♦ Concours Agro-Véto TB – 2018 – Informatique

Corrigé

1. Préliminaires

a. La matrice qui correspond au labyrinthe est `maze4` car l'entrée se trouve aux indices $(0,0)$ et la sortie aux indices $(6,2)$ et `maze4` est la seule matrice dont le terme d'indices $(0,0)$ est 2 et le terme d'indices $(6,2)$ est 3.

Remarque. En fait, aucune des 4 matrices n'est correcte car la troisième colonne de `maze4` est erronée... Les coefficients d'indices $2,2$, $2,3$ et $2,4$ devraient être des `0` et non pas des `1`.

b. Le chemin doit commencer en $[0,0]$ et se terminer en $[6,2]$ donc la seule possibilité est le `chemin1`.

c.

```
def entreeSortie(mat):
  n, p = np.shape(mat)
  L = [[0,0], [0,0]]
  for i in range(n):
    for j in range(p):
      if mat[i,j] == 2:
        L[0] = [i,j]
      if mat[i,j] == 3:
        L[1] = [i,j]
  return L
```

2. Recherche de chemin dans un labyrinthe parfait

a.

```
def vide(lab, x, y):
  n, p = np.shape(lab)
  if 0 <= x < n and 0 <= y < p and lab[x,y] not in
     [1,4]:
    return True
  else:
    return False
```

En fait, il vaudrait mieux écrire le code suivant :

```
def vide(lab, x, y):
  n, p = np.shape(lab)
  return 0 <= x < n and 0 <= y < p and lab[x,y] not
     in [1,4]
```

b.

```
def voisinsNonVisites (lab, i, j):
  liste = []
  cases = [[i+1,j], [i-1,j], [i,j-1], [i,j+1]]
  for k in range (4):
    x = cases [k][0]
    y = cases [k][1]
    if vide(lab, x, y):
      liste.append([x,y])
  return liste
```

c. Pour supprimer le dernier élément de la liste `chemin`, on peut utiliser la méthode `.pop` en écrivant `chemin.pop()` ou `chemin.pop(-1)`.

d.

```
def cheminSolution (lab):
  entree, sortie = entreeSortie(lab)
  i, j = entree
  chemin = [entree]
  lab[i,j] = 4
  while [i, j] != sortie:
    liste = voisinsNonVisites(lab, i, j)
    if len(liste) > 0:
      i, j = rd.choice(liste)
      lab[i,j] = 4
      chemin.append([i,j])
    else:
      chemin.pop()
      i, j = chemin[-1]
  return chemin
```

3. Compression de chemin

a. Le chemin compressé du labyrinthe de la figure 1 est

```
[[7,0], [0,0], [0,2], [4,2], [4,10], [0,10]]
```

b.

```
def direction(chemin,k):
  return [chemin[k+1][0] - chemin[k][0], chemin[k
     +1][1] - chemin[k][1]]
```

c.

```
def compression(chemin):
  dir = direction(chemin,0)
  chemin_compresse = [chemin[0]]
  for k in range (1, len(chemin)-1):
    if dir != direction(chemin, k):
      chemin_compresse.append(chemin[k])
      dir = direction (chemin, k)
  chemin_compresse.append (chemin[-1])
  return chemin_compresse
```