Partie 3 – Chemin d'un état initial à l'état final

Dans cette partie :

- On ne s'intéresse qu'au parcours en largeur.
- On souhaite déterminer une succession d'états permettant d'aller d'un état initial à l'état final, (et non pas tous les nœuds visités) i.e. le chemin permettant d'aller de la racine (état initial) à l'état final.

3.1 L'objectif de cette partie est de définir la fonction (bfsPath s) qui, à un état initial s, associe une liste d'états permettant d'aller de s à l'état final ef. Ci-dessous 3 exemples d'utilisation de cette fonction

```
-- a = [(3,2),(1,1),(2,2),(2,1),(3,1),(3,3),(2,3),(1,3),(1,2)]

> length (bfs a)
17
> length (bfsPath a)_
4
> putStr (toStr (bfsPath a))
 1 3 4
 8 2 0
 7 6 5

 1 3 0
 8 2 4
 7 6 5

 1 0 3
 8 2 4
 7 6 5

 1 2 3
 8 0 4
 7 6 5

-- e = [(2,2),(1,1),(2,1),(3,1),(2,3),(3,2),(3,3),(1,3),(1,2)]

> length (bfs e)
27
> length (bfsPath e)
5
> putStr (toStr (bfsPath e))
 1 2 3
 8 0 5
 7 4 6

 1 2 3
 8 4 5
 7 0 6

 1 2 3
 8 4 5
 7 6 0

 1 2 3
 8 4 0
 7 6 5

 1 2 3
 8 0 4
 7 6 5
```

```
-- s = [(2,3),(1,2),(1,1),(3,1),(3,2),(3,3),(2,2),(1,3),(2,1)]

> length (bfs s)  ==> 51
> length (bfsPath s)   ==> 6
> putStr (toStr (bfsPath s))
 2 8 3
 1 6 4
 7 0 5

 2 8 3
 1 0 4
 7 6 5

 2 0 3
 1 8 4
 7 6 5

 0 2 3
 1 8 4
 7 6 5

 1 2 3
 0 8 4
 7 6 5

1 2 3
8 0 4
7 6 5
```

NB. Cette Partie 3 est volontairement rédigée de façon plus ouverte. Toutes les fonctions ne sont pas décrites par le menu, et il vous appartient de définir les fonctions auxiliaires dont vous auriez besoin.

3.2 Retrouver le chemin allant d'un état initial e0 à l'état final ef

Pour pouvoir retrouver le chemin allant d'un état initial s à l'état final ef, on ne mémorise plus des états seuls, mais des couples (état visité, état père de l'état visité). Lorsque le parcours s'arrête, la liste de couples obtenue est parcourue pour retrouver le chemin allant de l'état initial s à l'état final ef.

a) Compléter la définition de la fonction (parent s xs) qui détermine le père de l'état s dans la liste de couples xs.

```
parent :: State ->
parent s ((x, p) : cs)
        | (s == x)
        | otherwise
```

b) Dans la liste de couples (s, parent de s), tous les états ont un père, sauf l'état initial e0. Afin de permettre un traitement homogène de tous les états, on définit un état virtuel noté nil qui servira de père à l'état e0. En déduire la définition de la fonction (findSolnPath s xs) qui permet de calculer le chemin allant de l'état initial e0 à l'état s dans la liste de couples xs.

```
nil :: State
nil = []

findSolnPath ::
findSolnPath s cs
     | ??? == nil =
     | otherwise
```

3.3 **Déterminer le chemin** allant d'un état initial s à l'état final ef en utilisant un parcours en largeur.

a) Commenter la définition ci-dessous de la fonction (bfsPath s) qui, à un état initial s, associe la liste des états permettant d'aller d'un état s à l'état final ef.

```
bfsPath :: State -> [State]
bfsPath s = bfsSolv2 [ (s, nil) ]  [ ]
```

b) En s'inspirant de la fonction (bfsSolv xs ys) qui effectue un parcours en largeur (cf Section 2.2), expliquer quel doit être le rôle de la fonction (bfsSolv2 xs ys) et compléter sa définition :

```
bfsSolv2 :: [(State, State)] -> [(State, State)] -> [State]
bfsSolv2 [] _ =
bfsSolv2 ((s,p):os) cs
   |s == ef =
   |otherwise = bfsSolv2 (add_Bfs2                              ) ((s,p) : cs)
        where remAlreadyVisited xs ys =
              add_Bfs2 xs ys =
```