T.C.

MARMARA UNIVERSITY

FACULTY of ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

CSE4082 Artificial Intelligence – Assignment 1

Knight's Tour Problem (KTP)

Group Members

130319659 - Cihan Erdoğanyılmaz

150120998 - Abdelrahman Zahran

150120997 - Mohamad Nael Ayoubi

1. Problem Definition

The Knight's Tour Problem involves finding a sequence of moves for a knight on an n x n chessboard such that:

1) The knight visits each square exactly once.
2) The tour starts from a given square (e.g., a1), and a solution must traverse all cells on the board without revisiting any.

Formal Definition:

- State Space: Set of all reachable board positions.
- Initial State: Knight starting at position (0,0).
- Goal State: All board cells are visited exactly once (total moves = n * n).
- Actions: Moves in L-shape, where a knight can move: {±2, ±1} or {±1, ±2} in the x and y directions.
- Transition Function: Valid moves must: Stay within board boundaries. Avoid revisiting already visited positions.

Environment Classification:

- Fully Observable: All board positions and moves are known.
- Deterministic: The next state is determined by the current state and chosen move.
- Discrete: Finite number of valid moves at each step.
- Single-agent: Only one knight moves on the board.
- Static: Board state does not change during the search process.

2. Search Methods

The program implements the following search algorithms using the general tree framework (treeSearch(Problem problem, String strategy) **line 243 in KnightTour.java**)

1) Breadth First Search: Explores all nodes level-by-level using a queue (FIFO). Ensures optimal solution (if exists) but may consume memory.
2) Depth First Search: Explores nodes depth-wise using a stack (LIFO).
3) DFS with Heuristic h1b [1]: Chooses moves that minimize the number of possible moves at the next step. (Warnsdorff's Rule)
4) DFS with Heuristic h1 [1]: Uses h1b for sorting but breaks ties based on the knight's distance to the nearest board corner.

3. Design and Implementation

The Java program contains the following components:

1) Node Class: Represents a single move of the knight with attributes:
    - x, y: Position on the board.
    - depth: Move count.
    - parent: Pointer to the previous node for backtracking.
    - Heuristic methods: calculateH1b() and nearestCornerDistance().
2) Problem Class: Contains the problem definition (board size, valid moves).
    - Methods: isGoal(Node node): Checks if the goal state is reached.
    - expand(Node node, heuristicType): Generates child nodes and applies heuristics for sorting.
    - isValid(Node node, int x, int y): Validates moves within board boundaries and checks for revisits.
3) Tree Search Function: Implements search strategies (BFS, DFS, DFS-h1b, DFS-h2):
    - Input: Search strategy, time constraint.
    - Output: Return status (A solution found, Timeout, Out of Memory, No solution exists). Solution path (if found). Number of nodes expanded. Execution time.
4) Main Method: Takes user inputs: board size (n), search method (a-d), and time constraint (t). Calls the appropriate search method and prints results.

General Tree Search Framework:

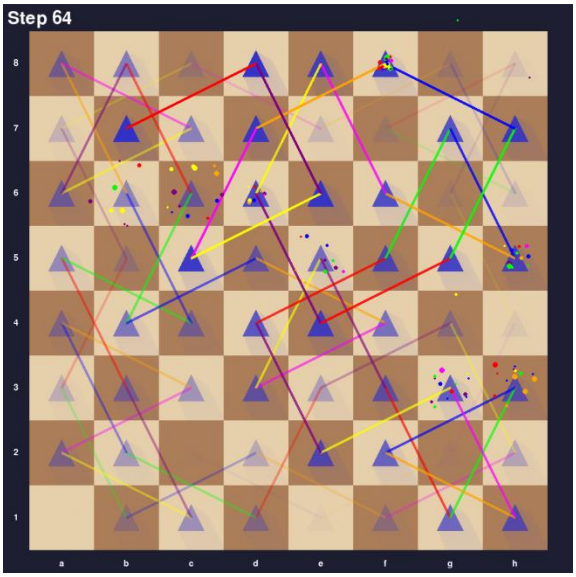
```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
   initialize the frontier (nodes to be visited) using the initial state of problem
   loop do
       if the frontier is empty then return failure
      choose a leaf node for expansion according to strategy and remove it from the
frontier
       if the node contains a goal state then return the corresponding solution
       expand the node and add the resulting nodes to the frontier
   end
```

4. Results

The results for each search method a-d, and for each board size n = 8, 16, 32, 41, and 52 with a time limit of 15 minutes are given in this section. Also, the output log files board.txt, path.txt, and moves.txt are also generated by the program and printed to terminal. The files of the runs are included in the zip folder under **Output** Directory.

To run visualization using the python script feed the **board.txt** and **path.txt** generated by the Java code to **Vizualization.py**. Please make sure that these scripts are under the same folder with the mentioned text files.

Example of visualization is shown below



For N=8, 16, 32, 41, 52 runs please proceed to Output – Tree Search Algorithm Directory We have provided Complete Runs with:

Board + Path + Moves (Total No. of Nodes Expanded + Time) in these log files

- **For each of the 4 algorithms (BFS, DFS, DFS-H1B, DFS-H2).**
- **Visualization Script that takes Board.txt + Path.txt as Input to visualize the N x N Board and display the Knight Tour. – For Large N Values – Feel Free to speed up the FPS value to get a faster Animation.**

- **In Short:**
- BFS for N = 8, 16, 32, 41, 52 – Out Of Memory
- DFS for N = 8, 16, 32, 41, 52 – Time Out.
- DFS-H1B for N = 8, 16, 32, 41, 52 – Solution Found, Hence for N = 41 Our H1B implementation is a complete implementation which guarantees to find a solution most of the time but in some runs we have received time out which have confirmed the claim in the Research Paper - Heuristic Strategies for the Knight Tour Problem that H1B fails to always find a solution for N = 41.
- DFS for N = 8, 16, 32, 41, 52 – Solution Found.

5. Maximum Board Size

The maximum board size that our program can find a solution within 15 minutes using DFS-h2 is:

This is a debatable point of view because it depends on the current computational resources in hand and it is capabilities to run large values of N. If we have unlimited resources we can guarantee the DFS-H2 strategy can find solution for the knight tour problem with any N value (any board size).

In The **Output Folder**, there is **Max in 15 Min** there is a couple of different runs for large N values - **Search Strategy: DFS-H2** - with the time constraint 15 min such as N = 245, N = 250, N = 255, N = 259. Before and After Each Run, we clear the workspace cache and reload the java environment to ensure the best run with the highest computational efficiency.