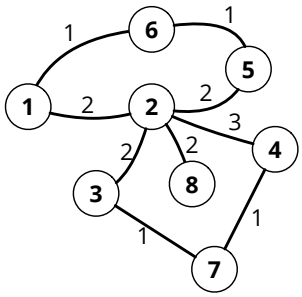
EXO0:

D'après le graphe ci-dessous, trouvez un « minimum spanning trees ». Il doit être minimum en terme de poids des arêtes (« edges »). En existe-t-il plusieurs ? Enumérez-les. Il s'agit d'un exercice sur papier = pas d'implémentation en Python.



Exo1:

Une application intéssante des graphes est le « social network analysis ». Imaginez que vous possédiez la liste de vos amis, ainsi que les amis de vos amis (ces derniers ne sont pas forcément dans vos amis) sous forme de graphe. Ce graphe doit permettre de:

1. Trouver parmis vos amis facebook ceux qui possèdent le plus d'amis
2. Découvrir que certains de vos amis se connaissent
3. Lister vos amis directes qui pourraient vous présenter une personne qui n'est pas dans votre liste d'amis
4. Trouvez une personne qui n'a pas d'ami.

Sur papier, comment modéliseriez vous le problème en utilisant un graph?

- A quoi correspondent les *ar*êtes (edges) et nœuds (vertices) ?
- Faut-il utiliser un graphe dirigé ?
- Décrivez, en terme de nœuds et d'arrêtes comment trouver les 4 éléments ci-dessus.

Exo2:

Dessinez un graphe dirigé (directed) G sachant que vertices = {1,6,3,5} et edges = {(1,6), (1,3), (3,3)}
A quoi correspond |G| ?
Est-ce un graphe dense ou clairsemé (sparse) ?

Exo3:

Sachant que G = { v = {1,6,3,5}, e = {(1,6), (1,3), (3,3)} }
Extraire un graphe qui est sous-graphe (subgraph) de G mais qui n'est pas un « spanning subgraph".

Exo4:

Analysez le code ci-dessous.

```
def magic_function(graph, start):
    explored = set()
    queue = [start]
```

```
    while len(queue) > 0:
        print ('queue: {0}'.format(queue))
        node = queue.pop(0)
        if node not in explored:
            explored.add(node)
            neighbours = graph[node]

            for neighbour in neighbours:
                queue.append(neighbour)
    return explored

graph = {'A': ['B', 'C', 'E'],
         'B': ['A','D', 'E'],
         'C': ['A', 'F', 'G'],
         'D': ['B'],
         'E': ['A', 'B','D'],
         'F': ['C'],
         'G': ['C'],
         'H': []}

print ('explored: {0}'.format(magic_function(graph, start='A')))
```

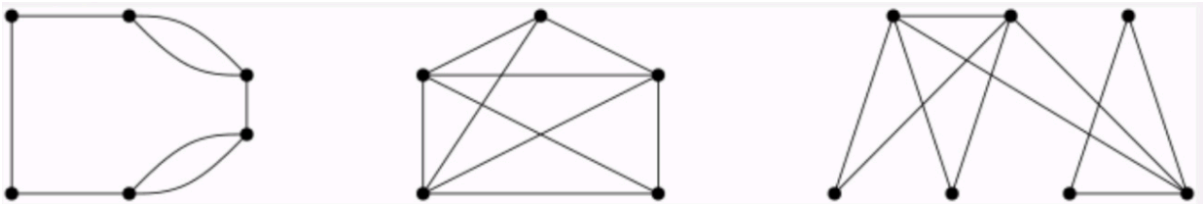
Lorsqu'on lance le programme, on obtient le résultat suivant (output) :

```
queue: ['A']
queue: ['B', 'C', 'E']
queue: ['C', 'E', 'A', 'D', 'E']
queue: ['E', 'A', 'D', 'E', 'A', 'F', 'G']
queue: ['A', 'D', 'E', 'A', 'F', 'G', 'A', 'B', 'D']
queue: ['D', 'E', 'A', 'F', 'G', 'A', 'B', 'D']
queue: ['E', 'A', 'F', 'G', 'A', 'B', 'D', 'B']
queue: ['A', 'F', 'G', 'A', 'B', 'D', 'B']
queue: ['F', 'G', 'A', 'B', 'D', 'B']
queue: ['G', 'A', 'B', 'D', 'B', 'C']
queue: ['A', 'B', 'D', 'B', 'C', 'C']
queue: ['B', 'D', 'B', 'C', 'C']
queue: ['D', 'B', 'C', 'C']
queue: ['B', 'C', 'C']
queue: ['C', 'C']
queue: ['C']
explored: ['A', 'B', 'C', 'E', 'D', 'F', 'G']
```

Exo3a : Quel est le nom de cet algorithme.
Exo3b : Pourquoi est-ce utile ?
Exo3c : Pourquoi est-ce que 'H' n'apparaît pas dans l'output alors qu'il est présent dans le graph?

Exo5:

1. Ci-dessous quelques exemples de graphes. Trouvez la relation entre la somme des degrés de toutes les noeuds (sum of the degrees of all the vertices) et le nombre d'arêtes (edges). Généraliser la relation.



2. Faites-de même avec quelques arbres (trees).

3. Trouvez le lien entre le nombre d'arêtes et le nombre de nœuds pour les graphes (de 1) et les arbres (de 2)
4. A partir du nombre de noeuds N et le degré de chaque noeuds D, écrivez un programme qui identifie s'il s'agit d'un arbre (en partant du principe que le graphe est connecté).

Input: Un entier N qui représente le nombre de noeuds et une liste d'entiers D qui représentent le degré de chaque noeuds.

Output: Output "True" s'il s'agit d'un arbre et "False" si ce n'est pas le cas.

Exo6:

Il existe une librairie qui permet de manipuler des graphes en Phyton : networkX.
Documentation: https://networkx.github.io/documentation/latest/tutorial.html

1. Installer la librairie (utiliser le tutorial de Pycharm pour installer des librairies)
2. Importer la librairie (import netwrokx as nx)

Exemple avec deux nœuds et une arête entre-eux.

```
import networkx as nx
import matplotlib.pyplot as plt

G = nx.Graph()  #creates a graph with the name G

G.add_node(1)    #Adds the first vertex to the graph
G.add_node(2)    #Adds vertex two to the graph

G.add_edge(1, 2)  #Adds an edge between vertex 1 and vertex 2

nx.draw(G)   #draws the graph
plt.show()   #Displays the graph on the screen
```

Utilisez la librairie pour trouver le chemin le plus court dans un graphe (en générant vous-même un graphe plus complexe que celui-ci).