Analyzing the Towers of Hanoi Problem

Jonathan Lee

November 2018

1 Project Description

This paper will discuss the various algorithms used in solving the Towers of Hanoi problem.

The Towers of Hanoi is a problem that involves some predetermined number of discs and three rods. The problem starts with the discs, each having a different size, arranged from biggest to smallest stacked on the first rod. The goal of this problem is to move the discs such that one ends up with all discs on the last rod, stacked from biggest to smallest with the bottom disc being the biggest, in as few turns as possible. The goal state is immediately known because it will mirror the initial the state. A legal move consists of moving one disc from a spoke to another. Also, a larger disc cannot be placed on top of a smaller disc at any point in this problem.

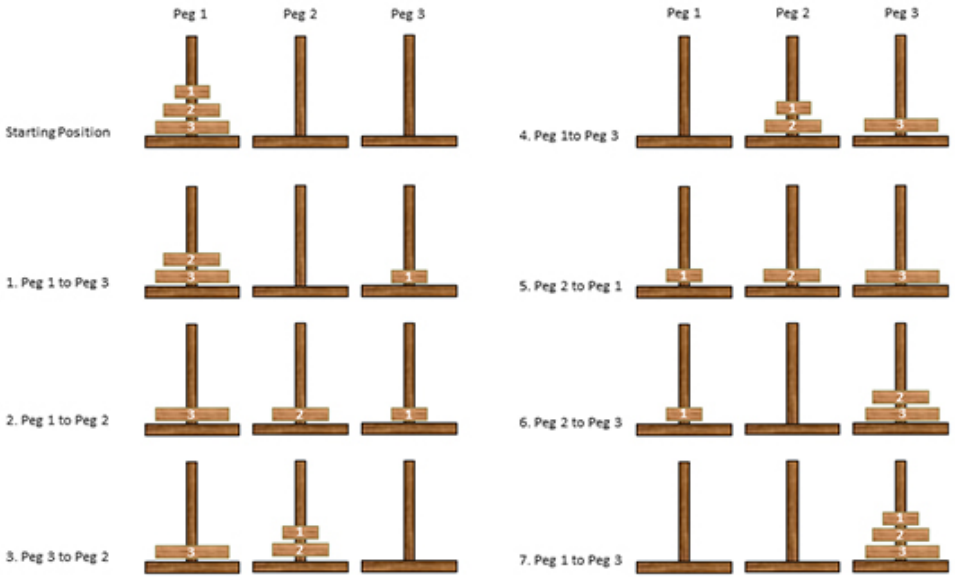

Figure 1: Towers solution when n = 3

For The Towers of Hanoi problem, we will analyze its performance using different versions of graph search. Graph search is ideal for the Towers problem, as opposed to tree search, because it guarantees no cycles in this problem. This means that it will not explore a state that has already been visited. This is extremely important because it eliminates redundant states and ensures a solution. In graph search, nodes are placed into the frontier. There is no need to maintain an explored set, a defining trait of tree search. Tree search is not recommended for the Towers problem because there is the possibility of infinitely looping through an already explored state.

We will analyze its performance when using breadth-first search (BFS), depth-first search (DFS), and iterative-deepening depth-first search (IDDFS). BFS is an uninformed search technique that starts at the root node and explores all nodes in a given depth before traversing to a deeper depth. This algorithm seeks to return an optimal solution, making it an ideal candidate for the Towers problem. DFS is another uninformed search search technique that traverses through each depth,

starting at the root, without needing to explore every single node at any given depth. When using a graph DFS on the Towers problem, it guarantees a solution but not necessarily the optimal solution. This technique may be more attractive to those interested in a solution without caring if the solution contains the fewest number of moves. IDDFS is a modified version of DFS that iteratively increases the max depth limit on search.

2 Experiments

The metrics used in these experiments include total number of nodes generated, maximum frontier count and depth. The total node count will give us a good sense of the time complexity for each algorithm. The frontier count serves as an indicator for space complexity. The depth will also provides useful information in determining the time complexity. For the Towers problem, the optimal number of moves is given by the equation $2^n - 1$, where n represents the number of discs. Therefore, its run-time is $O(2^n)$. The complexity of the problem takes increases exponentially as the number of discs increase. For each algorithm (graph BFS, graph DFS, and graph IDDFS) run five tests where the number of discs = 3, 6, 9, 10, and 11. We are also interested in the largest problem size for each algorithm. We will restrict our algorithms to cease searching if a problem takes longer than one hour to complete or if it runs out of memory, making a note if either of these events occur.

Experiments:

1. Graph BFS: total node count, maximum frontier and depth when n = 3, 6, 9, 10, 11
2. Graph DFS: total node count, maximum frontier and depth when n = 3, 6, 9, 10, 11
3. Graph IDDFS: total node count, maximum frontier and depth when n = 3, 6, 9, 10, 11
4. The largest problem size (number of discs) for Graph BFS, DFS, and IDDFS.