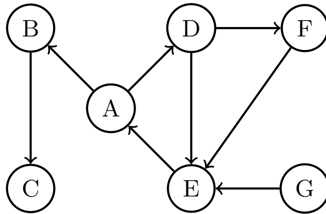
TP Python 10 - Graphes

Exercice 1 On considère le graphe suivant :



1) Représenter ce graphe en Python par un dictionnaire de listes d'adjacence (les sommets sont des caractères : `'A'`, `'B'`, etc.).

2) Écrire sa matrice d'adjacence (la numérotation des sommets doit correspondre à l'ordre alphabétique).

3) a) Parcourir à la main le graphe en profondeur en partant de A, puis en partant de G.

b) Même question avec un parcours en largeur.

Exercice 2 - Inverse d'un graphe

Dans cet exercice les graphes sont orientés et sont représentés par des dictionnaires d'adjacence.

1) Dessiner le graphe associé au dictionnaire `{1: [3, 4], 2: [3], 3: [1], 4: [2, 5], 5: [1]}`.

On appelle **inverse d'un graphe** le graphe obtenu en renversant toutes les arêtes du graphe de départ.

2) Donner le dictionnaire correspondant à l'inverse du graphe de la question 1.

3) Écrire une fonction `inverse` qui, recevant un graphe, renvoie son inverse.

Exercice 3 - Chemins dans un graphe

1) a) Dans cette question un graphe est représenté par le dictionnaire de ses listes d'adjacence. Écrire une fonction `est_chemin` qui, recevant un graphe et une liste de sommets, renvoie un booléen indiquant si cette liste définit un chemin ou non.

```
>>> G = {1: [2], 2: [1, 4, 5], 3: [4], 4: [2, 3, 5, 6], 5: [2, 4], 6: [4, 6]}
    # premier graphe du cours
>>> est_chemin(G, [1, 2, 4, 3])
True
>>> est_chemin(G, [1, 2, 5, 6])   # pas d'arête entre les sommets 5 et 6
False
```

b) Reprendre la question précédente lorsque les graphes sont représentés par leurs matrices d'adjacence. On supposera que les sommets sont numérotés à partir de 1.

```
>>> M = [[0, 1, 0, 0, 0, 0],
         [1, 0, 0, 1, 1, 0],
         [0, 0, 0, 1, 0, 0],
         [0, 1, 1, 0, 1, 1],
         [0, 1, 0, 1, 0, 0],
         [0, 0, 0, 1, 0, 1]]    # premier graphe du cours
>>> est_chemin(M, [1, 2, 4, 3])
True
>>> est_chemin(M, [1, 2, 5, 6])
False
```

2) a) Écrire une fonction `poids` qui, recevant un graphe pondéré représenté par un dictionnaire de dictionnaires d'adjacence et un chemin, renvoie le poids de celui-ci, s'il est valide, et `-1` sinon.

```
>>> G = {1: {2: 7, 3: 10}, 2: {1: 7, 3: 6, 4: 5, 5: 3}, 3: {1: 10, 2: 6, 5: 4},
         4: {2: 5}, 5: {2: 3, 3: 4}}  # troisième graphe du cours
>>> poids(G, [1, 2, 3, 5])
17
>>> poids(G, [1, 2, 3, 4])
-1
```

b) Même question lorsque le graphe est donné par sa matrice d'adjacence. On supposera que les sommets sont numérotés à partir de 1.

```
>>> M = [[0, 7, 10, 0, 0],
         [7, 0, 6, 5, 3],
         [10, 6, 0, 0, 4],
         [0, 5, 0, 0, 0],
         [0, 3, 4, 0, 0]]   # troisième graphe du cours
>>> poids(M, [1, 2, 3, 5])
17
>>> poids(M, [1, 2, 3, 4])
-1
```

Exercice 4

Dans cet exercice on supposera que les sommets sont numérotés à partir de 1.

1) Écrire une fonction `matrice_adjacence` qui, recevant le dictionnaire des listes d'adjacence d'un graphe non pondéré, renvoie sa matrice d'adjacence.

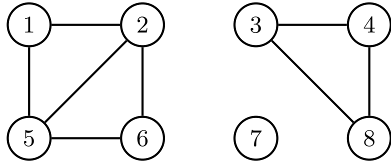
```
>>> G = {1: [2], 2: [1, 4, 5], 3: [4], 4: [2, 3, 5, 6], 5: [2, 4], 6: [4, 6]}
>>> matrice_adjacence(G)
[[0, 1, 0, 0, 0, 0], [1, 0, 0, 1, 1, 0], [0, 0, 0, 1, 0, 0], [0, 1, 1, 0, 1, 1],
 [0, 1, 0, 1, 0, 0], [0, 0, 0, 1, 0, 1]]
```

2) Écrire une fonction `listes_adjacence` qui, recevant la matrice d'adjacence d'un graphe non pondéré, renvoie le dictionnaire de ses listes d'adjacence.

```
>>> M = [[0, 1, 0, 0, 0, 0],
         [1, 0, 0, 1, 1, 0],
         [0, 0, 0, 1, 0, 0],
         [0, 1, 1, 0, 1, 1],
         [0, 1, 0, 1, 0, 0],
         [0, 0, 0, 1, 0, 1]]
>>> listes_adjacence(M)
{1: [2], 2: [1, 4, 5], 3: [4], 4: [2, 3, 5, 6], 5: [2, 4], 6: [4, 6]}
```

Exercice 5 - Composantes connexes

La **composante connexe** d'un sommet S d'un graphe non orienté G est l'ensemble des sommets reliés à S par un chemin de G. Les composantes connexes du graphe ci-dessous sont $\{1, 2, 5, 6\}$, $\{3, 4, 8\}$ et $\{7\}$.



1) Écrire une fonction `composante_connexe` qui, recevant le dictionnaire des listes d'adjacence d'un graphe non orienté et un de ses sommets, renvoie la composante connexe de celui-ci. Indication : utiliser un parcours de graphe (en largeur ou en profondeur).

```
>>> G = {1: [2, 5], 2: [1, 5, 6], 3: [4, 8], 4: [3, 8], 5: [1, 2, 6],
         6: [2, 5], 7: [], 8: [3, 4]}
>>> composante_connexe(G, 1)
[1, 2, 5, 6]
```

2) Écrire une fonction `est_connexe` qui, recevant le dictionnaire des listes d'adjacence d'un graphe non orienté, renvoie un booléen indiquant s'il est connexe ou non.

```
>>> est_connexe(G)
False
```

3) Écrire une fonction `composantes_connexes` qui, recevant le dictionnaire des listes d'adjacence d'un graphe non orienté, renvoie la liste de ses composantes connexes.

```
>>> composantes_connexes(G)
[[1, 2, 5, 6], [3, 4, 8], [7]]
```

Exercice 6 - Du coq à l'âne

Dans cet exercice on utilisera le fichier `mots.txt` du TP 8. On ne travaillera qu'avec des mots de trois lettres.

1) Créer la liste des mots de trois lettres du fichier.

2) Si cela n'a pas été fait au TP 8, écrire une fonction `voisins` qui, recevant une chaîne de caractères `mot`, renvoie la liste des mots qui diffèrent de `mot` d'une et une seule lettre.

```
>>> voisins('arc')
['ara', 'are', 'ars', 'art']
```

On peut ainsi définir un graphe dont les sommets sont les mots et où deux sommets sont reliés par une arête s'ils sont voisins au sens de la fonction précédente.

3) Écrire une fonction `creer_graphe` (sans paramètre) qui renvoie le dictionnaire des listes d'adjacence du graphe. Ne surtout pas afficher le résultat (on le stockera dans une variable `G`).

```
>>> G = creer_graphe()
>>> G['arc']
['ara', 'are', 'ars', 'art']
```

4) a) Combien le graphe a-t-il de sommets ? Combien d'arêtes ? En moyenne, combien chaque sommet a-t-il de voisins ?

b) Déterminer les mots qui n'ont pas de voisins.

c) Combien le graphe a-t-il de composantes connexes ? Quelle est la taille de sa plus grande composante connexe ?

5) Écrire une fonction `chemin` qui, recevant deux mots de trois lettres, renvoie le plus court chemin qui les relie, s'il existe, en utilisant l'algorithme de Dijkstra. On pourra utiliser la fonction `plus_court_chemin` du cours (elle est sur le site de la classe) en l'adaptant à un graphe non pondéré (en considérant que le poids de chaque arête est 1).

```
>>> chemin('coq', 'ane')
['coq', 'col', 'cil', 'ail', 'aie', 'ane']
```