Paradigmas de Programación

Práctica 10

Ejercicios:

1. Un preso ha escapado de su celda, pero, para obtener la libertad total, todavía tiene que cruzar el patio de la cárcel. El patio está dividido en baldosas, tiene una dimensión de $m \times n$ baldosas, y el preso sólo puede moverse baldosa a baldosa en horizontal y en vertical, nunca en diagonal. Y para escapar, debe buscar un camino desde la baldosa de la esquina noroeste, la $(1, 1)$, hasta la baldosa de la esquina sureste, la (m, n).

   En el patio hay unos robots de vigilancia dotados con sensores de movimiento. Y cada vez que detectan actividad en el patio, avanzan una baldosa. Cada uno de los robots puede efectuar dicho avance de alguna de estas cuatro maneras:

   (1) de norte a sur,

   (2) de sur a norte,

   (3) de oeste a este,

   (4) o de este a oeste.

   Si un robot llega a un borde del patio, digamos que "rebota". Es decir, en el siguiente movimiendo que efectúe, vuelve a la baldosa en la que estaba antes de alcanzar el borde, y cambia el sentido de su movimiento:

   - si era (1) pasa a ser (2), y viceversa;
   - y si era (3) pasa a ser (4), y viceversa.

   Las trayectorias de los robots podrían cruzarse, dando lugar a que dos o más robots caigan en la misma baldosa en un instante dado. No hay problema. Tienen unas pequeñas patas extensibles que les permiten situarse unos encima de otros en la misma baldosa, y suficiente inteligencia como para no autodestruirse entre ellos.

   El preso sabe que cada vez que avance una baldosa, los robots avanzarán también una baldosa exactamente en el mismo instante. Por tanto, antes de acometer la misión de cruzar el patio, lanza un objeto para que los robots efectúen un movimiento, y de esta forma adquiere la información de cómo están programados, lo cual le permitirá planificar su recorrido de escape.

   A la hora de realizar dicho recorrido, el preso sabe que si cae en una baldosa al mismo tiempo que algún robot cae en la misma baldosa, el robot hará explotar la bomba que lleva a bordo y el preso morirá. Por tanto, el preso debe buscar un camino que, en cada instante, no pase por baldosas ocupadas por robots. Y no puede repetir baldosas, es decir, no puede pasar por la misma baldosa más de una vez. Esto no garantiza necesariamente que el preso realice el camino más corto posible, pero sí impide que su algoritmo de planificación entre en un lazo sin fin que lo tenga dando vueltas por el patio indefinidamente.

   Sí puede, sin embargo, hacer lo siguiente: saltar a una baldosa donde "ahora mismo" hay un robot. ¿Por qué? Porque sabe que, en cuanto haga el gesto de iniciar el salto, el robot detectará el movimiento y abandonará la baldosa. Puede hacer esto incluso si el robot va a avanzar hacia la baldosa en la que está el preso, porque se supone que los robots son pequeños y el preso puede saltar por encima de ellos. En resumen, lo dicho: sólo debe evitar caer en una baldosa al mismo tiempo que algún robot cae en esa misma baldosa.

Considere la siguiente definición, como tipo soporte de los movimientos que pueden efectuar tanto el preso como los robots:

```
type move =
  North | South | East | West
```

Y seguidamente, para calcular la ruta de escape, implemente una función `tour` con tipo

```
int -> int -> ((int * int) * move) list -> move list
```

donde:

- los dos primeros parámetros son las dimensiones m y n del patio,
- el siguiente parámetro es la lista de las posiciones donde están los robots en el instante en el que el preso decide iniciar la ruta (junto con el tipo de movimiento que tienen programado),
- y el resultado a devolver es la lista ordenada de movimientos que tiene que efectuar el preso para escapar.

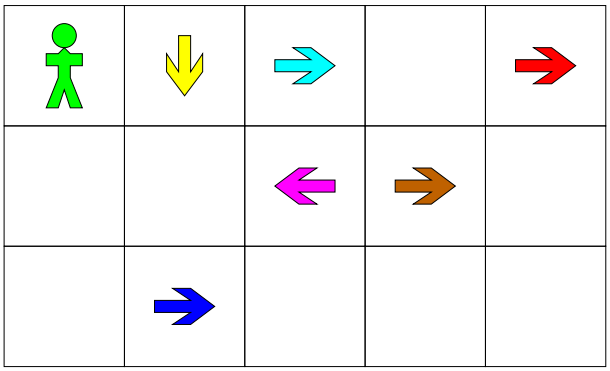
El problema podría tener varias soluciones. Es suficiente con devolver una cualquiera de ellas. Y si no existe ninguna ruta de escape posible, la función `tour` debe lanzar la excepción `Not_found`.

Realice su implementación en un archivo de texto con nombre `tour.ml`.

Ejemplo de ejecución:

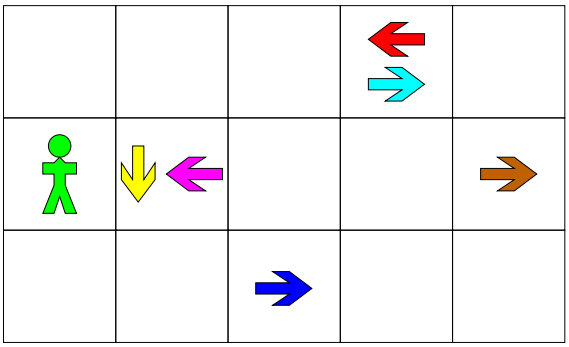
```
# tour 3 5 [((1, 2), South); ((1, 3), East); ((1, 5), East);
            ((2, 3), West); ((2, 4), East); ((3, 2), East)];;
- : move list =
  [South; South; East; East; North; East; South; East]
```

El instante inicial del escenario correspondiente a este ejemplo de ejecución podría representarse mediante la siguiente figura, donde el muñeco es el preso y las flechas son los robots:

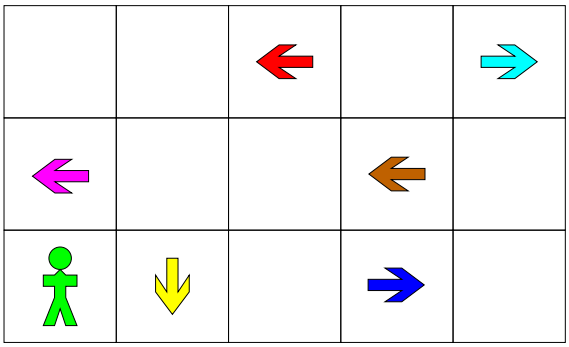


Instante 1

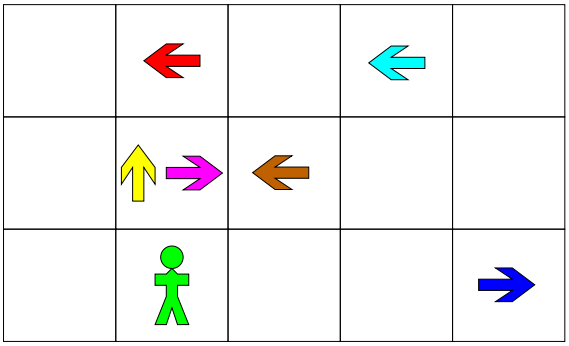
La evolución, instante a instante, de este escenario se muestra en la figura de la página siguiente.



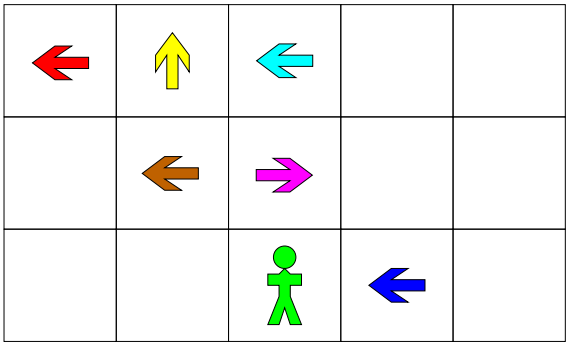
Instante 2



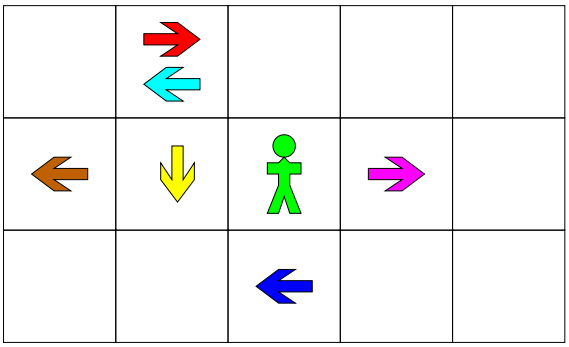
Instante 3



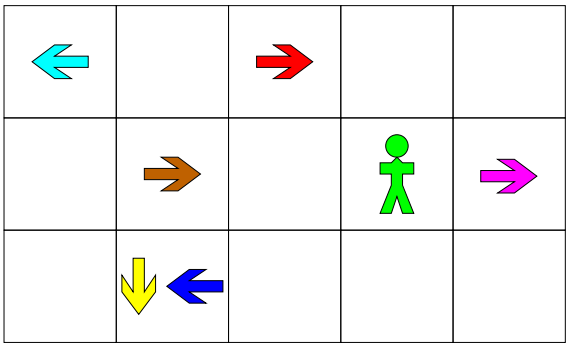
Instante 4



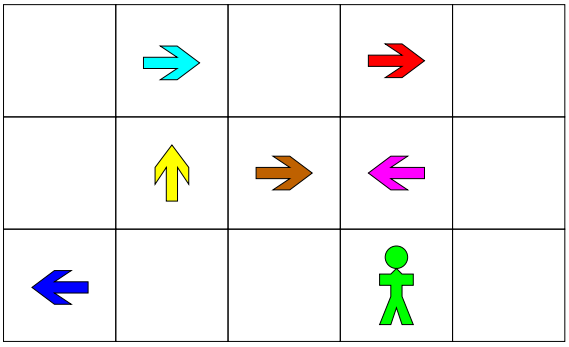
Instante 5



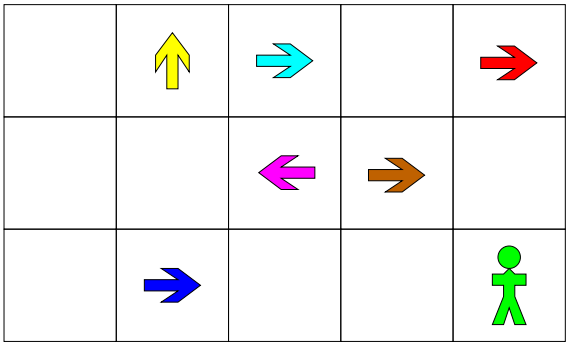
Instante 6



Instante 7



Instante 8



Instante 9

En las últimas páginas de este documento se incluyen otros ejemplos adicionales de posibles escenarios que también tienen solución.

2. Ejercicio opcional. Implemente una función `shortest` que tenga el mismo tipo que la función `tour` del ejercicio anterior, pero que calcule la lista de movimientos correspondiente a la ruta de escape más corta. Una vez más, puede que haya varias, pero sus longitudes deberían ser iguales, y es suficiente con devolver una cualquiera de

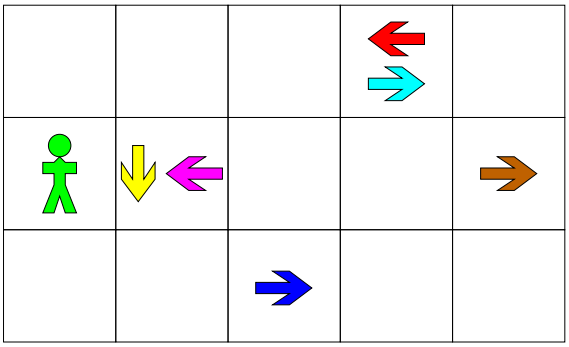
ellas. Y de nuevo, si no existe ninguna ruta de escape posible, la función debe lanzar la excepción `Not_found`. Realice su implementación en un archivo de texto con nombre `shortest.ml`.

Ejemplo de ejecución:

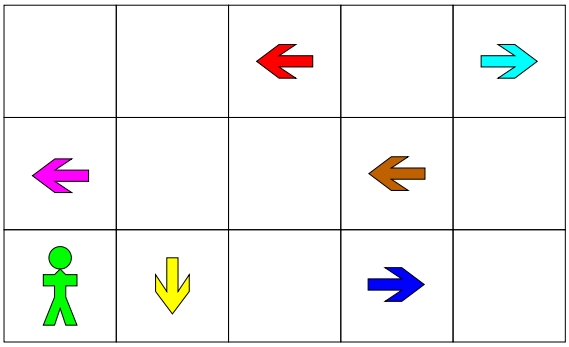
```
# shortest 3 5 [((1, 2), South); ((1, 3), East); ((1, 5), East);
                ((2, 3), West); ((2, 4), East); ((3, 2), East)];;
- : move list =
  [South; South; East; East; East; East]
```

Se trata del mismo escenario del ejemplo del ejercicio anterior. Sin embargo, puede observarse que el plan de movimientos obtenido es más corto (de hecho, su longitud es la mínima posible).

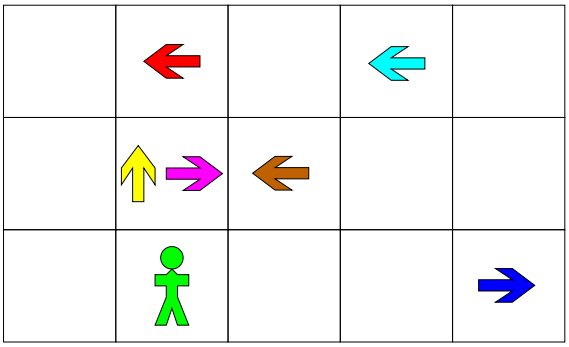
La evolución, en este caso, sería como sigue:



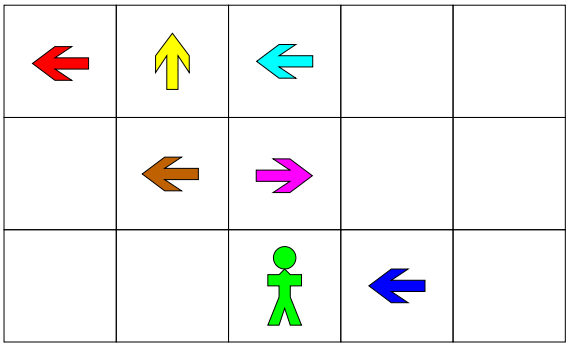
Instante 2



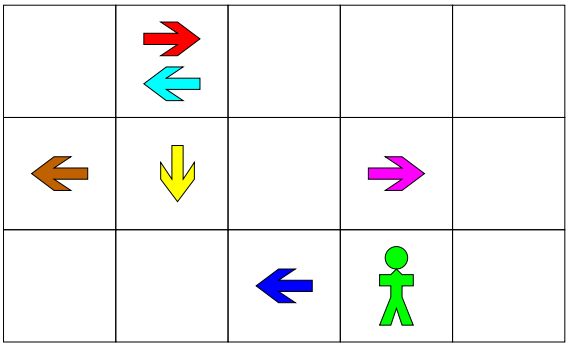
Instante 3



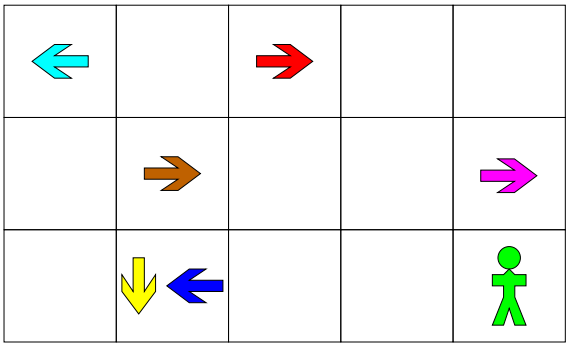
Instante 4



Instante 5



Instante 6



Instante 7

Otros escenarios que también tienen solución:

Escenario A (5 x 5 con 10 robots):

```
tour 5 5
  [((1, 3), East); ((1, 4), East); ((2, 3), North); ((2, 5), East);
   ((3, 5), North); ((4, 1), East); ((4, 2), West); ((4, 3), East);
   ((5, 3), West); ((5, 5), East)]
```

Escenario B (6 x 8 con 25 robots):

```
tour 6 8
  [((1, 2), East); ((1, 4), South); ((1, 5), East); ((1, 8), East);
   ((2, 3), South); ((2, 4), South); ((2, 8), East); ((3, 1), South);
   ((3, 2), East); ((3, 3), East); ((3, 4), East); ((3, 7), North);
   ((3, 8), East); ((4, 1), East); ((4, 2), West); ((4, 3), North);
   ((4, 5), South); ((4, 6), North); ((4, 7), West); ((5, 1), South);
   ((5, 4), North); ((5, 8), East); ((6, 2), North); ((6, 3), West);
   ((6, 5), East)]
```

Escenario C (8 x 12 con 50 robots):

```
tour 8 12
  [((1, 2), West); ((1, 4), West); ((1, 5), East); ((1, 6), North);
   ((1, 8), East); ((1, 9), West); ((1, 11), North); ((1, 12), West);
   ((2, 1), North); ((2, 4), North); ((2, 7), East); ((2, 8), South);
   ((2, 9), East); ((2, 10), North); ((3, 2), North); ((3, 3), South);
   ((3, 4), East); ((3, 8), South); ((3, 10), East); ((3, 11), South);
   ((4, 3), East); ((4, 5), North); ((4, 7), East); ((4, 9), North);
   ((4, 11), South); ((5, 3), South); ((5, 5), East); ((5, 6), South);
   ((5, 7), West); ((5, 10), North); ((6, 3), East); ((6, 5), North);
   ((6, 6), South); ((6, 7), North); ((6, 8), North); ((6, 10), South);
   ((6, 11), West); ((6, 12), West); ((7, 1), East); ((7, 2), North);
   ((7, 3), South); ((7, 5), South); ((7, 8), South); ((7, 10), East);
   ((7, 12), West); ((8, 3), North); ((8, 9), North); ((8, 10), West);
   ((8, 11), West); ((8, 12), South)]
```

Escenario D (10 x 15 con 80 robots):

```
tour 10 15
  [((1, 3), North); ((1, 4), North); ((1, 7), South); ((1, 8), South);
   ((1, 11), South); ((1, 12), East); ((1, 14), West); ((1, 15), West);
   ((2, 4), North); ((2, 7), South); ((2, 9), East); ((2, 10), South);
   ((2, 14), North); ((2, 15), North); ((3, 4), South); ((3, 6), South);
   ((3, 11), East); ((3, 12), West); ((3, 14), West); ((4, 3), West);
   ((4, 4), North); ((4, 5), West); ((4, 6), South); ((4, 8), North);
```

```
((4, 10), South); ((4, 11), South); ((4, 12), East); ((4, 14), East);
((4, 15), East); ((5, 1), West); ((5, 2), West); ((5, 3), East);
((5, 5), South); ((5, 6), East); ((5, 9), East); ((5, 11), South);
((5, 12), East); ((5, 14), South); ((5, 15), North); ((6, 2), West);
((6, 3), East); ((6, 4), East); ((6, 6), North); ((6, 9), East);
((6, 10), East); ((6, 11), East); ((6, 12), East); ((6, 13), East);
((6, 15), South); ((7, 1), North); ((7, 12), North); ((7, 13), South);
((7, 15), North); ((8, 1), South); ((8, 2), East); ((8, 4), East);
((8, 5), East); ((8, 6), South); ((8, 7), East); ((8, 9), East);
((8, 10), East); ((8, 14), West); ((8, 15), East); ((9, 1), South);
((9, 2), North); ((9, 3), West); ((9, 6), West); ((9, 9), East);
((9, 11), South); ((9, 14), North); ((9, 15), East); ((10, 1), West);
((10, 2), South); ((10, 9), North); ((10, 10), West); ((10, 11), South);
((10, 12), North); ((10, 13), North); ((10, 14), South); ((10, 15), South)]
```

Si implementa el ejercicio opcional, su función `shortest` debería resolver cada uno de los escenarios anteriores de la siguiente manera:

- Escenario A: en 10 movimientos.
- Escenario B: en 12 movimientos.
- Escenario C: en 26 movimientos.
- Escenario D: en 25 movimientos.