AI MID TERM LAB

Submitted By:

Shehzad Ahmad

(SP23-BSE-088)

Submitted To:

Mam Zeenat Zulfiqar

Date: October 24, 2025

This task involves solving the classic Missionaries and Cannibals river-crossing puzzle using three AI search algorithms — Breadth-First Search (BFS), Depth-First Search (DFS), and A*. The goal is to safely move all three missionaries and three cannibals from the left bank to the right bank using a boat that can carry two people at a time, without ever letting cannibals outnumber missionaries on either side.

Each algorithm represents the state as (M_left, C_left, Boat_position), generates valid successor states, and displays all intermediate states from start to goal.

BFS explores all states level by level to find the shortest path.

DFS explores deeper paths first.

A* uses a heuristic to find an optimal and efficient solution.

```
from collections import deque
import heapq

# ---------- Common Functions ----------
def is_valid_state(M, C):
    """Check if the state is valid."""
    if M < 0 or M > 3 or C < 0 or C > 3:
        return False
    if (M > 0 and M < C):  # left bank: missionaries outnumbered
        return False
    if (3 - M > 0 and 3 - M < 3 - C):  # right bank: missionaries
outnumbered
        return False
    return True


def get_successors(state):
    """Generate all valid successor states."""
    M, C, boat = state
    successors = []
    moves = [(1, 0), (2, 0), (0, 1), (0, 2), (1, 1)]  # possible boat
trips

    for m, c in moves:
        if boat == 'left':
            new_state = (M - m, C - c, 'right')
        else:
            new_state = (M + m, C + c, 'left')

        if is_valid_state(new_state[0], new_state[1]):
            successors.append(new_state)
    return successors


def print_path(path):
```

```python
    print("\nPath to goal:")
    for step in path:
        print(step)


# ---------- 1. Breadth-First Search ----------
def bfs(start, goal):
    queue = deque([(start, [start])])
    visited = set()

    while queue:
        state, path = queue.popleft()
        if state == goal:
            print("BFS Solution:")
            print_path(path)
            return path

        if state not in visited:
            visited.add(state)
            for next_state in get_successors(state):
                if next_state not in visited:
                    queue.append((next_state, path + [next_state]))


# ---------- 2. Depth-First Search ----------
def dfs(start, goal):
    stack = [(start, [start])]
    visited = set()

    while stack:
        state, path = stack.pop()
        if state == goal:
            print("\nDFS Solution:")
            print_path(path)
            return path

        if state not in visited:
            visited.add(state)
            for next_state in get_successors(state):
                if next_state not in visited:
                    stack.append((next_state, path + [next_state]))


# ---------- 3. A* Search ----------
def heuristic(state):
    """Estimate remaining people on left bank (missionaries +
cannibals)."""
    M, C, boat = state
    return M + C
```

```python
def a_star(start, goal):
    open_list = []
    heapq.heappush(open_list, (heuristic(start), 0, start, [start]))
    visited = set()

    while open_list:
        f, g, state, path = heapq.heappop(open_list)
        if state == goal:
            print("\nA* Solution:")
            print_path(path)
            return path

        if state not in visited:
            visited.add(state)
            for next_state in get_successors(state):
                new_g = g + 1
                new_f = new_g + heuristic(next_state)
                heapq.heappush(open_list, (new_f, new_g, next_state,
path + [next_state]))


# ---------- Run All Algorithms ----------
if __name__ == "__main__":
    start_state = (3, 3, 'left')
    goal_state = (0, 0, 'right')

    bfs(start_state, goal_state)
    dfs(start_state, goal_state)
    a_star(start_state, goal_state)
```

```
BFS Solution:

Path to goal:
(3, 3, 'left')
(3, 1, 'right')
(3, 2, 'left')
(3, 0, 'right')
(3, 1, 'left')
(1, 1, 'right')
(2, 2, 'left')
(0, 2, 'right')
(0, 3, 'left')
(0, 1, 'right')
(1, 1, 'left')
(0, 0, 'right')

DFS Solution:
```

```
Path to goal:
(3, 3, 'left')
(2, 2, 'right')
(3, 2, 'left')
(3, 0, 'right')
(3, 1, 'left')
(1, 1, 'right')
(2, 2, 'left')
(0, 2, 'right')
(0, 3, 'left')
(0, 1, 'right')
(0, 2, 'left')
(0, 0, 'right')

A* Solution:

Path to goal:
(3, 3, 'left')
(2, 2, 'right')
(3, 2, 'left')
(3, 0, 'right')
(3, 1, 'left')
(1, 1, 'right')
(2, 2, 'left')
(0, 2, 'right')
(0, 3, 'left')
(0, 1, 'right')
(0, 2, 'left')
(0, 0, 'right')
```

This code builds the full state graph (showing all valid moves) and highlights the BFS solution path from start to goal.

```
import networkx as nx
import matplotlib.pyplot as plt
from collections import deque

# ---------- Helper Functions ----------
def is_valid_state(M, C):
    if M < 0 or M > 3 or C < 0 or C > 3:
        return False
    if (M > 0 and M < C):
        return False
    if (3 - M > 0 and 3 - M < 3 - C):
        return False
    return True


def get_successors(state):
```

```python
    M, C, boat = state
    successors = []
    moves = [(1, 0), (2, 0), (0, 1), (0, 2), (1, 1)]

    for m, c in moves:
        if boat == 'left':
            new_state = (M - m, C - c, 'right')
        else:
            new_state = (M + m, C + c, 'left')

        if is_valid_state(new_state[0], new_state[1]):
            successors.append(new_state)
    return successors


# ---------- BFS Search ----------
def bfs(start, goal):
    queue = deque([(start, [start])])
    visited = set()

    while queue:
        state, path = queue.popleft()
        if state == goal:
            return path

        if state not in visited:
            visited.add(state)
            for next_state in get_successors(state):
                if next_state not in visited:
                    queue.append((next_state, path + [next_state]))
    return None


# ---------- Build State Graph ----------
def build_graph():
    G = nx.DiGraph()
    states = [(3, 3, 'left')]
    visited = set()

    while states:
        state = states.pop()
        if state not in visited:
            visited.add(state)
            for next_state in get_successors(state):
                G.add_edge(state, next_state)
                states.append(next_state)
    return G
```

```
# ---------- Visualize Graph ----------
def draw_graph(G, solution_path):
    pos = nx.spring_layout(G, seed=42)  # layout for visualization
    plt.figure(figsize=(12, 8))

    # Draw all nodes
    nx.draw_networkx_nodes(G, pos, node_color='lightblue',
node_size=700)
    nx.draw_networkx_edges(G, pos, arrows=True, edge_color='gray')
    nx.draw_networkx_labels(G, pos, font_size=8)

    # Highlight solution path
    path_edges = list(zip(solution_path, solution_path[1:]))
    nx.draw_networkx_edges(G, pos, edgelist=path_edges,
edge_color='red', width=2)
    nx.draw_networkx_nodes(G, pos, nodelist=solution_path,
node_color='orange', node_size=800)

    plt.title("Missionaries and Cannibals State Graph (BFS Path
Highlighted)")
    plt.axis('off')
    plt.show()


# ---------- Run ----------
if __name__ == "__main__":
    start_state = (3, 3, 'left')
    goal_state = (0, 0, 'right')

    G = build_graph()
    path = bfs(start_state, goal_state)

    if path:
        print("BFS Solution Path:")
        for step in path:
            print(step)
        draw_graph(G, path)
    else:
        print("No solution found.")

BFS Solution Path:
(3, 3, 'left')
(3, 1, 'right')
(3, 2, 'left')
(3, 0, 'right')
(3, 1, 'left')
(1, 1, 'right')
(2, 2, 'left')
(0, 2, 'right')
(0, 3, 'left')
```

```
(0, 1, 'right')
(1, 1, 'left')
(0, 0, 'right')
```

Missionaries and Cannibals State Graph (BFS Path Highlighted)