

## Hanoi towers in Prolog.

The Hanoi towers is a mathematical puzzle invented in 1883 by the French mathematician Édouard Lucas, in this game you have three towers and a set of N disks of different sizes. Each of the discs has a hole in the center that allows them to slide down the towers. Initially, the N disks are ordered from largest to smallest in one of the towers. The discs must be passed to another tower using the third as an auxiliary.

These movements must be made respecting the following rules:

1. Only one disc can intervene in each movement, therefore it will always be the top disc that must move.
2. A disc cannot be left on top of a smaller one.

Recursion is a tool used in mathematics and programming that allows a subprogram to call itself and is useful for solving definable problems on your own terms. Recursion is divided into 2 cases: The base case that defines the end of the recursion and represents an input to the problem for which the solution is readily known. The recursive case where a general rule is defined to solve the problem by reducing it to a smaller problem.

Taking these principles into account to solve the problem of the Towers of Hanoi, our base case will be when we only have one disk left to move and recursively the problem is solved by noting that when moving each disk from the source tower to the destination tower, all the disks must first be moved. n-1 disks stacked on top of the largest disk in the origin tower towards the auxiliary tower. Then move this larger disc from the origin tower to the destination tower and finally the n-1 that were stacked in the auxiliary tower to the destination tower.

In prolog, we represent the problem as the predicate `hanoi(N):- move(N, left, right, center).`, where we move N disks from the left to the right tower, using the center tower as an auxiliary in order to achieve the goal.

A recursive algorithm for solving this problem already exists, which is:

1. Move N-1 disks from the source to the auxiliary, using the target tower.
2. Move a disk from the source to the target.
3. Move N-1 disks from the auxiliary to the target, using the source tower.

The representation in prolog for this algorithm is:

```
%Classic algorithm by recursion, N = number of disks, X = Source, Y = Target, Z = Auxiliary
move(N, X, Y, Z) :-
    N>1, % If n is more than 1, to filter 0 and negative values
    M is N-1,
    move(M, X, Z, Y), % Move N-1 disks from the source to the auxiliary, using the target peg
```

```
move(1, X, Y, _), % Move a disk from the source to the target
```

```
move(M, Z, Y, X). % Move N-1 disks from the auxiliary to the target, using the source peg
```

For this solution to work, we need a base case. In this problem, it will be moving a single disk from one tower to another. As we can infer, we don't need the auxiliary tower in order to complete the operation.

In prolog, it will be represented as:

```
move(1, X, Y, _) :-
```

```
    write('Move top disk from '),
```

```
    write(X),
```

```
    write(' to '),
```

```
    write(Y),
```

```
    nl.
```

Now that we have the full representation of the problem, as well as a way to solve it, we can make the entire code in prolog:

```
% This program solves the towers of hanoi problem, using any number of disks and three pegs:  
Source, Target and Auxiliary.
```

```
%Base case, move one disk from the source to the target peg.
```

```
move(1, X, Y, _) :-
```

```
    write(X),
```

```
    write(' -> '),
```

```
    write(Y),
```

```
    nl.
```

```
%Classic algorithm by recursion, N = number of disks, X = Source, Y = Target, Z = Auxiliary
```

```
move(N, X, Y, Z) :-
```

```
    N>1, % If n is more than 1, to filter 0 and negative values
```

```
    M is N-1,
```

```
    move(M, X, Z, Y), % Move N-1 disks from the source to the auxiliary, using the target peg
```

```
    move(1, X, Y, _), % Move a disk from the source to the target
```

```
    move(M, Z, Y, X). % Move N-1 disks from the auxiliary to the target, using the source peg
```

```
% Assuming we call source, auxiliary and target right, center and left respectively, we can  
create a predicate that only takes the number of disks on the source:
```

```
hanoi(N):-
```

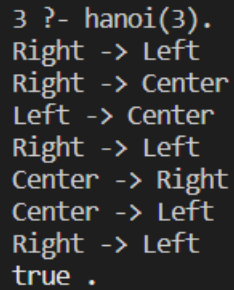
```
move(N, 'Right', 'Left', 'Center').
```

% To use the program, we input the number of disks on the hanoi() predicate

% Ex. for 4 disks, call hanoi(4)

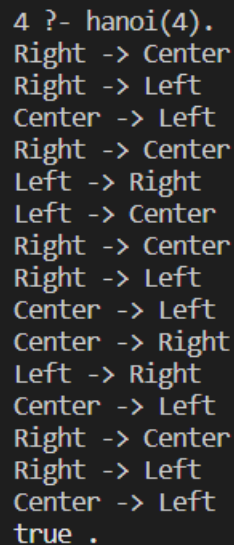
The code on a .pl file is located on: <https://github.com/AlexTheSleepy/hanoiTowers>.

When we run the program, we'll get the instructions to solve the problem with any number of disks.



```
3 ?- hanoi(3).  
Right -> Left  
Right -> Center  
Left -> Center  
Right -> Left  
Center -> Right  
Center -> Left  
Right -> Left  
true .
```

Figure 1: Screenshot of the program executing in a vscode terminal solving the Hanoi towers problem with 3 disks.



```
4 ?- hanoi(4).  
Right -> Center  
Right -> Left  
Center -> Left  
Right -> Center  
Left -> Right  
Left -> Center  
Right -> Center  
Right -> Left  
Center -> Left  
Center -> Right  
Left -> Right  
Center -> Left  
Right -> Center  
Right -> Left  
Center -> Left  
true .
```

Figure 2: Screenshot of the program executing in a vscode terminal, solving the Hanoi towers problem with 4 disks.

For example, in Figure 1 and 2 we can see the instructions of the problem with 3 and 4 disks respectively. As we can see, the recursive solution is being used. The “Right -> Left” in the middle is the separator of the first and second ‘blocks’; the `move(1, X, Y, _)` on the recursion. The first ‘block’ is moving  $N-1$  disks from the source to the auxiliary and the second ‘block’ is moving  $N-1$  from the auxiliary to the target. In general, each block is subdivided into two blocks by the middle instruction, until the blocks are one instruction

each. In that case, they are base cases, moving the top disk from any X tower to any other Y tower.

In conclusion, the Hanoi Towers problem is a good and well-known example of using recursion to solve a problem. It can be done easily, and deducting the algorithm isn't an issue, as putting some thought into it reveals it intuitively. All it takes is realizing the base case, and that iterating a couple of simple steps always leads to the solution. Since this is a logical problem, we can use a logic programming language like Prolog to automate the algorithm, since for a larger number of disks, the number of steps to reach the solution increase exponentially ( $O(n) = 2^n - 1$ ), and it's faster for a machine to generate the instructions on that order of magnitude than it is for a human. To complete the program, there was a need to understand the problem and the solution as well as the basic syntax of the language, which helped me to brush up on my knowledge of recursive programming, and concrete the new knowledge of Prolog by directly working with the language.

### References:

Educative Answers Team. (n.d.). *What is the Tower of Hanoi problem?* Educative: Interactive Courses for Software Developers. <https://www.educative.io/answers/what-is-the-tower-of-hanoi-problem>

M.Mirthula. (2021, April 28). *Tower of Hanoi — A Recursive approach*. Towards Data Science. <https://towardsdatascience.com/tower-of-hanoi-a-recursive-approach-12592d1a7b20#:~:text=The%20objective%20of%20the%20game>

*Torres de Hanoi*. (n.d.). <https://www.unca.edu.mx/Cprog/archivos/PSuperior2012.pdf>

*Torres de Hanoi (artículo) | Algoritmos*. (n.d.). Khan Academy. <https://es.khanacademy.org/computing/computer-science/algorithms/towers-of-hanoi/a/towers-of-hanoi>

Calvo, J. (2019, May 3). *Torres de Hanoi – Blog Europeanvalley*. Europeanvalley.es. <https://www.europeanvalley.es/noticias/torres-de-hanoi/>