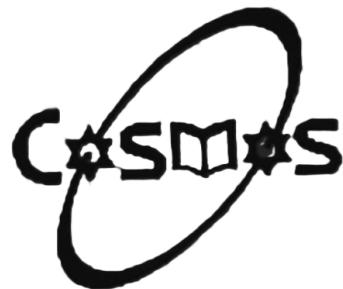


**Cosmos College of
Management & Technology**
(Affiliated to Pokhara University)

Sitapaila, Kathmandu



Artifical Intelligence (CMP 346) Lab Report

LAB REPORT NO: 05

LAB REPORT ON:
SEARCHING ALGORITHMS

SUBMITTED BY:-

Name: Kushal Prasad Joshi

Roll No: 230345

Faculty: Science and Technology

Group: B

SUBMITTED TO:-

Department: ICT

Lab Instructor: Mr. Ranjan Raj Aryal

Date of Experiment: 18th Janaury 2026

Date of Submission: 23rd Janaury 2026

TITLE

SEARCHING ALGORITHMS

OBJECTIVES

1. To implement breadth-first search algorithm using Python.
2. To implement depth-first search algorithm using Python.
3. To solve water jug problem using Python.

REQUIREMENTS

- Python
- VS Code

THEORY

BFS: Breadth-First Search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or an arbitrary node of a graph, sometimes referred to as a ‘search key’) and explores the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level.

DFS: Depth-First Search (DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

Water Jug Problem: The water jug problem is a classic problem in artificial intelligence and computer science. The problem involves two jugs with different capacities and the goal is to measure out a specific amount of water using these jugs. The problem can be solved using various search algorithms, such as BFS and DFS.

IMPLEMENTATION

```
1 from collections import deque
2
3 def bfs(graph, start):
4     visited = set()
5     queue = deque([start])
6
7     print("BFS Traversal Order:", end=' ')
8
9     while queue:
10         node = queue.popleft()
11         if node not in visited:
12             print(node, end=' ')
```

```

13     visited.add(node)
14     queue.extend(neighbor for neighbor in graph[node]
15                   if neighbor not in visited)
16
16 # Example usage:
17 if __name__ == "__main__":
18     graph = {
19         'A' : [ 'B', 'C' ],
20         'B' : [ 'D', 'E' ],
21         'C' : [ 'F' ],
22         'D' : [ ],
23         'E' : [ 'F' ],
24         'F' : [ ]
25     }
26
27     bfs(graph, 'A')

```

Listing 1: Breadth-First Search Implementation

```
1 BFS Traversal Order: A B C D E F
```

Listing 2: Output of BFS Implementation

```

1 def dfs(graph, start):
2     visited = set()
3     stack = [start]
4
5     print("DFS Traversal Order:", end=' ')
6
7     while stack:
8         node = stack.pop()
9         if node not in visited:
10            print(node, end=' ')
11            visited.add(node)
12            stack.extend(reversed([neighbor for neighbor in
13                           graph[node] if neighbor not in visited]))
14
14 # Example usage:
15 if __name__ == "__main__":
16     graph = {
17         'A' : [ 'B', 'C' ],
18         'B' : [ 'D', 'E' ],
19         'C' : [ 'F' ],
20         'D' : [ ],
21         'E' : [ 'F' ],
22         'F' : [ ]
23     }
24
25     dfs(graph, 'A')

```

Listing 3: Depth-First Search Implementation

```
1 DFS Traversal Order: A B D E F C
```

Listing 4: Output of DFS Implementation

```
1 ##### WATER JUG PROBLEM
  #####
2 # Problem Statement:
3   # Two jugs with capacities A and B
4   # Unlimited water supply
5   # Measure exactly T liters
6   # Allowed operations:
7     # Fill a jug
8     # Empty a jug
9     # Pour water from one jug to another
10
11 from collections import deque
12
13 def water_jug_problem(capacity_a, capacity_b, target):
14   # Set to keep track of already visited states
15   # Each state is represented as (water_in_jug_A,
16     water_in_jug_B)
17   visited = set()
18
19   # Queue for BFS
20   # Each element contains: (current_state_A,
21     current_state_B, path_taken)
22   queue = deque()
23
24   # Initial state: both jugs are empty
25   queue.append((0, 0, []))
26
27   # Continue BFS until queue is empty
28   while queue:
29     # Remove the front state from the queue
30     a, b, path = queue.popleft()
31
32     # If the target amount is found in either jug
33     if a == target or b == target:
34       # Add final state to path and return solution
35       path.append((a, b))
36       return path
37
38     # Skip state if already visited
39     if (a, b) in visited:
          continue
```

```

40     # Mark current state as visited
41     visited.add((a, b))
42
43     # Add current state to the path
44     path = path + [(a, b)]
45
46     # Generate all possible next states using allowed
        operations
47     next_states = [
48
49         # Fill Jug A completely
50         (capacity_a, b),
51
52         # Fill Jug B completely
53         (a, capacity_b),
54
55         # Empty Jug A
56         (0, b),
57
58         # Empty Jug B
59         (a, 0),
60
61         # Pour water from Jug A to Jug B
62         # Amount poured is the minimum of:
63         #   - water available in A
64         #   - remaining capacity in B
65         (a - min(a, capacity_b - b),
66          b + min(a, capacity_b - b)),
67
68         # Pour water from Jug B to Jug A
69         (a + min(b, capacity_a - a),
70          b - min(b, capacity_a - a))
71     ]
72
73     # Add all unvisited next states to the BFS queue
74     for state in next_states:
75         if state not in visited:
76             queue.append((state[0], state[1], path))
77
78     # If BFS completes without finding target
79     return None
80
81 # Example usage:
82 if __name__ == "__main__":
83     solution = water_jug_problem(
84         capacity_a = int(input("Enter capacity of Jug A: ")),
85         capacity_b = int(input("Enter capacity of Jug B: ")),
86         target = int(input("Enter target amount of water: ")))

```

```

87     )
88
89     print("Steps to reach target:", end=' ')
90     if solution:
91         for step in solution:
92             print(step, end=' ')
93     else:
94         print("No solution exists.")

```

Listing 5: Water Jug Problem Implementation

```

1 Enter capacity of Jug A: 4
2 Enter capacity of Jug B: 3
3 Enter target amount of water: 2
4 Steps to reach target: (0, 0) (0, 3) (3, 0) (3, 3) (4, 2)

```

Listing 6: Output of Water Jug Problem Implementation

RESULTS AND CONCLUSION

In this lab, we successfully implemented the Breadth-First Search (BFS) and Depth-First Search (DFS) algorithms in Python. We also solved the Water Jug Problem using the BFS approach. The implementations were tested with example graphs and jug capacities, yielding correct traversal orders and solution steps.