Lazy Queen

Sobre un tablero de ajedrez se han dispuesto varios peones y una dama. Se trata de saber si la dama puede comerse todos los peones de una “tacada” (es decir, sin pararse en ninguna de las casillas vacías)[1].

El problema es que nuestra dama es perezosa e intentará realizar esta maniobra con el menor coste posible para ella. El coste de moverse de una casilla a otra puede variar de un día a otro, o incluso según el estado de ánimo de la dama (pero permanece constante durante la “tacada”). En cualquier caso, este coste puede representarse con una función ***coste: int * int -> int * int -> int*** de modo que ***coste p1 p2*** daría el coste que supondría para la dama desplazarse de la casilla ***p1*** a la ***p2***. Puede asumirse que el coste nunca es negativo.

La reina nos pide que le calculemos un camino de coste mínimo para comerse todos los peones. La reina perezosa es paciente; pero no se equivoque, también es caprichosa y peligrosa; nuestra cabeza podría rodar.

Defina, en OCaml[2], una función

lazy_queen : (int * int -> int * int -> int) -> int * int -> (int * int) list -> (int * int) list

de modo que ***lazy_queen coste q_pos peones*** devuelva un recorrido de **coste mínimo** (según la función ***coste***) para comerse todos los peones de la lista ***peones*** si la dama está inicialmente situada en la casilla ***q_pos***. La lista que devuelve debe comenzar con la posición inicial de la dama y terminar con la posición del último peón comido.

Si ese recorrido no existe, la función debe provocar la excepción ***Not_found***.

Puede suponerse que los argumentos recibidos serán siempre válidos.

[1] Tenga en cuenta que una dama no puede comer a una pieza si se interpone otra en su camino.
[2] **La reina perezosa es también muy lista y funcional; así que ha prohibido el uso de valores mutables y cualquier módulo de la librería *Standard* que no sea el *List*.**

A modo ilustrativo considere, por ejemplo, las siguientes funciones de coste:

```
let costeh (_, j1) (_, j2) = abs (j2 - j1) (* anchos horizontes *)
let costev (i1, _) (i2, _) = abs (i2 - i1) (* caída libre *)
let coste1 (i1, j1) (i2 ,j2) = min (abs (i2 - i1)) (abs (j2 - j1)) (* las cuestas se hacen pesadas *)
let coste2 (i1, j1) (i2, j2) = max (abs (i2 - i1)) (abs (j2 - j1)) (* las cuestas no cuestan tanto *)
let coste3 (i1, j1) (i2, j2) = abs (i2 - i1) + abs (j2 - j1) (* bastante realista *)
let coste4 (i1, j1) (i2 ,j2) = abs (abs (i2 - i1) - abs (j2 - j1)) (* mejor en diagonal *)
let coste5 (i1, j1) (i2 ,j2) = max 0 (i2 - i1) + max 0 (j2 - j1) (* evita caer y tirar a la derecha *)
let coste6 (i1, j1) (i2 ,j2) = (* ¿es la diagonal racional? *)
    if i1 = i2 then abs (j2-j1) else if j1 = j2 then abs (i2-i1) else 3 * abs (j2-j1) / 2
```

Q	P	P
		P

```
# lazy_queen costeh (1,1) [(1, 3); (1,2); (3,3)];;
- : (int * int) list = [(1, 1); (1, 2); (1, 3); (3, 3)]
# lazy_queen costev (1,1) [(1, 3); (1,2); (3,3)];;
- : (int * int) list = [(1, 1); (1, 2); (1, 3); (3, 3)]
# lazy_queen coste1 (1,1) [(1, 3); (1,2); (3,3)];;
- : (int * int) list = [(1, 1); (1, 2); (1, 3); (3, 3)]
# lazy_queen coste2 (1,1) [(1, 3); (1,2); (3,3)];;
- : (int * int) list = [(1, 1); (1, 2); (1, 3); (3, 3)]
# lazy_queen coste3 (1,1) [(1, 3); (1,2); (3,3)];;
- : (int * int) list = [(1, 1); (1, 2); (1, 3); (3, 3)]
# lazy_queen coste4 (1,1) [(1, 3); (1,2); (3,3)];;
- : (int * int) list = [(1, 1); (3, 3); (1, 3); (1, 2)]
# lazy_queen coste6 (1,1) [(1, 3); (1,2); (3,3)];;
- : (int * int) list = [(1, 1); (1, 2); (1, 3); (3, 3)]
```

	P	P
	Q	
		P

```
# lazy_queen costeh (2,2) [(1, 3); (1,2); (3,3)];;
- : (int * int) list = [(2, 2); (1, 2); (1, 3); (3, 3)]
# lazy_queen coste1 (2,2) [(1, 3); (1,2); (3,3)];;
- : (int * int) list = [(2, 2); (1, 2); (1, 3); (3, 3)]
# lazy_queen coste3 (2,2) [(1, 3); (1,2); (3,3)];;
- : (int * int) list = [(2, 2); (1, 2); (1, 3); (3, 3)]
# lazy_queen coste4 (2,2) [(1, 3); (1,2); (3,3)];;
- : (int * int) list = [(2, 2); (3, 3); (1, 3); (1, 2)]
# lazy_queen coste5 (2,2) [(1, 3); (1,2); (3,3)];;
- : (int * int) list = [(2, 2); (3, 3); (1, 3); (1, 2)]
```

Q			P
	P		P

```
# lazy_queen costeh (1, 1) [(1, 4); (3, 4); (3, 2)];;
- : (int * int) list = [(1, 1); (1, 4); (3, 4); (3, 2)]
# lazy_queen coste1 (1, 1) [(1, 4); (3, 4); (3, 2)];;
- : (int * int) list = [(1, 1); (1, 4); (3, 4); (3, 2)]
# lazy_queen coste3 (1, 1) [(1, 4); (3, 4); (3, 2)];;
- : (int * int) list = [(1, 1); (1, 4); (3, 4); (3, 2)]
# lazy_queen coste4 (1, 1) [(1, 4); (3, 4); (3, 2)];;
- : (int * int) list = [(1, 1); (1, 4); (3, 2); (3, 4)]
# lazy_queen coste5 (1, 1) [(1, 4); (3, 4); (3, 2)];;
- : (int * int) list = [(1, 1); (1, 4); (3, 4); (3, 2)]
# lazy_queen coste6 (1, 1) [(1, 4); (3, 4); (3, 2)];;
- : (int * int) list = [(1, 1); (1, 4); (3, 4); (3, 2)]
```

Q		P	P
			P
P		P	

```
# lazy_queen costev (1,1) [(1, 4); (2, 4); (1, 3); (3, 3); (3, 1)];;
- : (int * int) list = [(1, 1); (1, 3); (1, 4); (2, 4); (3, 3); (3, 1)]
# lazy_queen coste1 (1,1) [(1, 4); (2, 4); (1, 3); (3, 3); (3, 1)];;
- : (int * int) list = [(1, 1); (3, 1); (3, 3); (1, 3); (1, 4); (2, 4)]
# lazy_queen coste4 (1,1) [(1, 4); (2, 4); (1, 3); (3, 3); (3, 1)];;
- : (int * int) list = [(1, 1); (3, 3); (2, 4); (1, 4); (1, 3); (3, 1)]
```

		P	P
		Q	P
P		P	

```
# lazy_queen coste1 (2,3) [(1, 4); (2, 4); (1, 3); (3, 3); (3, 1)];;
- : (int * int) list = [(2, 3); (2, 4); (1, 4); (1, 3); (3, 3); (3, 1)]
```

		P	P
			P
P		P	Q

```
# lazy_queen costeh (4,4) [(1, 4); (2, 4); (1, 3); (3, 3); (3, 1)];;
- : (int * int) list = [(4, 4); (2, 4); (1, 4); (1, 3); (3, 3); (3, 1)]
# lazy_queen costev (4,4) [(1, 4); (2, 4); (1, 3); (3, 3); (3, 1)];;
- : (int * int) list = [(4, 4); (3, 3); (3, 1); (1, 3); (1, 4); (2, 4)]
# lazy_queen coste1 (4,4) [(1, 4); (2, 4); (1, 3); (3, 3); (3, 1)];;
- : (int * int) list = [(4, 4); (2, 4); (1, 4); (1, 3); (3, 3); (3, 1)]
# lazy_queen coste2 (4,4) [(1, 4); (2, 4); (1, 3); (3, 3); (3, 1)];;
- : (int * int) list = [(4, 4); (3, 3); (2, 4); (1, 4); (1, 3); (3, 1)]
# lazy_queen coste3 (4,4) [(1, 4); (2, 4); (1, 3); (3, 3); (3, 1)];;
- : (int * int) list = [(4, 4); (2, 4); (1, 4); (1, 3); (3, 3); (3, 1)]
# lazy_queen coste4 (4,4) [(1, 4); (2, 4); (1, 3); (3, 3); (3, 1)];;
- : (int * int) list = [(4, 4); (3, 3); (2, 4); (1, 4); (1, 3); (3, 1)]
# lazy_queen coste5 (4,4) [(1, 4); (2, 4); (1, 3); (3, 3); (3, 1)];;
- : (int * int) list = [(4, 4); (2, 4); (1, 4); (1, 3); (3, 3); (3, 1)]
# lazy_queen coste6 (4,4) [(1, 4); (2, 4); (1, 3); (3, 3); (3, 1)];;
- : (int * int) list = [(4, 4); (3, 3); (2, 4); (1, 4); (1, 3); (3, 1)]
```

P			P
	Q		P
P			
	P		

```
# lazy_queen costev (2,2) [(1, 1); (1, 4); (2, 4); (3, 1); (4, 2)];;
- : (int * int) list = [(2, 2); (2, 4); (1, 4); (1, 1); (3, 1); (4, 2)]
# lazy_queen coste2 (2,2) [(1, 1); (1, 4); (2, 4); (3, 1); (4, 2)];;
- : (int * int) list = [(2, 2); (1, 1); (3, 1); (4, 2); (2, 4); (1, 4)]
# lazy_queen coste4 (2,2) [(1, 1); (1, 4); (2, 4); (3, 1); (4, 2)];;
- : (int * int) list = [(2, 2); (1, 1); (3, 1); (4, 2); (2, 4); (1, 4)]
# lazy_queen coste5 (2,2) [(1, 1); (1, 4); (2, 4); (3, 1); (4, 2)];;
- : (int * int) list = [(2, 2); (3, 1); (4, 2); (2, 4); (1, 4); (1, 1)]
# lazy_queen coste6 (2,2) [(1, 1); (1, 4); (2, 4); (3, 1); (4, 2)];;
- : (int * int) list = [(2, 2); (1, 1); (3, 1); (4, 2); (2, 4); (1, 4)]
```

P							
P							
	P	P		P			
	P						
		Q					
			P				
		P				P	P
			P	P			

```
# lazy_queen coste1 (5, 3) [(3, 2); (7, 3); (7, 8); (1, 1); (8, 4); (8, 5);
(2, 1); (3, 5); (4, 2); (6, 4); (3, 3); (7, 7)];;
- : (int * int) list =
[(5, 3); (4, 2); (3, 2); (2, 1); (1, 1); (3, 3); (3, 5); (8, 5); (8, 4);
 (6, 4); (7, 3); (7, 7); (7, 8)]
```

```
# lazy_queen coste3 (5, 3) [(3, 2); (7, 3); (7, 8); (1, 1); (8, 4); (8, 5);
(2, 1); (3, 5); (4, 2); (6, 4); (3, 3); (7, 7)];;
- : (int * int) list =
[(5, 3); (4, 2); (3, 2); (2, 1); (1, 1); (3, 3); (3, 5); (8, 5); (8, 4);
 (6, 4); (7, 3); (7, 7); (7, 8)]
```

```
# lazy_queen coste5 (5, 3) [(3, 2); (7, 3); (7, 8); (1, 1); (8, 4); (8, 5);
(2, 1); (3, 5); (4, 2); (6, 4); (3, 3); (7, 7)];;
- : (int * int) list =
[(5, 3); (4, 2); (3, 3); (7, 7); (7, 8); (7, 3); (6, 4); (8, 4); (8, 5);
 (3, 5); (3, 2); (2, 1); (1, 1)]
```

```
# lazy_queen coste6 (5, 3) [(3, 2); (7, 3); (7, 8); (1, 1); (8, 4); (8, 5);
(2, 1); (3, 5); (4, 2); (6, 4); (3, 3); (7, 7)];;
- : (int * int) list =
[(5, 3); (4, 2); (3, 2); (2, 1); (1, 1); (3, 3); (3, 5); (8, 5); (8, 4);
 (6, 4); (7, 3); (7, 7); (7, 8)]
```

									P
			Q					P	
P							p		
P	P		P		P				P
								P	
		P		P					
							P		
	P					P	P		

```
# lazy_queen costeh (2,4) [(4, 4); (6, 3); (3, 1); (10, 2); (1, 10); (10, 8); (6, 5);
(4, 6); (4, 10); (3, 8); (10, 7); (7, 8); (5, 9); (4, 1); (2, 9); (4, 2)];;
- : (int * int) list =
[(2, 4); (4, 4); (4, 6); (4, 10); (5, 9); (2, 9); (1, 10); (3, 8); (7, 8);
 (10, 8); (10, 7); (10, 2); (4, 2); (3, 1); (4, 1); (6, 3); (6, 5)]

# lazy_queen costev (2,4) [(4, 4); (6, 3); (3, 1); (10, 2); (1, 10); (10, 8); (6, 5);
(4, 6); (4, 10); (3, 8); (10, 7); (7, 8); (5, 9); (4, 1); (2, 9); (4, 2)];;
- : (int * int) list =
[(2, 4); (2, 9); (5, 9); (4, 10); (4, 6); (4, 4); (4, 2); (4, 1); (3, 1);
 (3, 8); (1, 10); (6, 5); (6, 3); (10, 7); (10, 2); (10, 8); (7, 8)]

# lazy_queen coste2 (2,4) [(4, 4); (6, 3); (3, 1); (10, 2); (1, 10); (10, 8); (6, 5);
(4, 6); (4, 10); (3, 8); (10, 7); (7, 8); (5, 9); (4, 1); (2, 9); (4, 2)];;
- : (int * int) list =
[(2, 4); (4, 4); (4, 6); (4, 10); (5, 9); (2, 9); (1, 10); (3, 8); (6, 5);
 (6, 3); (4, 1); (3, 1); (4, 2); (10, 2); (10, 7); (10, 8); (7, 8)]

# lazy_queen coste3 (2,4) [(4, 4); (6, 3); (3, 1); (10, 2); (1, 10); (10, 8); (6, 5);
(4, 6); (4, 10); (3, 8); (10, 7); (7, 8); (5, 9); (4, 1); (2, 9); (4, 2)];;
- : (int * int) list =
[(2, 4); (4, 4); (4, 6); (4, 10); (5, 9); (2, 9); (1, 10); (3, 8); (7, 8);
 (10, 8); (10, 7); (10, 2); (4, 2); (3, 1); (4, 1); (6, 3); (6, 5)]

# lazy_queen coste4 (2,4) [(4, 4); (6, 3); (3, 1); (10, 2); (1, 10); (10, 8); (6, 5);
(4, 6); (4, 10); (3, 8); (10, 7); (7, 8); (5, 9); (4, 1); (2, 9); (4, 2)];;
- : (int * int) list =
[(2, 4); (4, 6); (4, 4); (4, 2); (3, 1); (10, 8); (7, 8); (3, 8); (2, 9);
 (5, 9); (4, 10); (1, 10); (6, 5); (6, 3); (4, 1); (10, 7); (10, 2)]
```