Partie 2 du DM – les 3 parcours

Cette seconde partie a pour but de comparer les 3 parcours arborescents (en largeur d'abord, en profondeur d'abord et en meilleur d'abord présentés lors du CM 08. On commence par étudier le parcours en largeur et on demande ensuite de s'en inspirer pour définir les 2 autres parcours.

2.1 Préliminaires

a) En prenant comme état initial l'état e, montrer que e apparaît comme un descendant de lui-même. Pour cela, on représentera la partie correspondante de l'arbre de recherche.

b) Si l'on ne prend pas garde à ce phénomène, à quel problème devra-t-on faire face ?

2.2 Etude du parcours en largeur *(Breadth First Search)*

La fonction (bfs s), définie ci-dessous, réalise un parcours en largeur à partir d'un état initial s.

```
bfs :: State -> [State]
bfs s = bfsSolv [s] []

bfsSolv :: [State] -> [State] -> [State]
bfsSolv (s : ss) visited
        | s == ef   = reverse (s : visited)
        | otherwise = bfsSolv (add_Bfs (remAlreadyVisited (successeurs s) visited) ss) (s : visited)
                where
                        remAlreadyVisited xs ys = [x | x <- xs, not (elem x ys)]
                        add_Bfs xs ys = ys ++ xs
```

Questions. Commenter cette définition en indiquant :

a. A quoi correspond le premier paramètre de bfsSolv ?

b. Pourquoi on a besoin du second paramètre visited ?

c. Pourquoi fait-on un retournement de la liste dans la première équation de bfsSolv ?

d. On considère la liste (bfs s) ; quel sera son premier élément, quel sera son dernier élément ? que représente la longueur de cette liste ?

e. Commenter la définition de la fonction (add_Bfs xs ys) en indiquant comment sont gérés les successeurs d'un sommet s. S'agit-il d'une gestion de type FIFO (first in, first out) ou de type LIFO (last in, first out) ?

```
-- Premiers exemples de test du parcours en largeur

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

> length (bfs e)   ==>27

> length (bfs s)   ==> 51

> length (bfs ef)   ==> 1
```

Indications : N'hésitez pas à tester la fonction (bfs s). Prenez des exemples demandant un nombre faible de déplacements pour être résolus, commencer par 2, puis 3, puis 4, etc

Autre exemple de test du parcours en largeur

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

> putStr (toString a)
1 3 4
8 2 0
7 6 5

> bfs a
[[(3,2),(1,1),(2,2),(2,1),(3,1),(3,3),(2,3),(1,3),(1,2)],[(2,2),(1,1),(3,2),(2,1),(3,1),(3,3),(2,3),(1,3),(1,2)],[(3,1),(1,1),(2,2),(2,1),(3,2),(3,3),(2,3),(1,3),(1,2)],[(3,3),(1,1),(2,2),(2,1),(3,1),(3,2),(2,3),(1,3),(1,2)],[(2,1),(1,1),(3,2),(2,2),(3,1),(3,3),(2,3),(1,3),(1,2)],[(2,3),(1,1),(3,2),(2,1),(3,1),(3,3),(2,2),(1,3),(1,2)],[(1,2),(1,1),(3,2),(2,1),(3,1),(3,3),(2,3),(1,3),(2,2)],[(2,1),(1,1),(2,2),(3,1),(3,2),(3,3),(2,3),(1,3),(1,2)],[(2,3),(1,1),(2,2),(2,1),(3,1),(3,2),(3,3),(1,3),(1,2)],[(1,1),(2,1),(3,2),(2,2),(3,1),(3,3),(2,3),(1,3),(1,2)],[(3,1),(1,1),(3,2),(2,2),(2,1),(3,3),(2,3),(1,3),(1,2)],[(3,3),(1,1),(3,2),(2,1),(3,1),(2,3),(2,2),(1,3),(1,2)],[(1,3),(1,1),(3,2),(2,1),(3,1),(3,3),(2,2),(2,3),(1,2)],[(1,1),(1,2),(3,2),(2,1),(3,1),(3,3),(2,3),(1,3),(2,2)],[(1,3),(1,1),(3,2),(2,1),(3,1),(3,3),(2,3),(1,2),(2,2)],[(1,1),(2,1),(2,2),(3,1),(3,2),(3,3),(2,3),(1,3),(1,2)],[(2,2),(1,1),(2,1),(3,1),(3,2),(3,3),(2,3),(1,3),(1,2)]]

> length (bfs a)
17

> putStr (toStr (bfs a))

1 3 4
8 2 0
7 6 5

1 3 4
8 0 2
7 6 5

1 3 0
8 2 4
7 6 5

1 3 4
8 2 5
7 6 0

1 0 4
8 3 2
7 6 5

1 3 4
8 6 2
7 0 5

1 3 4
0 8 2
7 6 5

1 0 3
8 2 4
7 6 5

(suite haut colonne de droite)

1 3 4
8 2 5
7 0 6

0 1 4
8 3 2
7 6 5

1 4 0
8 3 2
7 6 5

1 3 4
8 6 2
7 5 0

1 3 4
8 6 2
0 7 5

0 3 4
1 8 2
7 6 5

1 3 4
7 8 2
0 6 5

0 1 3
8 2 4
7 6 5

1 2 3
8 0 4
7 6 5

2.3 Parcours en profondeur *(Depth First Search)*

Questions.

Montrer que l'on peut en déduire le parcours en profondeur (dfs s) en reprenant le schéma de programme de la section 2.2, et en remplaçant add_Bfs par add_Dfs que l'on définira.

Commenter la définition de la fonction (add_Bfs xs ys) en indiquant comment sont gérés les successeurs d'un sommet s. S'agit-il d'une gestion de type FIFO ou de type LIFO ?

```
dfs :: State -> [State]
dfs s = dfsSolv [s] []

dfsSolv :: [State] -> [State] -> [State]
dfsSolv (s : ss) visited
   | s == ef       = reverse (s : visited)
   | otherwise  = dfsSolv (add_Dfs (remAlreadyVisited (successeurs s) visited) ss) (s : visited)
        where
              remAlreadyVisited xs ys = [x | x <- xs, not (elem x ys)]
              add_Dfs xs ys =
```

Prenons comme exemples les états s et a :

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

> putStr (toString s)
 2 8 3
 1 6 4
 7 0 5

> putStr (toString a)
 1 3 4
 8 2 0
 7 6 5

> length (dfs s)   ==> 49 563

> length (dfs e)   ==> 75 029

> length (dfs a)   ==> 177 651

> length (dfs ef)   ==> 1
```

NB. Pour tester vous-même la fonction (dfs s), prenez des exemples où ef figure dans la branche la plus à gauche, commencer par une profondeur 2, puis 3, puis 4. En effet, comme le montrent les 2 exemples ci-dessus, le parcours en profondeur peut construire un nombre très élevé de nœuds, même si on peut résoudre très facilement le problème.

2.4 Parcours en meilleur d'abord *(Best First Search)*

Question. Montrer que l'on peut déduire le parcours en meilleur d'abord (bestfs s) en reprenant le schéma de programme de la section 2.2, et en remplaçant add_Bfs par add_Bestfs que l'on définira en utilisant comme fonction heuristique une fonction (h s) quelconque :

- pour tester avec h1 (somme des distances MANHATTAN), il suffira de définir : h=h1
- pour tester avec h2 (somme des distances de HAMMING), il suffira de définir : h=h2

Complétez les définitions suivantes :

```
-- pour définir la fonction heuristique h, choisir l'une des 2 définitions ci-dessous
-- h = h1
-- h = h2

bestfs :: State -> [State]
bestfs s = bestfsSolv [s] []

bestfsSolv :: [State] -> [State] -> [State]
bestfsSolv (s:ss) visited
    | s==ef      = reverse (s:visited)
    | otherwise = bestfsSolv (add_Bestfs (remAlreadyVisited (successeurs s) visited) ss) (s:visited)
 where
        remAlreadyVisited xs ys = [x | x <-xs, not (elem x ys)]

        add_Bestfs  []
        add_Bestfs  (x : xs)

        insert
        insert
            |
            | otherwise =
```

Indication. La fonction (insert s xs) insère l'état s à sa place dans la liste d'états xs ordonnée par valeurs croissante de la fonction h.

Ci-dessous, un exemple avec h=h2 (i.e. somme des distances de HAMMING)

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

-- xs est une liste d'états ordonnée suivant les valeurs croissantes de h
xs :: [State]
xs = [[(2,2),(1,1),(2,1),(3,1),(3,2),(3,3),(2,3),(1,3),(1,2)], [(2,1),(1,1),(2,2),(3,1),(3,2),(3,3),(2,3),(1,3),(1,2)],
[(3,1),(1,1),(2,2),(2,1),(3,2),(3,3),(2,3),(1,3),(1,2)], [(3,2),(1,1),(2,2),(2,1),(3,1),(3,3),(2,3),(1,3),(1,2)]]

> map h xs
[0,1,2,3]

> h s
4

> (insert s xs)
[[(2,2),(1,1),(2,1),(3,1),(3,2),(3,3),(2,3),(1,3),(1,2)],[(2,1),(1,1),(2,2),(3,1),(3,2),(3,3),(2,3),(1,3),(1,2)],[(3,1),(1,1),(2,
2),(2,1),(3,2),(3,3),(2,3),(1,3),(1,2)],[(3,2),(1,1),(2,2),(2,1),(3,1),(3,3),(2,3),(1,3),(1,2)],[(2,3),(1,2),(1,1),(3,1),(3,2),(3,
3),(2,2),(1,3),(2,1)]]

> map h (insert s xs)
[0,1,2,3,4]
```

-- Exemples avec h=h1 (somme distances MANHATTAN)

```
> a
 [(3,2),(1,1),(2,2),(2,1),(3,1),(3,3),(2,3),(1,3),(1,2)]
> e
[(2,2),(1,1),(2,1),(3,1),(2,3),(3,2),(3,3),(1,3),(1,2)]
> s
[(2,3),(1,2),(1,1),(3,1),(3,2),(3,3),(2,2),(1,3),(2,1)]

> length (bestfs a)
4
> bestfs a
[[(3,2),(1,1),(2,2),(2,1),(3,1),(3,3),(2,3),(1,3),(1,2)],[(3,1),(1,1),(2,2),(2,1),(3,2),(3,3),(2,3),(1,3),(1,2)],[(2,1),(1,
1),(2,2),(3,1),(3,2),(3,3),(2,3),(1,3),(1,2)],[(2,2),(1,1),(2,1),(3,1),(3,2),(3,3),(2,3),(1,3),(1,2)]]
> putStr (toStr (bestfs 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

> length (bestfs e)   ==> 5

> length (bestfs s)  ==> 6
```

-- Rappel des résultats obtenus avec bfs et dfs

--

```
> length (bfs a)   ==> 17

> length (bfs e)   ==> 27

> length (bfs s)   ==> 51

> length (dfs a)   ==> 177 651

> length (dfs e)   ==> 75 029

> length (dfs s)   ==> 49 563
```