TP 2025 graphes en `Python` : généralités et parcours

Dans ce TP, les graphes sont implémentés par des listes de listes d'adjacences.

```python
from __future__ import annotations
from collections import deque # pour le bfs
from typing import List, Sequence, Tuple, Deque
# ------------------------------------------------------------
# Représentation
# ------------------------------------------------------------
# Graphe orienté non pondéré, sommets indexés 0..n-1
# G[u] = liste des voisins sortants de u
Graph = List[List[int]]
g = [[0, 1, 4], [0, 2], [1, 3], [2, 4, 5], [0, 1, 3], [3]] #un exemple
```

1 Généralités

Question 1.

Écrire la fonction

```python
def degre_sortant(G: Graph, u: int) -> int:
```

qui renvoie le degré sortant du sommet u dans le graphe G.

Question 2.

Écrire la fonction

```python
def degre_entrant(G: Graph, u: int) -> int:
```

qui renvoie le degré entrant du sommet u dans le graphe G.

Question 3.

Écrire la fonction

```python
def transpose(G: Graph) -> Graph:
```

qui renvoie le graphe transposé du graphe G (inversion des arcs).

Question 4.

Écrire la fonction

```python
def matrice_adjacence(G: Graph) -> List[List[int]]
```

qui renvoie la matrice d'adjacence associée au graphe (0 : absence d'arc ; 1 : présence).

Question 5.

Écrire la fonction

```
def graphe_depuis_matrice(A: Sequence[Sequence[int]]) -> Graph:
```

qui construit un graphe à partir de sa matrice d'adjacence.

Question 6.

Écrire la fonction

```
def is_undirected(G: Graph) -> bool:
```

qui renvoie vrai si tout arc possède un arc retour.

2 BFS

On implante le parcours en largeur à 3 couleurs.

```
# Conventions :
#   - 0 = BLEU (non visité), 1 = VERT (en file), 2 = ROUGE (traité)
```

Question 7.

Écrire la fonction

```
def accessibles_depuis(G: Graph, s: int) -> List[int]:
```

qui renvoie la liste des accessibles depuis le sommet s dans le graphe G.

Question 8.

Écrire la fonction

```
def connected(G:Graph)->bool
```

qui renvoie vrai si le graphe (supposé non orienté) est connexe.

Question 9.

Écrire la fonction

```
def distances_depuis(G: Graph, s: int) -> List[Tuple[int, int]]
```

qui renvoie la liste des tuples (v, d) où v est accessible depuis s et d désigne la distance entre s et v. La liste retournée est triée par ordre croissant de distance.

3 DFS

On implante le DFS.

Question 10.

Écrire la fonction

```
def accessibles_depuis_dfs(G: Graph, s: int)->List:
```

qui renvoie la liste des accessibles depuis s en utilisant un DFS à 3 couleurs.

On définit l'exception :

```
class CYCLE(Exception):
    pass
```

Elle est soulevée lorsque l'exploration d'un graphe par DFS détecte un sommet.

Question 11.

Écrire la fonction

```
def cycle(G:Graph)->bool
```

qui renvoie un bouléen indiquant si le graphe possède un cycle.

```
g = [[1,2],[2],[],[5],[2,3,5],[4]]
cycle(g)
```

```
4->3->...->4 est un cycle
True
```

```
g2 = [[1,2],[2],[3],[],[2,3,5],[2]]
cycle(g2)
```

```
False
```