Coding

Denis TRYSTRAM

Lecture notes **Maths for Computer Science** – *MOSIG 1 – 2017*

1 Summary/Objective

Coding the instances of a problem is a tricky question that has a big influence on the way to obtain the solution. Even if Theory tells us that all *reasonable*[1] codings of instances are equivalent, this is an important practical problem.

2 Hanoi towers

2.1 The standard solution

Let start by revisiting the standard (recursive) version of the Hanoi towers problem.

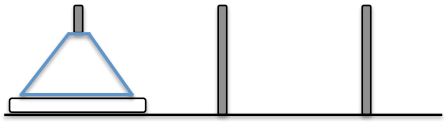

Figure 1: Initial position of the Hanoi towers.

The principle of the solving method is recalled in Figure 2 below. The disks are stacked on the first peg (taken as the left, denoted by D) and must be placed at the arrival peg (denoted by A) in minimum number of moves. The arrival is on the right peg if number of disks is odd and middle peg otherwise. The pegs are numbered respectively 0, 1 and 2 from left to right and any move is taken modulo 3 (which means for instance that the index of the peg right to peg 2 is 0).

It is forbidden to move a disk on top of a smaller one.

Let n denote the number of disks.

We detail the first moves of the solution in the figures 3 to 7.

The analysis of these first moves evidences some properties, in particular, the smallest disk moves every second steps. A more detailed observation shows that it moves successively on all the pegs $D \to A$, $A \to I$, $I \to D$, and so on. Using the coding of pegs, this corresponds to the cycling process:

[1] which means in polynomial memory space

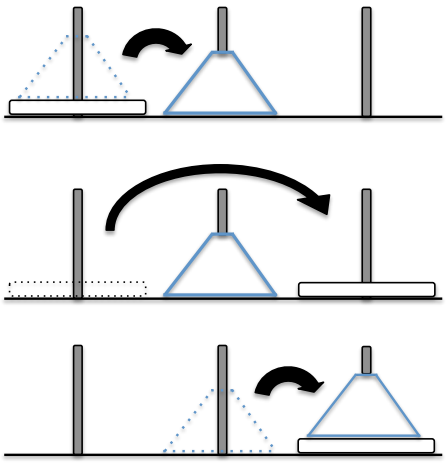

Figure 2: Principe of the recursive solution: move the $n-1$ smallest disks to the intermediate peg, move the largest one at its destination, move again the $n-1$ remaining disks on top of it.

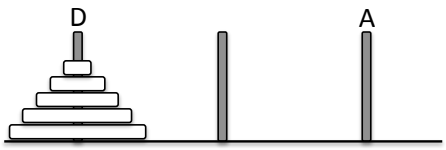

Figure 3: Initial position of the Hanoi tower puzzle composed of 5 disks. The arrival is on the right peg since the number of disks is odd.

$0 \to 2$, $2 \to 1$, $1 \to 3[3] = 0$. Going further shows that the second smallest disks moves every four steps (at step 2, 6, etc.) and we easily check that the i-th smallest disk moves every 2^i steps, while the largest disk moves only once. As the moves of the smallest disk are fixed, there is only one possible move left for the other disks (because of the constraint of moving only shorter disks on larger ones).

All these remarks above lead to a pretty simple analysis as shown in the next section.

Figure 4: First and second moves.

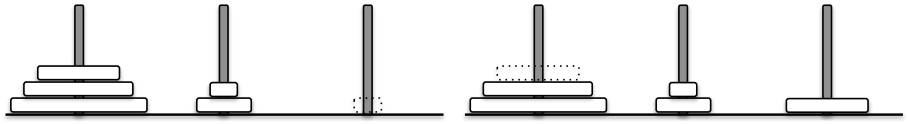

Figure 5: Moves 3 and 4.

2.2 Coding

Coding moves. The successive moves studied in the previous section may be coded using a binary representation. Let m be the move index. The natural way is to associate a binary digit to each disk where the most significant (leftmost) bit represents the largest disk as shown in Figure 8.

The rule for moving the disks have been briefly presented above, it is represented in Figure 9. More precisely, the smallest disk moves every second steps (odd m) and the other disks to move are determined by the number of times m can be divided by 2 (*i.e.* the number of 0 bits at the right of the configuration). The position of the disk to move is given by the rank of the first bit at 1 in the binary configuration of the move (see Figure 10). For instance, at move 12 $(01100)_2$, this rank is 3 (the third disk is moving).

The departure and arrival pegs (indexed by 0 and 2) for the mth move can also be determined elegantly from the binary representation of m using bitwise operations. Recall that the pegs are numbered 0, 1 and 2 from left to right and the disks are all stacked in peg 0 at the beginning, their destination is on peg 1 or 2 according to the parity of the number of disks. Move m is from peg (m AND m - 1) modulo 3 to peg ((m OR m - 1) + 1) modulo 3.

Coding positions. Here, we are interested in coding the positions of the disks at a given move instead of the moves. The bit string is read from left to right and each bit can be used to determine the location of the corresponding disk. More precisely:

- A 0 indicates that the largest disk is on the initial peg, while a 1 indicates that it is on the destination peg (right peg if number of disks is odd and middle peg otherwise). Observation: a straight sequence of 1's or 0's means that the corresponding disks are all on the same peg.

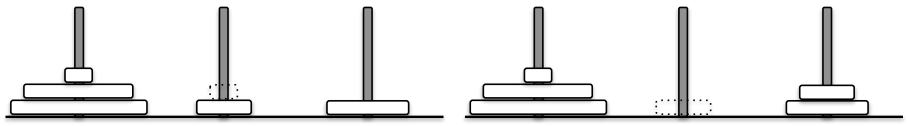

Figure 6: Moves 5 and 6.

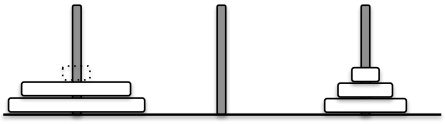

Figure 7: Move 7.

- A bit with a different value to the previous one means that the corresponding disk is located in one position to the left or right of the previous one. Whether it is left or right is determined by the following rule: Let k be the number of greater disks that are located on the same peg as their first greater disk. Add 1 to k if the largest disk is still on its initial peg (the left one). If k is even, the disk is located on the peg to the right, if it is odd, the disk is located one peg to the left (in case of even number of disks and vice versa otherwise).

Let us detail how this process works on a particular configuration: Move $(216)_{10} = (11011000)_2$. Figures 11 to 14 detail the positions of the successive disks.

As n is even, the arrival peg is the middle one. The bit representing the largest disk is 1, thus it is on the middle peg. The second disk is also coded by 1, so it is stacked on top of it, on the middle peg.

The coding of disk 3 is 0, which means that it is on another peg. Since $k = 1$ (it is odd), it is one peg to the left, *i.e.* on the left peg.

Disk 4 is coded by 1, so it is on another peg. Since k is odd (it is equal to 1), it is on the peg to the left, *i.e.* on the right peg according to the wraparound properties. The following bit is also equal to 1, it is stacked on top of it, on the same peg.

As the next disk (the 6th) is 0, it is on another peg. Since k is even ($k = 2$), the disk is one peg to the right, *i.e.* on the left peg. Both remaining disks (7th and 8th) are also 0, thus they are stacked on the left peg.

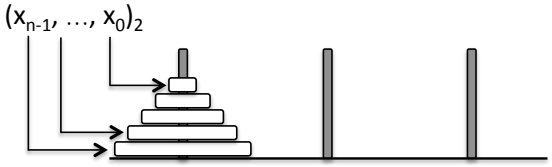

Figure 8: Labelling the disks by a bit string.

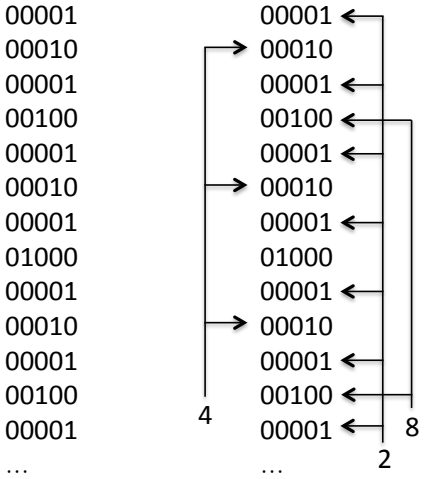

Figure 9: Position of the moved disks (left) and periodic pattern (right). The right bit corresponds to the smallest disk which moves every second moves, the second smallest moves every 4 moves, etc.

2.3 Using Gray code

Gray code is a binary coding system where the numbers are expressed by binary digits, but contrary to the classical positional number system, Gray code ensures that each number differs from its predecessor by exactly one bit changed. We can use this coding for describing the solution of the Hanoi Towers (see Appendix for the definition and properties of Gray codes).

If the moves are coded with a reflected Gray code (starting at zero) then the rank of the bit changed corresponds to the disk to move.

00001
00010
00011
00100
00101
00110
00111
01000
01001
01010
01011
01100
01101
...

Figure 10: The binary representation of the move index m gives the disk to move (rank of the first bit at 1 from the right – in blue).

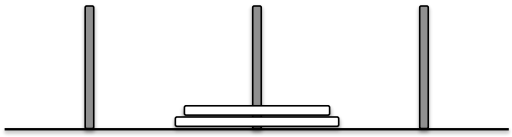

Figure 11: Position of the two largest disks.

3 Josephus

The second problem chosen to illustrate elegant coding is the Josephus' problem (also called survival problem) which was analyzed in detail in another chapter. Let us recall it briefly: n items are placed successively on a circle. Starting from the first item, we remove every second remaining item. The index of the last item is called $J(n)$.

In this problem, powers of 2 play again an important role. Let us use the binary representation of n and $J(n)$:
$n = \sum_{j=0}^{j=m} b_j.2^j = b_m.2^m + b_{m-1}.2^{m-1} + ... + b_1.2 + b_0$
$n = (1b_{m-1}...b_1b_0)_2$ since by definition of m $b_m = 1$
$k = (0b_{m-1}...b_1b_0)_2$ since $k < 2^m$
Thus, using the closed formula for $J(n)$:
$J(n) = (b_{m-1}...b_0b_m)_2$.
In other words, the solution is obtained by a simple shift of the binary representation of n.

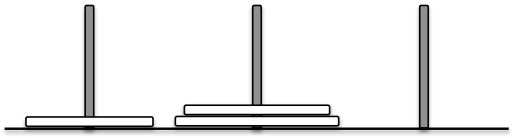

Figure 12: Position of the third disk.

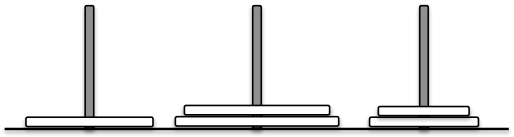

Figure 13: Position of the fourth and fifth disks.

Applied to $n = 41 = (101001)_2$
$J(41) = (010011)_2 = 19$.

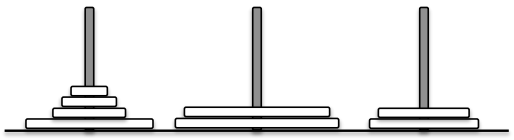

Figure 14: Position of the remaining disks.

Appendix: Gray codes

This code was invented by a french mathematician of the XIXth century (Louis Gros in 1872), reinvented by Franck Gray at Bell's lab in 1930.

There exist several variants of Gray codes. Let us present the most popular one, namely the *reflected Gray code* whose principle is depicted in Figure 15. The 1-bit Gray code is simply 0 and 1. The next one (for 2-bits) is obtained by mirroring the 1-bit code and prefix it by 0 and 1. The next ones are obtained similarly (see Figure 15).

The most important characteristic is the coding from one position to the next is to flip only one bit. There is a simple way to determine the bit that changes from a position to the next one:

if the number of bit at 1 in position i is even, then flip the last bit, otherwise, flip the bit left to the rightmost bit equal to 1.

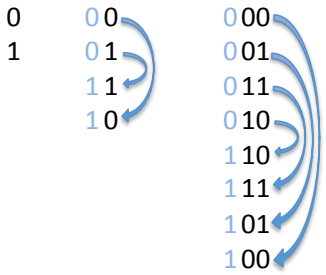

Figure 15: Construction of reflected Gray codes ($n = 1, 2, 3$).

Gray code can easily be determined from the classical binary representation as follows (see Figure 16):

$(x_{n-1}x_{n-2}...x_1x_0)_2$

shift right: $(0x_{n-1}x_{n-2}...x_1)_2$

Take the exclusive OR (bit-to-bit) between the binary code and its shifted number:

$(x_{n-1}(x_{n-2} \oplus x_{n-1})...(x_0 \oplus x_1))_G$.

For instance in the example of the figure, the binary code of $5 = (00101)_2$ is $(0 \oplus 0)(0 \oplus 0)(0 \oplus 1)(1 \oplus 0)(0 \oplus 1) = (00111)_G$.

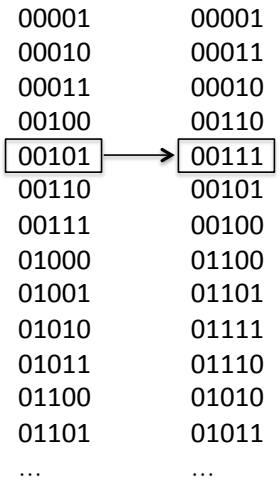

Figure 16: From binary to reflected Gray code.