A logo with text on it

Description automatically generated

**T.C.**

**MARMARA UNIVERSITY**

**FACULTY of ENGINEERING**

**COMPUTER ENGINEERING DEPARTMENT**

**CSE4082 – Artificial Intelligence**

**Assignment - 1**

**150120063 – Hamza Ali İSLAMOĞLU**

**Implementation Overview**

**Tree Search Algorithm Implementation:** The Tree Search algorithm, as discussed in class, is implemented in the following file and line range:

* File: **KnightTourSolver.java**
* Line Range: Lines **20** to **100** (in method **“solve”)**

This method contains the main logic for traversing the search space, generating nodes, and expanding them according to the selected search strategy (BFS, DFS, DFS with h1b, and DFS with h2).

**Heuristics and Node Selection**

The heuristics provided in the referenced paper are applied exclusively to the **Depth-First Search (DFS)** strategy for node selection. These heuristics are integrated directly into the **“generateChildren”** method without creating separate functions to comply with assignment requirements. For **Breadth-First Search (BFS)**, no heuristics are applied; nodes are simply added to the frontier in the order they are generated.

* Heuristic h1b: Prioritizes nodes based on the number of valid moves from a given position, sorting in ascending order.
* Heuristic h2: Combines the valid moves count with the distance to the nearest corner, ensuring nodes closer to corners are prioritized in case of a tie.

Implementation Details:

* File: **SearchAlgorithm.java**
* Line Range: Lines **46** to **90** (in method **“generateChildren”)**
* Integration: The heuristics are applied directly during child node generation for DFS only. BFS follows a simpler approach by adding children to the frontier without additional sorting.

**Problem Description**

**Formal Problem Statement:**

* **State Space:** The set of all possible knight positions on a chessboard.
* **Initial State:** The given starting position of the knight.
* **Actions:** Moves that the knight can make according to chess rules.
* **Transition Model:** A move from the current position to any reachable, unvisited square.
* **Goal State:** All squares of the board are visited exactly once.

**Environmental Classification:**

* **Deterministic:** The outcome of each move is predictable.
* **Fully Observable:** All squares of the chessboard are visible at all times.
* **Static:** The board does not change during the problem-solving process.
* **Discrete:** Both the state space and action set are finite and countable.

The Knight's Tour Problem (KTP) is a classic chessboard problem where a knight must visit each square of an chessboard exactly once. The problem involves finding a valid sequence of moves, starting from any given position, that satisfies this requirement.

**File Descriptions**

1. **Main.java:**
   * Handles user input for board size, starting position, move set selection, and search strategy.
   * Invokes the solver with the specified parameters.
2. **KnightTourSolver.java:**
   * Implements the Tree Search algorithm with BFS, DFS, DFS with h1b, and DFS with h2 strategies.
   * Manages the search process, including timing, node expansion, and solution reporting.
3. **Node.java:**
   * Represents individual board states with attributes for position, path cost, parent node, and visited squares.
4. **SearchAlgorithm.java:**
   * Provides common functionality for generating child nodes and validating moves.
   * Contains logic for applying heuristics during DFS.

**Constraints and Assumptions**

**Constraints:**

1. The Tree Search algorithm must be used for all strategies.
2. Heuristics from the paper are applied only to DFS.
3. The program must handle with a time limit of 15 minutes.
4. The chessboard coordinates follow the standard notation, e.g., **(1,1) for the lower-left corner.**

**Assumptions:**

1. The input is always valid and within the specified ranges.
2. The system running the program has sufficient memory for large board sizes.

**Output Description**

1. **Input Parameters:**
   * Board size (n) (max 361)
   * Search method (BFS, DFS, DFS with h1b, DFS with h2) (1, 2, 3, 4)
   * Starting position [(1,1) for the lower-left corner]
   * Time limit (15 minutes)
2. **Output Information:**
   * The search method and time limit used.
   * Status messages: “Solution Found”, “No Solution Exists”, “Timeout”, or “Out of Memory”.
   * Solution path (if found).
   * Total nodes expanded and created.
   * Time spent.

**Move Sets and Log Outputs**

To introduce variety in the knight's moves, five distinct move sets were designed. Each move set alters the order and direction of the knight's potential moves:

**Available Move Sets:**

* **Index 0:**
  + dx: [-2, -1, 1, 2, 2, 1, -1, -2]
  + dy: [1, 2, 2, 1, -1, -2, -2, -1]
* **Index 1:**
  + dx: [-2, -1, 2, 1, 2, 1, -1, -2]
  + dy: [1, 2, 1, 2, -1, -2, -2, -1]
* **Index 2:**
  + dx: [-2, 1, 2, -1, 2, 1, -2, -1]
  + dy: [1, 2, 1, 2, -1, -2, -1, -2]
* **Index 3:**
  + dx: [2, -1, 1, 2, -2, 1, -1, -2]
  + dy: [-1, 2, 2, 1, 1, -2, -2, -1]
* **Index 4:**
  + dx: [-1, -2, -2, -1, 1, 2, 2, 1]
  + dy: [-2, -1, 1, 2, 2, 1, -1, -2]

For each move set, separate log files were created, recording the details of the search process and solutions. Each log includes:

* **Search method used (BFS, DFS, DFS with h1b, DFS with h2).**
* **Time limit and board size.**
* **Solution path (if found).**
* **Total nodes expanded and created.**
* **Time spent.**
* **Failure messages (e.g., "Out of Memory", "Timeout").**

The MaxSizeTest.log file contains a record of the maximum board size successfully solved using the h2 heuristic without encountering an "Out of Memory" error. The largest board size achieved was , and the solution path for this size is also preserved in the log. **(361)**

A screen shot of a computer screen

Description automatically generated

**Figure 1:** Log output for maximum solvable board size (361x361) using the H2 heuristic. The solution path starts from position (1,1) and successfully visits all squares without encountering an 'Out of Memory' error. The move set used is X: [-2, -1, 2, 2, -1, -2, -1, 2] and Y: [1, 2, 1, 2, -1, -2, -2, -1].

A screen shot of a computer

Description automatically generated

**Figure 2:** Sample log output from program execution, showing board size, starting position, move set indices, algorithm choice, total nodes expanded, total nodes created, and time spent during execution. This log is a representative section.