TD 2

# Exercice 1

## Question 1

Graphe 1

|  |  |
| --- | --- |
| Pile | Nœud Courant |
| [2] | 2 |
| [0] | 0 |
| [1,3] | 3 |
| [1] | 1 |
| [4,5,6] | 6 |
| [4,5] | 5 |
| [4] | 4 |

Graphe 2

|  |  |
| --- | --- |
| Pile | Nœud Courant |
| [2] | 2 |
| [1] | 1 |
| [0,4,5] | 5 |
| [0,4,3] | 3 |
| [0,4] | 4 |
| [0,6] | 6 |
| [0] | 0 |

## Question 2

Graphe 1

|  |  |
| --- | --- |
| Pile | Nœud Courant |
| [1] | 1 |
| [2,3] | 3 |
| [2,1] | 1 |
| [2,3] | 3 |
| [2,1] | 1 |

Ça boucle …

Graphe 2

|  |  |
| --- | --- |
| Pile | Nœud Courant |
| [1] | 1 |
| [2,3] | 3 |
| [2,2,1] | 1 |

Ça boucle …

## Question 2.1

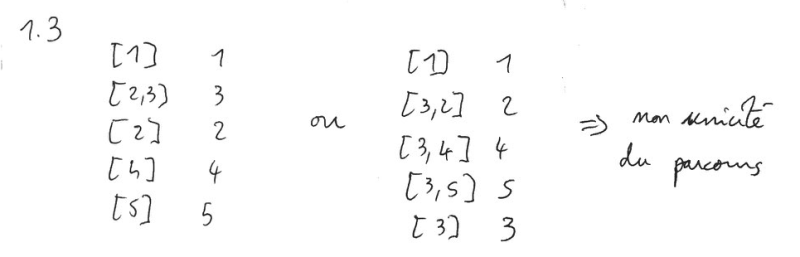
On peut observer que l’on boucle au niveau des sommets car on a un graphe non orienté.

## Question 2.2

Il faut modifier l’algo de sorte à ce que les graphes déjà visités ne se rajoute pas à la pile.

[Voir TD.py](TD.py)

Exécution du premier graphe de la question 2 avec l’algo parcours\_profondeur

[](https://cdn.discordapp.com/attachments/687981627929788429/692330092545966140/unknown.png)

# Exercice 2

## Question 2

*def* accessible (*graphe*, *sommet1*, *sommet2*) :

return sommet2 in parcours\_profondeur(graphe, sommet1)

## Question 3

*def* nb\_sommets\_accessibles(*graphe*, *sommet*):

return len(parcours\_profondeur(graphe, sommet))

# Exercice 3

## Question 1

Pas si on part du début.

## Question 2

Oui je l’ai vu

## Question 3

*def* cycle (*g*, *depart*) :

pile = [depart]

atteint = {depart}

while (len(pile) > 0):

noeud\_courant = pile.pop()

print noeud\_courant

if (accessible(g, noeud\_courant, depart)) :

return True

for i in g[noeud\_courant] :

if (i not in atteint) :

pile.append(i)

atteint.add(i)

return False

# Exercice 4

## Question 1

*def* parcours\_prof\_chaque\_sommet (*graphe*):

pile = graphe.nodes()

atteint = *set*()

while (len(pile) > 0):

noeud\_courant = pile.pop()

if noeud\_courant not in atteint :

for i in parcours\_profondeur(graphe, noeud\_courant) :

atteint.add(i)

return atteint

## Question 2

*def* nb\_composante\_connexe (*graphe*):

pile = graphe.nodes()

atteint = *set*()

nb\_comp = 0

while (len(pile) > 0):

noeud\_courant = pile.pop()

if noeud\_courant not in atteint :

nb\_comp += 1

for i in parcours\_profondeur(graphe, noeud\_courant) :

atteint.add(i)

return nb\_comp

## Question 3

*def* taille\_max\_composante\_connexe (*graphe*):

pile = graphe.nodes()

atteint = *set*()

taille\_comp\_con = 0

taille\_max\_comp\_con = 0

while (len(pile) > 0):

noeud\_courant = pile.pop()

if noeud\_courant not in atteint :

taille\_comp\_con += len(parcours\_profondeur(graphe, noeud\_courant))

for i in parcours\_profondeur(graphe, noeud\_courant) :

atteint.add(i)

if taille\_comp\_con > taille\_max\_comp\_con :

taille\_max\_comp\_con = taille\_comp\_con

taille\_comp\_con = 0

return taille\_max\_comp\_con

# Exercice 5

## Question 1

1-2-4-5

1-3-2-4-5

## Question 2

5-4-2-3

## Question 3

*def* chemin (*graphe*, *u*, *v*):

pile = [(u,[u])]

atteint = {u}

while pile :

courant, chem = pile.pop()

for vois in graphe[courant] :

if not vois in atteint :

atteint.add(vois)

pile.append((vois, chem[vois]))

if vois == v :

return chem+[vois]

return None