Dijkstra

Dado un grafo dirigido con ***n*** nodos (numerados de ***0*** a ***n-1***) con pesos en las aristas, se puede usar una matriz ***n x n*** para representar las aristas y sus pesos. Supongamos que los pesos de las aristas están representados por valores enteros (no negativos). Esta matriz podría representarse en *OCaml* con un valor ***w : int option array array***. De este modo si ***w.(i).(j)*** es ***None*** eso significaría que no existe arista del nodo ***i*** al ***j,*** mientras que si es ***Some n***, existiría tal arista y su peso sería ***n***.

Se trata de implementar una función

dijkstra : int option array array -> int -> int -> (int * int list) option,

de modo que el valor de ***dijkstra w i j*** sea ***None*** si no existe camino del vértice ***i*** al ***j*** en el grafo ***w,*** y sea ***Some (c, p)*** si es ***c*** es el coste (peso) mínimo para ir de ***i*** a ***j*** en el grafo ***w*** y ***p*** es uno de los caminos de coste mínimo entre ***i*** y ***j***. El camino mínimo se representa con una lista de nodos que necesariamente ha de empezar con ***i*** y terminar con ***j*** y debe contener en medio todos los nodos (debidamente ordenados) por los que hay que pasar para llegar de ***i*** a ***j*** con el coste mínimo.

La función ***dijkstra*** no debe modificar en modo alguno el vector que recibe como argumento, y su comportamiento no debe depender de ningún valor externo a la función (aparte de su propio argumento). Debe comprobarse que la matriz sea cuadrada y que no contiene valores negativos (para que pueda aplicarse el *algoritmo de Dijkstra*) y que ***i*** y ***j*** son nodos válidos dentro del grafo (i.e. sus valores están entre *0* y *n-1*) en caso contrario debe activarse la excepción ***Invalid_argument "dijkstra"***.

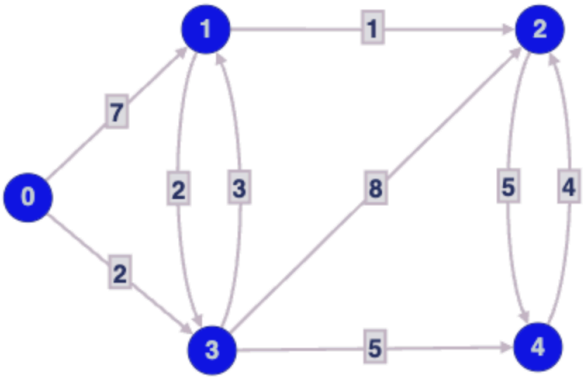
Para implementar este algoritmo se necesita tener alguna manera de representar una cola de prioridades mínimas. Aunque una simple lista de pares *(prioridad, valor)* podría servir para este fin, resultaría más conveniente disponer de una estructura más eficiente. El módulo ***MinPriQueue***, cuyo código fuente se adjunta a este enunciado, proporciona una implementación funcional de estas colas basada en montículos binarios, que resulta bastante eficiente. Puede compilar el código fuente de este módulo (`ocamlc -c minPrioQueue.mli minPrioQueue.ml`) y cargarlo en el compilador interactivo *ocaml* con

el comando `#load "minPrioQueue.cmo"`. En el archivo *minPrioQueue.mli* puede verse la interfaz de este módulo que contiene sólo un tipo de dato abstracto para representar las colas y tres valores de significado bastante obvio. Naturalmente, estos valores pueden usarse, si se desea, en la definición de la función ***dijkstra***.

Escriba la definición de la función ***dijkstra*** en un archivo ***dijkstra.ml*** que debe compilar sin errores con la orden

`ocamlc -c dijkstra.mli dijkstra.ml`

Ejemplo




```
# let w = let w = Array.make_matrix 5 5 None in
  w.(0).(1) <- Some 7; w.(0).(3) <- Some 2; w.(1).(2) <- Some 1; w.(1).(3) <- Some 2;
  w.(2).(4) <- Some 5; w.(3).(1) <- Some 3; w.(3).(2) <- Some 8; w.(3).(4) <- Some 5;
  w.(4).(2) <- Some 4; w;;
val w : int option array array =
  [|[|None; Some 7; None; Some 2; None|];
    [|None; None; Some 1; Some 2; None|];
    [|None; None; None; None; Some 5|];
    [|None; Some 3; Some 8; None; Some 5|];
    [|None; None; Some 4; None; None|]|]
# dijkstra w 0 5;;
Exception: Invalid_argument "dijkstra".
# dijkstra w 0 2;;
- : (int * int list) option = Some (6, [0; 3; 1; 2])
# dijkstra w 1 1;;
- : (int * int list) option = Some (0, [1])
# dijkstra w 0 2;;
- : (int * int list) option = Some (6, [0; 3; 1; 2])
# dijkstra w 0 4;;
- : (int * int list) option = Some (7, [0; 3; 4])
# dijkstra w 3 2;;
- : (int * int list) option = Some (4, [3; 1; 2])
# dijkstra w 2 3;;
- : (int * int list) option = None
```