Paradigmas de Programación

Práctica 11

El "problema del caballo" (en inglés, *knight's tour*) es un antiguo problema matemático que parte de una cuadrícula de $m \times n$ casillas y de un caballo de ajedrez situado en una casilla cualquiera (x, y), donde $1 \leq x \leq m$ y $1 \leq y \leq n$. El problema consiste en averiguar si el caballo, efectuando los saltos en L propios de esta pieza de ajedrez, es capaz de pasar por todas las casillas de la cuadrícula, sin repetir ninguna. En caso afirmativo, la secuencia de casillas visitadas constituye una solución del problema.

En esta práctica consideraremos dos variantes del problema original:

1. La primera de estas variantes considera una casilla de inicio y otro casilla de finalización, pero esta vez no buscamos un camino que cubra todas las casillas de la cadrícula, sino un camino cualquiera que permita llegar de una casilla a otra, eso sí, mediante los saltos en L propios del caballo de ajedrez. Por tanto, escriba en un fichero `tour.ml` una función

   ```
   tour : int -> int -> (int * int) -> (int * int) -> (int * int) list
   ```

   que dados dos enteros (para especificar las dimensiones m y n de la cuadrícula), un par de enteros (para especificar las coordenadas (x_i, y_i) de la casilla en la que está situado inicialmente el caballo), y otro par de enteros (para especificar las coordenadas (x_f, y_f) de la casilla final en la que debe quedar situado el caballo), devuelva una lista de casillas que constituya una solución de esta variante del problema del caballo para esa cuadrícula y para esas posiciones inicial y final. Las casillas inicial y final deben aparecer al principio y al final de la lista, respectivamente. Y la lista no puede contener casillas repetidas, es decir, solo se puede pasar una vez por cada una de las casillas del recorrido obtenido. En caso de que dicha solución no exista, la función activará la excepción `Not_found`.

   ```
   # tour 3 3 (1,1) (2,2);;
   Exception: Not_found.

   # tour 5 5 (1,1) (5,5);;
   - : (int * int) list =
   [(1, 1); (2, 3); (4, 2); (2, 1); (1, 3); (3, 2); (5, 1); (4, 3); (3, 1);
    (1, 2); (2, 4); (4, 5); (5, 3); (4, 1); (2, 2); (3, 4); (5, 5)]
   ```

   En el segundo de los ejemplos de ejecución anteriores, la solución no es única, así que la lista obtenida podría ser diferente.

2. (Ejercicio opcional) La segunda variante también considera una casilla de inicio y una casilla de finalización, pero esta vez buscamos un camino minimal (es decir, un camino tal que no haya otro más corto) entre ambas casillas (y también mediante movimientos en L). Por tanto, escriba en un fichero `shortest.ml` una función que resuelva esta nueva problemática. Una vez más, las casillas inicial y final deben aparecer en la lista, y la lista no puede contener casillas repetidas. Y de nuevo, la función activará la excepción `Not_found` cuando no exista solución para los parámetros recibidos.

   ```
   # shortest_tour 5 5 (1,1) (5,5);;
   - : (int * int) list = [(1, 1); (2, 3); (1, 5); (3, 4); (5, 5)]
   ```

   En el ejemplo de ejecución anterior, la solución no es única, así que la lista obtenida podría ser diferente, pero deberá ser de la misma longitud.

Nota Importante: Realice las implementaciones de los ejercicios 1 y 2 de esta práctica en los ficheros `tour.ml` y `shortest.ml`, respectivamente. Cuando se solicite la entrega de esta práctica, cada alumno deberá enviar únicamente estos ficheros. Sea muy cuidadoso a la hora de crear los ficheros, y **respete los nombres indicados**. Además, **estos ficheros deben compilar sin errores** con órdenes: `ocamlc -c tour.ml` y `ocamlc -c shortest.ml`.