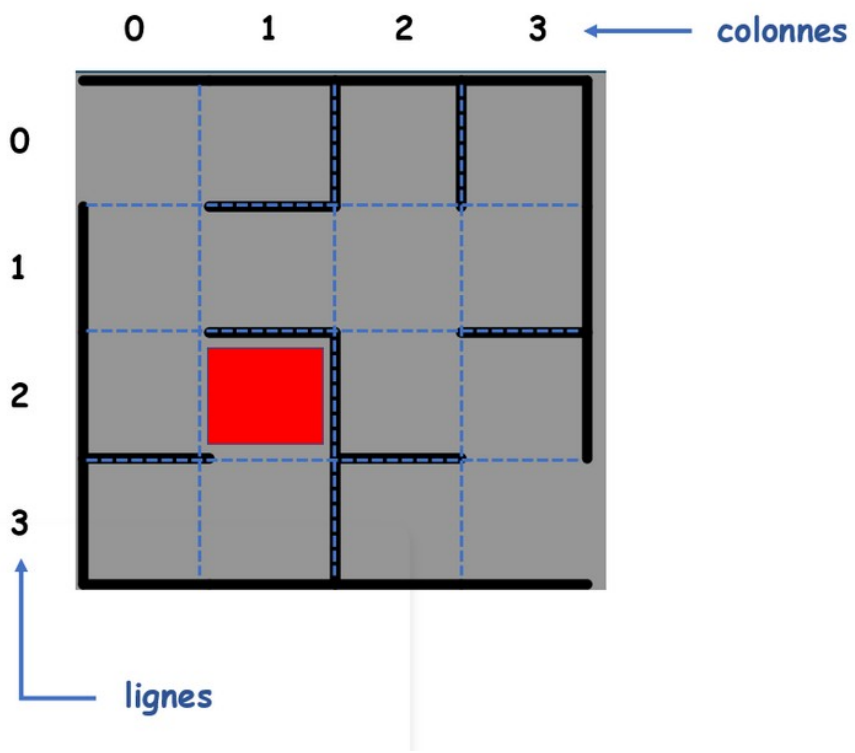
Mini projet résolution de labyrinthe

On choisit de coder les murs d'un labyrinthe avec l'implémentation suivante :

```
murs1 = [[[True,False,False,False],[True,True,True,False],[True,True,False,True],[True,True,False,True]],
        [[False,False,False,True],[True,False,True,False],[False,False,False,False],[False,True,True,True]],
        [[False,False,True,True],[True,True,False,False],[False,False,True,True],[True,True,False,False]],
        [[True,False,True,True],[False,True,True,False],[True,False,True,True],[False,False,True,False]]]
```



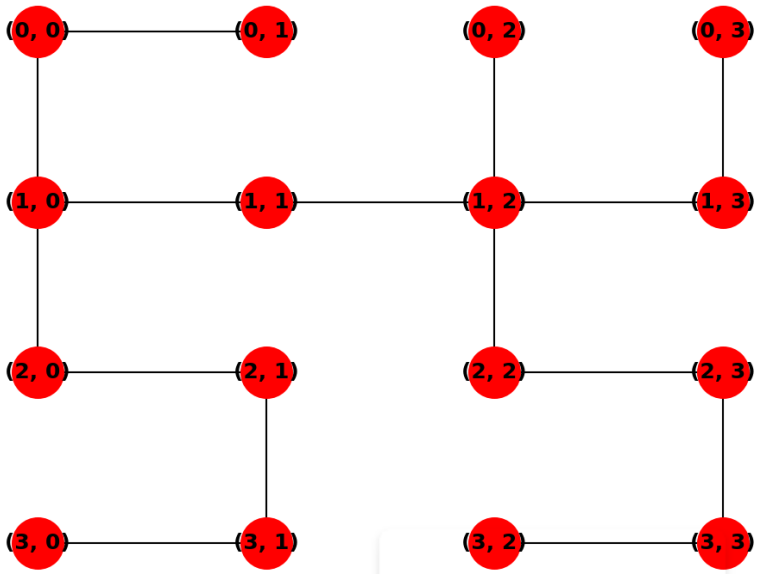
Description des éléments de la liste représentant une `case`

ici c'est la case murs[2][1] et elle est égale à

`[True, True, False, False] :`

- Indice 0 : valeur `True` : booléen qui indique ici la **présence** d'un mur **en haut** de la case.
- Indice 1 : valeur `True` : booléen qui indique ici la **présence** d'un mur **à droite** de la cas
- Indice 2 : valeur `False` : booléen qui indique ici **la non présence** de mur **en bas** de la case.
- Indice 3 : valeur `False` : booléen qui indique ici **la non présence** de mur **à gauche** de la case.

1) Implémenter la méthode **creer_graphe** qui permet à partir de la connaissance des murs d'établir le graphe des déplacements possibles ci-dessous.



G = {(0, 0): [(0, 1), (1, 0)], (0, 1): [(0, 0)], (0, 2): [(1, 2)], (0, 3): [(1, 3)], (1, 0): [(0, 0), (1, 1), (2, 0)], (1, 1): [(1, 2), (1, 0)], (1, 2): [(0, 2), (1, 3), (2, 2), (1, 1)], (1, 3): [(0, 3)], (2, 0): [(1, 0), (2, 1)], (2, 1): [(3, 1), (2, 0)], (2, 2): [(1, 2), (2, 3)], (2, 3): [(3, 3), (2, 2)], (3, 0): [(3, 1)], (3, 1): [(2, 1), (3, 0)], (3, 2): [(3, 3)], (3, 3): [(2, 3), (3, 2)]}

Chaque sommet est nommé par ses coordonnées.
La fonction **dessiner_graphe_labyrinthe** est déjà implémentée.dans le fichier **dessiner_graphe.py**.
Aller voir son code pour comprendre les paramètres attendus.

2) Implémenter la méthode **déterminer_chemin** :

Pour cela effectuez un parcours en largeur, depuis le départ (0,0) et arrêtez vous dès que l'arrivée (3,3) est atteinte. Pour pouvoir retrouver le chemin, notez dans un dictionnaire **pere** chaque fois que vous enfilez un sommet **a** adjacent à un sommet **s** celui qui était son père (pere[a] =s). Cela vous permettra de déterminer le chemin arrivant à (3 ,3) en remontant la filiation jusqu'à (0,0).
Il ne vous restera alors qu'à inverser l'ordre pour obtenir le chemin.

```
Python 3.7.9 (bundled)
>>> %Run labyrinthe.py
  [(0, 0), (1, 0), (1, 1), (1, 2), (2, 2), (2, 3), (3, 3)]
>>>
```