ITC – TD n°12-2

Prise en main des graphes orientés

Le présent TD a pour but de prendre en main les définitions et les outils que nous utiliserons tout au long des chapitres sur les graphes.

> Il est recommandé de garder les fonctions que vous écrirez dans chaque TD sous la main : elles pourront servir de brique élémentaires / fonctions utilitaires pour d'autres TD.

On fournit un module proposant plusieurs graphes pré-codés ainsi que des méthodes d'affichage. On pourra les charger avec l'instruction

```
import module_graphes as gr
```

Représentation des données

Dans tous nos travaux, les graphes sont représentés par un dictionnaire de dictionnaires d'adjacence, les sommets ayant pour type str ; tous les sommets u sont des clés valides du graphe g, v est une clé valide de $g[u]$ uniquement si g contient l'arc (u, v) ; dans ce cas $g[u][v]$ est le poids de l'arc (u, v), noté $w_{(u,v)}$ dans le sujet (ou 1 si le graphe n'est pas pondéré). Lorsqu'un fonction « prend en paramètre un graphe », elle prend donc en paramètre un dictionnaire `{str: {str: int}}`.

Les informations supplémentaires sont stockées dans des dictionnaires ayant pour clés les sommets.

I. Échauffement

On peut récupérer et visualiser un graphe orienté de test avec les instructions

```
g = gr.recuperer_graphe('g2', oriente=True)
gr.afficher_graphe(g)
```

1 – Écrire deux fonctions `degré_entrant(g, u)` et `degré_sortant(g, u)` qui prennent en paramètre un graphe et un sommet et renvoient le degré entrant ou sortant de ce sommet.

2 – Écrire une fonction `est_orienté(g)` prenant en paramètres un graphe, qui renvoie `True` si le graphe est orienté, et `False` dans le cas contraire.

3 – Un graphe orienté peut avoir au plus $n_{\max} = n_S^2$ arcs ; la densité d'un graphe orienté est définie comme $d = n_A/n_{\max}$. Écrire une fonction `densité(g)` qui calcule la densité du graphe fourni. Quelle est la complexité de cette fonction ?

Un trou noir est un sommet de degré entrant $n_s - 1$ et de degré sortant 0.

4 – Dessiner un graphe contenant un trou noir. Créer ce graphe sous Python.

5 – Écrire une fonction `trou_noir(g)` qui renvoie `''` si le graphe ne contient pas de trou noir, et le nom du sommet si un trou noir est présent.

6 – Quelle est la complexité de votre fonction ?

II. Créer des graphes

7 – Écrire une fonction `désorienter(g)` qui prend en paramètre un graphe orienté, et renvoie un graphe non orienté équivalent ; si le graphe de départ est pondéré et que $w_{(u,v)} \neq w_{(v,u)}$, on prendra comme poids pour l'arête dans le graphe non orienté la moyenne $(w_{(u,v)} + w_{(v,u)})/2$.

8 – Écrire une fonction `transposée(g)` qui prend en paramètre un graphe orienté $G = S, A$, et renvoie un graphe $G' = S, A'$ orienté renversé, c'est-à-dire tel que

$$\forall (u, v) \in A\,, \quad (v, u) \in A'$$

en conservant la pondération s'il y en a une.

9 – Le carré G^2 d'un graphe $G = (S, A)$ est le graphe (S, A^2) avec A^2 l'ensemble des arêtes (u, w) telles que $(u, v) \in A$ et (v, w) $in A$; autrement dit, il existe un chemin de longueur exactement deux dans G pour aller de u à w. Écrire une fonction `carré(g)` qui calcule et renvoie le graphe carré de g.

III. Arbres orientés

On peut obtenir un arbre orienté de test avec

```
a = gr.recuperer_graphe('g6')
gr.afficher_graphe(a)
```

10 – Écrire une fonction `arbre_construire_précédents` qui prend en paramètre un arbre orienté et renvoie un dictionnaire `(str, str)` qui associé à chaque sommet le sommet précédent dans l'arbre, et `None` pour la racine.