Table of Contents

Introducción 1.1

Descripción 1.2

Pseudocódigo y complejidad 1.3

Análisis del tiempo de ejecución 1.4

Informe: Torres de Hanói

Introducción

En este documento trataremos el informe realizado sobre la práctica 2 de Diseño y Análisis de Algoritmos sobre el juego de *Las Torres de Hanói*. El informe contendrá una introducción , una descripción del problema, el pseudocódigo y un análisis detallado, y un análisis del tiempo de ejecución.

Torres de Hanói

Las Torres de Hanói es un rompecabezas o juego matemático inventado en 1883 por el matemático francés Édouard Lucas. Este juego de mesa solitario se trata de un juego con un número de discos de radio creciente que se apilan insertándose en una de las tres estacas de un tablero. El objetivo del juego es crear la pila en otra de las estacas siguiendo ciertas reglas. El problema es muy conocido en la ciencia de la computación y aparece en muchos libros de texto como introducción a la teoría de algoritmos.

La fórmula para encontrar el número de movimientos necesarios para transferir n discos del poste A al poste C es: ***2n - 1.***

> Credits: Wikipedia.org

Descripción del problema

> Credits: Diseño y Análisis de Algoritmos, Curso 2016-2017

Las Torres de Hanói o Torres de Brahma es un juego o puzzle matemático. Aunque hay diferentes variantes, la tradicional, y más conocida, consiste en tres varillas y un número "*n*" de discos de diferentes tamaños (radios) que pueden deslizarse sobre cada una de las varillas. El juego comienza con los "*n*" discos apilados de mayor a menor radio en forma de cono en una de las varillas. El objetivo del juego, ilustrado en la Figura 1, es mover la pila de discos completa a otra varilla, teniendo en cuenta las siguientes reglas:

- **Sólo se puede mover un disco a la vez.**
- **Cada movimiento consiste en tomar uno de los discos ubicados en el tope de una de las pilas y ubicarlo en el tope de otra pila.**
- **No se puede colocar un disco de mayor radio sobre otro de menor radio.**

Con *"n= 3"* discos, el juego puede resolverse, como mínimo, en siete movimientos. El mínimo número de movimientos requeridos para resolver el juego teniendo en cuenta un total dendiscos es igual a ***2n−1***. Por ejemplo, en el caso de n= 64 discos, el número mínimo de movimientos es igual a 1,8446743e+ 19.

Efectivamente, tal y como hemos podido comprobar desde n = 3 hasta n = 15 con nuestro programa el numero de movimientos efectuados para cada n es el siguiente:

> **También podemos observar como el tiempo de ejecución del método Hanoi.playGame() va aumentando cada vez.**
>
> *See*: Análisis del tiempo de ejecución

```
$ java Hanoi 3 0
Final: C = [3, 2, 1]
Tiempo de ejecucion: 1 milisegundos.
Numero total de movimientos: 7 movimientos.

$ java Hanoi 4 0
Final: C = [4, 3, 2, 1]
Tiempo de ejecucion: 1 milisegundos.
Numero total de movimientos: 15 movimientos.

$ java Hanoi 5 0
```

```
Final: C = [5, 4, 3, 2, 1]
Tiempo de ejecucion: 1 milisegundos.
Numero total de movimientos: 31 movimientos.

$ java Hanoi 6 0
Final: C = [6, 5, 4, 3, 2, 1]
Tiempo de ejecucion: 1 milisegundos.
Numero total de movimientos: 63 movimientos.

$ java Hanoi 7 0
Final: C = [7, 6, 5, 4, 3, 2, 1]
Tiempo de ejecucion: 1 milisegundos.
Numero total de movimientos: 127 movimientos.

$ java Hanoi 8 0
Final: C = [8, 7, 6, 5, 4, 3, 2, 1]
Tiempo de ejecucion: 1 milisegundos.
Numero total de movimientos: 255 movimientos.

$ java Hanoi 9 0
Final: C = [9, 8, 7, 6, 5, 4, 3, 2, 1]
Tiempo de ejecucion: 2 milisegundos.
Numero total de movimientos: 511 movimientos.

$ java Hanoi 10 0
Final: C = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
Tiempo de ejecucion: 3 milisegundos.
Numero total de movimientos: 1023 movimientos.

$ java Hanoi 11 0
Final: C = [11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
Tiempo de ejecucion: 5 milisegundos.
Numero total de movimientos: 2047 movimientos.

$ java Hanoi 12 0
Final: C = [12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
Tiempo de ejecucion: 6 milisegundos.
Numero total de movimientos: 4095 movimientos.

$ java Hanoi 13 0
Final: C = [13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
Tiempo de ejecucion: 8 milisegundos.
Numero total de movimientos: 8191 movimientos.

$ java Hanoi 14 0
Final: C = [14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
Tiempo de ejecucion: 10 milisegundos.
Numero total de movimientos: 16383 movimientos.

$ java Hanoi 15 0
Final: C = [15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
Tiempo de ejecucion: 14 milisegundos.
```

```
Numero total de movimientos: 32767 movimientos.
```

Pseudocódigo y complejidad

El método divide y vencerás consiste en descomponer el problema que hay que resolver en una serie de subproblemas, resolver estos subproblemas y combinarlos después para obtener la solución del problema original. Lo importante es que los subproblems son del mismo tipo que el problema original, pero de menor tamaño, y se resuelven usando la misma técnica. De esta forma, el método se expresa de manera natural mediante un algoritmo recursivo, con esquema:

```
divide_venceras(p: problema)
    dividir(p, p1, p2, ..., pk)
    for i = 1, 2, ..., k
        si = resolver (pi)
    solución = combinar(s1,s2,...,sk)
```

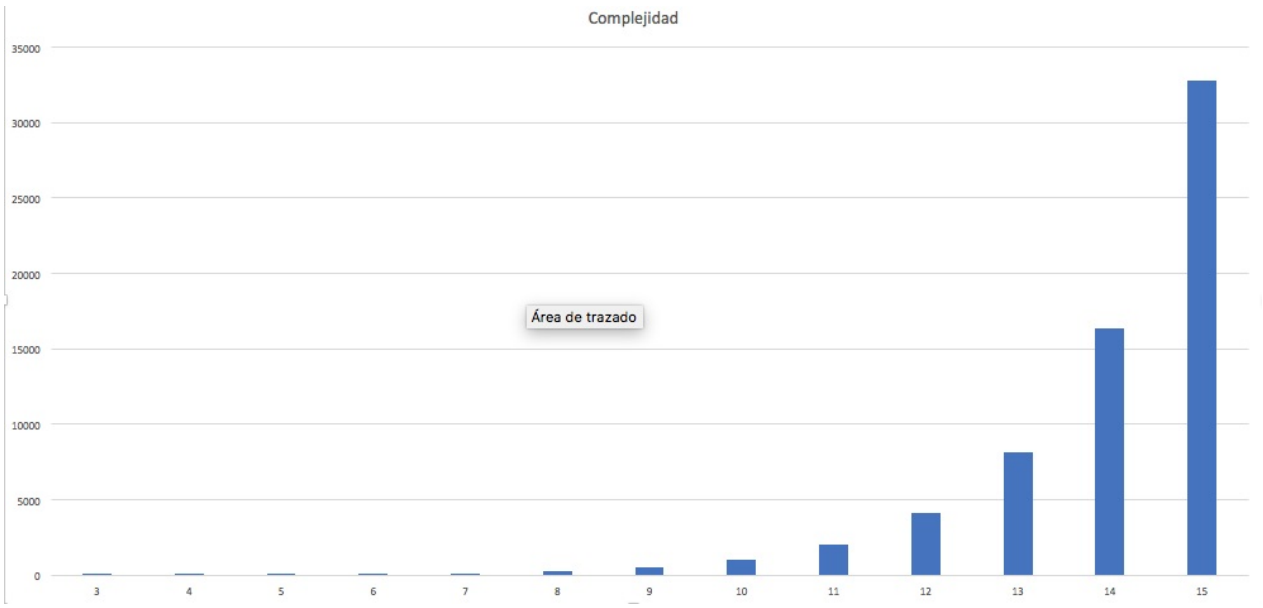
Siguiendo el esquema dado anteriormente, por consiguiente, podemos definir el pseudocódigo de *Las Torres de Hanoi* de la siguiente manera:

```
hanoi(n, A, B, C: int)
    if n==1 then
        mover(A,C)
    else
        Hanoi(n-1, A, C, B)
        mover(A,C)
        Hanoi(n-1, B, A, C)
```

Siendo el primer *if nuestro ad hoc* y siendo la función `mover(X,Y)` nuestra manera de resolver y combinar los sub-problemas.

Con respecto a la ***complejidad***, viene dada por el número de movimientos. Después de ver el análisis de movimientos realizado en la Descripción, podemos comprobar que viene a ser *Θ(2^n)*.

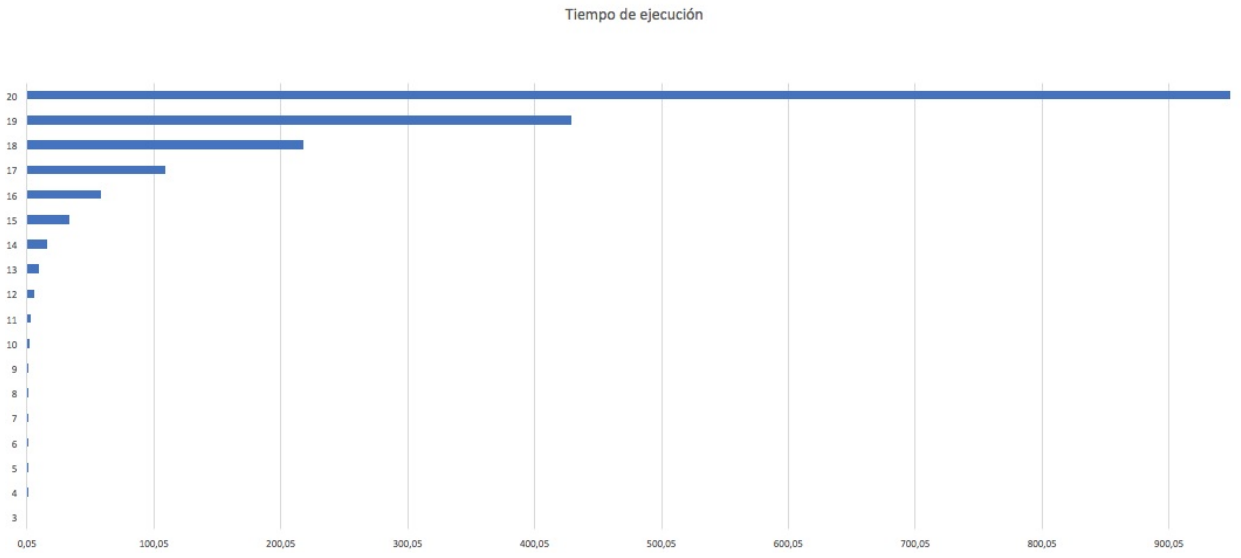
A continuación se muestra una gráfica de la complejidad desde n=3 hasta n=15. Podemos observar que es similar a la dada en Análisis del tiempo de ejecución, ya que al ser el tiempo en realizar un movimiento una constante, el tiempo total que tarda en computar el problema viene dado por el número de movimientos.


Complejidad
35000
30000
25000
20000
15000
10000
5000
0
Área de trazado
3
4
5
6
7
8
9
10
11
12
13
14
15

Análisis del tiempo de ejecución

A continuación mostramos una tabla con diferentes tamaños de problema y diferentes tiempos de ejecución en el modo visual, para apreciar una mayor diferencia:

Entrada del programa	Tiempo de ejecución
java Hanoi 3 1	5 milisegundos
java Hanoi 4 1	12 milisegundos
java Hanoi 5 1	19 milisegundos
java Hanoi 6 1	31 milisegundos
java Hanoi 7 1	49 milisegundos
java Hanoi 8 1	73 milisegundos
java Hanoi 9 1	133 milisegundos
java Hanoi 10 1	185 milisegundos
java Hanoi 11 1	314 milisegundos
java Hanoi 12 1	578 milisegundos
java Hanoi 13 1	979 milisegundos
java Hanoi 14 1	1566 milisegundos
java Hanoi 15 1	3304 milisegundos
java Hanoi 16 1	5803 milisegundos
java Hanoi 17 1	10886 milisegundos
java Hanoi 18 1	21828 milisegundos
java Hanoi 19 1	42886 milisegundos
java Hanoi 20 1	94827 milisegundos


Tiempo de ejecución
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
0,05
100,05
200,05
300,05
400,05
500,05
600,05
700,05
800,05
900,05