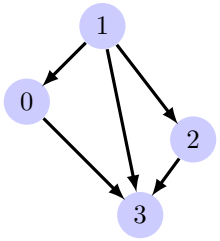
DS1

Le temps imparti pour ce devoir est d'une heure. L'énoncé comporte 2 pages. Tout document ou matériel électronique est interdit.

La présentation des copies et la qualité de la rédaction seront largement prises en compte dans l'évaluation. En particulier, vous devrez faire apparaître clairement les indentations dans les codes produits, qui doivent chacun être accompagnés des commentaires suffisants à la compréhension des algorithmes utilisés. Ne séparez pas vos codes sur plusieurs pages. Si vous repérez ce qui vous semble être une erreur d'énoncé, vous devez le signaler sur votre copie et poursuivre votre composition. Si vous ne parvenez pas à résoudre une question, vous pouvez tout de même utiliser toute fonction proposée dans l'énoncé dans les questions ultérieures de l'exercice.
Le devoir est constitué de deux exercices indépendants.

Exercice 1 - Graphe océan

Dans un graphe orienté $G = (S, A)$ sans boucles, une source universelle est un sommet u tel que, pour tout $v \in S \backslash \{u\}$, on a $(u, v) \in A$ et $(v, u) \notin A$. Le but de cet exercice est la détection de sources universelles, selon la représentation du graphe.

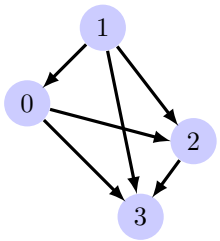


Graphe G_1

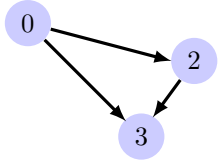
Par exemple, le sommet 1 est une source universelle dans le graphe ci-dessus.

1. Donner la matrice d'adjacence `M` et la liste d'adjacence `L` du graphe orienté G_1 donné en exemple.
2. Peut-il exister plusieurs sources universelles pour un même graphe ?
3. Écrire une fonction `source_mat` qui prend en entrée un graphe représenté par sa matrice d'adjacence et renvoie `True` si le graphe contient une source universelle, et `False` sinon.
4. Déterminer la complexité de `source_mat`.
5. Écrire une fonction `source_list` qui prend en entrée un graphe représenté par une liste d'adjacence et renvoie `True` si le graphe contient une source universelle, et `False` sinon.
6. Déterminer la complexité de la fonction `source_list` et comparer avec `source_mat`.

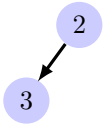
Un graphe orienté est un océan s'il ne comporte qu'un seul sommet ou s'il contient une source universelle v et que $G \backslash v$ est encore un océan. Ici $G \backslash v$ est le graphe obtenu à partir de G en supprimant le sommet v ainsi que tous les arcs qui lui sont incidents.



G_2



$G_2 \setminus s_1$



$(G_2 \setminus s_1) \setminus s_0$

$((G_2 \setminus s_1) \setminus s_0) \setminus s_2$

Dans l'exemple ci-dessus, le sommet 1 du graphe G_2 est une source universelle. On le retire et on remarque que le sommet 0 est une source universelle du graphe G_2 privé du sommet 1. On le supprime et on constate que le sommet 2 est une source universelle du graphe obtenu. Il ne reste plus qu'un seul sommet après avoir ôté le sommet 2. On peut donc en conclure que G_2 est un océan.

7. Écrire une fonction `supprime` qui prend en entrée un graphe G, représenté de la manière la plus appropriée, et un sommet v de G, et qui renvoie le graphe $G\backslash v$.

8. Écrire une fonction `ocean` qui prend en entrée un graphe, représenté de la manière la plus appropriée, et qui renvoie `True` si le graphe est un océan, et `False` sinon.

9. Combien de graphes orientés à n sommets sont des océans ? Proposer une autre méthode pour tester si un graphe est un océan.

Exercice 2 - Algorithmes de contrôle

Une liste X est trié par ordre croissant si `X[i] < X[i+1]` pour tout $i \in [\![0, n-1]\!]$, où n est le nombre d'éléments de la liste X. On va étudier plusieurs algorithmes permettant de tester si une liste est triée.

1. On considère les deux fonctions suivantes :

```
def fonction1(X):
        n = len(X)
        res = True
        for i in range(n):
                if X[i] >= X[i+1]
                        res = False
        return(res)
```

et

```
def fonction2(X):
        n = len(X)
        for i in range(n-1):
                if X[i] < X[i+1]:
                        return(True)
                else:
                        return(False)
```

Dire, pour chacune d'entre elles, si la fonction permet d'indiquer correctement si une liste est triée. Dans le cas contraire, proposer des modifications pour que la fonction réponde correctement à la question de savoir si une liste est triée ou non.

2. Quelle est la complexité A_n de la fonction `fonction2` éventuellement modifiée ? On ne considérera que le nombre de comparaisons en fonction de la longueur n de la liste.

3. Si `X` est une liste de longueur $N \geq 2$, que représentent les commandes Python `X[0:N//2]` et `X[N//2:]` ?

4. Élaborer une fonction récursive `test(X)` permettant de vérifier qu'un tableau X est trié ou non, pour lequel, si la liste X a une longueur N supérieure ou égale à 2 , on appelle récursivement `test(X[0:N//2])` et `test(X[N//2:])`.

5. Estimer la complexité B_n de `test`, en nombre de comparaisons, pour une liste de longueur $n = 2^p$, avec $p \in \mathbb{N}^*$.

6. Comment procéder lorsque n n'est pas une puissance de 2 dans la question précédente ? Commentez finalement l'intérêt de la fonction récursive `test`.