Question 8.6 from CTCI: Towers of Hanoi

Benjamin De Brasi

September 27, 2017

Question

Towers of Hanoi: In the classic problem of the Towers of Hanoi, you have 3 towers and N disks of different sizes which can slide onto any tower. The puzzle starts with disks sorted in ascending order of size from top to bottom (Le., each disk sits on top of an even larger one) . You have the following constraints:

1. Only one disk can be moved at a time.
2. A disk is slid off the top of one tower onto another tower.
3. A disk cannot be placed on top of a smaller disk.

Write a program to move the disks from the first tower to the last using Stacks.

Explanation and Algorithm

This problem sounds like a good candidate for the Base Case and Build approach.

Let's start with the smallest possible example : n = 1.

Case n = 1. Can we move Disk 1 from Tower 1 to Tower 3? Yes.

1. We simply move Disk 1 from Tower 1 to Tower 3.

Case n = 2. Can we move Disk 1 and Disk 2 from Tower 1 to Tower 3? Yes.

1. Move Disk 1 from Tower 1 to Tower 2
2. Move Disk 2 from Tower 1 to Tower 3
3. Move Disk 1 from Tower 2 to Tower 3

Note how in the above steps, Tower 2 acts as a buffer, holding a disk while we move other disks to Tower 3.

Case n = 3. Can we move Disk 1,2, and 3 from Tower 1 to Tower 3? Yes.

1. We know we can move the top two disks from one tower to another (as shown earlier), so let's assume we've already done that. But instead, let's move them to Tower 2.
2. Move Disk 3 to Tower 3.
3. Move Disk 1 and Disk 2 to Tower 3. We already know how to do this-just repeat what we did in Step 1.

Case n = 4. Can we move Disk 1, 2,3 and 4 from Tower 1 to Tower 3? Yes.

1. Move Disks 1,2, and 3 to Tower 2. We know how to do that from the earlier examples.
2. Move Disk 4 to Tower 3.
3. Move Disks 1,2 and 3 back to Tower 3.

Remember that the labels ofTower 2 and Tower 3 aren't important. They're equivalent towers. So, moving disks to Tower 3 with Tower 2 serving as a buffer is equivalent to moving disks to Tower 2 with Tower 3 serving as a buffer.

Hints

1. Try the Base Case and Build approach.
2. You can easily move the smallest disk from one tower to another. It 's also pretty easy to move the smallest two disks from one tower to another. Can you move the smallest three disks?
3. Think about moving the smallest disk from tower X=0 to tower Y=2 using tower Z=l as a temporary holding spot as having a solution for f (1, X=0, Y=2, Z=l). Moving the smallest two disks is f (2, X=0, Y=2, Z=l) . Given that you have a solution for f(l, X=8, Y=2, Z=l) and f(2, X=8, Y=2, Z=l) , can you solve f(3, X=0, Y=2, Z=l)?
4. Observe that it doesn't really matter which tower is the source, destination, or buffer. You can do f(3, X=0, Y=2, Z=l) by first doingf(2, X=0, Y=l, Z=2) (moving two disks from tower 0 to tower 1, using tower 2 as a buffer), then moving disk 3 from tower 0 to tower 2, then doing f(2, X=l, Y=2, Z=0) (moving two disks from tower 1 to tower 2, using tower 0 as a buffer) . How does this process repeat?

5. If you 're having trouble with recursion, then try trusting the recursive process more. Once you've figured out how to move the top two disks from tower 0 to tower 2, trust that you have this working. When you need to move three disks, trust that you can move two disks from one tower to another. Now, two disks have been moved. What do you do about the third?

Code

```
\*First code for a node and stack *\

public class Node{
   private int data;
   private Node next;

   public Node(){
      data = 0;
      next = null;
   }

   public Node(int a){
      data = a;
      next = null;
   }
}

public class Stack{
   private Node head;
   private int length;

   public Stack(){
      length = 0;
      head = null;
   }

   public push(Node a){
      length++;
      if(head == null)
         head = a;
      else{
         Node tmp = head;
         while(tmp.next != null)
            tmp = tmp.next;

         tmp.next = a;
      }
```

```
   }

   public Node pop(){
      length--;
      tmp = head;
      head = head.next;
      return tmp;
   }

   public int peek(){
      return head.data;
   }

   public boolean isEmpty(){
      if(head == null)
         return false;
      else
         return true;
   }

   public int size(){
      return length;
   }
```

```
\*Code that solves Towers of Hanoi *\

public void Hanoi(int n, Stack source, Stack buffer, Stack target){

   if(n > 0)
      return;

   Hanoi(n-1, source, target,buffer);
   target.push(source.pop());
   Hanoi(n-1, buffer, source, target);

}
```

Run Time analysis

$$O(2^n)$$

Sources

Question, answer and other material taken from Cracking the Coding Interview 6th edition by Gayle Laakmann McDowell.