P4. Tower of Hanoi

[[홍원표 21701065]]

1. 문제 분석

P4 is similar to the original Tower of Hanoi problem, but the **destination rod is not set**. Also, the disks are distributed in **arbitrary locations**. (However, the **pre-condition** that **only smaller disks** can be placed **on top of larger disks** still exists.) There are **N number of Disks,** and the destination rod is set as K. There are only **3 rods** available. The distributed disks are given as inputs. Output should be the **minimum number of disk moves** to stack all disks on the destination rod.

1. 문제 풀이

The key point of this problem is to **eliminate all disks (FROM THE BIGGEST DISK)** until there is none left. The way to eliminate disk is check whether the **biggest disk is on the destination rod**. If **it's on** the destination rod, there's simply **NO NEED to MOVE the disk**, so we can solve the **problem as if the disk doesn't exist**.

If the biggest disk is **NOT on the destination rod**, the rods must go through this **particular condition**: if the **biggest disk number *i* is on rod 1**, **every other disks** should be **stacked on the rod 2**, and rod 3, which is the **destination rod, must be empty. Only this way**, the **disk *i*** can be **placed on the destination rod**. And from then on, it requires **2^(N-1) disk-moves** to solve the problem.

Also, we must set the **new destination** for the **decreased** number of disks. If the **disk *i*** was **on the destination rod**, the new destination rod for disk i-1 should **ALSO be the same one as the previous**. This simply means that **they're well stacked**, and we **would not need to worry** about them. If the **disk *i* was NOT on the destination rod**, we need to **check WHERE** the disk i was, and the **new destination rod should be the left over rod**. Taking the example from above, if the **previous destination rod was rod 3**, and **disk *i* was on rod 1**, the **new destination rod should be rod 2**. This way, we can make the **particular condition** faster.

Repeating this until all disks are gone will give us certain sum of powers of 2. This repeats N times.

1. 문제 풀이 분석

For N number of disks. Since I did not use math.h, I had to solve the power of 2 recursively. 2^N takes N-1 recursions. Also, in order to solve this problem, we must know where disks are in stacked order. By using queue, we just need N spaces. Thus, Time Complexity: O(N * (N-1) / 2) = **O(N^2).** Space Complexity: **O(N)**

1. Application

This can be applied to any algorithm to solve **combinatorial puzzles**, as long as the puzzle requires some "required" steps to follow such as the **particular condition** stated above. **Rubik's Cube** can be counted as examples.