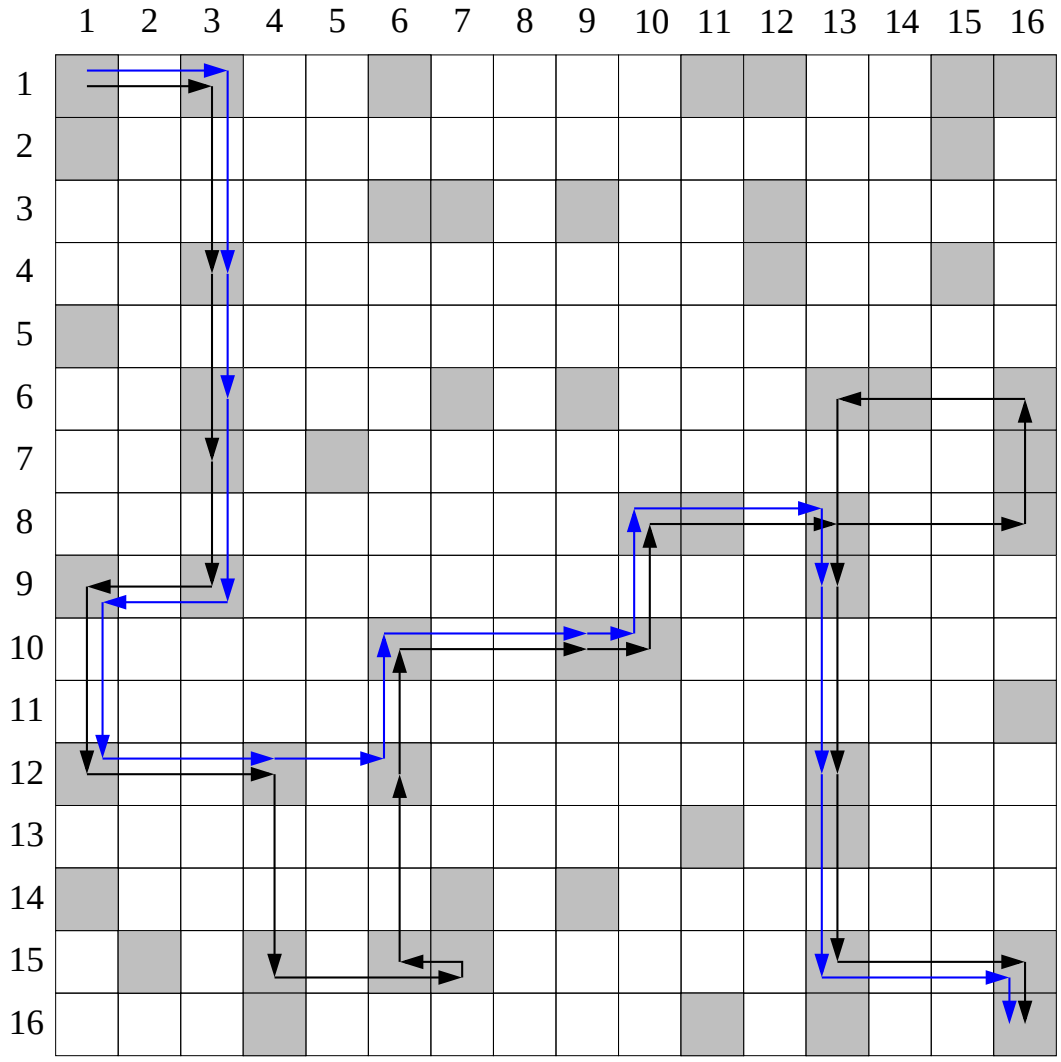
Paradigmas de Programación

Práctica 10

Ejercicios:

1. Dicen que antiguamente una ardilla podía cruzar la península ibérica de norte a sur, saltando de árbol en árbol, sin tocar el suelo. Imagine que dibujamos una rejilla que cubra toda la península, y que sombreamos aquellas casillas en las que hay un árbol, para obtener algo similar al tablero de la figura. Abordaremos entonces la problemática de encontrar un camino para la ardilla, que viaje (solo horizontal y verticalmente) desde la esquina superior izquierda hasta la esquina inferior derecha, que pase solo por las casillas sombreadas, y que además no repita ninguna. Adicionalmente, sabemos que la ardilla tiene una capacidad de salto de n casillas, donde $n > 0$. Por ejemplo, si n es 3, la ardilla puede saltar desde una determinada casilla a todas aquellas que se encuentren a distancia 1, 2 o 3 de esa casilla.



Escriba en un fichero `tour.ml` una función

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

que dados dos enteros mayores que 0 (para especificar las dimensiones m y n del tablero), una lista de pares de enteros (para especificar las casillas sombreadas, es decir, las que

contienen árboles), y un entero adicional (para especificar la capacidad de salto de la ardilla), devuelva una lista de casillas ordenadas de tal forma que constituyan una solución al problema de viajar desde la casilla $(1, 1)$ hasta la (m, n) saltando de árbol en árbol. Las casillas inicial y final deben aparecer al principio y al final de la lista, respectivamente, y la lista no puede contener casillas repetidas. En caso de que dicha solución no exista, la función activará la excepción `Not_found`.

Ejemplo de ejecución:

```
# let trees =
  [(1,1); (1,3); (1,6); (1,11); (1,12); (1,15); (1,16); (2,1); (2,15); (3,6);
   (3,7); (3,9); (3,12); (4,3); (4,12); (4,15); (5,1); (6,3); (6,7); (6,9);
   (6,13); (6,14); (6,16); (7,3); (7,5); (7,16); (8,10); (8,11); (8,13);
   (8,16); (9,1); (9,3); (9,13); (10,6); (10,9); (10,10); (11,16); (12,1);
   (12,4); (12,6); (12,13); (13,11); (13,13); (14,1); (14,7); (14,9); (15,2);
   (15,4); (15,6); (15,7); (15,13); (15,16); (16,4); (16,11); (16,13);
   (16,16)];;
val trees : (int * int) list = ...

# tour 16 16 trees 3;;
- : (int * int) list =
[(1, 1); (1, 3); (4, 3); (7, 3); (9, 3); (9, 1); (12, 1); (12, 4); (15, 4);
 (15, 7); (15, 6); (12, 6); (10, 6); (10, 9); (10, 10); (8, 10); (8, 13);
 (8, 16); (6, 16); (6, 13); (9, 13); (12, 13); (15, 13); (15, 16); (16, 16)]
```

Esta solución se corresponde con el camino de color negro de la figura de la página anterior. En este caso, la solución no es única, así que la lista obtenida podría ser diferente.

El fichero `tour.ml` debe compilar sin errores con la orden `ocamlc -c tour.mli tour.ml`.

2. (Ejercicio opcional) Una variante del mismo problema considera las mismas casillas de inicio y de finalización, pero esta vez buscamos un camino minimal en número de saltos, es decir, un camino tal que no haya otro con menos saltos entre ambas casillas, y también usando solo casillas sombreadas y sin repetir ninguna. Por tanto, escriba en un fichero `shortest.ml` una función `shortest_tour` que tenga el mismo tipo que la función `tour` y 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.

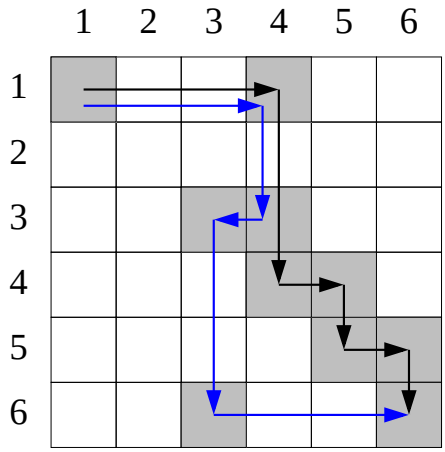
   Ejemplo de ejecución:

```
# shortest_tour 16 16 trees 3;;
- : (int * int) list =
[(1, 1); (1, 3); (4, 3); (6, 3); (9, 3); (9, 1); (12, 1); (12, 4); (12, 6);
 (10, 6); (10, 9); (10, 10); (8, 10); (8, 13); (9, 13); (12, 13); (15, 13);
 (15, 16); (16, 16)]
```

   Esta solución se corresponde con el camino de color azul de la figura de la página anterior. En este caso, la solución tampoco es única, así que la lista obtenida podría ser diferente, pero deberá ser de la misma longitud.

   Y tenga en cuenta además que un camino minimal en número de saltos no tiene porqué ser al mismo tiempo un camino minimal en distancia, es decir, en número de casillas recorridas. Por ejemplo, en la figura de la página siguiente se puede observar que el

camino de color negro realiza 6 saltos y recorre 11 casillas, mientras que el camino de color azul realiza 5 saltos pero recorre 13 casillas.



Lo que nos interesa en esta práctica es obtener un camino con el menor número de saltos posible, no el camino más corto, lo cual sería una problemática totalmente distinta.

Así pues, para este apartado opcional, en el tablero de esta nueva figura, una vez más, la solución correcta sería el camino de color azul.

El fichero `shortest.ml` debe compilar sin errores con la orden `ocamlc -c shortest.mli shortest.ml`.