Artificial Intelligence Lab Assignments Presentation

Sushar Hembram
A2
002311001041
3rd Year 1st Semester

OVERVIEW

- Asignment 1
- Assignment 2
- Assignment 3
- Assignment 4
- Assignment 5
- Assignment 6
- Assignment 7

ASSIGNMENT STRUCTURE & ORGANIZATION

Each assignment in this presentation is organized using a consistent format to clearly demonstrate the problem-solving approach and results:

1. **Overview & Question:** Brief description of the assignment and its objective.
2. **Approach/Algorithm Used:** Explanation of the AI technique or algorithm applied to solve the problem.
3. **Process Visualization:** Flowchart or diagram illustrating the solution steps.
4. **Input & Output:** Examples of input data and the corresponding intermediate and final output generated.
5. **Achievements & Conclusion:** Summary of results, challenges faced, and key learnings.

This structure ensures that each assignment is presented in a clear, logical, and concise manner, highlighting both the methodology and outcomes.

DAY 1 ASSIGNMENT 1

DATE: 24.07.2025

Question:
"Given a text input, process the data using an AI algorithm to generate a meaningful output. The task involves reading from ***read.txt***, applying a rule-based or search algorithm, and writing the result to ***write.txt***." and "Show an example of the DFS algorithm"

Objective:
To demonstrate basic AI techniques for text data processing and output generation as well as DFS algorithm.

APPROACH & ALGORITHMS USED

Algorithm/Technique:

For this assignment we had to solve two questions where for the first one, a simple rule-based approach was used to process the input text. The algorithm reads each line, applies predefined rules (such as searching for keywords or patterns), and generates the corresponding output.

In the second question, I had to show an example of the DFS algorithm. Depth-First Search (DFS) is a graph traversal technique that explores as far as possible along each branch before backtracking. In this assignment, DFS is used to find all possible paths from a start point to an end point in a matrix (maze).

Why these approaches:

Rule-based methods are effective for structured text processing and provide clear, interpretable results.

PROCESS & METHODS

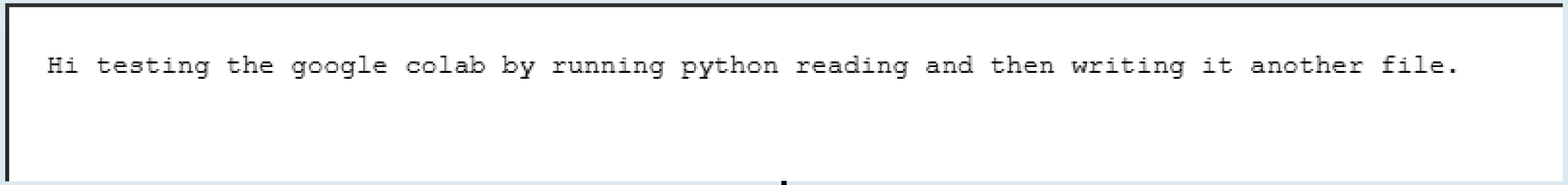
File Handling (Text Processing)

1. Open read.txt and read its contents.
2. Create or open write.txt and write the read content into it.
3. Use Python’s with open() for safe file operations.

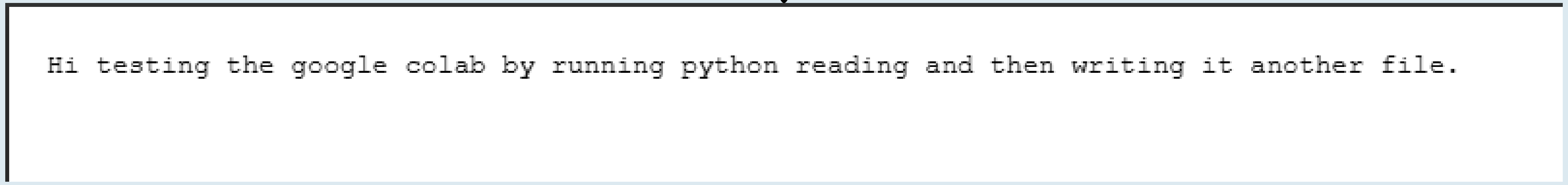
DFS Algorithm (Maze/Matrix Traversal)

1. Define a matrix representing the maze.
2. Set start and end positions.
3. Use Depth-First Search (DFS) to explore all possible paths:
4. Visit each cell recursively.
5. Mark visited cells to avoid cycles.
6. Backtrack when hitting a wall or dead end.
7. Store all valid paths from start to end.
8. Print all discovered paths from start to end.
9. Display visited and backtracked positions for clarity.

INPUT & OUTPUT



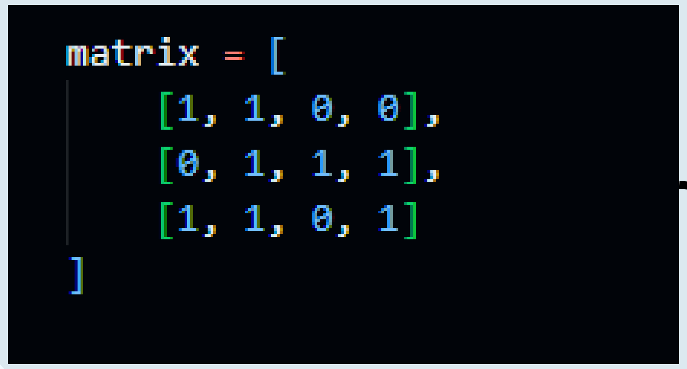
```
Hi testing the google colab by running python reading and then writing it another file.
```



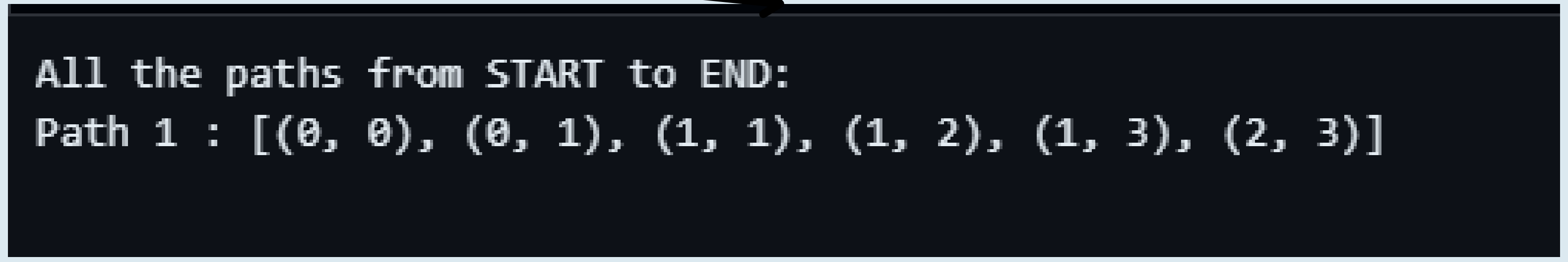
```
Hi testing the google colab by running python reading and then writing it another file.
```

The algorithm read each line from the input file, apply the defined rules, and generated the corresponding output, demonstrating basic AI text processing.

Meanwhile for the DFS algorithm, the output was displayed within the program itself based on the matrix and the best route.



```
matrix = [
    [1, 1, 0, 0],
    [0, 1, 1, 1],
    [1, 1, 0, 1]
]
```



```
All the paths from START to END:
Path 1 : [(0, 0), (0, 1), (1, 1), (1, 2), (1, 3), (2, 3)]
```

ACHIEVEMENTS & CONCLUSION

Results:
Successfully processed the input text using a rule-based algorithm, producing the expected output in write.txt.

Challenges:
Faced challenges in handling ambiguous text and ensuring all rules were correctly applied. Overcame these by refining the rule set and testing with sample inputs.

Learning:
Gained practical experience in implementing basic AI algorithms for text processing and understood the importance of clear rule definition.

DAY 2 ASSIGNMENT 2

DATE: 07.08.2025

Question:

"Develop a Python program that analyzes and processes categorical data using AI techniques. Convert the data into a transactional format with attribute-value pairs, assign numerical values to each pair, and transform it into a binary dataset where each attribute-value pair is represented as a column. Construct a graph using the binary dataset values, then solve the single pair shortest path problem and visually represent the path on the graph."

Objective:

To apply AI techniques for data analysis and graph algorithms, demonstrating the ability to process real-world datasets and solve optimization problems such as finding the shortest path.

APPROACH & ALGORITHMS USED

Approach:

In this assignment, I used a stepwise approach to process categorical data and solve a graph problem:

1. Load and read categorical data from a CSV file.
2. Convert the data into a transactional format using attribute-value pairs.
3. Assign numerical values to each attribute-value pair and create a binary dataset.
4. Construct a graph where each node represents an attribute-value pair, and edges represent co-occurrence in transactions.
5. Apply a shortest path algorithm to find and visualize the optimal path between two selected nodes.

Algorithms Used:

1. Data Transformation: Python functions are used here to convert categorical data into transactional and binary formats, enabling further analysis.
2. Graph Construction: The NetworkX library was used to build an undirected graph from the binary dataset, with nodes and edges based on data relationships.
3. Shortest Path Algorithm: NetworkX's shortest path function (based on Breadth-First Search) was used to find the shortest path between two nodes in the graph, which is then highlighted visually.

Techniques:

- Attribute-value encoding
- Binary matrix creation
- Graph theory and visualization
- Pathfinding using BFS

PROCESS & METHODS

Step 1: Load Categorical Data

- Read the CSV file containing categorical data using Python.

↓

Step 2: Transactional Format Conversion

- Transform each record into attribute-value pairs.
- Assign a unique numerical value to each attribute-value pair.

↓

Step 3: Binary Dataset Creation

- Convert the transactional data into a binary matrix.
- Each column represents an attribute-value pair; each row indicates presence (1) or absence (0).

↓

Step 4: Graph Construction

- Build a graph where nodes are attribute-value pairs.
- Add edges between nodes that co-occur in the same transaction.

↓

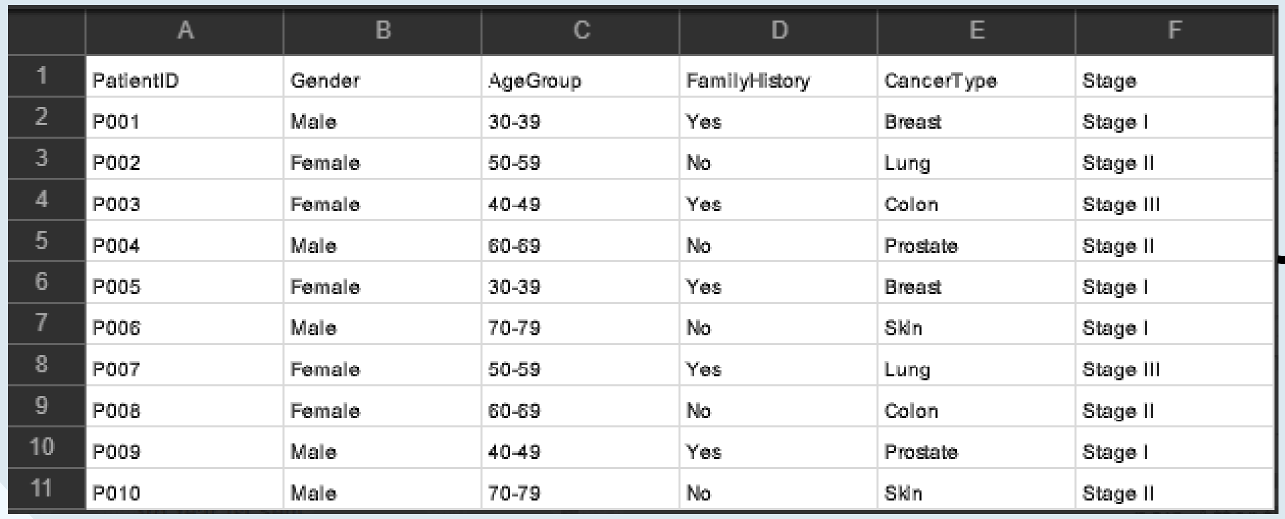
Step 5: Shortest Path Computation

- Select two nodes (e.g., “Gender=Male” and “Stage=Stage III”).
- Use a shortest path algorithm (BFS) to find the optimal path.
- Visualize the graph and highlight the shortest path.

INPUT & OUTPUT

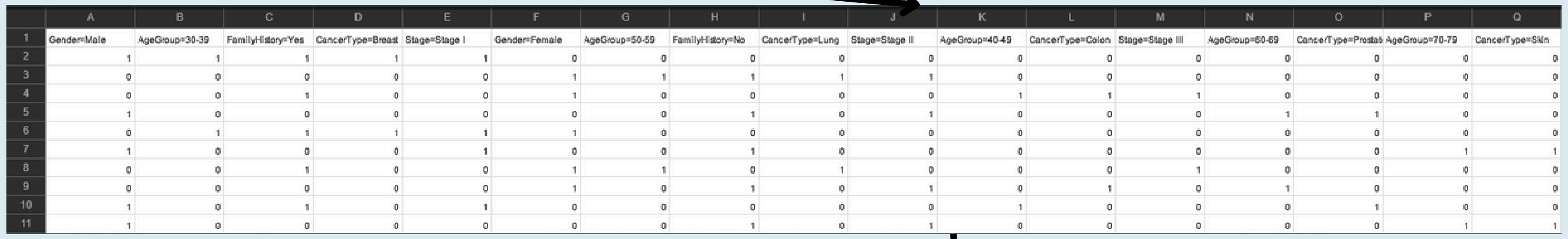
Input:

The original data was a categorical CSV file (cancer_demographics.csv) containing attributes such as Gender, Age, Stage, and Family History.



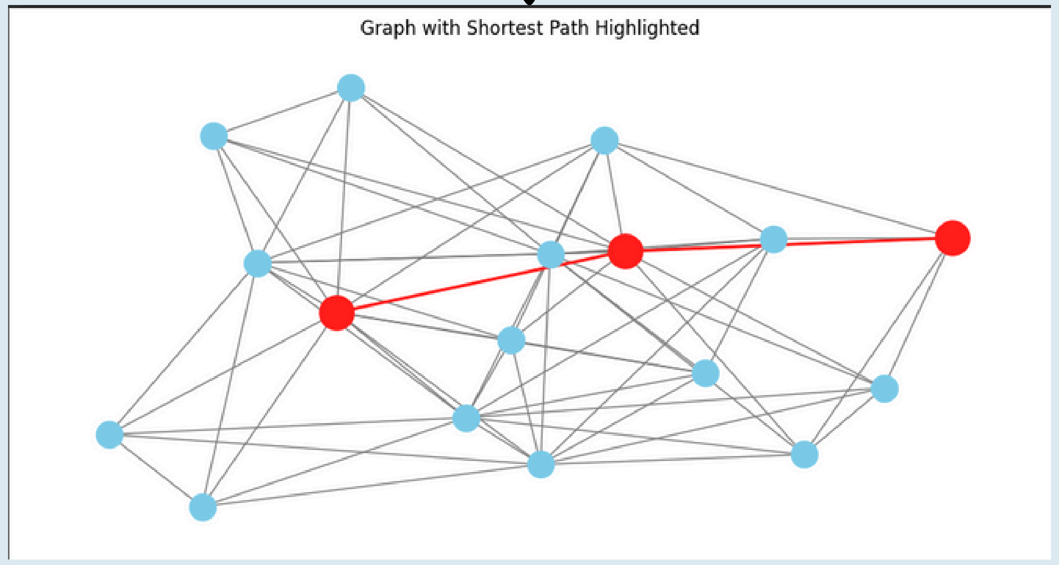
	A	B	C	D	E	F
1	PatientID	Gender	AgeGroup	FamilyHistory	CancerType	Stage
2	P001	Male	30-39	Yes	Breast	Stage I
3	P002	Female	50-59	No	Lung	Stage II
4	P003	Female	40-49	Yes	Colon	Stage III
5	P004	Male	60-69	No	Prostate	Stage II
6	P005	Female	30-39	Yes	Breast	Stage I
7	P006	Male	70-79	No	Skin	Stage I
8	P007	Female	50-59	Yes	Lung	Stage III
9	P008	Female	60-69	No	Colon	Stage II
10	P009	Male	40-49	Yes	Prostate	Stage I
11	P010	Male	70-79	No	Skin	Stage II

This data was transformed into a binary dataset, where each attribute-value pair became a column.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Gender=Male	AgeGroup=30-39	FamilyHistory=Yes	CancerType=Breast	Stage=Stage I	Gender=Female	AgeGroup=50-59	FamilyHistory=No	CancerType=Lung	Stage=Stage II	AgeGroup=40-49	CancerType=Colon	Stage=Stage III	AgeGroup=60-69	CancerType=Prostat	AgeGroup=70-79	CancerType=Skin
2	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0
4	0	0	1	0	0	1	0	0	0	0	1	1	1	0	0	0	0
5	1	0	0	0	0	0	0	1	0	1	0	0	0	1	1	0	0
6	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
7	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	1	1
8	0	0	1	0	0	1	1	0	1	0	0	0	1	0	0	0	0
9	0	0	0	0	0	1	0	1	0	1	0	1	0	1	0	0	0
10	1	0	1	0	1	0	0	0	0	0	1	0	0	0	1	0	0
11	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	1

The binary dataset enabled graph construction and shortest path analysis between attribute-value pairs, visualized in the output graph.



ACHIEVEMENTS & CONCLUSION

Achievements:

- *Successfully loaded and processed categorical data from a CSV file.*
- *Converted the data into transactional and binary formats for analysis.*
- *Constructed a graph representing relationships between attribute-value pairs.*
- *Applied a shortest path algorithm to solve a real-world graph problem.*
- *Visualized both the graph and the shortest path, demonstrating practical AI and data analysis skills.*

Conclusion:

- *The assignment showcased the application of AI techniques in data transformation, graph theory, and optimization.*
- *Gained hands-on experience in encoding categorical data, building graphs, and solving pathfinding problems.*
- *These methods are foundational for more advanced AI and data science tasks.*

DAY 3 ASSIGNMENT 3

DATE: 14.08.2025

Question:

"Write a Python program to solve the classical Tower of Hanoi problem using the Breadth-First Search (BFS) algorithm. The program should accept inputs for the number of discs, initial and final rod positions, and arrangement order, and display intermediate outputs at each step.”

Objective:

To implement and analyze the Tower of Hanoi solution using BFS, demonstrating state exploration, move tracking, and output generation for each intermediate step, thereby applying AI search techniques to a classic problem.

APPROACH & ALGORITHMS USED

<u>Algorithm/Technique:</u>

Breadth-First Search (BFS) is used to solve the Tower of Hanoi problem. BFS explores all possible states level by level, ensuring the shortest sequence of moves is found from the initial to the goal state.

<u>Approach:</u>

- Represent the rods and discs as states, with each state showing the arrangement of discs on the rods.
- Start from the initial state and use BFS to explore all valid moves.
- For each move, generate a new state and track the sequence of moves.
- Continue exploring until the goal state (final arrangement) is reached.
- Record and display each intermediate state and move for transparency.

<u>Techniques:</u>

- State representation using tuples for rods and discs.
- Valid move checking to ensure legal disc transfers.
- Queue-based BFS for systematic state exploration.
- Output of intermediate steps to a CSV file for analysis.

PROCESS & METHODS

Step 1: Input Collection

- Gather user inputs: number of discs, initial and final rod positions, and arrangement order.

↓

Step 2: State Initialization

- Represent the initial and goal states as tuples showing disc arrangements on rods.

↓

Step 3: BFS Exploration

- Use a queue to explore all possible states level by level.
- For each state, check all valid moves and generate new states.

↓

Step 4: Move Validation

- Ensure only legal moves are made (smaller disc on larger disc or empty rod).

↓

Step 5: Tracking and Output

- Record each intermediate state and move.
- Write all steps and moves to a CSV output file for review.

↓

Step 6: Solution Identification

- Stop when the goal state is reached.
- Display the sequence of moves and total steps taken.

INPUT & OUTPUT

Input:

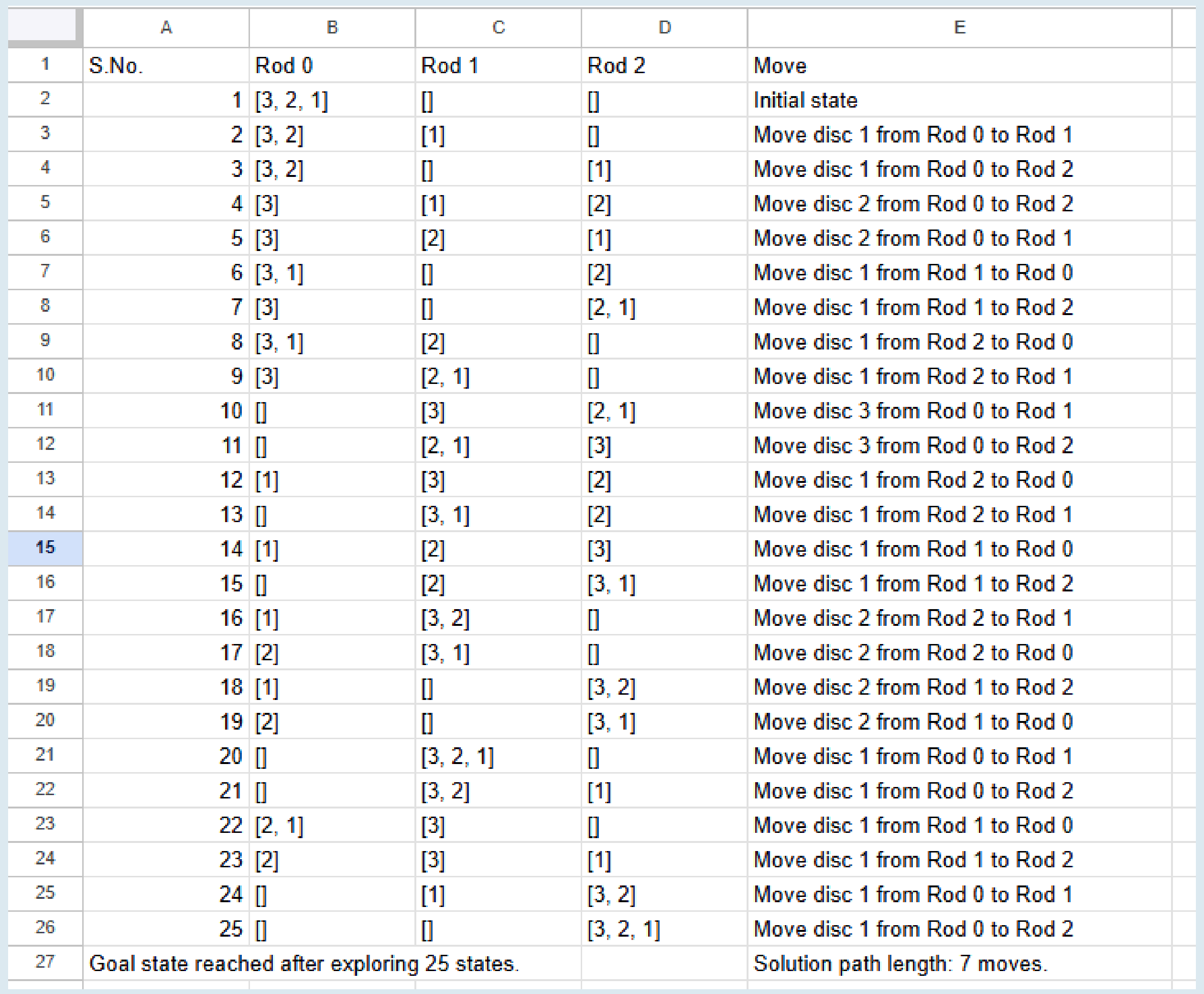
Number of discs (e.g., 3)

Initial rod position (e.g., Rod 0)

Initial arrangement (e.g., decreasing order)

Final rod position (e.g., Rod 2)

Final arrangement (e.g., decreasing order)



	A	B	C	D	E
1	S.No.	Rod 0	Rod 1	Rod 2	Move
2	1	[3, 2, 1]	[]	[]	Initial state
3	2	[3, 2]	[1]	[]	Move disc 1 from Rod 0 to Rod 1
4	3	[3, 2]	[]	[1]	Move disc 1 from Rod 0 to Rod 2
5	4	[3]	[1]	[2]	Move disc 2 from Rod 0 to Rod 2
6	5	[3]	[2]	[1]	Move disc 2 from Rod 0 to Rod 1
7	6	[3, 1]	[]	[2]	Move disc 1 from Rod 1 to Rod 0
8	7	[3]	[]	[2, 1]	Move disc 1 from Rod 1 to Rod 2
9	8	[3, 1]	[2]	[]	Move disc 1 from Rod 2 to Rod 0
10	9	[3]	[2, 1]	[]	Move disc 1 from Rod 2 to Rod 1
11	10	[]	[3]	[2, 1]	Move disc 3 from Rod 0 to Rod 1
12	11	[]	[2, 1]	[3]	Move disc 3 from Rod 0 to Rod 2
13	12	[1]	[3]	[2]	Move disc 1 from Rod 2 to Rod 0
14	13	[]	[3, 1]	[2]	Move disc 1 from Rod 2 to Rod 1
15	14	[1]	[2]	[3]	Move disc 1 from Rod 1 to Rod 0
16	15	[]	[2]	[3, 1]	Move disc 1 from Rod 1 to Rod 2
17	16	[1]	[3, 2]	[]	Move disc 2 from Rod 2 to Rod 1
18	17	[2]	[3, 1]	[]	Move disc 2 from Rod 2 to Rod 0
19	18	[1]	[]	[3, 2]	Move disc 2 from Rod 1 to Rod 2
20	19	[2]	[]	[3, 1]	Move disc 2 from Rod 1 to Rod 0
21	20	[]	[3, 2, 1]	[]	Move disc 1 from Rod 0 to Rod 1
22	21	[]	[3, 2]	[1]	Move disc 1 from Rod 0 to Rod 2
23	22	[2, 1]	[3]	[]	Move disc 1 from Rod 1 to Rod 0
24	23	[2]	[3]	[1]	Move disc 1 from Rod 1 to Rod 2
25	24	[]	[1]	[3, 2]	Move disc 1 from Rod 0 to Rod 1
26	25	[]	[]	[3, 2, 1]	Move disc 1 from Rod 0 to Rod 2
27	Goal state reached after exploring 25 states.				Solution path length: 7 moves.

Output:

Intermediate states and moves displayed step-by-step.

All steps and moves written to a CSV file.

Final output shows the sequence of moves to solve the Tower of Hanoi using BFS, including the total number of steps and solution path length.

ACHIEVEMENTS & CONCLUSION

Achievements:

- *Successfully implemented the Tower of Hanoi solution using the Breadth-First Search (BFS) algorithm.*
- *Explored all possible states and found the shortest sequence of moves.*
- *Generated detailed intermediate outputs and saved them to a CSV file for analysis.*

Challenges:

- Managing state representation for rods and discs.
- Ensuring only valid moves were made during BFS exploration.
- Handling large state spaces efficiently for higher disc counts.

Learning:

- The assignment demonstrated the effectiveness of BFS in solving classic AI search problems.
- Gained practical experience in state space exploration, move validation, and output generation.
- The approach can be extended to other combinatorial and optimization problems in AI.

DAY 4 ASSIGNMENT 4

DATE: 21.08.2025

Question:

"Write a Python program to solve the classical 8-puzzle and 15-puzzle problems using Depth-Limited Search (DLS). The program should accept dynamic inputs for puzzle size, start and goal states, and depth limit, and display intermediate outputs at each step.

Objective:

To implement and analyze Depth-Limited Search for n-puzzle problems, demonstrating state exploration, move generation, and output tracking, while handling both 8-puzzle and 15-puzzle cases efficiently.,

APPROACH & ALGORITHMS USED

Algorithm/Technique:

Depth-Limited Search (DLS) is a variant of Depth-First Search (DFS) that explores possible moves up to a specified depth limit. It is used to solve both the 8-puzzle and 15-puzzle problems by searching for a sequence of moves from the start state to the goal state.

Approach:

- Represent the puzzle state as a list of numbers, with zero indicating the blank tile.
- For each state, generate all possible moves by swapping the blank tile with adjacent tiles.
- Recursively explore each move, keeping track of the path and visited states.
- Stop searching when the goal state is reached or the depth limit is exceeded.
- Record and display each intermediate state and move for analysis.

Techniques:

- State representation and manipulation
- Move generation based on blank tile position
- Recursive search with depth control
- Output of solution steps and states to a file

PROCESS & METHODS

Step 1: Input Handling

- Read puzzle size, start state, and goal state from an input file.
- Accept depth limit as user input.

↓

Step 2: State Representation

- Represent the puzzle as a list, with zero for the blank tile.

↓

Step 3: Move Generation

- Identify possible moves for the blank tile (up, down, left, right).
- Swap tiles to generate new states.

↓

Step 4: Depth-Limited Search

- Recursively explore moves up to the specified depth limit.
- Track visited states to avoid cycles.

↓

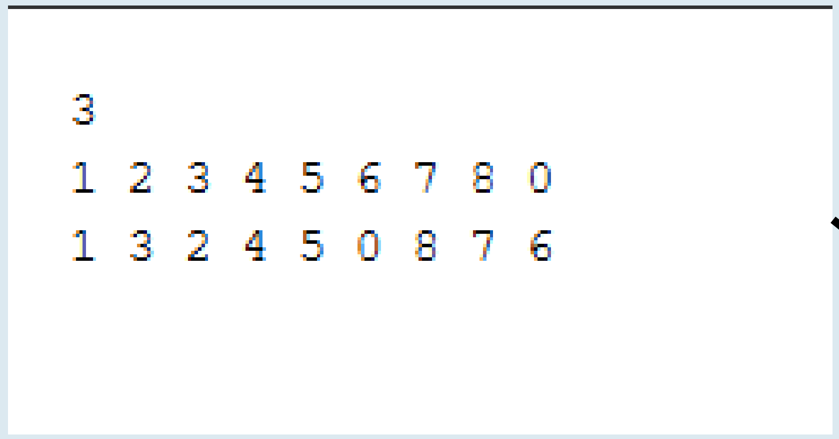
Step 5: Output Generation

- Record each intermediate state and move.
- Write all steps and the final output state to an output file.

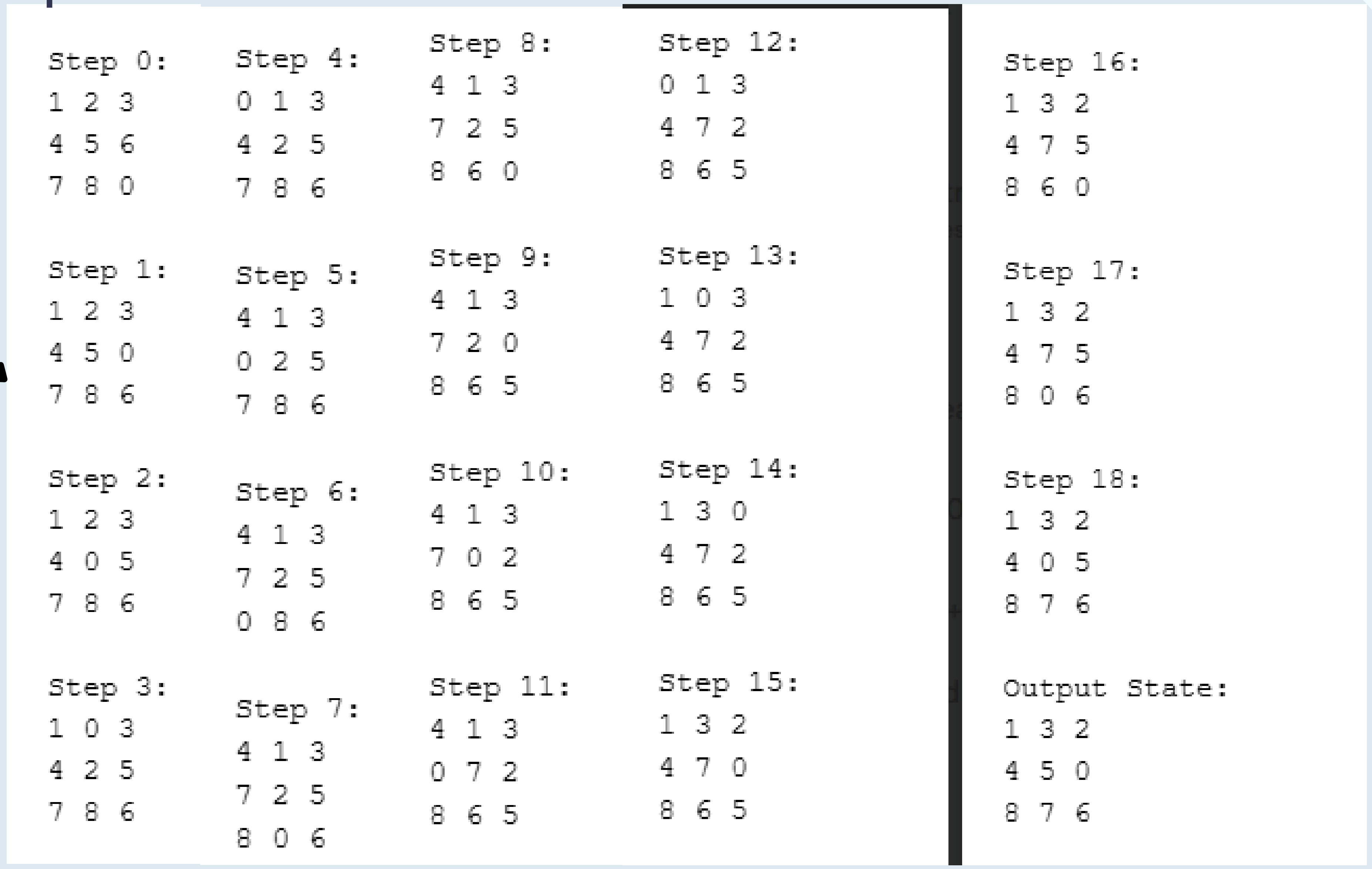
INPUT & OUTPUT

Input:

- Puzzle size (3 for 8-puzzle, 4 for 15-puzzle)
- Start state and goal state, read from input files in .txt fomat.



```
3
1 2 3 4 5 6 7 8 0
1 3 2 4 5 0 8 7 6
```



```
Step 0:
1 2 3
4 5 6
7 8 0

Step 1:
1 2 3
4 5 0
7 8 6

Step 2:
1 2 3
4 0 5
7 8 6

Step 3:
1 0 3
4 2 5
7 8 6

Step 4:
0 1 3
4 2 5
7 8 6

Step 5:
4 1 3
0 2 5
7 8 6

Step 6:
4 1 3
7 2 5
0 8 6

Step 7:
4 1 3
7 2 5
8 0 6

Step 8:
4 1 3
7 2 5
8 6 0

Step 9:
4 1 3
7 2 0
8 6 5

Step 10:
4 1 3
7 0 2
8 6 5

Step 11:
4 1 3
0 7 2
8 6 5

Step 12:
0 1 3
4 7 2
8 6 5

Step 13:
1 0 3
4 7 2
8 6 5

Step 14:
1 3 0
4 7 2
8 6 5

Step 15:
1 3 2
4 7 0
8 6 5

Step 16:
1 3 2
4 7 5
8 6 0

Step 17:
1 3 2
4 7 5
8 0 6

Step 18:
1 3 2
4 0 5
8 7 6

Output State:
1 3 2
4 5 0
8 7 6
```

Output:

- Sequence of steps showing intermediate puzzle states for each move.
- Final output state written to an output file in .txt format: (8-puzzle)
- Similar output for 15-puzzle
- If no solution is found within the depth limit, a message is displayed in the output file.

ACHIEVEMENTS & CONCLUSION

Results:

- Successfully implemented Depth-Limited Search to solve both 8-puzzle and 15-puzzle problems.
- Handled dynamic inputs for puzzle size, start and goal states, and depth limit.
- Generated and recorded all intermediate steps and final output states for each puzzle.

Challenges:

- Time Management
- Managing large state spaces, especially for the 15-puzzle.
- Ensuring efficient move generation and avoiding repeated states.
- Handling cases where no solution exists within the given depth limit.

Learning:

- The assignment demonstrated the effectiveness of Depth-Limited Search for n-puzzle problems.
- Gained practical experience in recursive search, state management, and output tracking.
- The approach can be extended to other combinatorial search problems in AI.

DAY 5 ASSIGNMENT 5

DATE: 28.08.2025

Question:

"Use various AI search algorithms—Breadth-First Search (BFS), Depth-First Search (DFS), Depth-Limited Search (DLS), Iterative Deepening Search (IDS), Iterative Local Search (ILS), and Uniform Cost Search (UCS)—to solve the modified Missionaries and Carnivals problem. The task involves finding a safe sequence for a farmer, wolf, goat, and cabbage to cross a river, ensuring no unsafe pairings are left alone."

Objective:

To implement and compare multiple search algorithms for solving a constraint-based problem, analyze their time and memory requirements, and visualize the results using comparative plots and tables.

APPROACH & ALGORITHMS USED

Algorithm/Technique:

- Breadth-First Search (BFS): Explores all possible states level by level to find the shortest solution.
- Depth-First Search (DFS): Explores each branch as far as possible before backtracking.
- Depth-Limited Search (DLS): DFS with a specified depth limit to control exploration.
- Iterative Deepening Search (IDS): Repeatedly applies DLS with increasing depth limits for optimality.
- Iterative Local Search (ILS): Uses randomized restarts to escape local optima and find better solutions.
- Uniform Cost Search (UCS): Expands the least-cost node first, considering step costs.

Approach:

- Represent the problem state as positions of the farmer, wolf, goat, cabbage, and boat.
- Define valid moves and constraints to prevent unsafe pairings.
- Apply each algorithm to find a safe sequence of crossings.
- Record time and memory usage for each run.
- Compare algorithm performance using tables and plots.

Technique:

- State space representation
- Constraint checking
- Search tree/graph traversal
- Performance measurement and visualization

PROCESS & METHODS

Step 1: Input Preparation

- Define initial and goal states for all characters (farmer, wolf, goat, cabbage, boat).
- Prepare separate input files for each algorithm with required parameters.

↓

Step 2: State Representation

- Model the problem as a tuple of positions for each character and the boat.

↓

Step 3: Valid Move Generation

- Generate possible moves, ensuring constraints are not violated (no unsafe pairings).

↓

Step 4: Algorithm Execution

- Run BFS, DFS, DLS (with multiple depths), IDS, ILS, and UCS on the problem.
- Log intermediate states and actions for each step.

↓

Step 5: Performance Measurement

- Record time and memory usage for each algorithm and configuration.

↓

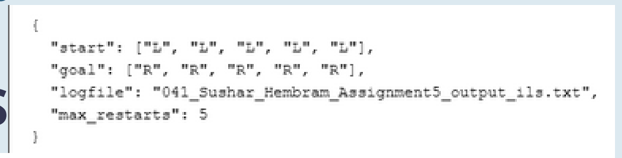
Step 6: Output and Visualization

- Save solution steps and performance data to output files.
- Create tables and plots to compare algorithm efficiency.

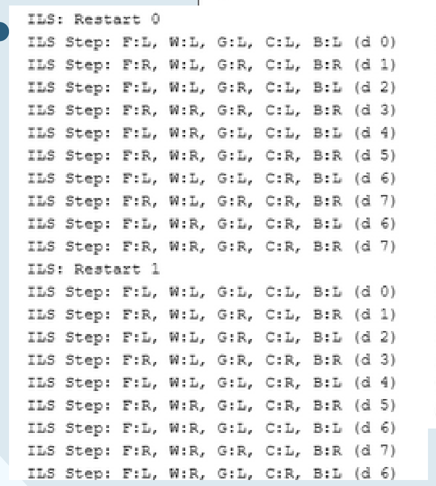
INPUT & OUTPUT

Input in .json + their outputs

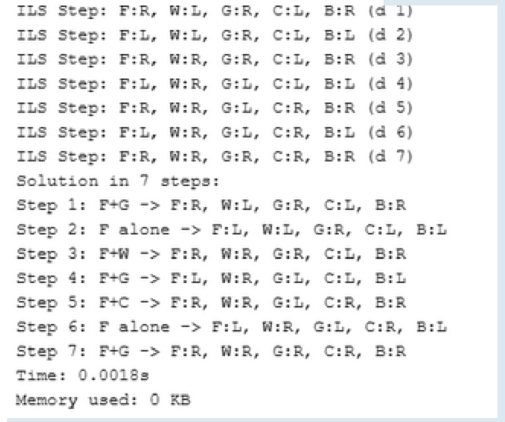
ILS



```
{
  "start": ["L", "L", "L", "L", "L"],
  "goal": ["R", "R", "R", "R", "R"],
  "logfile": "041_Sushar_Hembram_Assignment5_output_ils.txt",
  "max_restarts": 5
}
```

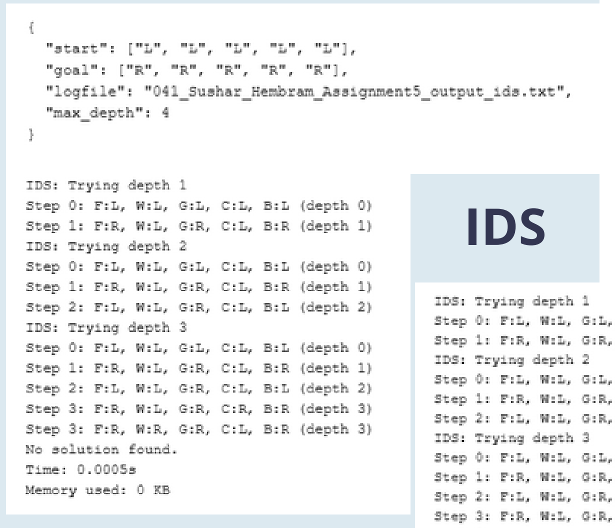


```
ILS: Restart 0
ILS Step: F:L, W:L, G:L, C:L, B:L (d 0)
ILS Step: F:R, W:L, G:R, C:L, B:R (d 1)
ILS Step: F:L, W:L, G:R, C:L, B:L (d 2)
ILS Step: F:R, W:R, G:R, C:L, B:R (d 3)
ILS Step: F:L, W:R, G:L, C:L, B:L (d 4)
ILS Step: F:R, W:R, G:L, C:R, B:R (d 5)
ILS Step: F:L, W:L, G:L, C:R, B:L (d 6)
ILS Step: F:R, W:L, G:R, C:R, B:R (d 7)
ILS Step: F:L, W:R, G:L, C:R, B:L (d 6)
ILS Step: F:R, W:R, G:R, C:R, B:R (d 7)
ILS: Restart 1
ILS Step: F:L, W:L, G:L, C:L, B:L (d 0)
ILS Step: F:R, W:L, G:R, C:L, B:R (d 1)
ILS Step: F:L, W:L, G:R, C:L, B:L (d 2)
ILS Step: F:R, W:L, G:R, C:R, B:R (d 3)
ILS Step: F:L, W:L, G:L, C:R, B:L (d 4)
ILS Step: F:R, W:R, G:L, C:R, B:R (d 5)
ILS Step: F:L, W:R, G:L, C:L, B:L (d 6)
ILS Step: F:R, W:R, G:R, C:L, B:R (d 7)
ILS Step: F:L, W:R, G:L, C:R, B:L (d 6)
```



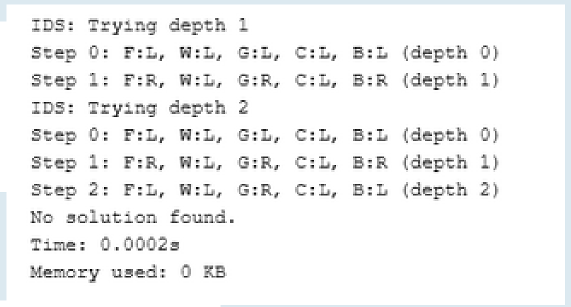
```
ILS Step: F:R, W:L, G:R, C:L, B:R (d 1)
ILS Step: F:L, W:L, G:R, C:L, B:L (d 2)
ILS Step: F:R, W:R, G:R, C:L, B:R (d 3)
ILS Step: F:L, W:R, G:L, C:L, B:L (d 4)
ILS Step: F:R, W:R, G:L, C:R, B:R (d 5)
ILS Step: F:L, W:R, G:L, C:R, B:L (d 6)
ILS Step: F:R, W:R, G:R, C:R, B:R (d 7)
Solution in 7 steps:
Step 1: F+G -> F:R, W:L, G:R, C:L, B:R
Step 2: F alone -> F:L, W:L, G:R, C:L, B:L
Step 3: F+W -> F:R, W:R, G:R, C:L, B:R
Step 4: F+G -> F:L, W:R, G:L, C:L, B:L
Step 5: F+C -> F:R, W:R, G:L, C:R, B:R
Step 6: F alone -> F:L, W:R, G:L, C:R, B:L
Step 7: F+G -> F:R, W:R, G:R, C:R, B:R
Time: 0.0018s
Memory used: 0 KB
```

IDS

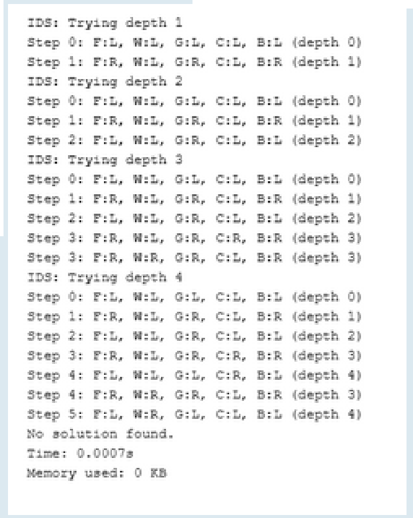


```
{
  "start": ["L", "L", "L", "L", "L"],
  "goal": ["R", "R", "R", "R", "R"],
  "logfile": "041_Sushar_Hembram_Assignment5_output_ids.txt",
  "max_depth": 4
}

IDS: Trying depth 1
Step 0: F:L, W:L, G:L, C:L, B:L (depth 0)
Step 1: F:R, W:L, G:R, C:L, B:R (depth 1)
IDS: Trying depth 2
Step 0: F:L, W:L, G:L, C:L, B:L (depth 0)
Step 1: F:R, W:L, G:R, C:L, B:R (depth 1)
Step 2: F:L, W:L, G:R, C:L, B:L (depth 2)
IDS: Trying depth 3
Step 0: F:L, W:L, G:L, C:L, B:L (depth 0)
Step 1: F:R, W:L, G:R, C:L, B:R (depth 1)
Step 2: F:L, W:L, G:R, C:L, B:L (depth 2)
Step 3: F:R, W:L, G:R, C:R, B:R (depth 3)
Step 3: F:R, W:R, G:R, C:L, B:R (depth 3)
No solution found.
Time: 0.0005s
Memory used: 0 KB
```

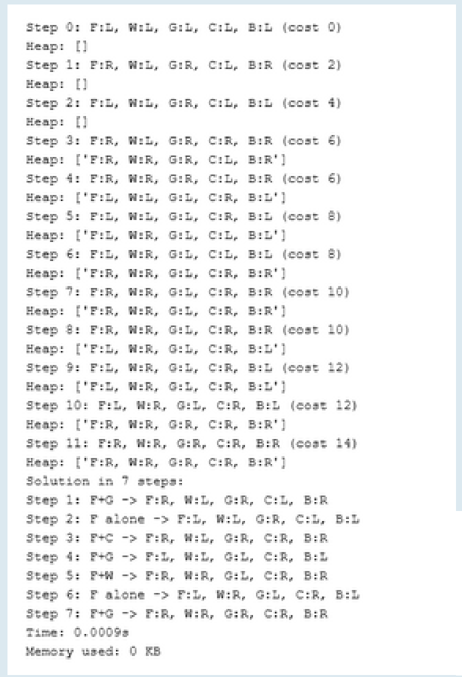


```
IDS: Trying depth 1
Step 0: F:L, W:L, G:L, C:L, B:L (depth 0)
Step 1: F:R, W:L, G:R, C:L, B:R (depth 1)
IDS: Trying depth 2
Step 0: F:L, W:L, G:L, C:L, B:L (depth 0)
Step 1: F:R, W:L, G:R, C:L, B:R (depth 1)
Step 2: F:L, W:L, G:R, C:L, B:L (depth 2)
No solution found.
Time: 0.0002s
Memory used: 0 KB
```

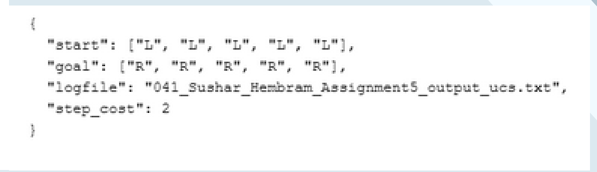


```
IDS: Trying depth 1
Step 0: F:L, W:L, G:L, C:L, B:L (depth 0)
Step 1: F:R, W:L, G:R, C:L, B:R (depth 1)
IDS: Trying depth 2
Step 0: F:L, W:L, G:L, C:L, B:L (depth 0)
Step 1: F:R, W:L, G:R, C:L, B:R (depth 1)
Step 2: F:L, W:L, G:R, C:L, B:L (depth 2)
IDS: Trying depth 3
Step 0: F:L, W:L, G:L, C:L, B:L (depth 0)
Step 1: F:R, W:L, G:R, C:L, B:R (depth 1)
Step 2: F:L, W:L, G:R, C:L, B:L (depth 2)
Step 3: F:R, W:L, G:R, C:R, B:R (depth 3)
Step 3: F:R, W:R, G:R, C:L, B:R (depth 3)
IDS: Trying depth 4
Step 0: F:L, W:L, G:L, C:L, B:L (depth 0)
Step 1: F:R, W:L, G:R, C:L, B:R (depth 1)
Step 2: F:L, W:L, G:R, C:L, B:L (depth 2)
Step 3: F:R, W:L, G:R, C:R, B:R (depth 3)
Step 4: F:L, W:L, G:L, C:R, B:L (depth 4)
Step 4: F:R, W:R, G:R, C:L, B:R (depth 3)
Step 5: F:L, W:R, G:L, C:L, B:L (depth 4)
No solution found.
Time: 0.0007s
Memory used: 0 KB
```

UCS

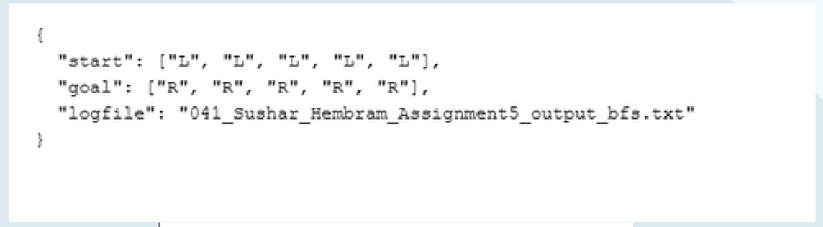


```
Step 0: F:L, W:L, G:L, C:L, B:L (cost 0)
Heap: []
Step 1: F:R, W:L, G:R, C:L, B:R (cost 2)
Heap: []
Step 2: F:L, W:L, G:R, C:L, B:L (cost 4)
Heap: []
Step 3: F:R, W:L, G:R, C:R, B:R (cost 6)
Heap: ['F:R, W:R, G:R, C:L, B:R']
Step 4: F:R, W:R, G:R, C:L, B:R (cost 6)
Heap: ['F:L, W:L, G:L, C:R, B:L']
Step 5: F:L, W:L, G:L, C:R, B:L (cost 8)
Heap: ['F:L, W:R, G:L, C:L, B:L']
Step 6: F:L, W:R, G:L, C:L, B:L (cost 8)
Heap: ['F:R, W:R, G:L, C:R, B:R']
Step 7: F:R, W:R, G:L, C:R, B:R (cost 10)
Heap: ['F:R, W:R, G:L, C:R, B:R']
Step 8: F:R, W:R, G:L, C:R, B:R (cost 10)
Heap: ['F:L, W:R, G:L, C:R, B:L']
Step 9: F:L, W:R, G:L, C:R, B:L (cost 12)
Heap: ['F:L, W:R, G:L, C:R, B:L']
Step 10: F:L, W:R, G:L, C:R, B:L (cost 12)
Heap: ['F:R, W:R, G:R, C:R, B:R']
Step 11: F:R, W:R, G:R, C:R, B:R (cost 14)
Heap: ['F:R, W:R, G:R, C:R, B:R']
Solution in 7 steps:
Step 1: F+G -> F:R, W:L, G:R, C:L, B:R
Step 2: F alone -> F:L, W:L, G:R, C:L, B:L
Step 3: F+C -> F:R, W:L, G:R, C:R, B:R
Step 4: F+G -> F:L, W:L, G:L, C:R, B:L
Step 5: F+W -> F:R, W:R, G:L, C:R, B:R
Step 6: F alone -> F:L, W:R, G:L, C:R, B:L
Step 7: F+G -> F:R, W:R, G:R, C:R, B:R
Time: 0.0009s
Memory used: 0 KB
```

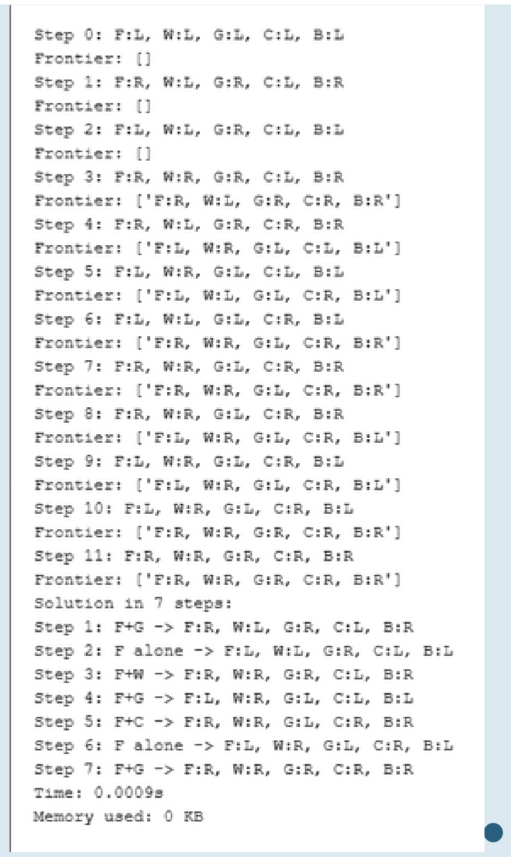


```
{
  "start": ["L", "L", "L", "L", "L"],
  "goal": ["R", "R", "R", "R", "R"],
  "logfile": "041_Sushar_Hembram_Assignment5_output_ucs.txt",
  "step_cost": 2
}
```

BFS

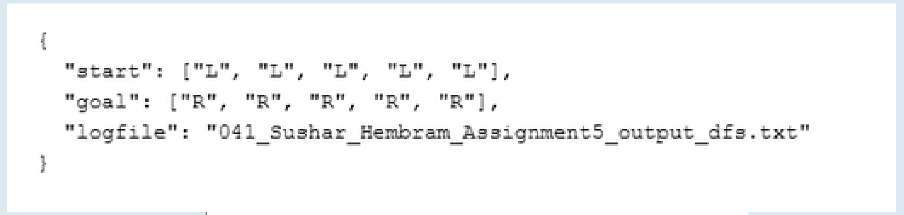


```
{
  "start": ["L", "L", "L", "L", "L"],
  "goal": ["R", "R", "R", "R", "R"],
  "logfile": "041_Sushar_Hembram_Assignment5_output_bfs.txt"
}
```

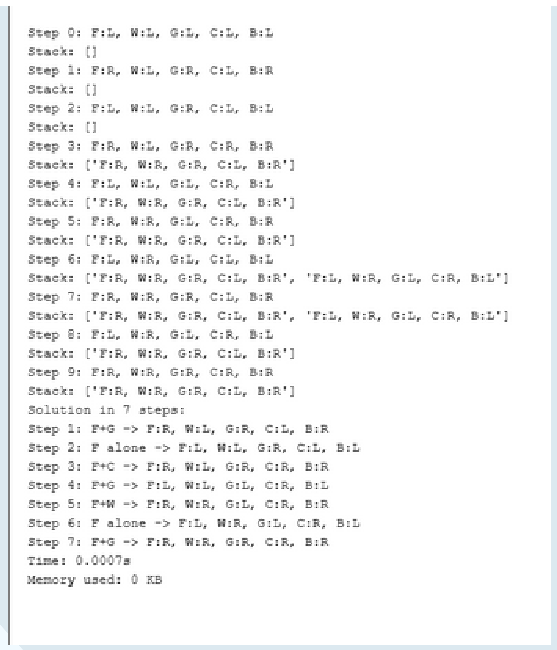


```
Step 0: F:L, W:L, G:L, C:L, B:L
Frontier: []
Step 1: F:R, W:L, G:R, C:L, B:R
Frontier: []
Step 2: F:L, W:L, G:R, C:L, B:L
Frontier: []
Step 3: F:R, W:R, G:R, C:L, B:R
Frontier: ['F:R, W:L, G:R, C:R, B:R']
Step 4: F:R, W:L, G:R, C:R, B:R
Frontier: ['F:L, W:R, G:L, C:L, B:L']
Step 5: F:L, W:R, G:L, C:L, B:L
Frontier: ['F:L, W:L, G:L, C:R, B:L']
Step 6: F:L, W:L, G:L, C:R, B:L
Frontier: ['F:R, W:R, G:L, C:R, B:R']
Step 7: F:R, W:R, G:L, C:R, B:R
Frontier: ['F:R, W:R, G:L, C:R, B:R']
Step 8: F:R, W:R, G:L, C:R, B:R
Frontier: ['F:L, W:R, G:L, C:R, B:L']
Step 9: F:L, W:R, G:L, C:R, B:L
Frontier: ['F:L, W:R, G:L, C:R, B:L']
Step 10: F:L, W:R, G:L, C:R, B:L
Frontier: ['F:R, W:R, G:R, C:R, B:R']
Step 11: F:R, W:R, G:R, C:R, B:R
Frontier: ['F:R, W:R, G:R, C:R, B:R']
Solution in 7 steps:
Step 1: F+G -> F:R, W:L, G:R, C:L, B:R
Step 2: F alone -> F:L, W:L, G:R, C:L, B:L
Step 3: F+W -> F:R, W:R, G:R, C:L, B:R
Step 4: F+G -> F:L, W:R, G:L, C:L, B:L
Step 5: F+C -> F:R, W:R, G:L, C:R, B:R
Step 6: F alone -> F:L, W:R, G:L, C:R, B:L
Step 7: F+G -> F:R, W:R, G:R, C:R, B:R
Time: 0.0009s
Memory used: 0 KB
```

DFS

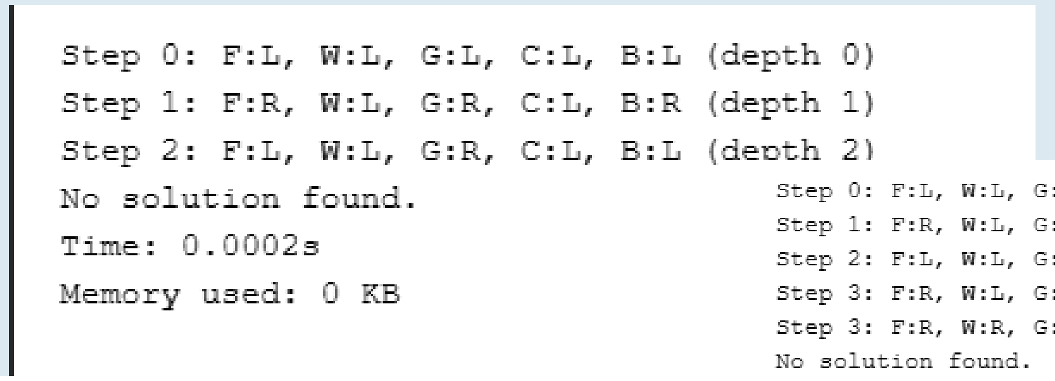


```
{
  "start": ["L", "L", "L", "L", "L"],
  "goal": ["R", "R", "R", "R", "R"],
  "logfile": "041_Sushar_Hembram_Assignment5_output_dfs.txt"
}
```

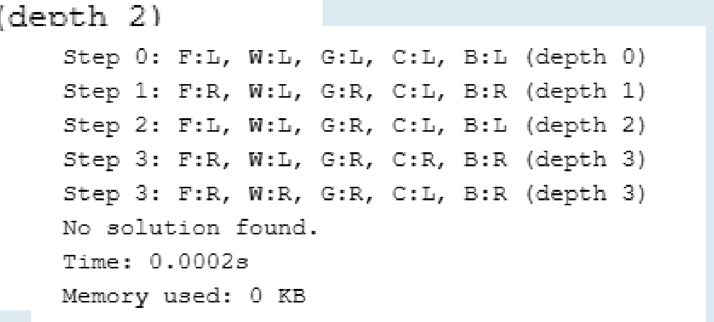


```
Step 0: F:L, W:L, G:L, C:L, B:L
Stack: []
Step 1: F:R, W:L, G:R, C:L, B:R
Stack: []
Step 2: F:L, W:L, G:R, C:L, B:L
Stack: []
Step 3: F:R, W:L, G:R, C:R, B:R
Stack: ['F:R, W:R, G:R, C:L, B:R']
Step 4: F:L, W:L, G:L, C:R, B:L
Stack: ['F:R, W:R, G:R, C:L, B:R']
Step 5: F:R, W:R, G:L, C:R, B:R
Stack: ['F:R, W:R, G:R, C:L, B:R']
Step 6: F:L, W:R, G:L, C:L, B:L
Stack: ['F:R, W:R, G:R, C:L, B:R', 'F:L, W:R, G:L, C:R, B:L']
Step 7: F:R, W:R, G:R, C:L, B:R
Stack: ['F:R, W:R, G:R, C:L, B:R', 'F:L, W:R, G:L, C:R, B:L']
Step 8: F:L, W:R, G:L, C:R, B:L
Stack: ['F:R, W:R, G:R, C:L, B:R']
Step 9: F:R, W:R, G:R, C:R, B:R
Stack: ['F:R, W:R, G:R, C:L, B:R']
Solution in 7 steps:
Step 1: F+G -> F:R, W:L, G:R, C:L, B:R
Step 2: F alone -> F:L, W:L, G:R, C:L, B:L
Step 3: F+C -> F:R, W:L, G:R, C:R, B:R
Step 4: F+G -> F:L, W:L, G:L, C:R, B:L
Step 5: F+W -> F:R, W:R, G:L, C:R, B:R
Step 6: F alone -> F:L, W:R, G:L, C:R, B:L
Step 7: F+G -> F:R, W:R, G:R, C:R, B:R
Time: 0.0007s
Memory used: 0 KB
```

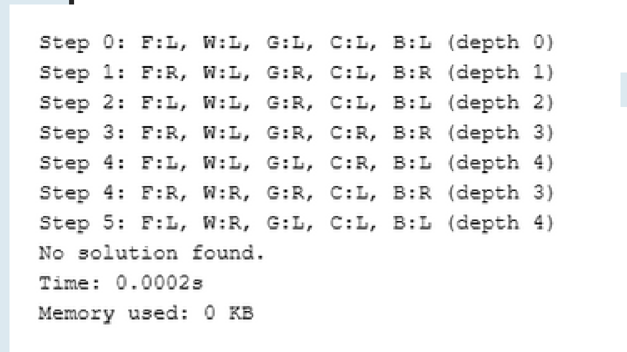
DLS



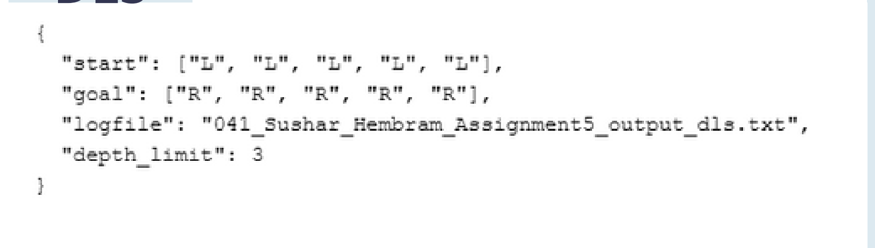
```
Step 0: F:L, W:L, G:L, C:L, B:L (depth 0)
Step 1: F:R, W:L, G:R, C:L, B:R (depth 1)
Step 2: F:L, W:L, G:R, C:L, B:L (depth 2)
No solution found.
Time: 0.0002s
Memory used: 0 KB
```



```
Step 0: F:L, W:L, G:L, C:L, B:L (depth 0)
Step 1: F:R, W:L, G:R, C:L, B:R (depth 1)
Step 2: F:L, W:L, G:R, C:L, B:L (depth 2)
Step 3: F:R, W:L, G:R, C:R, B:R (depth 3)
Step 3: F:R, W:R, G:R, C:L, B:R (depth 3)
No solution found.
Time: 0.0002s
Memory used: 0 KB
```



```
Step 0: F:L, W:L, G:L, C:L, B:L (depth 0)
Step 1: F:R, W:L, G:R, C:L, B:R (depth 1)
Step 2: F:L, W:L, G:R, C:L, B:L (depth 2)
Step 3: F:R, W:L, G:R, C:R, B:R (depth 3)
Step 4: F:L, W:L, G:L, C:R, B:L (depth 4)
Step 4: F:R, W:R, G:R, C:L, B:R (depth 3)
Step 5: F:L, W:R, G:L, C:L, B:L (depth 4)
No solution found.
Time: 0.0002s
Memory used: 0 KB
```

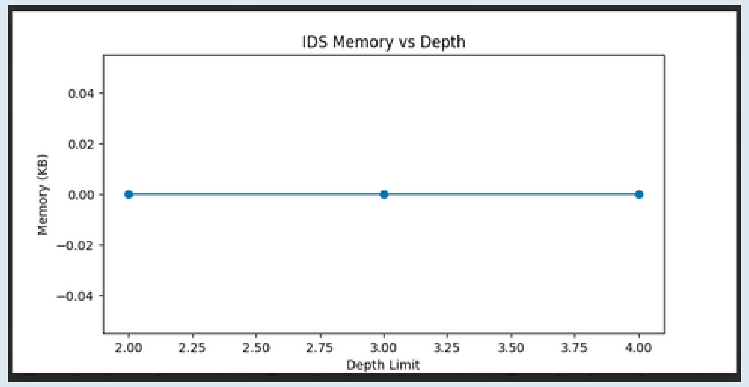


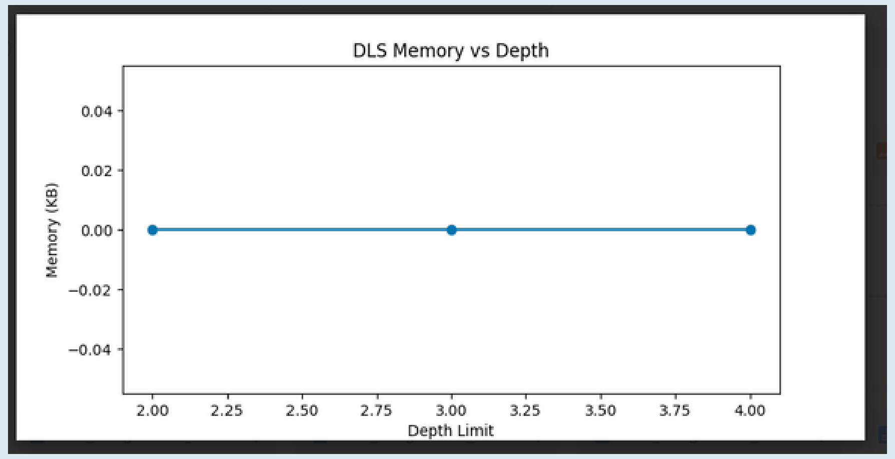
```
{
  "start": ["L", "L", "L", "L", "L"],
  "goal": ["R", "R", "R", "R", "R"],
  "logfile": "041_Sushar_Hembram_Assignment5_output_dls.txt",
  "depth_limit": 3
}
```

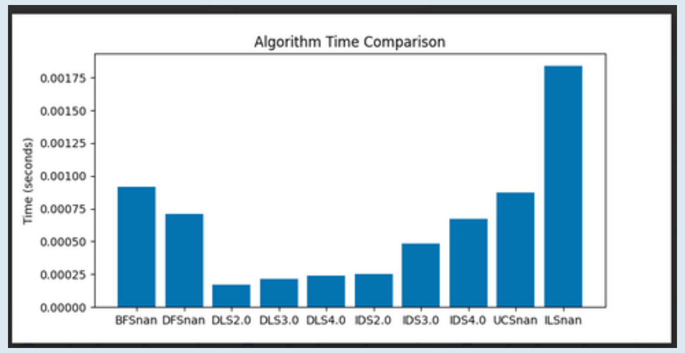
ACHIEVEMENTS & CONCLUSION

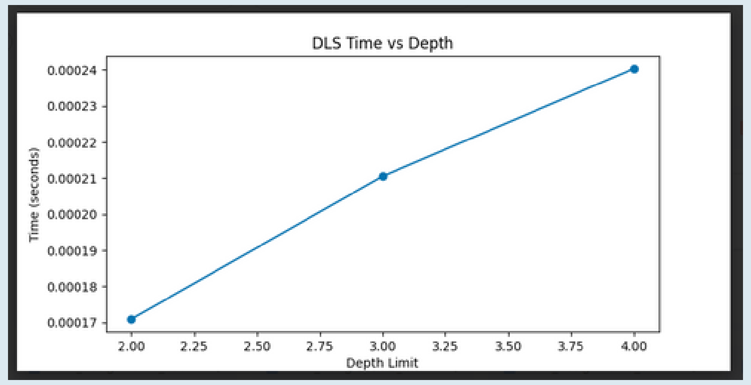
Achievements:

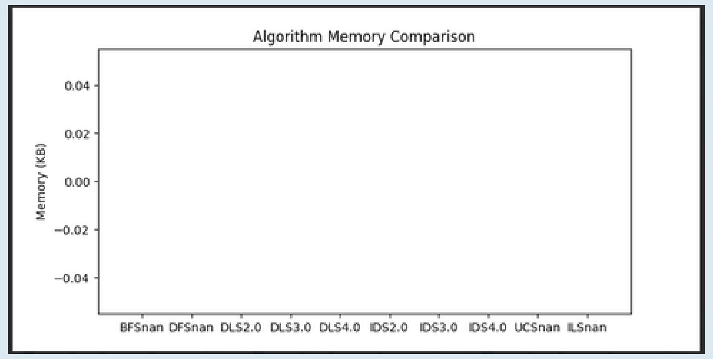
- Successfully implemented and solved the Missionaries and Carnivals problem using six different AI search algorithms.
- Collected and compared time and memory usage for each algorithm and configuration.
- Visualized algorithm performance with comparative graphs and summary tables (CSV).
- Demonstrated strengths and limitations of each search method in terms of efficiency and resource usage.

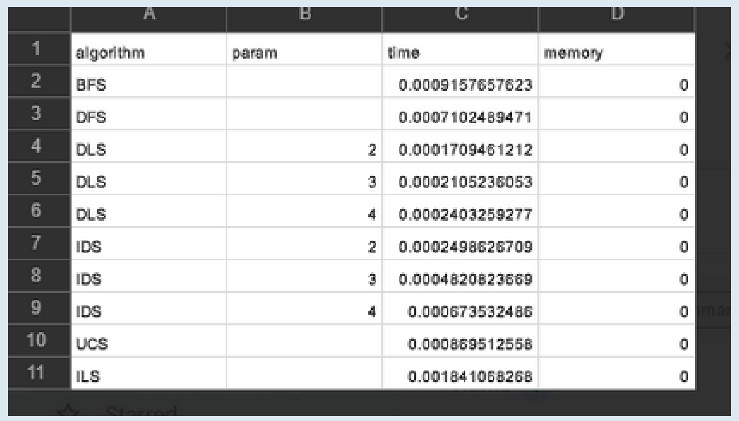












	A	B	C	D
1	algorithm	param	time	memory
2	BFS		0.0009157657623	0
3	DFS		0.0007102489471	0
4	DLS	2	0.0001709461212	0
5	DLS	3	0.0002105236053	0
6	DLS	4	0.0002403259277	0
7	IDS	2	0.0002498626709	0
8	IDS	3	0.0004820823669	0
9	IDS	4	0.000673532486	0
10	UCS		0.000869512558	0
11	ILS		0.001841068268	0

Conclusion:

- The assignment provided practical insights into the behavior and efficiency of various search algorithms.
- Comparative analysis using graphs and tables highlighted which algorithms are optimal for different constraints.
- This approach can guide algorithm selection for similar AI problems in the future.

DAY 7 ASSIGNMENT 6

DATE: 11.09.2025

Question:
"Write a Python program to solve the classical Tic-Tac-Toe problem using Best First Search. The program should handle both 3x3 and 4x4 boards, accept dynamic inputs, and use a suitable heuristic function to guide the search for a winning sequence”

Objective:
To implement and analyze Best First Search for Tic-Tac-Toe, demonstrating heuristic-based decision making, dynamic input handling, and output generation for both board sizes, with clear tracking of intermediate and final states.

APPROACH & ALGORITHMS USED

Algorithm/Technique:

Best First Search is used to solve the Tic-Tac-Toe problem. The algorithm explores possible board states by always expanding the most promising node, as determined by a heuristic function.

Approach:

- Represent the board as a 3x3 or 4x4 grid, with dynamic input for initial state and player.
- Generate all possible moves for the current player, creating new board states.
- Use a heuristic function to evaluate each state, prioritizing moves that maximize the player's chances of winning.
- Expand nodes in order of their heuristic value until a winning sequence is found or all possibilities are exhausted.
- Record intermediate steps and final solution for analysis.

Techniques:

- Heuristic evaluation based on possible winning lines for each player.
- Priority queue for node expansion.
- Dynamic handling of board size and player input.

PROCESS & METHODS

Step 1: Input Handling

- Read board size, initial board state, and starting player from an input file.

↓

Step 2: State Representation

- Model the board as a grid (3x3 or 4x4) with cells marked as X, O, or empty.

↓

Step 3: Move Generation

- For each empty cell, generate possible moves for the current player.

↓

Step 4: Heuristic Evaluation

- Use a heuristic function to score each board state based on potential winning lines.

↓

Step 5: Best First Search Execution

- Expand the most promising board state using a priority queue.
- Track explored states and avoid revisiting.

↓

Step 6: Output Generation

- Record each step and board state in an output file.
- Display the winning sequence or report if no solution is found.

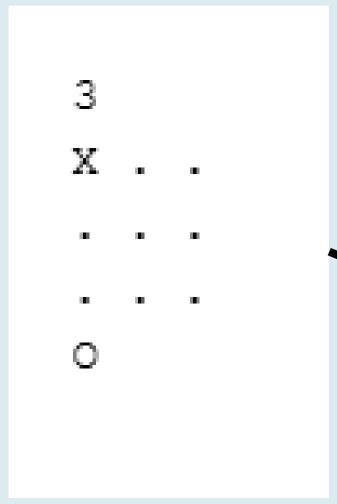
INPUT & OUTPUT

Input:

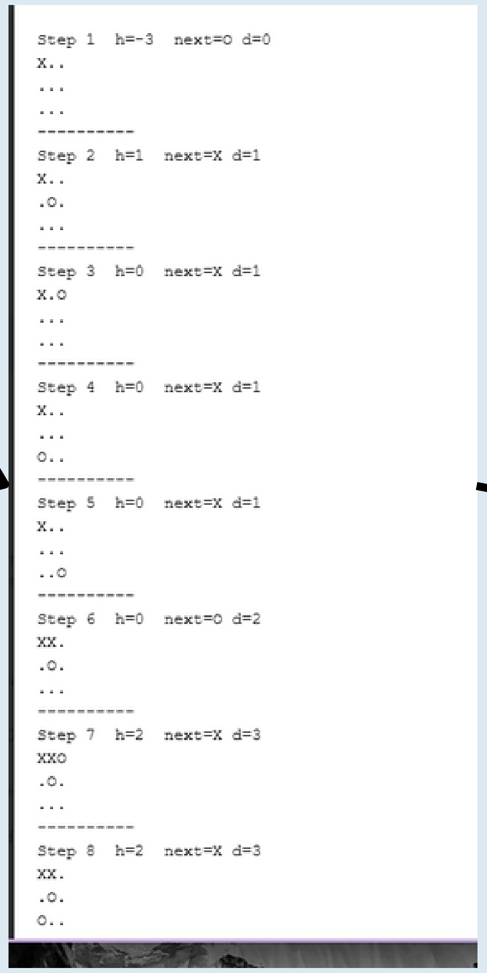
- Board size (3x3 or 4x4), initial board state, and starting player, read from an input file.

Output:

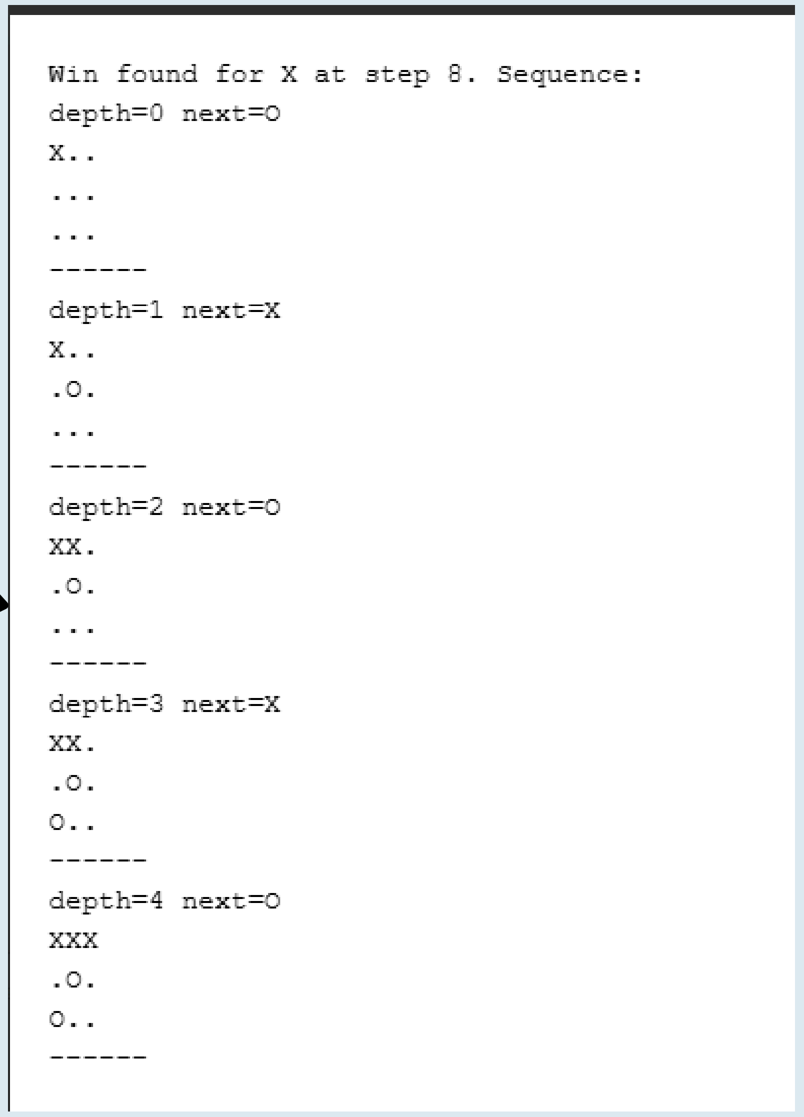
- Sequence of board states and moves leading to a win, recorded in the output file.
- Intermediate steps and heuristic values displayed in a steps file.



```
3
X . .
. . .
. . .
O
```



```
Step 1  h=-3  next=O d=0
X..
...
...
----------
Step 2  h=1  next=X d=1
X..
.O.
...
----------
Step 3  h=0  next=X d=1
X.O
...
...
----------
Step 4  h=0  next=X d=1
X..
...
O..
----------
Step 5  h=0  next=X d=1
X..
...
..O
----------
Step 6  h=0  next=O d=2
XX.
.O.
...
----------
Step 7  h=2  next=X d=3
XXO
.O.
...
----------
Step 8  h=2  next=X d=3
XX.
.O.
O..
```



```
Win found for X at step 8. Sequence:
depth=0 next=O
X..
...
...
------
depth=1 next=X
X..
.O.
...
------
depth=2 next=O
XX.
.O.
...
------
depth=3 next=X
XX.
.O.
O..
------
depth=4 next=O
XXX
.O.
O..
------
```

ACHIEVEMENTS & CONCLUSION

Results:

- Successfully implemented Best First Search to solve Tic-Tac-Toe for both 3x3 and 4x4 boards.
- Designed and applied a heuristic function to guide the search efficiently.
- Recorded all intermediate steps and winning sequences in output files.

Challenges:

- Designing an effective heuristic to evaluate board states.
- Managing the increased complexity and state space for larger boards.
- Ensuring dynamic input handling and avoiding repeated states.

Conclusion:

- The assignment demonstrated the power of heuristic-based search in game solving.
- Gained practical experience in AI search strategies and state evaluation.
- The approach can be extended to other board games and decision-making problems.

DAY 8 ASSIGNMENT 7

DATE: 18.09.2025

Question:

"Write a Python program to solve the Travelling Salesman Problem (TSP) using any iterative optimization methods. The program should find the shortest possible route starting from a given city, using a suitable heuristic function and dynamic input parameters."

Objective:

To implement and analyze the optimization method used by the student for TSP, demonstrating iterative improvement, heuristic-based decision making, and output generation of both intermediate and final solutions, with clear tracking of optimization progress.

APPROACH & ALGORITHMS USED

Algorithm/Technique:
Simulated Annealing, an iterative optimization technique, is used to solve the Travelling Salesman Problem (TSP). The algorithm explores possible tours, accepting both improvements and occasional worse solutions to escape local minima, guided by a temperature parameter that gradually decreases.

Approach:

- Represent cities and distances as a matrix, with dynamic input for start city and parameters.
- Generate an initial tour using the Nearest Neighbor heuristic.
- Iteratively generate neighboring tours by swapping cities.
- Use a heuristic function (tour cost) to evaluate each tour.
- Accept new tours based on cost and current temperature, allowing probabilistic jumps to avoid local optima.
- Gradually reduce temperature to focus the search and converge to an optimal solution.
- Record intermediate and final results for analysis.

Techniques:

- Iterative improvement and probabilistic acceptance
- Heuristic evaluation of tour cost
- Dynamic input handling and output tracking

PROCESS & METHODS

Step 1: Input Handling

- Read the distance matrix, start city, initial temperature, cooling rate, and number of iterations from the input file..

↓

Step 2: Initial Tour Generation

- Create an initial tour using the Nearest Neighbor heuristic.

↓

Step 3: Simulated Annealing Iteration

- For each iteration:
  - Generate a neighboring tour by swapping two cities.
  - Calculate the cost of the new tour.
  - Decide whether to accept the new tour based on cost and current temperature (probabilistic acceptance).

↓

Step 4: Temperature Update

- Gradually decrease the temperature using the cooling rate.

↓

Step 5: Output Generation

- Record intermediate results (tour, cost, temperature) in an output file.
- Save the best tour and its cost as the final solution.

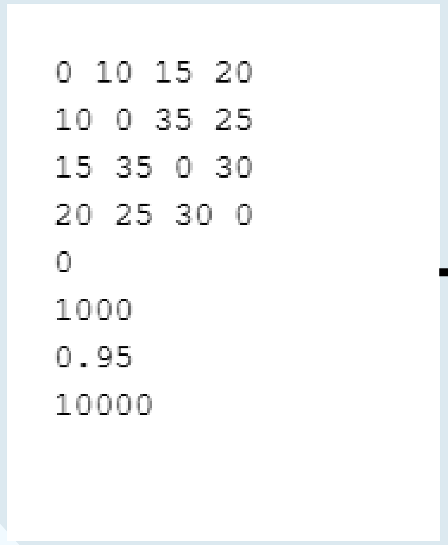
INPUT & OUTPUT

Input:

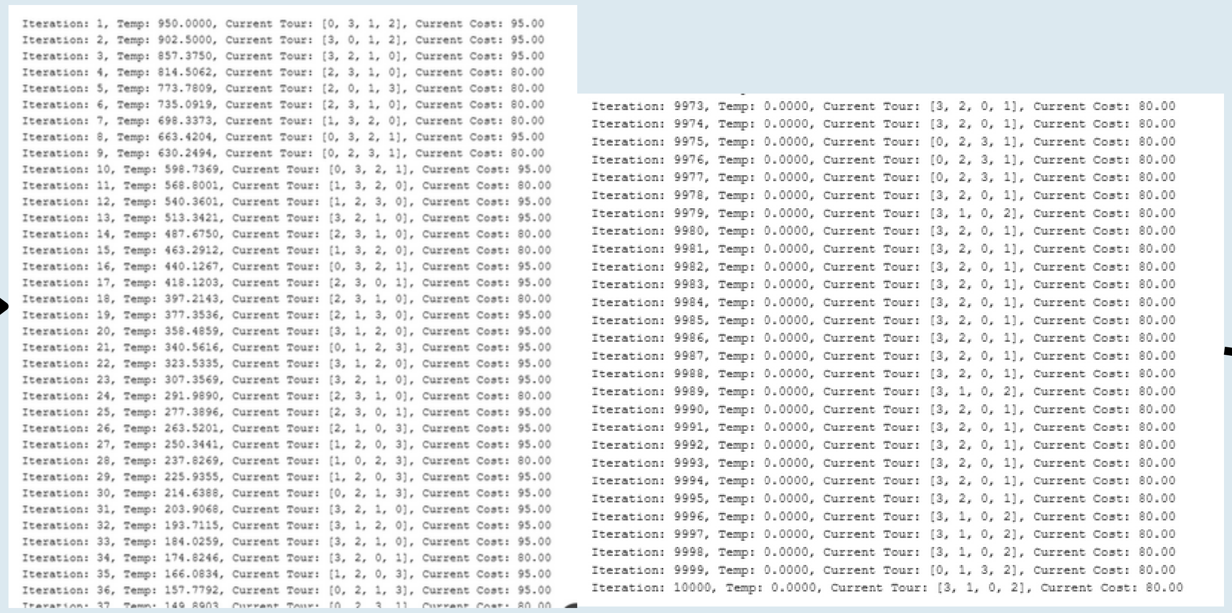
- **Distance matrix between cities, start city, initial temperature, cooling rate, and number of iterations, read from an input file.**

Output:

- Intermediate results for each iteration: current tour, cost, and temperature, recorded in an output file.
- Final output: best tour found and its total cost, saved in a separate output file.

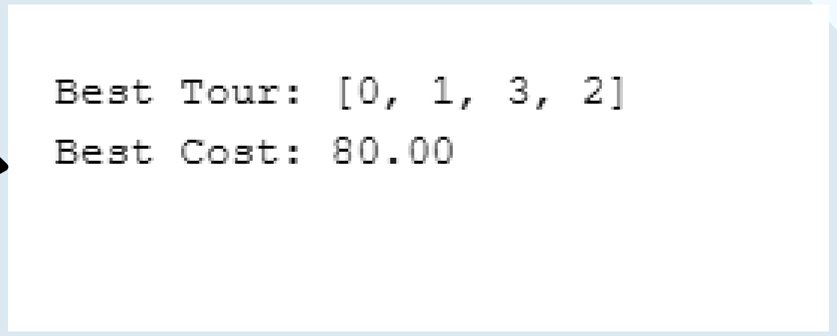


```
0 10 15 20
10 0 35 25
15 35 0 30
20 25 30 0
0
1000
0.95
10000
```



```
Iteration: 1, Temp: 950.0000, Current Tour: [0, 3, 1, 2], Current Cost: 95.00
Iteration: 2, Temp: 902.5000, Current Tour: [3, 0, 1, 2], Current Cost: 95.00
Iteration: 3, Temp: 857.3750, Current Tour: [3, 2, 1, 0], Current Cost: 95.00
Iteration: 4, Temp: 814.5062, Current Tour: [2, 3, 1, 0], Current Cost: 80.00
Iteration: 5, Temp: 773.7809, Current Tour: [2, 0, 1, 3], Current Cost: 80.00
Iteration: 6, Temp: 735.0919, Current Tour: [2, 3, 1, 0], Current Cost: 80.00
Iteration: 7, Temp: 698.3373, Current Tour: [1, 3, 2, 0], Current Cost: 80.00
Iteration: 8, Temp: 663.4204, Current Tour: [0, 3, 2, 1], Current Cost: 95.00
Iteration: 9, Temp: 630.2494, Current Tour: [0, 2, 3, 1], Current Cost: 80.00
Iteration: 10, Temp: 598.7369, Current Tour: [0, 3, 2, 1], Current Cost: 95.00
Iteration: 11, Temp: 568.8001, Current Tour: [1, 3, 2, 0], Current Cost: 80.00
Iteration: 12, Temp: 540.3601, Current Tour: [1, 2, 3, 0], Current Cost: 95.00
Iteration: 13, Temp: 513.3421, Current Tour: [3, 2, 1, 0], Current Cost: 95.00
Iteration: 14, Temp: 487.6750, Current Tour: [2, 3, 1, 0], Current Cost: 80.00
Iteration: 15, Temp: 463.2912, Current Tour: [1, 3, 2, 0], Current Cost: 80.00
Iteration: 16, Temp: 440.1267, Current Tour: [0, 3, 2, 1], Current Cost: 95.00
Iteration: 17, Temp: 418.1203, Current Tour: [2, 3, 0, 1], Current Cost: 95.00
Iteration: 18, Temp: 397.2143, Current Tour: [2, 3, 1, 0], Current Cost: 80.00
Iteration: 19, Temp: 377.3536, Current Tour: [2, 1, 3, 0], Current Cost: 95.00
Iteration: 20, Temp: 358.4859, Current Tour: [3, 1, 2, 0], Current Cost: 95.00
Iteration: 21, Temp: 340.5616, Current Tour: [0, 1, 2, 3], Current Cost: 95.00
Iteration: 22, Temp: 323.5335, Current Tour: [3, 1, 2, 0], Current Cost: 95.00
Iteration: 23, Temp: 307.3569, Current Tour: [3, 2, 1, 0], Current Cost: 95.00
Iteration: 24, Temp: 291.9890, Current Tour: [2, 3, 1, 0], Current Cost: 80.00
Iteration: 25, Temp: 277.3896, Current Tour: [2, 3, 0, 1], Current Cost: 95.00
Iteration: 26, Temp: 263.5201, Current Tour: [2, 1, 0, 3], Current Cost: 95.00
Iteration: 27, Temp: 250.3441, Current Tour: [1, 2, 0, 3], Current Cost: 95.00
Iteration: 28, Temp: 237.8269, Current Tour: [1, 0, 2, 3], Current Cost: 80.00
Iteration: 29, Temp: 225.9355, Current Tour: [1, 2, 0, 3], Current Cost: 95.00
Iteration: 30, Temp: 214.6388, Current Tour: [0, 2, 1, 3], Current Cost: 95.00
Iteration: 31, Temp: 203.9068, Current Tour: [3, 2, 1, 0], Current Cost: 95.00
Iteration: 32, Temp: 193.7115, Current Tour: [3, 1, 2, 0], Current Cost: 95.00
Iteration: 33, Temp: 184.0259, Current Tour: [3, 2, 1, 0], Current Cost: 95.00
Iteration: 34, Temp: 174.8246, Current Tour: [3, 2, 0, 1], Current Cost: 80.00
Iteration: 35, Temp: 166.0834, Current Tour: [1, 2, 0, 3], Current Cost: 95.00
Iteration: 36, Temp: 157.7792, Current Tour: [0, 2, 1, 3], Current Cost: 95.00
Iteration: 37, Temp: 149.8903, Current Tour: [0, 2, 3, 1], Current Cost: 80.00
```

```
Iteration: 9973, Temp: 0.0000, Current Tour: [3, 2, 0, 1], Current Cost: 80.00
Iteration: 9974, Temp: 0.0000, Current Tour: [3, 2, 0, 1], Current Cost: 80.00
Iteration: 9975, Temp: 0.0000, Current Tour: [0, 2, 3, 1], Current Cost: 80.00
Iteration: 9976, Temp: 0.0000, Current Tour: [0, 2, 3, 1], Current Cost: 80.00
Iteration: 9977, Temp: 0.0000, Current Tour: [0, 2, 3, 1], Current Cost: 80.00
Iteration: 9978, Temp: 0.0000, Current Tour: [3, 2, 0, 1], Current Cost: 80.00
Iteration: 9979, Temp: 0.0000, Current Tour: [3, 1, 0, 2], Current Cost: 80.00
Iteration: 9980, Temp: 0.0000, Current Tour: [3, 2, 0, 1], Current Cost: 80.00
Iteration: 9981, Temp: 0.0000, Current Tour: [3, 2, 0, 1], Current Cost: 80.00
Iteration: 9982, Temp: 0.0000, Current Tour: [3, 2, 0, 1], Current Cost: 80.00
Iteration: 9983, Temp: 0.0000, Current Tour: [3, 2, 0, 1], Current Cost: 80.00
Iteration: 9984, Temp: 0.0000, Current Tour: [3, 2, 0, 1], Current Cost: 80.00
Iteration: 9985, Temp: 0.0000, Current Tour: [3, 2, 0, 1], Current Cost: 80.00
Iteration: 9986, Temp: 0.0000, Current Tour: [3, 2, 0, 1], Current Cost: 80.00
Iteration: 9987, Temp: 0.0000, Current Tour: [3, 2, 0, 1], Current Cost: 80.00
Iteration: 9988, Temp: 0.0000, Current Tour: [3, 2, 0, 1], Current Cost: 80.00
Iteration: 9989, Temp: 0.0000, Current Tour: [3, 1, 0, 2], Current Cost: 80.00
Iteration: 9990, Temp: 0.0000, Current Tour: [3, 2, 0, 1], Current Cost: 80.00
Iteration: 9991, Temp: 0.0000, Current Tour: [3, 2, 0, 1], Current Cost: 80.00
Iteration: 9992, Temp: 0.0000, Current Tour: [3, 2, 0, 1], Current Cost: 80.00
Iteration: 9993, Temp: 0.0000, Current Tour: [3, 2, 0, 1], Current Cost: 80.00
Iteration: 9994, Temp: 0.0000, Current Tour: [3, 2, 0, 1], Current Cost: 80.00
Iteration: 9995, Temp: 0.0000, Current Tour: [3, 2, 0, 1], Current Cost: 80.00
Iteration: 9996, Temp: 0.0000, Current Tour: [3, 1, 0, 2], Current Cost: 80.00
Iteration: 9997, Temp: 0.0000, Current Tour: [3, 1, 0, 2], Current Cost: 80.00
Iteration: 9998, Temp: 0.0000, Current Tour: [3, 1, 0, 2], Current Cost: 80.00
Iteration: 9999, Temp: 0.0000, Current Tour: [0, 1, 3, 2], Current Cost: 80.00
Iteration: 10000, Temp: 0.0000, Current Tour: [3, 1, 0, 2], Current Cost: 80.00
```



```
Best Tour: [0, 1, 3, 2]
Best Cost: 80.00
```

ACHIEVEMENTS & CONCLUSION

Results:

- Successfully implemented Simulated Annealing to solve the Travelling Salesman Problem.
- Found an optimized tour with minimal cost using iterative improvement and probabilistic acceptance.
- Recorded and analyzed intermediate and final results for solution quality.

Challenges Faced:

- Designing an effective cooling schedule and heuristic for convergence.
- Managing randomness and ensuring reproducibility of results.
- Handling large iteration counts for better optimization without excessive computation time.

Conclusion:

- The assignment demonstrated the power of iterative optimization and heuristic-guided search for complex problems.
- Simulated Annealing provided a practical approach to escape local minima and find near-optimal solutions.
- The methodology can be extended to other combinatorial optimization tasks in AI.

REFERENCE

Link Name: https://github.com/UNKN0WN006/artificial-intelligence-lab

Brief Description: This Github repository contains all the codes and writeups shared and discussed.

THANK YOU

FOR YOUR ATTENTION