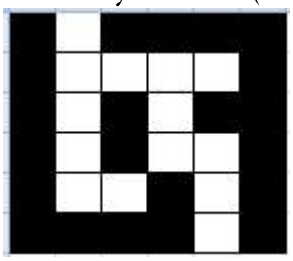
Un second exemple - Résoudre un labyrinthe

On représentera le labyrinthe à l'aide d'un tableau (liste de listes).

Le labyrinthe (noir pour les murs...) (0 pour les murs...)



```
laby=[[0,1,0,0,0,0],
      [0,1,1,1,1,0],
      [0,1,0,1,0,0],
      [0,1,0,1,1,0],
      [0,1,1,0,1,0],
      [0,0,0,0,1,0]]
```

- On repère une case par ses coordonnées (i,j), on y accède dans le tableau par `laby[i][j]`
- L'entrée se fait par la case (0,1) et la sortie par la case (5,4)

? QUESTION 1:

Compléter le code pour obtenir le nombre de lignes et le nombre de colonnes de ce tableau:

```
lignes=len(     )
colonnes=len(     )
```

Nous aurons besoin de créer une copie de ce tableau pour pouvoir travailler dessus sans altérer le tableau `laby`

À FAIRE 4:

Tester ce code

```
from copy import deepcopy

laby=[[0,1,0,0,0,0],
      [0,1,1,1,1,0],
      [0,1,0,1,0,0],
      [0,1,0,1,1,0],
      [0,1,1,0,1,0],
      [0,0,0,0,1,0]]

T=deepcopy(laby)
T[3][2]='hello'

for ligne in laby:
    print(ligne)
print("----------------")
for ligne in T:
    print(ligne)
```

Objectif: L'objectif est d'écrire un programme qui détermine s'il existe un chemin de l'entrée vers la sortie en se déplaçant vers le haut, le bas, la gauche ou la droite (mais pas en diagonale).

Étape 1 :

À faire 5:

Voici une fonction qui prend en paramètre un tableau `T` et un tuple `v`

```
def voisins(T,v):
    V=[]
    i,j=v[0],v[1]
    for a in (-1,1):
        if 0<=i+a<lignes:
            if T[i+a][j]==1:
                V.append((i+a,j))
        if 0<=j+a<colonnes:
            if T[i][j+a]==1:
                V.append((i,j+a))
    return V
```

Question 2:

Que retourne cette fonction ?

.....
.....
.....
.....
.....
.....

Question 3:

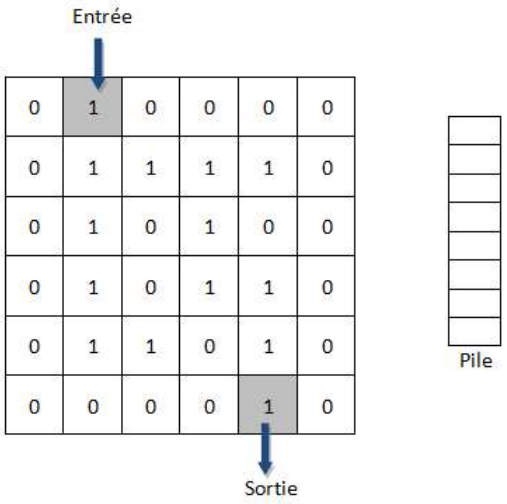
Expliquer l'affichage provoqué par cette instruction : `print(voisins(laby,(0,1))`

.....
.....
.....
.....
.....
.....

Spécifier la fonction

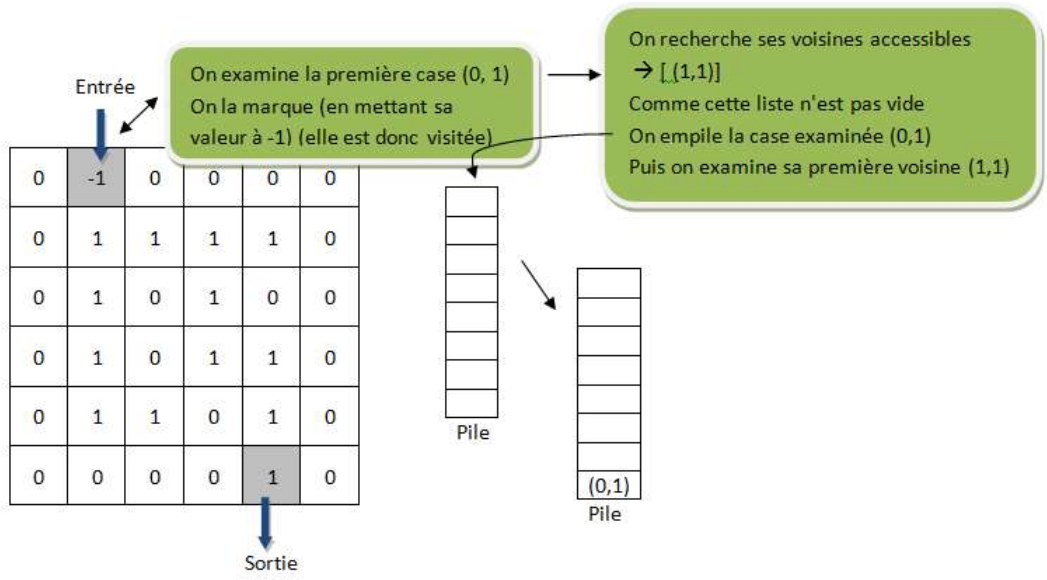
Étape 2 : parcours du labyrinthe

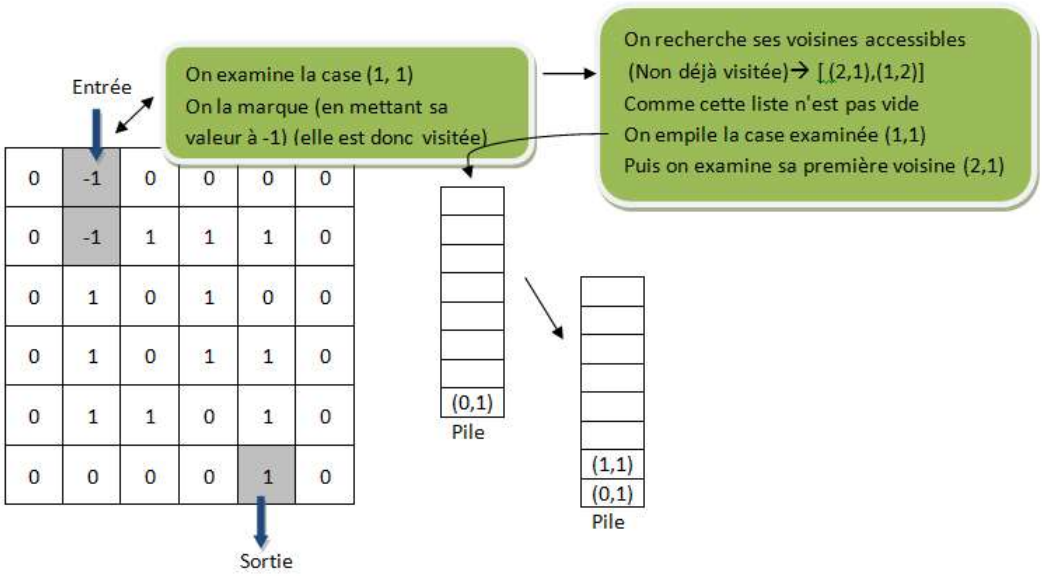
L'idée est de parcourir le labyrinthe depuis l'entrée, en utilisant une pile pour stocker le chemin, pour pouvoir dépiler lorsque le chemin n'aboutit pas et redémarrer sur une autre voie..
Un schéma sera sans doute plus efficace qu'un long discours...



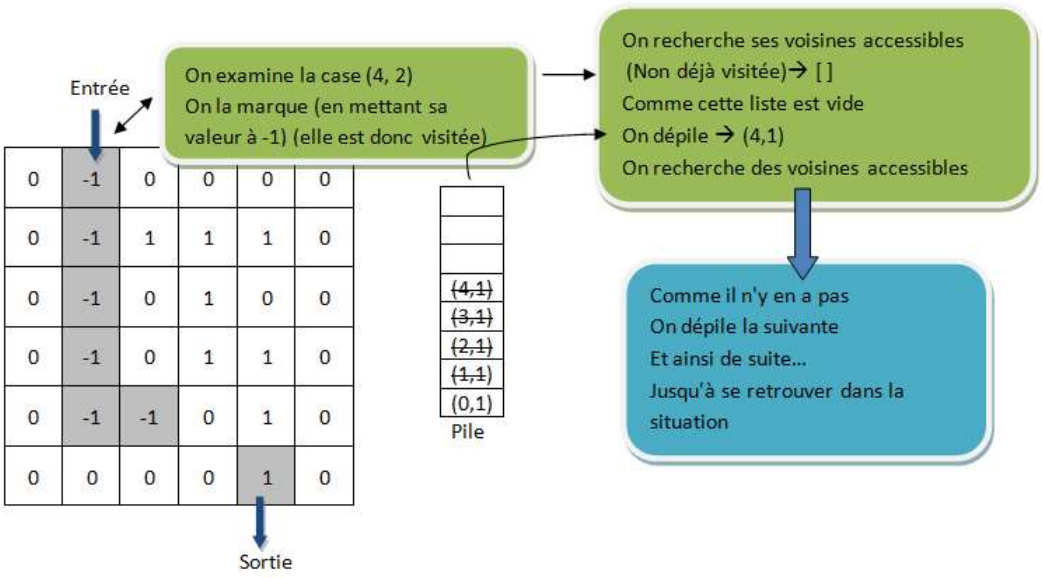
On dispose d'une copie de notre tableau et d'une pile vide

Le processus...

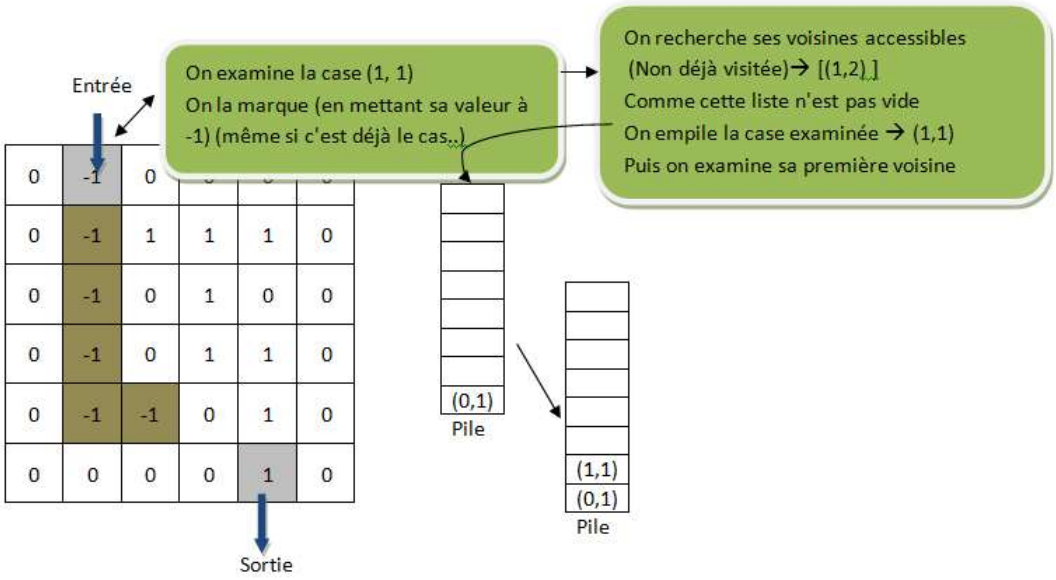




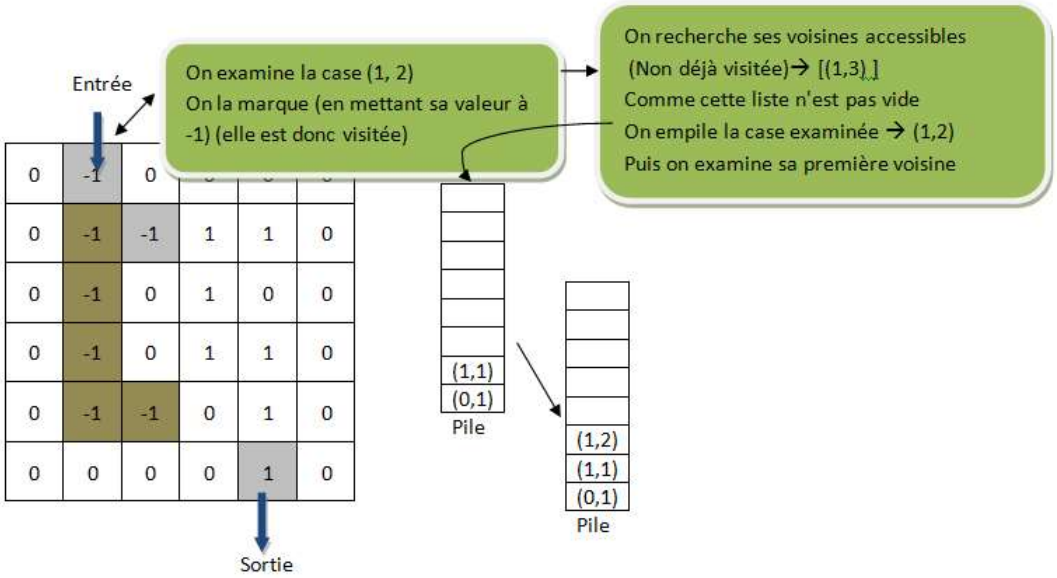
Et ainsi de suite jusqu'à ce que l'on tombe sur une impasse..



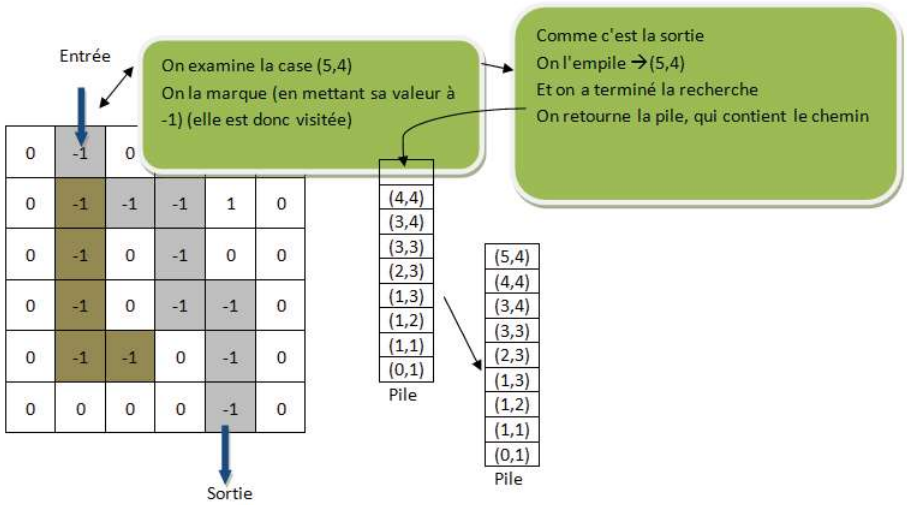
On examine la dernière case dépilée...



On poursuit l'exploration...



Et ainsi de suite jusqu'à la sortie...



L'algorithme :

À faire 6:

Compléter l'algorithme :

```
fonction parcours( laby , entree, sortie)
    T <-- une copie de ........
    p <-- Pile()
    v <-- entree
    on met à .... la valeur de la case.... dans ...
    recherche <--True
    tant que recherche est vrai
        vois <-- voisins (...,...)
        si la liste ... est vide
            si la pile vide --> renvoyer False
            sinon v <--.......................
        sinon
            on ........ p  avec ....
            v <-- le 1er ...................
            on met à .... la valeur de la case.... dans ...
            si v = ......
                On ........ p  avec......
                recherche <--.......
    return ....
```

Question 4:

Si au cours de l'exécution la pile se trouve vide, que cela signifie-t-il?

.....

.....

.....

Réaliser le programme (mettre en évidence le chemin) et tester le avec d'autres labyrinthes

Prolongement possible : Pour un projet on pourrait imaginer un programme qui génère un labyrinthe et qui le résout...