Graphes - Parcours

1 Premières applications

Exercice 1. *Connexité*

Implémentez une fonction ayant la spécification suivante :

```
def est_connexe(g : list[list[int]]) -> bool :
    """
    Entrée : g un graphe non-orienté représenté par sa liste d'adjacence.
    Sortie : un booléen vrai si et seulement si g est connexe
    """
```

Vous utiliserez pour ce faire un parcours en profondeur utilisant une pile implémenté par une liste Python.

Donnez sa complexité.

Exercice 2. *Calcul de l'ensemble des accessibles*

On introduit les structures de file (et pile) `deque` du module `collections`.

Importez l'objet deque dans votre fichier sans importer tout le module collection.

Les opérations suivantes sont en O(1).

```
F = deque() # création de file vide
F.append('N') # enfiler des éléments
F.append('a')
F.append('d')
x = F.popleft() # défiler un élément
```

Implémentez une fonction ayant la spécification suivante :

```
def accessibles(g : list[list[int]], s : int) -> list[int] :
    """
    Entrée : g un graphe orienté représenté par sa liste d'adjacence, et s un
↪   sommet de g.
    Sortie : la liste des accessibles dans g depuis s.
    """
```


Vous utiliserez pour ce faire un parcours en largeur utilisant une file implémenté par un objet `deque`.

Donnez sa complexité.

Exercice 3. *Cyclicité*

Implémentez une fonction ayant la spécification suivante :

```
def cyclique(g : list[list[int]]) -> bool :
    """
    Entrée : g un graphe orienté représenté par sa liste d'adjacence.
    Sortie : un booléen vrai si et seulement si g possède un cycle.
    """
```

Vous utiliserez pour ce faire un parcours en profondeur *récursif* (le seul, le vrai).

Donnez sa complexité.

Exercice 4. *Plus court chemin*

Implémentez une fonction ayant la spécification suivante :

```
def plus_court(g : list[list[int]], u : int, v : int) -> int :
    """
    Entrée : g un graphe orienté représenté par sa liste d'adjacence, u et v
↪   deux sommets de g
    Sortie : la longueur du plus court chemin entre u et v dans g, -1 si v
↪   n'est pas accessible depuis u.
    """
```

Vous choisirez le parcours le plus adapté pour faire cela.

Donnez sa complexité.

Exercice 5. *Cycles*

Modifiez l'algorithme testant l'existence d'un cycle dans un graphe pour qu'il renvoie un cycle lorsqu'il en a trouvé un.

2 2-coloration

Une 2-coloration d'un graphe $G = (S, A)$ consiste à choisir pour chaque sommet du graphe une couleur, rouge ou noir par exemple, de telle sorte que deux sommets voisins n'est pas la même couleur. Formellement, Une 2-coloration d'un graphe $G = (S, A)$ est une fonction $f \in \{0, 1\}^S$ vérifiant,

$$\forall (s, t) \in A, f(s) \neq f(t)$$

Une 2-coloration sera représentée en Python par une liste de booléen de taille $n = |S|$.

On suppose ici G non-orienté et *connexe*.

Exercice 6. *Vérification*

Implémentez une fonction ayant la spécification suivante :

```
def est_2coloration(g : list[list[bool]], coloration : list[int]) -> bool :
    """
    Entrée : g un graphe non-orienté représenté par sa liste d'adjacence, u et
↪   v deux sommets de g
    Sortie : est-ce que coloration représente une 2-coloration valide du graphe
↪   g
    """
```

Exercice 7. *Recherche*

1. Soit $s \in G$. Montrer que s'il existe une 2-coloration f de G telle que $f(s) = 0$ alors il existe une 2-coloration g de G telle que $g(s) = 1$.
2. Montrer que s'il existe une 2-coloration de G alors il en existe exactement 2.
3. Proposez et implémentez un algorithme qui calcule la 2-coloration d'un graphe, si elle existe, et renvoie `None` sinon.

3 Cycles Eulériens - Bonus très difficile

Un cycle simple dans un graphe, est un cycle ne passant pas deux fois par les mêmes arêtes.

Un graphe $G = (S, A)$ possède un cycle eulérien si et seulement si il existe un cycle simple passant par chaque arête de G. On dit alors que G est eulérien.

On suppose ici G non-orienté.

1. Montrer que si G possède un cycle eulerien, alors les degrés de tous ses sommets sont pairs.
2. Montrer, par récurrence sur le nombre d'arêtes de G, que si tous les degrés des sommets de G sont pairs, alors G possède un cycle eulérien.
3. Supposons G eulérien. Montrer que pour tout cycle simple non vide C de G, C est un cycle eulerien ou il existe un sommet s de C qui est l'extrémité d'une arête n'étant pas dans C.
4. Montrer que pour tout cycle simple C et C' de G n'ayant aucune arête en commun et ayant un sommet s de G, en commun, on peut construire un cycle simple englobant toutes les arêtes de C et C'.
5. Proposez et implémentez un algorithme permettant de trouver un cycle eulérien dans un graphe, si il existe.