Assignment 5
Write-up: My Approach to the River Crossing AI Search Problem

Problem Representation and Variables

State Representation

I represented each state as a tuple of five elements: the positions of the farmer, wolf, goat, cabbage, and boat. Each can be either 'L' (left bank) or 'R' (right bank). For example, ('L', 'L', 'L', 'L', 'L') means everyone is on the left bank. This compact tuple encodes the entire configuration of the problem at any step and makes it easy to check for valid moves and to log the state at each step.

Input and Output Files

To keep my code flexible and reproducible, I used JSON input files for all parameters. Each algorithm has its own input file, specifying the start and goal states, the output log file name, and any algorithm-specific parameters (like depth_limit for DLS, max_depth for IDS, max_restarts for ILS, or step_cost for UCS). This way, I could easily experiment with different settings without changing the code.

Each algorithm writes its intermediate steps and final solution to a dedicated output file (e.g., *041_Sushar_Hembram_Assignment5_output_bfs.txt*). These log files contain all the steps, the solution (if found), and the time and memory used.

Algorithmic Variables

Frontier Structures:

queue: Used in BFS (FIFO queue)
stack: Used in DFS, DLS, IDS, ILS (LIFO stack)
heap: Used in UCS (priority queue/min-heap)

Visited Set:

visited: Set to keep track of already explored states to avoid cycles and redundant work.

Path and Actions:

path: List of states from the start to the current state.
actions: List of actions taken to reach each state.

Logging and Performance Variables

Logging:

logfile: The output file where all steps and results are written.
log_step(logfile, msg): Function to append messages to the log file.

Performance:
get_memory_kb(): Function to get current memory usage in KB.
time.time(): Used to measure elapsed time for each algorithm.
mem_before, ***mem_after***: Memory usage before and after running an algorithm.
t0, t1: Start and end times for measuring execution duration.
Algorithm Parameters
All parameters are read from the input files:

depth_limit: For DLS (Depth-Limited Search)
max_depth: For IDS (Iterative Deepening Search)
max_restarts: For ILS (Iterative Local Search)
step_cost: For UCS (Uniform Cost Search)
This parameterization allows me to control the behavior of each algorithm without changing the code.

Process Flow in My Notebook

Input Handling
Each algorithm reads its own input file, which specifies the start and goal states, output file, and any algorithm-specific parameters.

Algorithm Execution
For each algorithm (BFS, DFS, DLS, IDS, UCS, ILS):

The algorithm is called with the parameters from its input file.
The search proceeds, expanding states and logging each step to its output file.
The path and actions are tracked for solution reconstruction.
Time and memory usage are measured before and after the run.

Logging
Every step (state expansion, frontier/stack/heap contents, actions) is logged to the output file. At the end, the solution path (if found), total time, and memory used are also logged.

Results Aggregation
After all algorithms run, their results (algorithm name, parameter, time, memory) are collected into a list of dictionaries. This list is converted to a pandas DataFrame for easy analysis and saved as a CSV file.

Visualization
Four plots are generated:

- Bar chart of time for all algorithms.
- Bar chart of memory for all algorithms.
- Line plot of time vs. depth for DLS and IDS.
- Line plot of memory vs. depth for DLS and IDS.

Plots are saved as PNG files and displayed in the notebook.

Output Download
Output files, plots, and the summary CSV can be downloaded for submission or further analysis.

Explanation of the Notebook Structure
My notebook is organized into the following sections:

- **Input File Insertion:** I use the Colab file upload utility to upload all required input files (JSON format) for each algorithm.

- **Imports and Utility Functions:** I import all necessary libraries (time, resource, json, deque, matplotlib, pandas) and define utility functions for memory measurement, state validation, state transitions, and logging.

- **Algorithm Implementations:** I implement BFS, DFS, DLS, IDS, UCS, and ILS as separate functions. Each function logs every step and the current state of the frontier (queue, stack, or heap).

- **Main Function:** The ***run_and_log*** function handles running each algorithm, measuring time and memory, and logging results. It returns a dictionary with the algorithm name, parameter, time, and memory used.

- **Running All Algorithms:** I read each input file, run the corresponding algorithm, and collect the results. For DLS and IDS, I run each with three different depth limits and create separate output files for each run.

- **Results Table and Plots**: I aggregate all results into a pandas DataFrame, print it, and save it as a CSV file. I then generate and save all required plots for time and memory comparisons.

- **Download Section:** I provide code to download all output files, plots, and the summary CSV from Colab.

Design Choices and Rationale

I chose to keep input and output files separate for each algorithm to avoid confusion and make debugging easier. Parameterizing everything via input files makes my experiments reproducible and easy to modify. Logging every step, not just the final answer, gives a transparent view of the search process and helps with troubleshooting.

I also made sure to use efficient data structures for each algorithm (queues for BFS, stacks for DFS/DLS/IDS/ILS, and a heap for UCS) to reflect their theoretical properties in practice.

Conclusion

By parameterizing my code, logging every step, and systematically measuring performance, I was able to not only solve the river crossing problem but also compare the algorithms in a rigorous way. The results and visualizations provide clear evidence of the strengths and weaknesses of each approach, and the modular design means I can easily extend this framework to other search problems in the future.

How My Notebook Works

****Note** : The files are supposed to present somewhere in device say in downloads so that you are supposed to upload by selecting the input files**

Input files (JSON) are uploaded and specify all parameters for each algorithm.
Each algorithm reads its input, runs, and writes a detailed log to its own output file.
All steps, solutions, time, and memory are logged for transparency.
Results are collected into a summary table and visualized with four plots.
All files (logs, plots, summary) can be downloaded for submission or further analysis.