ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

HS 2023

Prof. R. Wattenhofer
Robin Fritsch

Computational Thinking
Sample Solutions to Exercise 1

1 Towers of Hanoi

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

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 (the largest disk that has already been placed on the bottom of the final rod will not influence the rest of the game in any way!). For moving this tower of $n-1$ disks, the former helping rod will act as our new starting rod, and the former starting rod will act as our new helping rod.

2 Nim Game

a) In principle Nim is a game with recursive nature. Let us assume we have a function `is_won(n)` that tells us if the next 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, for `nim(3)` the starting player wins by taking all three sticks. → notebook

However, `nim(2)` is always lost: the only options are taking one stick or halving the stack, both of which leave the other player to play with one stick left. We already know that the starting player wins with one stick. Hence, the starting player always loses with two sticks (against optimal 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 lose, we can win by creating this situation, thus returning True. If *all* options make the new starting player win, we will lose 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!

b) 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 we find at least one move that makes the new starting player lose, we return `True`, otherwise we return `False`.