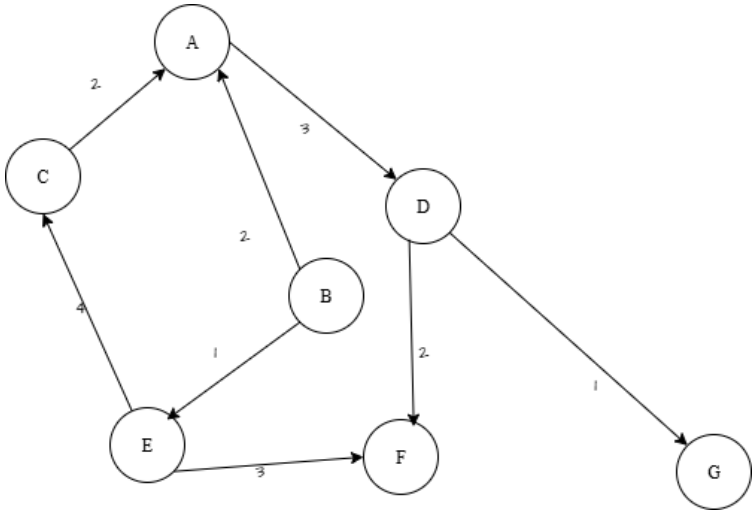

Figure 2: Search graph for Q2.

(a) [10 pts] Perform Depth-First Search (DFS) from B to G.

(a.1) Return the DFS search table.

(a.2) What is the path returned?

(a.3) State the node expansion order.

(a.4) What is the cost of the returned path?

(b) [5 pts] Perform Breadth-First Search (BFS) from B to G.

(b.1) Return the BFS search table.

(b.2) What is the path returned?

(b.3) State the node expansion order.

(b.4) What is the cost of the returned path?

(c) [5 pts] Perform Uniform-Cost Search (UCS) from B to G.

(c.1) Return the UCS search table.

(c.2) What is the path returned?

(c.3) What is the node expansion order?

(c.4) What is the cost of the returned path?

Question 3: Uninformed Search and Informed Search [20 pts]

Using the provided graph with **six** states and the indicated connection costs, apply the following search strategies. Work through each method manually, showing the order of node expansions, the path returned, and the corresponding path cost. Do not use programming or code implementation in your solution.

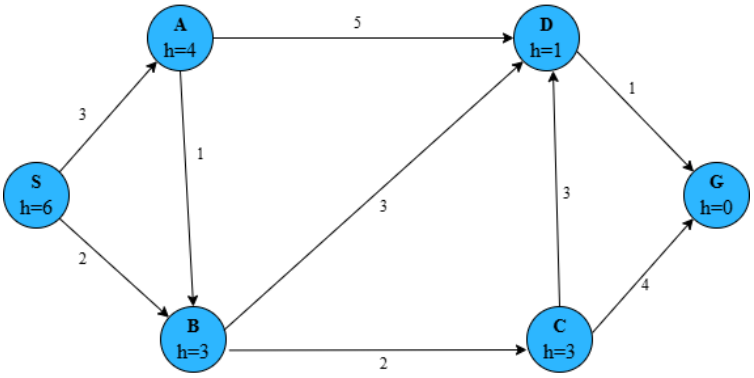

Figure 3: Search graph for Q3.

(a) [**5 pts**] Perform Depth-First Search (DFS) from S to G.

(a.1) State the path returned

(a.2) Compute the total path cost.

(b) [**5 pts**] Perform Breadth-First Search (BFS) from S to G.

(b.1) State the path returned

(b.2) Compute the total path cost.

(c) [**5 pts**] Perform Uniform-Cost Search (UCS) from S to G.

(c.1) State the path returned

(c.2) Compute the total path cost.

(d) [**5 pts**] Perform A* search from S to G.

(d.1) State the path returned

(d.2) Compute the total path cost.

Question 4: [10pts] A* Search

The graph shown below includes link costs (on the edges) and heuristic estimates (next to the states). The arrows are undirected. Let state M be the **start** and state P be the **goal**.

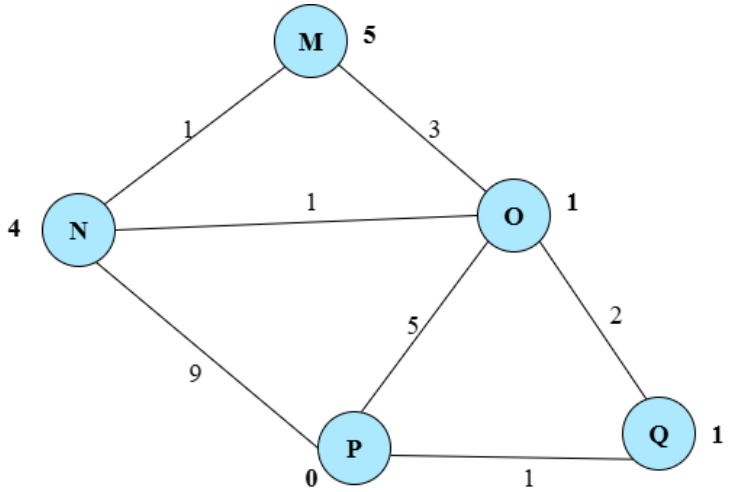

Figure 4: Search graph for Q4.

Please transcribe *only* the information requested into the table given below.

Path to State Expanded	**Length of Path** $g(n)$	**Total Estimated Cost** $f(n)$	**Expanded List**
M	0	5	(M)

Simulate the **A* search algorithm** on this graph using a strict expanded list. For each expansion step, provide:

(a) The path from the start state to the node being expanded.

(b) The actual path cost $g(n)$ (sum of link costs along the path).

(c) The total estimated cost $f(n) = g(n) + h(n)$.

(d) The current contents of the expanded list (states in order).

Question 5: [40pts] Search

In this question, your objective is to implement search algorithms in Python that enable Pacman to navigate a maze and reach the food. You are provided with the maze layout, the starting point, and the food location, and your goal is to determine a valid path.

Code Files

Filename	Purpose
`search.py`	You will implement your algorithm in this file.
`searchAgents.py`	This file will be used to call your implemented algorithms.
`pacman.py`	This file is the main program that runs the Pacman game.
`util.py`	This file consists of useful data structures.
`game.py`	Defines how Pacman is played. Describes AgentState, Agent, Direction, Grid.
`graphicsDisplay.py`, `graphicsUtils.py`, `textDisplay.py`, `ghostDisplay.py`, `keyboardAgents.py`, `layout.py`	These files are used to render the Pacman game. You can ignore these files.

Acknowledgement: The base code is borrowed from the Introduction to AI course of the University of California, Berkeley.

1. Instructions

a. Environment

Use Python 3.6+ to run the game:

```
python pacman.py
```

b. Searching Function Format

```
def search_fn(problem):
    # Your task: implement a searching function
    # The function will return a list of actions
```

The problem is an instance of the `PositionSearchProblem` class from `searchAgents.py`, and supports:

- `getStartState()`
- `isWall(current)`

- `isGoalState(current)`
- `getSuccessors(current)`

Example:

```
# search.py
def tinyMazeSearch(problem):
    from game import Directions
    s = Directions.SOUTH
    w = Directions.WEST
    return [s, s, w, s, w, w, s, w]
```

Run:

```
python pacman.py -l tinyMaze -p SearchAgent -a fn=tinyMazeSearch
```

Fails for:

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=tinyMazeSearch
# It will be failed
```

2. Searching Algorithms

Implement search algorithms that guide Pacman to the food, ensuring he collides with walls at least once but no more than twice. If Pacman hits zero walls or more than two, the attempt will be considered a failure.

Algorithms to implement:

- Depth First Search (DFS)
- Breadth First Search (BFS)
- Uniform Cost Search (UCS)
- A* Search (Manhattan Heuristic)

Use `Queue`, `PriorityQueue`, and `Stack` from `util.py`.

a. Depth First Search (10 points)

```
# search.py
def depthFirstSearch(problem):
    YOUR CODE HERE
```

Test DFS:

```
python pacman.py -l tinyMaze -p SearchAgent -a fn=dfs
python pacman.py -l mediumMaze -p SearchAgent -a fn=dfs
python pacman.py -l bigMaze -p SearchAgent -a fn=dfs
```

b. Breadth First Search (10 points)

```
# search.py
def breadthFirstSearch(problem):
   YOUR CODE HERE
```

Test BFS:

```
python pacman.py -l tinyMaze -p SearchAgent -a fn=bfs
python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs
python pacman.py -l bigMaze -p SearchAgent -a fn=bfs
```

c. Uniform Cost Search (10 points)

```
# search.py
def uniformCostSearch(problem):
   YOUR CODE HERE
```

Test UCS:

```
python pacman.py -l tinyMaze -p SearchAgent -a fn=ucs
python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs
python pacman.py -l bigMaze -p SearchAgent -a fn=ucs
```

d. A* Search (10 points)

```
# search.py
def nullHeuristic(state, problem=None):
    # A heuristic function estimates the cost from the current state to the nearest
    # goal in the provided SearchProblem. This heuristic is trivial.
    return 0

def aStarSearch(problem, heuristic=nullHeuristic):
    YOUR CODE HERE
```

Test A*:

```
python pacman.py -l tinyMaze -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic
python pacman.py -l mediumMaze -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic
python pacman.py -l bigMaze -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic
```