Algorithms Design Project No. 13

Bleoancă Diana Iulia

June 4, 2018

University: Automatics and Computers, Craiova
First Year
Group: C.E 1.1A
Coordinating teachers:
Costin Bădică and Alex Becheru

1 Introduction

The Red Emperor has ordered for the wedding of his son a cake win N stories. His cooks have set up the cake on a silver plate. However, the emperor wants the cake to be place on a golden plate. Unfortunately, the cooks have no idea how to move the cake from one plate to another, due to the immensity of the cake. However, they have figured out that only one additional plate can be used, a bronze plate. This application solve the cooks dilemma and reconstruct the cake exactly as it was in the begging, but on the golden plate.

This problem can be solved with Tower of Hanoi problem.

2 Problem statements

The Tower of Hanoi is a mathematical game or puzzle. It consists of three rods and a number of disks of different sizes, which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape.

The objective of this puzzle is to move the entire cake to another plate, obeying the following simple rules:

1. Only one story can be moved at a time.
2. Each move consists of taking the upper story from one of the stacks and placing it on top of another stack or on an empty plate.
3. No story may be placed on top of a smaller story.

We need to display every move.

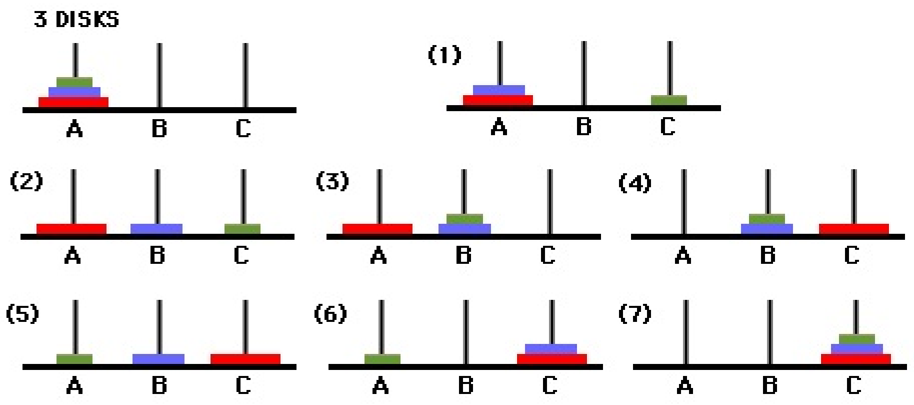


Figure 1: Steps example for the algorithm

2.1 Example IO

Example input : 5

Example output :

```
Move floor 1 from silver plate to golden plate
Move floor 2 from silver plate to bronze plate
Move floor 1 from golden plate to bronze plate
Move floor 3 from silver plate to golden plate
Move floor 1 from bronze plate to silver plate
Move floor 2 from bronze plate to golden plate
Move floor 1 from silver plate to golden plate
Move floor 4 from silver plate to bronze plate
Move floor 1 from golden plate to bronze plate
Move floor 2 from golden plate to silver plate
Move floor 1 from bronze plate to silver plate
Move floor 3 from golden plate to bronze plate
Move floor 1 from silver plate to golden plate
Move floor 2 from silver plate to bronze plate
Move floor 1 from golden plate to bronze plate
Move floor 5 from silver plate to golden plate
Move floor 1 from bronze plate to silver plate
Move floor 2 from bronze plate to golden plate
Move floor 1 from silver plate to golden plate
Move floor 3 from bronze plate to silver plate
Move floor 1 from golden plate to bronze plate
Move floor 2 from golden plate to silver plate
Move floor 1 from bronze plate to silver plate
Move floor 4 from bronze plate to golden plate
Move floor 1 from silver plate to golden plate
Move floor 2 from silver plate to bronze plate
Move floor 1 from golden plate to bronze plate
Move floor 3 from silver plate to golden plate
Move floor 1 from bronze plate to silver plate
Move floor 2 from bronze plate to golden plate
Move floor 1 from silver plate to golden plate
```

3 Pseudocode Algorithms

3.1 Iterative method

3.1.1 General

To store the data for every plate I use a data structure with three fields defined by me. The first field represents the top - the variable which keeps the number of elements of that certain plate. The second field - array of ints - stores the indices of the elements from that plate. The third field stores the name of each plate in a string. in order to create the structures I use my defined function create_stack which has as parameters my data structure and a string which transfers the name of the plate. This function returns the data structure created and has O(1) complexity.

For adding elements on a plate I use a function named push defined by me and it increases the top by 1 and also adds the element on the top position of the array. This function has as parameters the data structure in which the element is added.

Moving an element from a plate to another plate is done by the function pop having as parameters the structure defined by me. Pop returns the element to be moved and decreases the top by 1.

Pop and push also has constant time complexity.

Using a function called fill_array the cake is arranged on the initial plate in decreasing order. This process takes O(n) time.

For printing the moves generated in my program I use a function called print which takes as parameters the two plates which takes part in my move: the plate from which I take the element from and the plate where I put it, and the index of the moved story.

3.1.2 Moves

This function choose and implement the legal movement between two plates. First, we need to pop the last story from each of them in order to find out in each case i find myself at that moment. If one of the popped element is equal with 0, that means that we have no story on that plate and we need to push other story. In other case, when both of the popped elements exist, we need to compare them to put the little one above the big one.

Parameters are: from_plate $--$ the structure that represents the source plate and to_plate $--$ the structure that represents the destination plate.

This function has only decisional conditions. In this way, the complexity time is O(const.). Depending on the case, we have:

1. in best case , when the first condition is fulfilled, the complexity is O(5);
2. in worst case , when the last condition is fulfilled, the complexity is O(9);

```
MOVE_FROM_PLATE ($from\_plate$, $to\_plate$)
1. top_first_plate ← pop(from_plate)
2. top_second_plate ← pop(to_plate)
3. if top_first_plate = 0 then
4.     push($from\_plate$, $top\_second\_plate$)
5.     print the move
6. if top_second_plate = 0 then
7.     push($to\_plate$, $top\_first\_plate$)
8.     print the move
9. if top_first_plate > top_second_plate then
10.    push($from\_plate$, $top\_first\_plate$)
11.    push($from\_plate$, $top\_second\_plate$)
12.    print the move
13. if top_first_plate < top_second_plate then
14.     push($to\_plate$, $top\_second\_plate$)
15.     push($to\_plate$, $top\_first\_plate$)
16.     print the move
```

Figure 2: Move decision

3.1.3 Algorithm function

We have to calculate the total number of moves like 2 at power n - 1, when n is number of stories. Iterating through the no of moves, we check the following condition. If the iterator modulo 3 is equal with 1 , we do the movement of top story between source plate and destination plate. If the iterator modulo 3 is equal with 2, we do the movement top story between source plate and auxiliary plate. If the iterator is divided by 3, then the legal movement is between auxiliary plate and destination plate. If number of disks is even then interchange destination plate and auxiliary plate.

The parameters are: all the structures that represent the three plates in this order: silver plate, bronze plate and golden plate.

Time complexity of this function in O(m * 2^{n}),

where n is number of stories and m represents the number of conditions that go through until finds one which fulfills it. $1 \leqslant m \leqslant 4$

```
TOWER_OF_HANOI ($from\_plate$, $aux\_plate$, $to\_plate$)
1. total_no_of_moves ← 2 $at\ power\ n - 1$
2. for iterator ← 1, $total\_no\_of\_moves$ do
3. if iterator % 3 = 1 then
4.    if no_of_stories % 2 = 1 then
5.       move_from_plate($from\_plate$ , $to\_plate$)
6.    else move_from_plate($from\_plate$ , $aux\_plate$)
7. if iterator % 3 = 2 then
8.    if no_of_stories % 2 = 1 then
9.       move_from_plate($from\_plate$ , $aux\_plate$)
10.   else move_from_plate($from\_plate$ , $to\_plate$)
11. if iterator % 3 = 0 then
12.    move_from_plate($aux\_plate$ , $to\_plate$)
```

Figure 3: Generating the moves to solve the problem

3.2 Recursive method

3.2.1 General

In this solution, we recurse on the largest story to be moved. We have a recursive function that takes as a parameter the floor that is the largest floor in the cake we want to move. At the top level, we want to move the entire tower, so we want to move all stories from silver plate to bronze plate. We can break this into three basic steps:

1. Move stories n - 1 and smaller from silver plate to bronze plate, using golden plate as a spare. We do this by recursively using the same procedure. After finishing this, we have all the other stories excepted the biggest one on the bronze plate.

2. Now, with all the smaller disks on the spare plate, we move last floor from silver to golden plate.

3. Finally, we move floor n - 1 and smaller from additional plate to golden plate recursively using the same procedure.

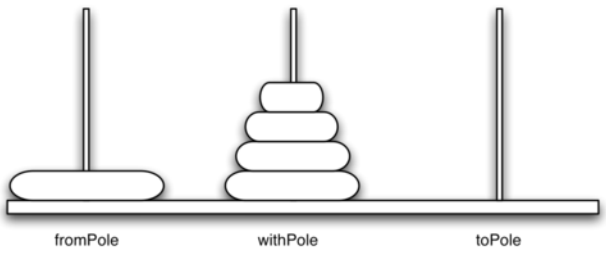


Figure 4: Example after first recursion

3.2.2 Algorithm function

When story is equal with 1 it means that it is the smallest. In this case we can just move the story directly. In the other cases, we follow the three-step recursive procedure.

The parameters are : an integer - number of stories and the three structures plates, in this order: source plate, auxiliary plate and destination plate.

Time complexity of this function in O($2^{n} - 1$),

where n is the number of stories.

```
TOWER_OF_HANOI ( $no\_of\_stories$, strings $silver\_plate$, $bronze\_plate$, $golden\_plate$ )
1. if $no\_of\_story = 1$ then
2.     print the move
3.     return
4. tower_of_hanoi ( $no\_of\_story$ - 1, strings $silver\_plate$, $golden\_plate$, $bronze\_plate$ )
5. print the move
6. tower_of_hanoi ( $no\_of\_story$ - 1, strings $bronze\_plate$, $silver\_plate$, $golden\_plate$ )
```

Figure 5: Recursive method

3.3 Generator input

I make a random generator to generate ten different inputs for the problem. For first five integers, I use standard function rand() % 15 + 5 to generate small values because tower of Hanoi has an exponential time execution. In this way, the generated values are: $5 \leq no\_of\_stories \leq 19$.

For the last five integers, I have a rand() % 60 + 10 for such large value. In this way, the generated values are: $10 \leq no\_of\_stories \leq 69$.

All this values are put on a file.

4 Application design

4.1 The high level architectural overview of the application

The recursive method applies "*Divide and Conquer*" strategy , while the second method follows step by step all the moves, with some conditions. So, the first method is better on time complexity, while the other one has better memory performances.

In terms of performance, Tower of Hanoi is really slow for large input data because it has exponential-time complexity.

4.2 The specification of the input

We need to move a cake with n stories from one plate to another. N is the single variable that we ask for and represents the number of the stories of which the cake is made. N is by default an integer and usually it isn't too large due to the exponential time complexity.

4.3 The specification of the output

The aim of the program is to show every step needed to move the cake from the silver plate to the golden plate, respecting the above mentioned rules. We need to display the index number of the story and both plates we make use of: the one that we take the story from and the one we put the story on.

4.4 The list of all the modules in the application and their description

The Tower of hanoi task has long been used in computer science classes to introduce the notion of recursion. This puzzle is a classic toy problem in artificial intelligence for evaluating search and planning algorithms.

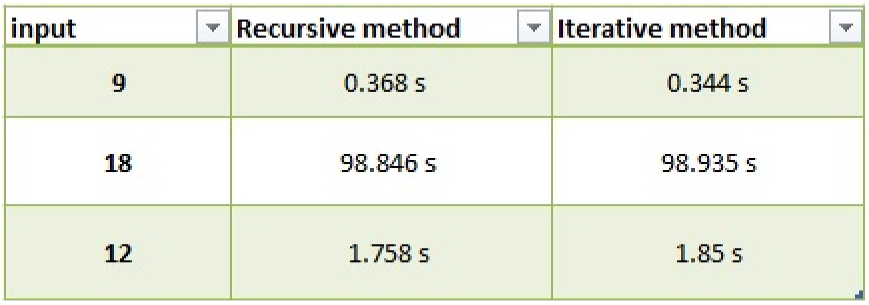
A number of models of TOH problem solving have been developed within the cognitive architecture, which combines symbolic and sub-symbolic computational mechanisms. These models share a common focus on the psychological reality of the goal stack. Other two models of TOH problem solving have been developed in a hybrid cognitive architecture that marries a symbolic production system interpreter with connectionist computational mechanisms such as thresholds, activations, weights, and parallel processing.

5 Results and Conclusions

The Tower of Hanoi problem can be solved in a variety of ways, with a wide variation in efficiency.

A simple solution for the toy puzzle is to alternate moves between the smallest piece and a non-smallest piece with strictly steps. Iterative solution is easily to understand, having a logic algorithm. The functions have concrete descriptions and they respect the single responsability principles.

The most known method of TOH is the recursive algorithm. The key to solving a problem recursively is to recognize that it can be broken down into a collection of smaller sub-problems, to each of which that same general solving procedure that we are seeking applies, and the total solution is then found in some simple way from those sub-problems' solutions. This method is harder to understand, but, regarding the time-complexity and the length of the code, this way is the most efficient.



input	Recursive method	Iterative method
9	0.368 s	0.344 s
18	98.846 s	98.935 s
12	1.758 s	1.85 s

Figure 6: Steps example for the algorithm

It is funny how I used to struggle solving this puzzle game when I was a child, and now I am capable to make the computer solve this puzzle for me and not only solving it, but doing it in the most efficient way.

References

[1] https://en.wikipedia.org/wiki/Tower_of_Hanoi *Tower of Hanoi* .

[2] https://www.geeksforgeeks.org/c-program-for-tower-of-hanoi/ https://www.geeksforgeeks.org/iterative-tower-of-hanoi/ *Algorithm ideas* .

[3] https://etd.library.vanderbilt.edu/available/etd-04032006-120209/unrestricted/Varma2006-dissertation.pdf *A COMPUTATIONAL MODEL OF TOWER OF HANOI PROBLEM SOLVING - DOCUMENTATION* .

[4] LaTeXproject site, http://latex-project.org/