TP - graphes

K. Bertet – J.L. Guillaume – A. Huchet

Tous les TP sont à faire en python, de préférence avec jupyter-notebook. Les logiciels principaux sont installés sur les machines virtuelles ubuntu de l'université.
Si vous décidez d'installer des choses sur vos machines, vous aurez besoin d'installer graphviz au minimum.

1. Prise en main de Graphviz

Cette partie est facultative, elle montre comment visualiser des graphes simplement. Le format dot utilisé peut-être importé dans networks (cf partie 2).

Graphviz[1] est une suite logicielle open source d'édition et de visualisation de graphes qui inclut :
- Dot : un langage de description de graphes
- dot, neato, circo, etc. : différents algorithmes de dessin de graphe
- gvedit : un logiciel graphique d'édition de graphes (à installer si nécessaire)

Exercice 1. Créer le graphe suivant avec un éditeur de texte puis le visualiser avec différentes méthodes (dot, circo, etc.)

```
// exemple de graphe orienté (dans un fichier nommé graphe.dot)
digraph G {
  a - > c
  a - > d
  d - > e
  b - > d
}

// exemple de visualisation (avec le logiciel dot)
dot -Tpng graphe.dot -o graphe.png
```

Exercice 2. Consulter le manuel d'utilisation du langage dot pour :
- Colorier un arc en spécifiant son attribut *color*
- Associer un label à un arc et à un sommet
- Modifier le graphe afin qu'il soit non orienté

2. Prise en main de Networkx

Networkx est une bibliothèque python permettant d'instancier et de manipuler des graphes, de les visualiser au format dot. Consulter la documentation.
Exercice 3. Copiez-collez et exécutez le code ci-dessous dans un notebook (il faut lancer jupyter-notebook) pour vérifier que tout fonctionne. Que se passe-t-il si vous réexécutez le code

```
%matplotlib inline
import matplotlib.pyplot as plt
from networkx import nx
```

[1] http://www.graphviz.org

```
# Génération d'un graphe aléatoire à 10 sommets et 20 arêtes
G = nx.gnm_random_graph(10,20)
# Visualisation du graphe
nx.draw_networkx(G)
plt.show()
```

Exercice 4. (A ne pas faire, sauf sur machine perso avec package pygraphviz correctement installé) Ecrivez du code python pour charger le graphe au format dot de l'exercice 1 et le visualiser.

Exercice 5. Ecrivez une fonction python qui créé et retourne le graphe orienté des diviseurs sur l'ensemble des entiers compris entre 2 et n (n passé en paramètre) possédant : un sommet pour chaque entier entre 2 et n ; un arc de l'entier x vers l'entier y si y est un multiple de x.

- https://networkx.org/documentation/stable/reference/introduction.html#networkx-basics
- Exemple d'exécution attendue pour le graphe des diviseurs de 2 à 10 :

```
In [12]: def diviseurs(n):
             ... à compléter ...

In [13]: G = diviseurs(10)
         nx.draw_networkx(G)
         plt.show()
```

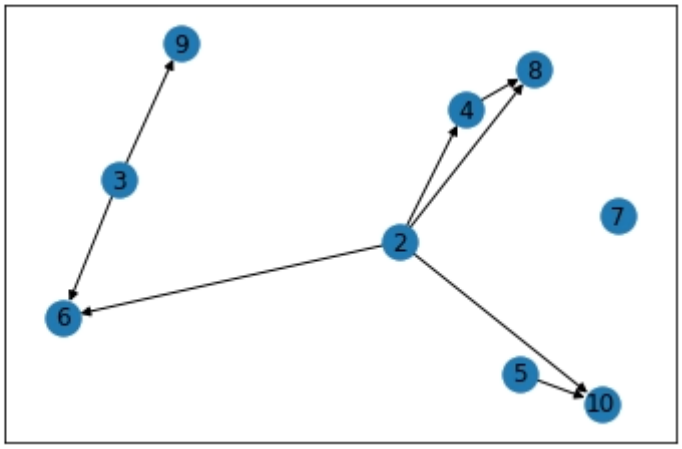

Figure 1: représentation du graphe des diviseurs de 2 à 10

Exercice 6. Que peut-on dire d'un sommet dont le degré entrant est nul ? D'un sommet dont le degré sortant est nul ?

Exercice 7. Mettre en place un mécanisme de coloriage des arcs et nœuds d'un graphe. Vous pouvez pour cela faire un premier dessin avec draw_networkx puis redessiner les arêtes par-dessus (à la même position, cf second paramètre de la fonction draw_networkx_edges). Testez pour obtenir quelque chose comme sur la figure 2 (les arêtes sont en bleu, de largeur 8 et avec une opacité de 0,5).

- https://networkx.org/documentation/stable/reference/drawing.html

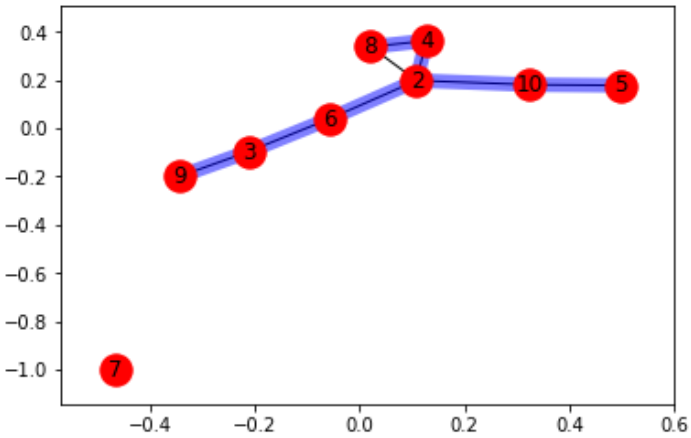

Figure 2: Coloration des arcs d'un BFS partant du sommet 4 sur le graphe des diviseurs.

Exercice 8. Si vous ne l'avez pas fait dans l'exercice précédent, écrivez une fonction mydraw pour dessiner un graphe en mettant certaines arêtes en avant.

```
def mydraw(G, edges):
    ...
```

3. Algorithmes de parcours de graphe

Exercice 9. Implémenter l'algorithme de parcours en largeur d'un graphe qui prend un graphe et un sommet de départ et retourne la liste des arêtes (ou arcs) de l'arbre du parcours en largeur partant du sommet de départ.

- On supposera que le graphe est (fortement) connexe pour n'implémenter que la boucle interne du BFS.
- Vous pouvez utiliser un ensemble de sommets visités plutôt que de colorer les sommets durant le parcours. Ainsi si un sommet est dans l'ensemble il a déjà été visité sinon il ne l'a pas été.

```
import queue
def bfs(G, source):
    q = queue.Queue() # file pour stocker les sommets durant la visite
    visited = {source} # ensemble des sommets déjà visités
    ...
```

Exercice 10. Visualiser l'arbre du parcours en largeur d'un graphe en utilisant ce que vous avez fait dans l'exercice 7 (comme sur la Figure 2).

```
G = nx.gnm_random_graph(10,15)
edges = bfs(G,5)
mydraw(G, edges)
```

Exercice 11. Coder, tester et visualiser l'arbre du parcours en profondeur (avec une file ou en récursif) selon votre préférence.

4. Bonus - Algorithmes de parcours de graphe

Exercice 12. Implémenter et tester les algorithmes vus au TD1. Il faudra très probablement modifier les codes précédents pour inclure au minimum les temps de fin de visite des sommets. A chaque fois vous pouvez visualiser le résultat.

- le tri topologique ;
- le calcul des composantes connexes d'un graphe non orienté qui retourne les sommets sous forme d'une partition composé d'un ensemble par composante ;

- le calcul des composantes fortement connexes d'un graphe orienté.