Theory of Computer Science

Ex.4 Algorithms:
The iterative solver for Hanoi Tower in pseudocode

Artur Oleksiński
27.04.2021

1. Introduction

The task provided to us by the tutor was simple, to implement
the iterative Hanoi Tower. During the initial analysis of the problem, I have started to ask questions about the pattern of the Hanoi tower problem.
It led to a deeper analysis of the problem, and to create the report.
I am fully aware that marks will be given only basing on the pseudocode, all additional work was done in my spare time as an interest.

2. Analysis of the problem

I. Original Rules of the puzzle

- Only one disk may be moved at a time.
- Each move consists of taking the most upper disk from one of the stack and placing it at the top of another stack or empty pole.
- The disks cannot be placed on top of smaller disks.

II. Research

In the beginning, I have searched for useful pieces of information about Hanoi tower puzzle. During the search, I've observed that the majority of implementations and approaches were done using the recursive approach with a limited description of how actually Hanoi tower is being solved.
That's the main reason for the creation of this paper.

III. Main analysis

The analysis of the problem started with disassembling the recursive implementation of The Hanoi tower puzzle provided to us by the tutor.

```
procedure Hanoi(n: integer, A: stack, B: stack, C: stack)
# first stack argument is the source pole
# second - the helper pole
# third - the destination pole
begin
 if n > 0 then
 begin
   Hanoi(n-1, A, C, B)    # 1)
   C.push(A.pop())        # 2)
   Hanoi(n-1),B, A, C)    # 3)
 end
end
```

Figure 1. Pseudocode provided by the tutor during the Theory of Computer Science Exercises

The example shows the notes of an expanded recursive algorithm for 3-disks and 3-poles problem.

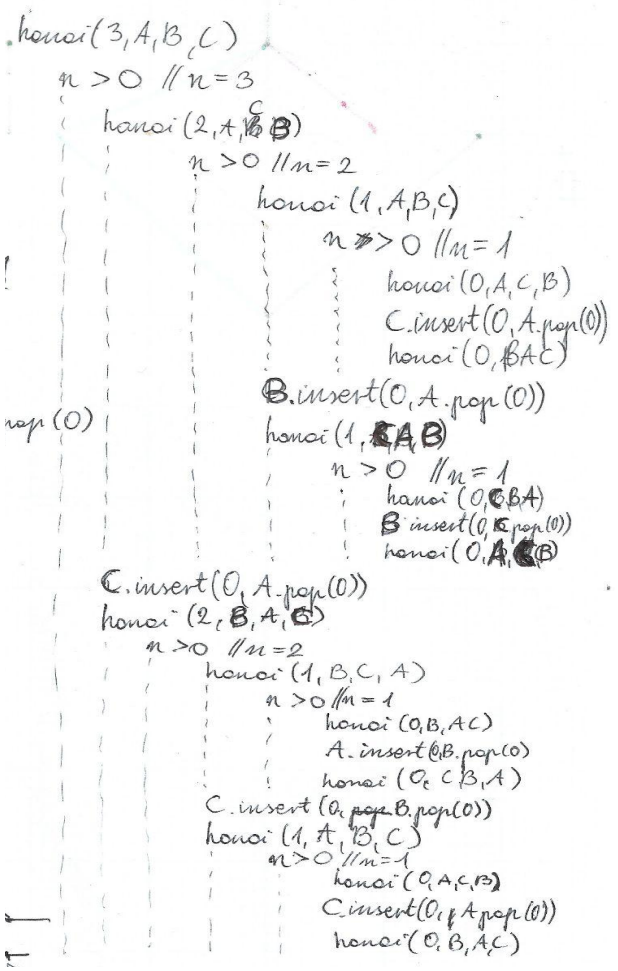

Figure 2. Part of the development notes showing the expanded recursive algorithm for 3-disks and 3-poles.

In the beginning, one looking at Figure 2 could say that the algorithm is highly randomized in choosing the destination pole but actually, it is deeply connected with the nature of the puzzle.

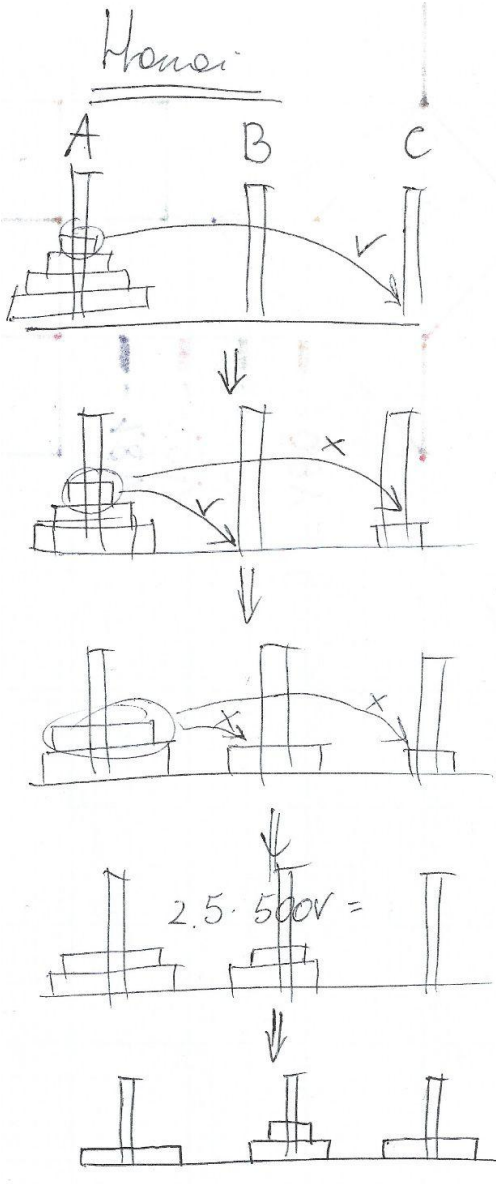

Figure 3.*

In the next step, I wanted to observe the procedure from a bot-like perspective.
During the analysis of every move and how decisions are made, I have developed two additional constraints to the original Rules of The Hanoi tower.

The first additional rule:
One can't move a disk that was moved in the previous turn.

The first additional rule assures that the bot will not be stuck in the infinite loop or even back already made move. During the analysis this rule became obvious
because we want to move forward with solving the puzzle and not moving back.

With this set of rules, the bot can easily do most of the moves, but there is still the problem of the first move and the problem of two available moves.
On the development notes and first version of the prototyping code, these are written as separate rules but later in the further analysis,
these rules are merged.

*Figure 3. Part of the development notes showing the process of solving the puzzle.

IV. The Fractal series

In this part the main problems are:

- The problem of the first move
- The problem of two available moves.

Before I started developing the rules for the bot I observed that the choice of the first pole is constant for even and an odd number of poles separately.
If the number of poles is even, one should choose
the available place from the left.
If the number is odd, the choice is to put the disk on the right.

In development notes, it is rule number 5* (where stars represent the additional rules for Hanoi tower).

Another problem is the moment when the bot has two available spaces to put the disk. At the start, I have tried to map choosing of the pole to the number of steps at which the problem appears. This was the wrong approach and I have started to search for better alternatives.

During the analysis decomposition of recursive implementation
of the Hanoi tower puzzle, the important thing to notice was the depth of the recursion which strictly corresponds to the nature of The Hanoi tower puzzle.

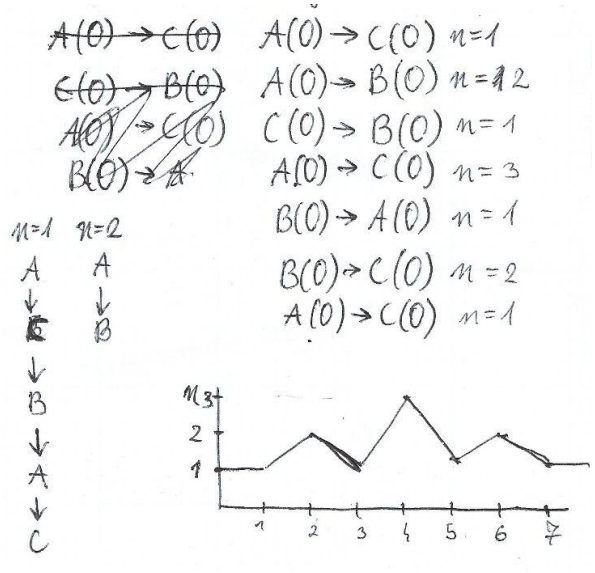

Figure 4. Piece of development notes about fractal series for n = 3.

A list of all the moves with their depths in recursion creates a particular pattern which can be written as:

```
# Python code
def fractal_series(n: int) -> tuple:
|   series = [1]
|   for i in range(2, n+1):
|   |   series = [series[:], [i], series[:]]
|   |   series = list(chain.from_iterable(series))
|   return tuple(series)
```

Where chain.from_iterables(series) eliminates multi dimensionalism of the list.
Eg. [[1],[1]] ⇒ [1,1]

The fractal series is built with the sequence n-1, n, and n-1 and it looks like this:
n = 1
array = 1

n=2
array = 1, 2, 1

n = 3
array = 1, 2, 1, 3, 1, 2, 1

n = 4
array = 1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, 1

and so on.

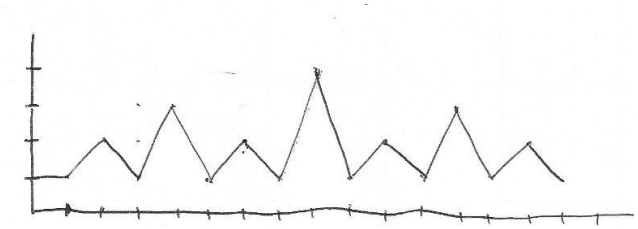

Figure 5. Fractal series for n = 4

Using fractal series we obtain information
what move should be made if there are two free places to put the disk.

This creates the last additional rule:

For an even number of disks - if fractal series at i is even,
put the disk to the most right pole, if it is odd then put it on the left.

For an odd number of disks - if fractal series at i is even,
put the disk to the most left pole, if it is odd then put it on the right.

This provides to the bot all necessary information and can be used for n number of disks,
always finding the most optimal way of solving the puzzle.
On the next pages is a handwritten solution from development notes
of 4-disks 3-poles Hanoi tower.

Prototype code and all the notes can be found on:
https://github.com/ArturOle/Hanoi_iterative

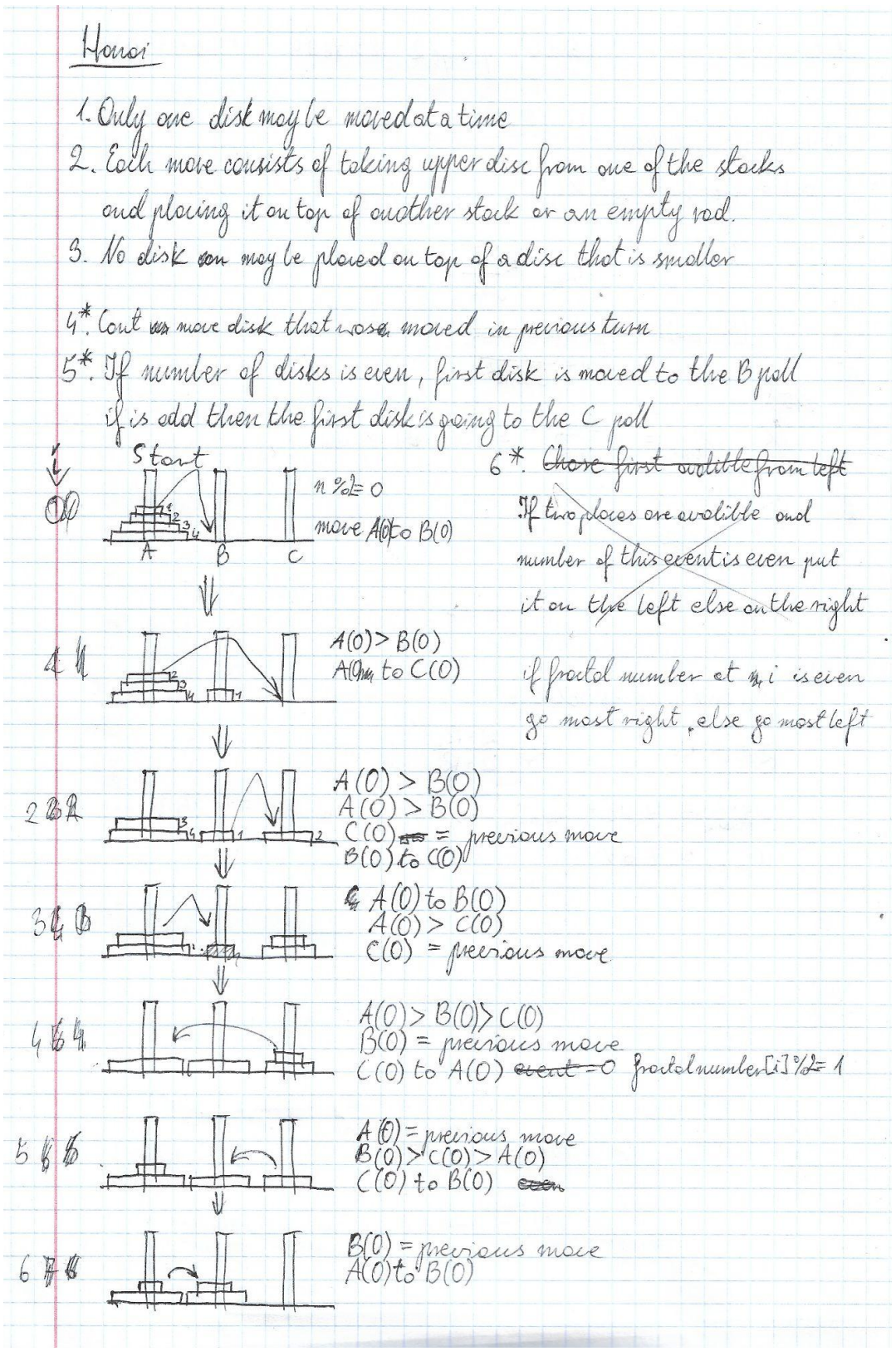

Figure 6. The first page of handwritten solution from development notes.

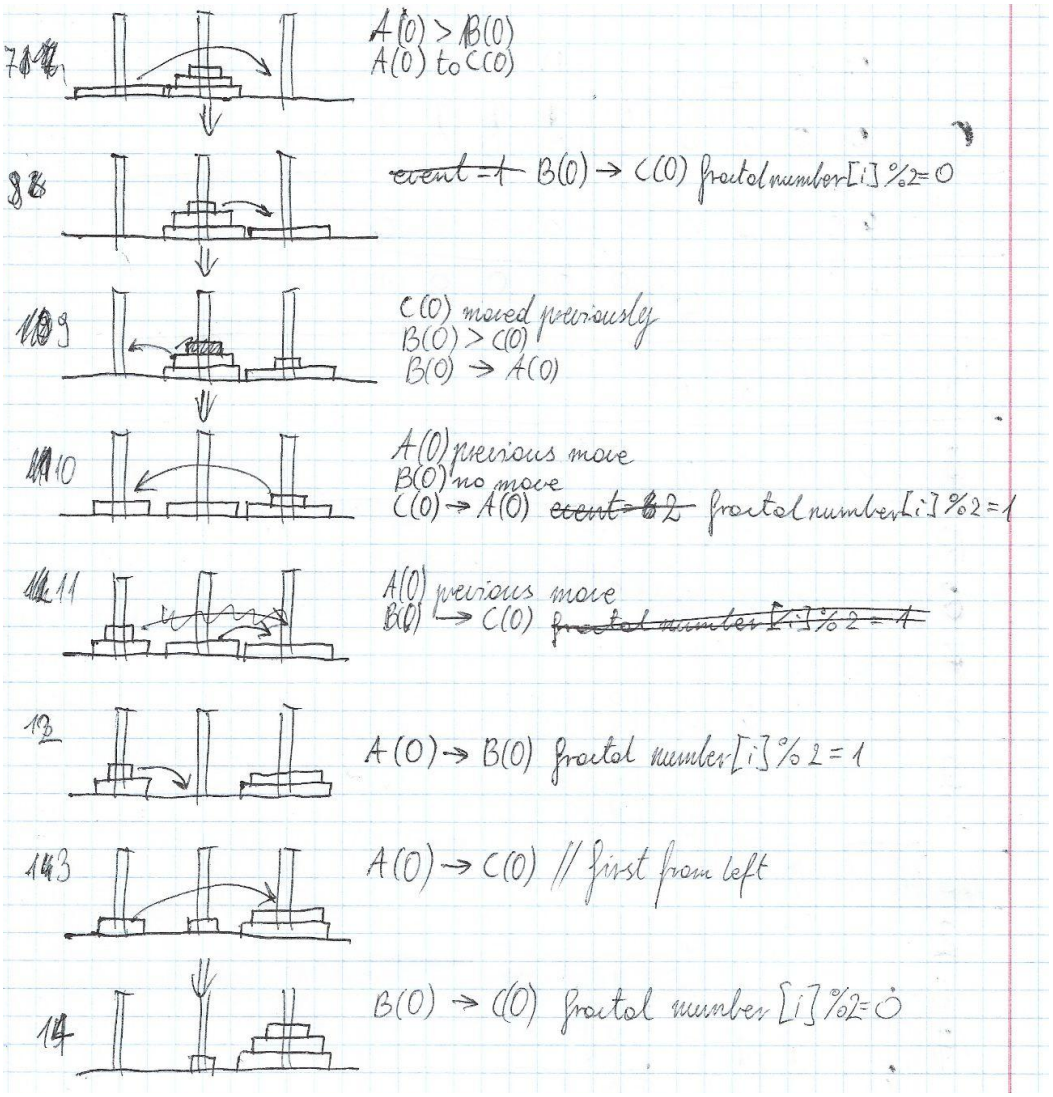

Figure 7. The first page of handwritten solution from development notes.

3. Pseudocode

I am aware that it is not the most efficient implementation, there are many flaws but it works in the minimal possible number of steps. 2^N -1

Artur Oleksiński

Task 4: Pseudocode for iterative Hanoi tower solving.

```
function Hanoi (N: integer, A: stack, B: stack, C: stack)
var fractal_series : array
var A,B,C : stack, stack, stack
var previous : integer
var step : integer
var polls : array of stack [1:3]
var even : integer
begin
    even := N % 2
    polls := [A,B,C]
    previous := 0
    step := 0 ; fractal_series := fractalseries[N];
    while step < pow(2,N)-1 do:
    begin
        i=0
        for poll in polls[1:3] do:
            if poll[0] == previous:
                pass
            else if poll[0] < polls[(i+1)%3][0] and poll[0] < polls[(i+2)%3][0]:
                if (fractal_series[step]-even)%2:
                    polls[(i+1)%3].push(poll.pop())
                    previous = polls[(i+1)%3][0]
                    break
                else:
                    polls[(i+2)%3].push(poll.pop())
                    previous = polls[(i+2)%3][0]
                    break
            else if poll[0] < polls[(i+1)%3][0]:
                polls[(i+1)%3].push(poll.pop())
                previous = polls[(i+1)%3][0]
                break
            else if poll[0] < polls[(i+2)%3][0]:
                polls[(i+2)%3].push(poll.pop())
                previous = polls[(i+2)%3][0]
                break
            i+=1
        step+=1
    end
    return polls
end
```