Recursion

1.1 Reductions

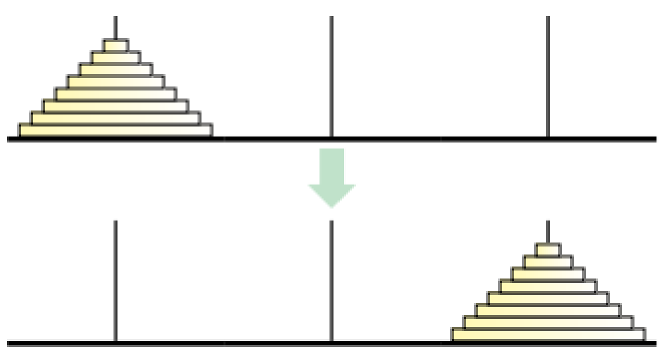
- Reduce one problem X to another problem Y
- Solving for Y is a black box for X, meaning X is independent of Y.
- We can assume that the black box solves Y correctly.

1.2 Simplify and Delegate

- Recursion is a type of reduction.
- If the problem can be solved directly, then solve it directly.
- If not, then reduce the problem to a simpler version of the same problem.
- In other words, it is either simple enough to be solved in one step, or it is complex and you need to simplify the problem and "give it to someone else".
  - The book calls "someone else" the *Recursion Fairy*.
  - AKA the *Induction Hypothesis*.
- Because we are simplifying/reducing the problem with recursion, we need a **base case** that can be solved without any further reduction.
  - We cannot have an infinite sequence.

1.3 Tower of Hanoi

- Tower of Hanoi Problem: How can we move a tower of n disks from one peg to another, using a third spare peg as an occasional placeholder, without ever placing a disk on top of a smaller disk?



- Strategy:
  - We want to move the entire tower from one peg to another.
    - We can't move the bottom disk because the smaller disks are on top.
  - We need to move $n-1$ disks off from the n-th disk to a placeholder disk.

    - Then, we move the n-th disk to its destination.
    - Finally, we move the $n-1$ disks from the placeholder to the destination.
- Our strategy allowed us to reduce the problem from n disks to $n-1$ disks.
    - We solved it for n, but we don't know how to solve it for $n-1$. This is okay because how $n-1$ gets solved is a black box for us.
    - We can hand it off the problem to the *Recursion Fairy* and we don't have to worry about it anymore.
- When $n=0$, the problem is *trivial* (omg leiss reference) and we don't have to reduce the problem anymore.
- We **shouldn't** unearth the recursive sequence and try to see how it all works out. Our **only** task is to reduce the problem to a simpler version of it, or to solve it directly if it is possible.
- In terms of Induction, our base case is $n=0$, and for any $n \geq 1$, the Inductive Hypothesis implies that our algorithm correctly moves the top $n-1$ disks.
- Psuedocode for the Recursive Hanoi Algorithm:

```
Hanoi(n, src, dst, tmp):
if n>0
        Hanoi(n-1, src, tmp, dst)
        move disk n from src to dst
        Hanoi(n-1, tmp, dst, src)
```

- If $n>0$, then first we move $n-1$ disks from the starting(src) peg to the placeholder(tmp) peg, using the destination(dst) peg as a temporary peg. Then, we move the n-th disk from our starting peg to our destination peg with a single, simple move. Finally, we move back all the $n-1$ disks from the temporary peg to the destination peg by calling `Hanoi()` recursively.
- Let $T(n)$ denote the amount of moves it takes to transfer n disks. This is equivalent to the running time of our algorithm.
    - Our vacuous base case implies that $T(0)=0$, and the more general recursive algorithm implies that $T(n)=2T(n-1)+1$ for any $n \geq 1$.
    - We can easily guess that the general equation for T is $T(n)=2^n-1$ by writing out the first few values of n.
        - Prove this is true with a simple induction proof.
    - If we had $n=64$ disk, moving a tower of 64 disks would take $2^{64}-1=18,446,744,073,709,551,615$ moves. At a single move per second, it would take around 585 billion years to complete the algorithm.