# TP 3

## Parcours en profondeur

### Exercice 1. Rappels

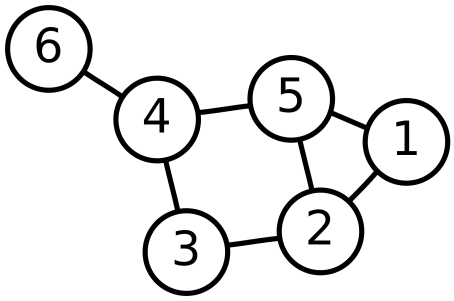
**1.** Rappelez l'algorithme de parcours en profondeur d'un graphe.

**2.** Proposez un programme python qui implémente le parcours en profondeur d'un graphe.

### Exercice 2. Réseau ferré

*Dans cet exercice, pour chaque question vous devez proposer et implémenter une fonction python qui permette d'y répondre.*

Le graphe ci-dessous représente un réseau ferré reliant six villes numérotées de 1 à 6.




**1.** Implémentez l'algorithme de parcours en profondeur sur le réseau ferré en partant de la ville 6.

**2.** Suite à des travaux de rénovation, les lignes reliant les villes 4-5 et 3-2 ont été coupées. Bob qui habite la ville 6 souhaite rendre visite à Alice qui habite la ville 1.

**2.1** Bob peut-il se rendre chez Alice ?

**2.2** Bob et Alice décident alors de prendre le train chacun de leur côté. Combien de villes Bob peut-il visiter ? Même question pour Alice.

**2.3** Combien de trains faut-il au minimum pour que le réseau ferré puisse être entièrement parcouru ?

**3.** Les travaux sont terminés.

**3.1** Cinq amis habitant la ville 1 décident de visiter chacun une ville différente. Proposez un chemin pour chacun d'entre eux.

**3.2** Existe-t-il un cycle dans ce réseau ferré ?

## Exercice 3. Carte au trésor

Le code ci-dessous a pour effet de créer un graphe G1 avec des trésors sur les noeuds (un trésor est un entier).

```
G1=networkx.Graph()
G1.add_edges_from([("a","f"),("a","b"),("f","c"),("f","e"),
("f","d"),("c","e"),("c","b"),("c","d"),("e","d")])
G1.node["a"]["tresor"]=3
G1.node["f"]["tresor"]=4
G1.node["c"]["tresor"]=5
G1.node["e"]["tresor"]=1
G1.node["d"]["tresor"]=2
G1.node["b"]["tresor"]=0
```

**1.** Écrivez une fonction `tresor(G,n)` prenant en entrée un graphe et un noeud et renvoyant le trésor présent sur le noeud n dans G.

**2.** Écrivez une fonction `parcours_tresor(G,n)` qui prend en entrée un graphe et un noeud de départ, et parcourt le graphe en choisissant à chaque étape de visiter le noeud encore non visité et aperçu ayant le plus grand trésor. La fonction doit renvoyer la liste des noeuds visités dans l'ordre de visite.

## Exercice 4. Tout à l'égout

Considérons que nous avons un graphe G représentant une ville. On souhaite créer dans cette ville un réseau de tout à l'égout de façon à ce que toute la ville soit couverte par le tout à l'égout. Il faut donc choisir un ensemble d'arêtes du graphe G qui "couvre" toute la ville.

**1.** Écrivez une fonction `est_couvrant(G,liste)` qui prend en entrée un graphe G et un ensemble d'arêtes et renvoie True si cet ensemble couvre le graphe et False sinon.

**2.** Écrivez une fonction `couvrant(G)` prenant en entrée un graphe G et renvoyant un ensemble d'arêtes qui couvre G.

**3.** On va maintenant chercher à ce que le réseau de tout à l'égout soit le plus petit possible. Écrivez une fonction `plus_court(G)` qui prend en entrée un graphe et renvoie un ensemble d'arêtes couvrant G et de taille minimum (tel qu'on ne puisse pas enlever une arête).