ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

HS 2020   Prof. R. Wattenhofer
                                                                    Devillez Henri

Computational Thinking
Exercise 1 Solutions

1   Towers of Hanoi

Let us first look at an easy problem: we want to move a tower of size 1 (aka only the smallest   → notebook
disk) to a final rod given a helping rod. The smallest disk we can always move on any rod (it can
never be placed on a smaller one).

Now let us look at the case where we have a tower of more than one disk, let's say n disks. The
largest disk has to go to the very bottom of the final tower, otherwise we would place it later on a
smaller disk. To do this, we need to move the entire tower to the helping rod (using the final rod
as auxiliary structure). Then, the largest disk can be moved to the free final rod. Now we need
to move the remaining tower from the helping rod to the final rod. This tower has $n - 1$ disks,
so we can use recursion and do the same trick again. Now, the former helping row is our starting
rod and the starting rod is our new helping rod.

2   Nim Game

Part 1

In principle Nim is a game with recursive nature. Let's think we have a function `is_won(n)` that   → notebook
tells us if the player to move can win the game with n sticks. For example in our game `nim(1)`
would return `True` since the starting player takes the last stick and the other player cannot move
and loses. Similarly, `nim(3)` is won by taking all three sticks but `nim(2)` is always lost. The only
options are taking one stick or halving the stack, which leaves the other player to play on one
stick left. We already know that the starting player wins on one stick. Thus playing on two sticks
always looses against best play.

This gives us the recursive idea to play Nim: we check all available options given to us: removing
1, removing 3, halving an even stack. If *any* option makes the then starting player loose we can
win by creating this situation, thus returning True. If *all* options make the new starting player
win, we will loose and return false.

Moreover, we can use dynamic programming to solve this problem faster. We store all intermediate
results `nim(n)` that we already computed. Now we only need to look at those results instead
of invoking recursive calls. The solution implementation shows both the dynamic programming
version and the recursive variant. If you want, play with increasing values of n to see the difference
in speed!

Part 2

The idea for this game is very similar to the first variant. We check every possible move in any
of the three piles and if the find at least one move that makes the new starting player loose, we
return `True`, otherwise we return `False`.