*Rappel sur les listes et les piles* :

Une pile sur le principe LIFO ( LAST IN FIRST OUT) : le dernier élément ajouté est le premier à être retiré. Ex : pop() en python

Une basée sur le principe FIFO ( FIRST-IN-FIRST-OUT) : le premier élément ajouté est le premier à être retiré. Ex ; pop(0).

Dans le cas d’un gestion de Liste en tant que Pile et file, on préférera utiliser une deque ( double-ended queue) : combine le principe de la pile (popRight) et de la file ( popLeft)

*Utilisation de l’algo BFS :*

L’algo Breadth First Search ou parcours en largeur permet le parcours d’un graphe ou d’un arbre de la manière suivante :

* On commence par explorer un nœud source, puis ses successeurs, puis les successeurs non explorés des successeurs, etc. Il permet de calculer la distance de tous les nœuds depuis un nœud source dans un graphe non pondéré.

Implémentation :

# init d'un set, et d'une deque

# une deque est une liste qui est à la fois une pile et une file

def BFS(graphe, root)

visited, queue = set(), collections.deque([root])

visited.add(root)

while queue

# le vertex est le noeud courant = premier de la liste

vertex = queue.popleft()

print(str(vertex) + "", end="")

for neighboor in graph[0]: # [1,2]

# Si les noeuds ne sont pas déja visités

if neighboor not in visited:

# on les ajoute aux noeuds visités

visited.add(neighboor)

# on les ajoute à deque :

queue.append(neighboor)

On passe à la fonction le graphe ou tableau, ainsi que le nœud racine

* On init un set qui correspond aux nœuds visités, et une deque à laquelle on passe le nœud racine.
* On ajoute le nœud racine à la liste des nœuds visités.
* Tant que la liste deque de nœuds n’est pas vide :
* On pop le premier nœud de la deque
* Dans le graphe, le noeud racine est associé à la liste de ses nœuds adjacents.
* On passe donc le nœud qu’on a popé dans le graphe, puis on parcourt la liste des nœuds adjacents
* Si le nœud n’est pas dans la liste des nœuds visités, alors on l’ajoute.
* On l’ajoute également à la deque

Dans la pratique :

* On fait un set ou dico contenant la liste des nœuds ainsi que les nœuds adjacents :
* graph = {0: [1, 2], 1: [2], 2: [3], 3: [1, 2]}
* Puis n passe ce graphe à la fonction BFS, et on modifie l’état des cellules en fonction de la logique de notre code.