Algorithmes et Structures de données | 2021

Journée 5 – Vendredi 26 mars

Exercice 1 – Création de graphes en Python

Le but de cet exercice est de vous familiariser avec l'utilisation du module NetworkX pour la création de graphes en Python. Pour ce faire, on vous demande de vous référer à la documentation en ligne afin de créer les deux graphes suivants :

a. un graphe non-dirigé composé de 5 sommets numérotés de 1 à 5, en utilisant la méthode `add_note(…)`, et des arêtes (1, 2), (1, 3), (1, 5), en utilisant la méthode `add_edge(…)`;

b. un graphe dirigé composé des arêtes ("s", "a"), ("a", "l"), ("l", "u"), ("u", "t"), ("t", "à"), ("à", "v"), ("v", "o"), ("o", "u"), ("u", "s"), en utilisant la méthode `add_edges_from(…)`.

Indication. Afin de visualiser vos graphes, utilisez l'opération `plot(…)` et `display()` qui vous sont fournies dans le fichier template. Pour cette exercice, utilisez `plot(…)` avec un seul paramètre, à savoir le graphe à afficher.

Attention. Chaque appel à `plot(…)` ouvre une fenêtre distincte affichant le graphe passé en paramètre, mais cette fenêtre ne devient visible qu'après l'appel à `display()`. De plus, même si vous faites plusieurs appels à `plot(…)`, il suffit d'un seul appel à `display()` pour afficher toutes les fenêtres. En fonction du système d'exploitation sous-jacent, il se peut que certaines fenêtres (voire toutes) soient superposées et que vous deviez les déplacer afin de voir chaque graphe que vous avez affiché. Finalement, votre programme ne se termine qu'une fois que vous avez fermé toutes les fenêtres.

Exercice 2 – Manipulation des éléments d'un graphe en Python

Le but de cet exercice est de vous familiariser avec l'utilisation du module NetworkX pour la manipulation des éléments d'un graphe. Pour ce faire, on vous demande de partir des deux graphes de l'exercice précédent et d'afficher les informations suivantes sur la console :

a. la liste des sommets et des arêtes de ces deux graphes, via leurs propriétés `nodes` et `edges`;

b. la liste d'adjacence des sommets 1 et 4 pour le premier graphe, et "a" et "u" pour le second.

Indication. Soit `G` un graphe et `s` un sommet de `G`, la liste d'adjacence de `s` est obtenue en utilisant l'expression Python suivante `list(G[s])`.

Exercice 3 – Ajout d'attributs aux éléments d'un graphe

Le but de cet exercice est de vous familiariser avec l'utilisation du module NetworkX pour l'ajout d'attributs aux éléments d'un graphe. Pour ce faire, on vous demande de partir du second graphe de l'exercice 1 (graphe dirigé) et d'implémenter les extensions suivantes :

a. ajouter un attribut `weight` de type entier, pris aléatoirement entre 1 et 10, à chaque arête;

b. ajouter un attribut `ascii` à chaque sommet, en lui attribuant comme valeur un booléen indiquant si le sommet est une chaîne de caractères purement ascii (sans caractère accentué) ou non;

c. ajouter un attribut `parent` à chaque sommet au moyen d'une boucle, en lui attribuant comme valeur le sommet de l'itération précédente (le sommet de la première itération n'a pas de parent);

d. afficher sur la console pour chaque sommet du graphe, la liste de tous ses attributs;

e. afficher sur la console pour chaque sommet du graphe, uniquement l'attribut `ascii`;

f. créer un nouveau graphe à partir de l'attribut `parent`, en liant chaque sommet du graphe initial au sommet se trouvant dans son attribut `parent` (le résultat est donc un arbre).

Indication. Pour obtenir un nombre aléatoire entre 1 et 10, utiliser `random.randint(1, 10)` et pour savoir si une chaîne est purement ascii, utilisez la fonction `is_ascii(…)` fournie dans le template.

Exercice 4 – Implémentation en Python de la recherche en largeur

Le but de cet exercice est d'implémenter l'algorithme de recherche en largeur (BFS) vu au cours, dans le langage Python. Pour ce faire, on vous demande d'écrire une fonction **`bfs(G,s)`** en vous appuyant sur le module NetworkX et en vous inspirant des exercices précédents.

Afin de valider votre implémentation, on vous suggère, à la fin de la recherche en largeur d'abord, de construire un arbre à partir de l'attribut **`parent`** et de le retourner. Comme graphe d'entrée, on vous suggère de construire celui qui est utilisé dans les slides pour illustrer l'algorithme BFS.

Indication. Afin de vous faciliter la tâche, le fichier template définit la variable `infinity` et importe le module `queue` (https://docs.python.org/3/library/queue.html). Pour cet algorithme, vous n'avez besoin que de l'opération `put(…)`, qui correspond à ENQUEUE, de l'opération `get()`, qui correspond à DEQUEUE, et de l'opération `empty()`, qui permet de tester si la queue est vide.

De plus, pour cette exercice, utilisez la fonction `plot(…)` avec deux paramètres, le premier étant le graphe passé à la fonction `bfs(G,s)` et le second l'arbre retourné par cette fonction. La figure 1 montre ce qui devrait être affiché.

Exercice optionnel. Estimez la complexité temporelle de l'algorithme BFS, sachant que les opérations ENQUEUE et DEQUEUE ont une complexité temporelle de $O(1)$.

Exercice 5 – Implémentation en Python de l'algorithme de Kruskal

Le but de cet exercice est d'implémenter l'algorithme de Kruskal permettant de trouver un arbre couvrant minimal (MST) dans le langage Python.

Pour ce faire, on vous demande d'écrire une fonction **`mst(G)`** en vous appuyant sur le module NetworkX et en vous inspirant des exercices précédents. Comme graphe d'entrée, on vous suggère de construire celui qui est utilisé dans les slides pour illustrer l'algorithme de Kruskal.

Indication. Afin de vous faciliter la tâche, le fichier template fournit une classe `DisjointSet`, qui permet de créer une famille d'ensembles disjoints deux à deux (disjoint sets) utilisée dans l'algorithme de Kruskal. Pour ce faire, on l'initialise avec une liste de sommets et on dispose ensuite des opérations `find(…)` et `union(…)`. Finalement, comme pour l'exercice précédent, utilisez la fonction `plot(…)` avec deux paramètres, le premier étant le graphe passé à la fonction `mst(G)` et le second l'arbre retourné par cette fonction. La figure 2 montre ce qui devrait être affiché.

Figure 1 – Affichage attendu pour l'exercice 4

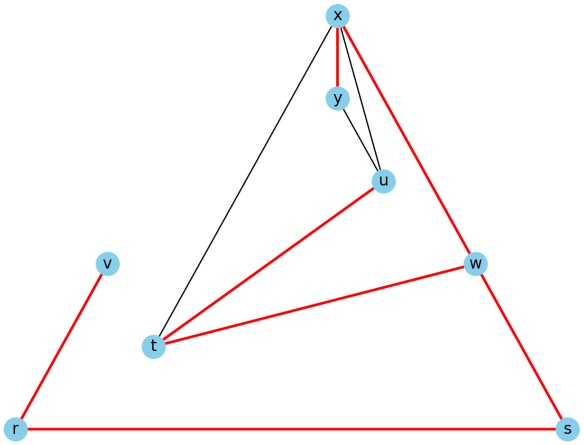

Figure 2 – Affichage attendu pour l'exercice 5