

## Comment sortir d'un labyrinthe ?

Dans ce TD vous allez écrire un algorithme permettant de sortir d'un labyrinthe. Ce labyrinthe sera représenté sous forme de matrice 2D (coordonnées discrètes). Le point de départ sera toujours de coordonnées (0, 0) et le point d'arrivée le point en bas à droite (dans le cas de l'exemple donné ci-dessous (9, 9)); les murs de valeur 1 sont infranchissables; on ne peut pas non plus sortir de la matrice. On peut aller dans 8 directions possibles : haut, bas, gauche, droite, et leurs diagonales correspondantes.

**Exercice 1.1** – Dans un premier temps recopiez le programme ci-dessous qui charge le labyrinthe et une fonction d'affichage; identifiez un des chemins conduisant à la sortie (fichier `code/load.py`) :

```
labyrinthe = [[0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
               [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
               [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
               [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
               [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
               [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
               [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
               [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
               [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
               [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
               [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]

def affiche_labyrinthe(l):
    print('\n'.join([''.join(['{:4}'.format(item) for item in row])
                      for row in l]))
```

**Exercice 1.2** – Proposez un algorithme de décision qui indique si il existe un chemin reliant l'entrée et la sortie. Une approche simple est le parcours en largeur, avec un mécanisme de mémorisation afin d'éviter de re-parcourir les noeuds déjà parcouru.

**Exercice 1.3** – On sait qu'il existe un ou plusieurs chemins pour sortir, mais pas forcément lequel. Rajoutez une structure de données permettant de mémoriser le chemin parcouru, et une fonction d'affichage de ce chemin parcouru. Un chemin ressemble à la suite de couples comme suit :

```
chemin : [(0, 0), (1, 0), (2, 1), (3, 2), (4, 3), (5, 4),
          (5, 5), (6, 6), (7, 7), (8, 8), (9, 9)]

étapes : 11
```

**Exercice 1.4** – Enfin on a trouvé un chemin.. mais pas forcément le plus court ! Pour cela, améliorez votre algorithme afin de prendre en compte la minimisation de 1) la distance vers la sortie, et 2) la distance déjà parcourue. Conseil : utilisez la distance de Manhattan pour calculer la distance vers la sortie, pour la distance parcourue il s'agit

Vous pourrez utiliser le module `PriorityQueue` pour gérer une file de priorité qui renvoie la valeur minimale insérée avec votre information (fichier `code/priorite.py`) :

---

```
from queue import PriorityQueue
```

```
file_prio = PriorityQueue()  
file_prio.put((2, "Bob"))  
file_prio.put((1, "Alice"))
```

```
while not file_prio.empty():  
    print(file_prio.get())
```

---

**Exercice 1.5** – Illustrez vos différents algorithmes avec différents labyrinthes afin de tester (un labyrinthe vite, sans issue, sans mur, etc.) et de montrer leur efficacité (minimisation du nombre de cases visitées, parcours optimal).