Hanoi

Analice hasta su total comprensión el siguiente fragmento de código OCaml.

```
let rec hanoi n ori des =
    (* n número de discos, 1 <= ori <= 3, 1 <= dest <= 3, ori <> des *)
    if n = 0 then "" else
    let otro = otro ori des in
    hanoi (n-1) ori otro ^ move (ori, des) ^ hanoi (n-1) otro des
```

Con esta definición, ***hanoi n ori des*** devuelve, codificados en un string, los movimientos necesarios para resolver el problema de las “Torres de Hanoi” con *n* discos, partiendo del poste *ori* y teniendo que dejarlos todos en el poste *des* (los postes se suponen numerados del *1* al *3*).

```
# hanoi 1 1 3;;
- : string = " 1 ---2--> 3 \n"
# hanoi 2 3 1;;
- : string = " 1    2 <- 3 \n 1 <--2--- 3 \n 1 <- 2    3 \n"
```

Se dispone, además de una redefinición de esta función que permite contemplar el caso de que el poste de origen sea el mismo que el de destino (en ese caso, el problema está ya resuelto sin necesidad de realizar ningún movimiento).

```
let hanoi n ori des =
    if n = 0 || ori = des then "\n"
    else hanoi n ori des
```

Adicionalmente, también está definida una función ***print_hanoi*** que muestra por la salida estándar una representación de los movimientos, después de haber comprobado la validez de los argumentos.

```
let print_hanoi n ori des =
    if n < 0 || ori < 1 || ori > 3 || des < 1 || des > 3
       then print_endline  " ** ERROR ** \n"
       else print_endline (" ========== \n" ^
                           hanoi n ori des ^
                           " ========== \n");;
```

Observe los siguientes ejemplos de uso.

```
# print_hanoi 3 1 4;;
 ** ERROR **

- : unit = ()
# print_hanoi 3 1 1;;
 ===========
 ===========

- : unit = ()
# print_hanoi 0 2 3;;
 ===========
 ===========

- : unit = ()
# print_hanoi 3 3 2;;
 ===========
 1    2 <- 3
 1 <--2--- 3
 1 <- 2    3
 1    2 <- 3
 1 ---2--> 3
 1 -> 2    3
 1    2 <- 3
 ===========

- : unit = ()
```

Lamentablemente, se han perdido las definiciones de las funciones ***move*** y ***otro***. Reconstruya estas definiciones, de modo que las funciones descritas anteriormente se comporten exactamente como en los ejemplos mostrados. Para ello, complete el archivo ***hanoi.ml*** suministrado junto a este enunciado. **No está permitido modificar de manera alguna las definiciones que se conservan completas.**

Como seguramente sabrá, el número de movimientos necesario para resolver las *Torres de Hanoi* con *n* discos (si el poste inicial y final son distintos) es exactamente $2^n - 1$.

Añada al archivo ***hanoi.ml*** la definición de una función ***nth_hanoi_move: int -> int -> int -> int -> int * int***, de modo que ***nth_hanoi_move n nd ori des*** devuelva el el ***n***-ésimo movimiento de la secuencia de movimientos necesarios para resolver el problema con *nd* discos desde el poste ***ori*** al poste ***des***. Dado que el mayor int es $2^{Sys.int\_size - 1} - 1$, solo se pretende que funcione correctamente si ***0 ≤ nd < Sys.int_size*** y ***ori <> des***.

Si el resultado es el par ***(i, j)*** esto indicará que el ***n***-ésimo movimiento consiste en llevar el disco en la cima del poste ***i*** a la cima del poste ***j***.

```
# nth_hanoi_move 1 3 1;;
- : int -> int * int = <fun>
# nth_hanoi_move 1 2 3 1;;
- : int * int = (3, 2)
# nth_hanoi_move 2 2 3 1;;
- : int * int = (3, 1)
# nth_hanoi_move 3 2 3 1;;
- : int * int = (2, 1)
# nth_hanoi_move 16 6 2 3;;
- : int * int = (2, 1)
# nth_hanoi_move 32 6 2 3;;
- : int * int = (2, 3)
# nth_hanoi_move 15 6 2 3;;
- : int * int = (1, 3)
# nth_hanoi_move 32 6 2 3;;
- : int * int = (2, 3)
# nth_hanoi_move 63 6 2 3;;
- : int * int = (1, 3)
# nth_hanoi_move max_int (Sys.int_size - 1) 3 2;;
- : int * int = (1, 2)
```

Ya que el número de movimientos para resolver Torres de Hanoi con *n* discos es $2^n - 1$, la complejidad de un algoritmo que enumere esos movimientos ha de ser $O(2^n)$. Por ello, debemos esperar que el tiempo empleado para calcular estos movimientos con la función ***hanoi*** se duplique cada vez que se incrementa en 1 el número de discos.

En OCaml podemos utilizar la pseudo-función Sys.time para consultar el tiempo de CPU consumido por el proceso. Así, la función crono, definida a continuación, puede usarse para cronometrar el tiempo que consume la aplicación de cualquier función a cualquier argumento

```
let crono f x =
    let t = Sys.time () in
    let _ = f x in
    Sys.time () -. t
```

Utilice esta función para ver como crecen los tiempos de aplicación de hanoi cuando el número de discos aumenta de, digamos, 20 a 25. Si su máquina es muy lenta o muy rápida quizá deba cambiar algo este intervalo. Considere repetir la operación varias veces para conseguir valores más fiables. Añada los resultados al final del archivo ***hanoi.ml*** (como comentario) y responda a la pregunta de si avalan o no lo esperado teóricamente.

La complejidad de la función ***nth_hanoi_move*** debía ser, sin embargo *O(nd)* (siendo *nd* el número de discos). Compruébelo empíricamente (e incluya también su experimento, como comentario, en el archivo *hanoi.ml*), teniendo en cuenta que, al tener que ser *nd* < *Sys.intsize* (seguramente *63* en su equipo), cualquier aplicación válida de la función debería resultar prácticamente instantánea. **Si no es así, debería considerar la redefinición de esta función.**