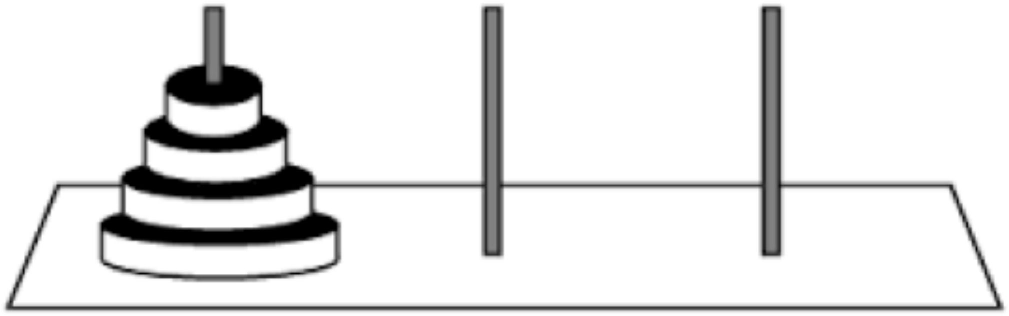
Assignment 1: Tower of Hanoi
Due: September 29th by beginning of class

The Game

The Tower of Hanoi is a puzzle that provides us with the opportunity to look at some of the basic issues in problem solving. The game consists of three pegs and an arbitrary number of disks of descending size. In the initial state, the disks are stacked from largest to smallest on the leftmost peg. The goal is to get all of the disks on the rightmost peg.

It is a classic exponential problem in that the minimum number of moves to solve the problem is $2^n - 1$, where n is the number of disks.



In order to play this game, you have to obey the following simple rules:

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
3. No disk may be placed on top of a smaller disk.

The Assignment

The assignment is to build a system that takes two numbers (the number of pegs and the number of starting disks) and produces a solution to the problem.

You can work in teams of up to three people. You are going to hand in the homework in the form of two documents:

- The code itself
- A trace of the code running that shows its progress and path and when it is finished, displays the solution it developed.

I am working with the graders to figure out how we are going to turn in work.

Issues

In order to do this you will need to think about the following:

- How to represent the state of the game? That is, how to represent the pegs and the disks that are associated with each of them?
- How to test for when you are reached the end state?
- How do you implement the one action you are allowed to make?
- How do you make sure that you avoid returning to a state that you have been in previously and thus avoid getting caught in a recurring loop?

In order to avoid repeating states, you are going to have to implement some sort of memory of past states that you have been in. Also, given that you will sometimes find yourself in a state in which there may be no legal move from a given state that does not result in a repeated state, you are going to have to be able to back track a state you were in earlier and take a different action. When you do so, you still want to maintain information about the states that you passed through.

There is a set of issues that I want you to be thinking about as you address this problem.

- How much of the world do you have to represent in order to capture the state of the game?
- How do you test for completion?
- How to you implement the constraints on each move (the rules of the game)?
- Is there a way to test your progress? That is, is there an evaluation function that gives you a better score, as you get closer to the solution?

Search

This is a classic search problem. In order to solve it, you can take one of three approaches:

- *Depth first search* in which you commit to a path and then back up if you fail. Easy to implement and only requires that you can back track and hold onto past states you have encountered.
- *Breadth first search* in which you expand the space of possible solutions by adding a step to each of the initial possibilities that you have taken. This

means that you maintain and expand upon all possible solutions to length N until you find the right one.

- *Best first search*, which is essentially depth first, but you always try to take an action that gets you closer to the solution. This requires an incremental evaluation function.