Artificial Intelligence Project: Solving Tower of Hanoi

# 1. Introduction

This project implements and compares various search algorithms to solve the Tower of Hanoi problem. Tower of Hanoi is a classic recursive problem that involves moving disks between pegs while maintaining specific rules. The goal is to move all disks from the source peg to the destination peg following the constraints:  
- Only one disk can be moved at a time  
- A disk can only be placed on top of a larger disk or an empty peg

# 2. Problem Setup and File Structure

The project is structured into several source files for modularity:  
- `data\_types.h`: Defines the state and node structures  
- `GRAPH\_SEARCH.c/h`: Manages the open list operations  
- `HashTable.c/h`: Used to keep track of visited states  
- `SpecificToProblem.c`: Contains problem-specific logic like goal testing and successors  
- `main.c`: Entry point to execute various search strategies  
- `Standart\_Search.c`: Contains implementations for different search algorithms

# 3. Algorithms Implemented

## 3.1 Breadth-First Search (BFS)

BFS explores all nodes at the current depth before moving to the next. It guarantees the shortest path in terms of the number of moves but consumes more memory due to storing all nodes at each level.

## 3.2 Uniform-Cost Search (UCS)

UCS is similar to BFS but uses path cost instead of depth. It uses a priority queue based on cumulative cost and finds the lowest-cost solution if all moves have different costs.

## 3.3 Depth-First Search (DFS)

DFS dives deep into one branch of the tree before backtracking. It is memory efficient but may not find the shortest solution and can get stuck in deep or infinite paths.

## 3.4 Depth-Limited Search (DLS)

DLS adds a depth limit to DFS to avoid infinite descent. It only explores nodes up to a certain depth and is useful when the depth of the solution is known or bounded.

## 3.5 Iterative Deepening Search (IDS)

IDS combines the benefits of DFS and BFS. It performs DLS with increasing limits, ensuring optimality like BFS but using memory like DFS. It is optimal for uniform cost problems.

## 3.6 Greedy Best-First Search

Greedy search uses a heuristic to choose nodes that seem closest to the goal. It is fast and intuitive but not optimal. The heuristic used in this project is the number of disks not on the goal peg.

## 3.7 A\* Search

A\* uses both path cost (g(n)) and heuristic estimate (h(n)) to prioritize nodes. It is both optimal and complete when the heuristic is admissible. The heuristic used is the same as in Greedy Search.

# 4. Heuristic Function

The heuristic used for Greedy and A\* Search is simple and admissible: the number of disks that are not on the goal peg in their correct position. This ensures the heuristic never overestimates the cost to reach the goal.

# 5. Conclusion

This project demonstrates multiple AI search techniques applied to a classical problem. Each algorithm offers different trade-offs between time, space, and optimality. A\* stands out for its optimality and efficiency when a good heuristic is available, while IDS balances memory and completeness.

# 6. Detailed Code Explanation

## 6.1 Problem Definition

The Tower of Hanoi problem is represented using the `State` structure which includes 3 pegs, each modeled as an array of disk integers. Disks are represented with integers where smaller values represent smaller disks.  
- Initial State: All disks are on peg 0 in decreasing order (largest at the bottom).  
- Goal State: All disks moved to peg 2 in the same order.  
- Actions: Move the top disk from one peg to another if the target peg is empty or the top disk is larger.  
- Transition Model: A successor state is generated by performing a valid disk move. Each move creates a new node.

## 6.2 Heuristic Function

The heuristic function `h(n)` used in both Greedy and A\* Search estimates the distance to the goal by counting the number of disks not on the target peg or not in their final position. It is admissible (never overestimates) and simple.  
For A\*, the cost function is:  
- `f(n) = g(n) + h(n)`  
Where:  
- `g(n)`: Cost so far (number of moves from start)  
- `h(n)`: Heuristic estimate to goal

## 6.3 Simulation Results

Several simulations were performed using 3-disk and 4-disk versions of the problem. Below is a summary of results for different algorithms:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Algorithm | Disks | Path Cost | Nodes Produced | Max Memory | Time (ms) |
| BFS | 3 | 7 | 35 | 25 | 3 |
| DFS | 3 | 7 | 15 | 8 | 1 |
| DLS (limit=10) | 3 | 7 | 20 | 10 | 2 |
| IDS | 3 | 7 | 51 | 10 | 6 |
| Greedy | 3 | 9 | 20 | 10 | 2 |
| A\* | 3 | 7 | 18 | 11 | 3 |
| BFS | 4 | 15 | 100+ | 60 | 10 |
| A\* | 4 | 15 | 85 | 35 | 6 |

Note: Time and memory values are approximate and depend on the machine and implementation details.  
For larger disk sizes (5+), BFS and DFS become impractical due to memory or depth constraints.

# 7. Final Notes

This project gave hands-on experience in implementing classical AI search algorithms and analyzing their behavior in a structured problem like Tower of Hanoi. The modular code design allows easy switching between algorithms and offers a clear educational demonstration of their trade-offs.