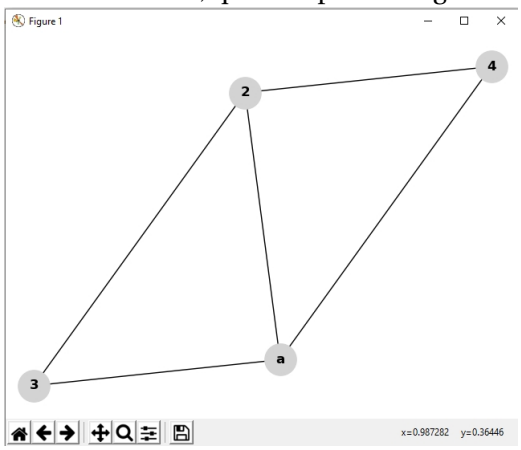
Représenter un graphe avec Networkx

*Ce document a pour objectif de mettre rapidement au point une fonction / méthode pour représenter graphiquement avec la bibliothèque **netxorkx**. Il n'a pas l'intention d'être exhaustif, et une consultation de la documentation officielle est nécessaire pour aller plus loin...*

A ma connaissance il existe deux bibliothèques qui permettent de représenter des graphes en Python : **networkx** et **graphviz**. Dans ce cours nous allons nous concentrer sur la première, qui permet non seulement de représenter simplement des graphes, mais aussi d'appliquer des algorithmes spécifiques aux graphes. Voici un exemple simple d'utilisation de cette bibliothèque. A noter qu'il faut aussi pour l'affichage importer la bibliothèque **pyplot**.

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

g = nx.Graph()

g.add_node("a")
g.add_node(2)
g.add_node(3)
g.add_node(4)

g.add_edge("a",2, weight = 6)
g.add_edge("a",3)
g.add_edge("a",4)
g.add_edge(2,3)
g.add_edge(2,4)
g.add_edge(2,4)

nx.draw(g, with_labels = True, font_weight = 'bold',
node_size = 800, node_color = 'lightgrey')

plt.show()
```

Ci-contre le résultat, que l'on peut enregistrer au format png :



Quelques exercices sur cette partie :

1. écrire une fonction (ou une méthode en POO) affiche_graphe(dic) prend quand comme argument une liste d'adjacence et qui affiche le graphe avec **networks**.

   Vous pourrez tester l'affiche de graphe ci-dessous.

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

2. écrire une fonction graphe_proba(n,p) qui prend comme argument un entier n et un réel p tel que $0 \leq p \leq 1$ et qui renvoie la liste d'adjacence d'un graphe qui comprend n sommets, et chaque sommet a la probabilité p d'avoir un lien avec un autre sommet. Tester la fonction en affichant le graphe.

Une proposition pour chaque fonction demandée...

Une proposition pour la fonction affiche_graphe(dic). On remarquera que suivant le type de graphe que l'on veut, orienté ou non, on utilisera **Graph()** ou **DiGraph()**. Je n'ai pas donné de correction pour afficher à partir d'une matrice d'adjacence, car on a une fonction qui permet de passer d'une matrice à une liste d'adjacence...

```
def affiche_graphe(dic):
    """
    affiche un graphe sous forme de dico
    avec networkx
    """
    g = nx.DiGraph()
    # on peut utiliser Graph(), DiGraph(),
    #  MultiGraph() ou MultiDiGraph()
    if dic is None :
        return "Graphe vide"
    for noeud in dic :
        g.add_node(noeud)
    for noeud in dic :
        if dic[noeud] is None : continue
        for som in dic[noeud]:
            g.add_edge(noeud,som)
    nx.draw(g, with_labels = True, font_weight = 'bold', node_size = 800,
     node_color = 'yellow')
    plt.show()
```

Une proposition pour la fonction graphe_proba(n,p) :

```
def graphe_proba(n, p):
    """
    renvoie un graphe de n sommets
    où chaque elem à une proba p
    d'avoir un lien avec un autre elem
    renvoi sous forme de dico
    """
    dico = {}
    if n <= 0 :
        return dico
    for i in range(n):
        dico[i] = None
    for noeud in dico :
        suivants = []
        for cible in dico :
            if noeud == cible : continue
            proba = random()
            if proba < p :
                suivants.append(cible)
        dico[noeud] = suivants
    return dico
```

Voici une amélioration de la fonction d'affichage d'un graphe, avec la possibilité d'afficher un chemin sous forme d'une liste de sommets.

```
def affiche_graphe_chemin(dic, chem):
    """
    affiche un graphe sous forme de dico et un chemin
    avec networkx
    """
    g = nx.MultiDiGraph()
    if dic is None :
        return "Graphe vide"
    for noeud in dic : # on ajoute les noeuds
        g.add_node(noeud)
    for noeud in dic : # on trace les arcs
        if dic[noeud] is None : continue
        for lien in dic[noeud]:
            g.add_edge(noeud, lien, color = 'blue', weight = 1)
    if len(chem) > 1 : # on repasse le chemin en couleur
        for i in range(len(chem)-1):
            g.add_edge(chem[i], chem[i+1] ,color = 'red' ,weight = 5)
    # on affiche le graphique
    colors = nx.get_edge_attributes(g,'color').values()
    weights = nx.get_edge_attributes(g,'weight').values()
    nx.draw(g,
            with_labels = True,
            #font_weight = 'bold',
            node_size = 300 ,
            node_color = 'yellow',
            edge_color = colors,
            width=list(weights))
    plt.show()

affiche_graphe_chemin(monGraphe, "AB")
```

Le résultat en affichant le chemin de A vers B...