EXPERIMENT NO. 3

Student Name and Roll Number: Piyush Gambhir – 21CSU349
Semester /Section: Semester-V – AIML-V-B (AL-3)
Link to Code: NCU-Lab-Manual-And-End-Semester-Projects/NCU-CSL347-AAIES-Lab_Manual at main · Piyush-Gambhir/NCU-Lab-Manual-And-End-Semester-Projects (github.com)
Date: 26.08.2023
Faculty Signature:
Grade:

Objective(s):

- Understand what State Space Search is.
- Study about how state spaces work and how state space search algorithms work.
- Implement State Space Search for solving a real-world problem.

Outcome:

Student will be familiarized with the State Space Search algorithm.

Problem Statement:

Implement a basic state space search program in Python to solve the classic "Missionaries and Cannibals" puzzle. The goal is to implement a simple program that finds a sequence of valid moves to safely transport three missionaries and three cannibals across a river, following specific constraints.

Background Study:

State space search is a problem-solving technique that navigates through a set of possible states to find a solution. It represents the problem as a graph or tree, with each node representing a state and edges as valid transitions. Algorithms like BFS and DFS are used to explore the state space efficiently, finding solutions for puzzles, games, and optimization tasks by minimizing search effort and avoiding revisiting already explored states using queues and sets. In the "Missionaries and Cannibals" puzzle, state space search helps identify a valid sequence of moves to safely transport individuals while respecting constraints.

Question Bank:

1. What is the state space search technique?
   State space search technique involves systematically exploring the possible states of a problem to find a solution. It's commonly used in AI and computer science to solve problems by representing the problem's possible configurations as states in a graph or tree and then applying search algorithms to traverse and find the optimal solution.

2. Discuss the role of Breadth-First Search (BFS) in solving the "Missionaries and Cannibals" puzzle.

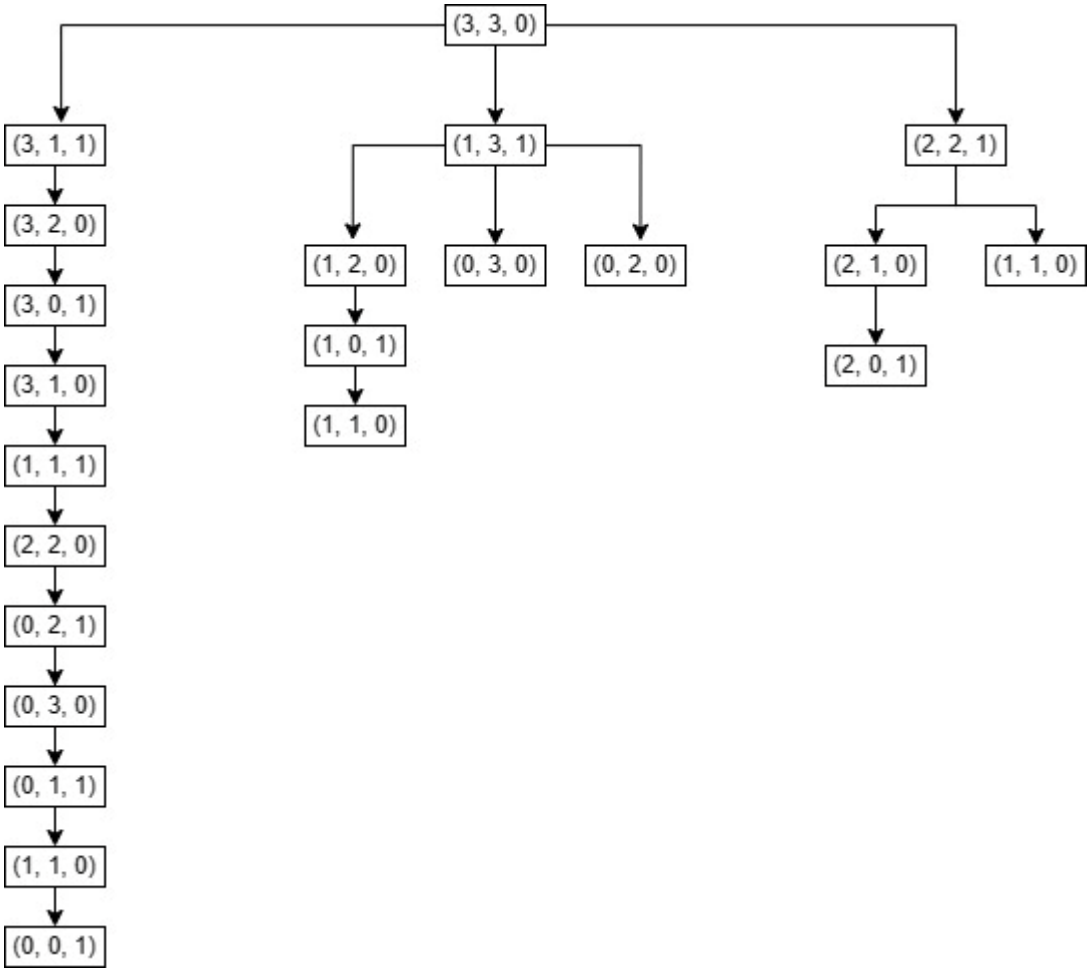
Breadth-First Search (BFS) plays a crucial role in solving the "Missionaries and Cannibals" puzzle by exploring the state space in a level-by-level manner. In this puzzle, the goal is to move three missionaries and three cannibals from one side of a river to the other using a boat, ensuring that at no point on either side there are more cannibals than missionaries, or the cannibals will eat the missionaries.

Breadth-First Search (BFS) plays a crucial role in solving the "Missionaries and Cannibals" puzzle by exploring the state space in a level-by-level manner. In this puzzle, the goal is to move three missionaries and three cannibals from one side of a river to the other using a boat, ensuring that at no point on either side there are more cannibals than missionaries, or the cannibals will eat the missionaries.

Student Work Area

Algorithm/Flowchart/Code/Sample Outputs

State Space Tree



Manual Solving

DELTA
Date / /
Pg No.

0 → Left 1 → Right

1. (3,3,0) → (3,1,1)

Moving 2 cannibals to the right

2. (3,1,1) → (3,2,0)

Moving 1 cannibal back to the left

3. (3,2,0) → (3,0,1)

Moving 2 cannibals to the right

4. (3,0,1) → (3,1,0)

Move 1 cannibals to the left

5. (3,1,0) → (1,1,1)

Move 2 missionaries to the right

6. (1,1,1) → (2,2,0)

Move 1 Missionary and 1 cannibal back to the left

7. $(2,2,0) \longrightarrow (0,2,1)$

Move 2 missionaries to the right.

8. $(0,2,1) \longrightarrow (0,3,0)$

Move 1 cannibal back to the left.

9. $(0,3,0) \longrightarrow (0,1,1)$

Move 2 cannibals to the right

10. $(0,1,1) \longrightarrow (1,1,0)$

Move 1 ~~cannibal~~ missionary back to left

11. $(1,1,0) \longrightarrow (0,0,1)$

Move 1 missionary and 1 cannibal to the right.

$(0,0,1)$ is the required state.

Code

Experiment 3

Problem Statement

Implement a basic state space search program in Python to solve the classic "Missionaries and Cannibals" puzzle. The goal is to implement a simple program that finds a sequence of valid moves to safely transport three missionaries and three cannibals across a river, following specific constraints.

Code

Imports Needed

```
[1] ✓ 0.0s
```

```python
from collections import deque
```

Python

Define the goal state

```
[2] ✓ 0.0s
```

```python
GOAL_STATE = (0, 0, 1)
```

Python

Check if a state is valid

```
[3] ✓ 0.0s
```

```python

def is_valid_state(state):
    (ml, cl, b) = state
    mr = 3 - ml
    cr = 3 - cl
    # Ensure no negative counts and no count greater than 3
    if not (0 <= ml <= 3 and 0 <= cl <= 3 and 0 <= mr <= 3 and 0 <= cr <= 3):
        return False
    # Cannibals shouldn't outnumber missionaries on either bank
    if (ml < cl and ml != 0) or (mr < cr and mr != 0):
        return False
    return True
```

Python

Generate possible next states

```python
def generate_next_states(state):
    (ml, cl, boat) = state
    possible_moves = [(2, 0), (1, 0), (1, 1), (0, 1), (0, 2)]

    next_states = []

    for move in possible_moves:
        if boat == 0:
            next_state = (ml - move[0], cl - move[1], 1)
        else:
            next_state = (ml + move[0], cl + move[1], 0)

        if is_valid_state(next_state):
            next_states.append(next_state)

    return next_states
```

[4] ✓ 0.0s Python

Breadth-First Search function

```python
def bfs(initial_state):
    explored = set()
    queue = deque([(initial_state, [])])

    while queue:
        current_state, path = queue.popleft()

        if current_state == GOAL_STATE:
            return path + [current_state]

        explored.add(current_state)

        for next_state in generate_next_states(current_state):
            if next_state not in explored:
                queue.append((next_state, path + [current_state]))

    return None

```

[5] ✓ 0.0s Python

Output:

Solve the puzzle

```python
# Solve the puzzle
initial_state = (3, 3, 0)
solution_path = bfs(initial_state)

if solution_path:
    print("Solution Path:")
    for state in solution_path:
        print(state)
else:
    print("No solution found.")
```

[6] ✓ 0.0s Python

```
Solution Path:
(3, 3, 0)
(2, 2, 1)
(3, 2, 0)
(3, 0, 1)
(3, 1, 0)
(1, 1, 1)
(2, 2, 0)
(0, 2, 1)
(0, 3, 0)
(0, 1, 1)
(1, 1, 0)
(0, 0, 1)
```