R308 - Consolidation de la programmation : TP4-5
9 octobre 2022

Modélisation de graphes

Un graphe est un ensemble de nœuds (on parle aussi de sommets, d'acteurs ou de sites) qui sont reliés entre eux par des liens (on parle aussi d'arrêtes, d'arcs, ou de lignes). Formellement, un graphe G est un couple $G = (V, E)$ où V est un ensemble de nœuds et $E = \{\{x, y\}|x, y \in V\}$ un ensemble de liens qui relient les nœuds. Les graphes sont largement utilisés pour modéliser différents systèmes d'interaction ou de connexions comme les réseaux informatiques, les réseaux sociaux, les réseaux routiers, etc.

Différents types de graphes peuvent être définis. On cite par exemple :

- **Les graphes simples non dirigés.** Dans ce type de graphes les liens entre les couples de nœuds sont symétriques. C'est par exemple le cas d'un lien d'amitié dans un réseau social comme Facebook (ou les amitiés sont censées être réciproques !). Dans un graphe simple, un nœuds ne peut pas être relié à lui même (pas de boucle) et deux nœuds ne peuvent pas avoir plus d'un lien entre eux.
- **Les graphes simples dirigés**. Dans ce type de graphes les liens sont orientés. Si $x \rightarrow y$ on n'a pas nécessairement le lien inverse $y \rightarrow x$. C'est par exemple le cas d'un graphe qui modélise le web où les nœuds sont les pages web et les liens représentent les liens html. C'est aussi le cas de certains réseaux sociaux comme twitter (une personne peut suivre une autre mais l'inverse n'est pas forcément vrai !).
- Les graphes bipartites où il existe une partition de l'ensemble V en deux ensembles $\top$ et $\bot$ de sort que les liens relient des nœuds appartenant à un sous ensemble différent !

Il existe une grande variété de types de graphes. nous allons nous intéresser au cas simple des graphes simple non dirigés. La figure 1 illustre quelques types de graphes.

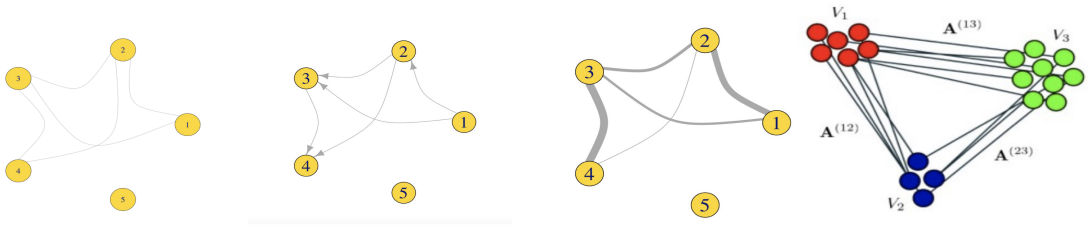

FIGURE 1 – Différents types de graphes

Quelques propriétés de graphes

Nous définissons dans la suite quelques propriétés basiques des graphes :

— L'**ordre** d'un graphe désigne le nombre de nœuds. On a donc $Ordre(G) = |V|$.

— La **taille** d'un graphe désigne le nombre de liens. $Taille(G) = |E|$.

— La **densité** d'un graphe désigne la probabilité d'avoir un lien entre deux nœuds du graphe sachant la taille et l'ordre du graphe. Dit autrement, c'est le ratio du nombre de liens par rapport au nombre potentiel de liens. Il est facile de vérifier que pour un graphe simple non dirigé d'ordre n, le nombre de liens potentiels est $\frac{n\times(n-1)}{2}$. Donc, si m est le nombre de liens (ou le taille du graphe), on a la densité égale à $\frac{2\times m}{n\times(n-1)}$

— Un nœud x est dit **voisin** d'un nœud y si $\{x, y\} \in E$

— Le **degré** d'un nœud est le nombre de ses voisins.

— La **distribution de degrés** d'un graphe est l'histogramme de la liste des degrés des nœuds du graphe. L'histogramme donne pour chaque valeur de degré possible le nombre de nœuds qui ont cette valeur pour degrés.

— Un chemin simple entre deux nœuds n_0 et n_n est une séquence de nœuds $n_0, n_1, \ldots, n_n$ où chaque deux nœuds successifs n_i, n_{i+1} sont des voisins et tous les nœuds appartenant au chamin sont différents entre eux.

— Un graphe G est **connexe** si il existe un chemin entre n'importe quel couple de nœuds.

Considérons le graphe non dirigé illustré à la figure 2. L'ordre de ce graphe est 5, sa taille est 5 aussi et sa densité est 0.5. Le graphe n'est pas connexe comme aucun chemin ne mène au nœud 5. La distribution de degrés de ce graphe et : $\{0 : 1, 1 : 0, 2 : 2, 3 : 2\}$

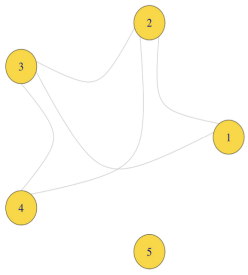

FIGURE 2 – Exemple de graphe non dirigé

Représentation de graphes

Différents schéma de représentation de graphes peuvent être utilisés. Un schéma classique est de représenter un graphe par un dictionnaire où les clés sont les nœuds du graphe et chaque clé est associé à un liste qui contient les voisins du nœud donné en clé. Par exemple la structure du graphe illustré à la figure 2 serait la suivante :
{
$n_0 : (n_1, n_2, n_3)$
$n_1 : (n_0, n_2, n_3)$
$n_2 : (n_0, n_1, n_3)$
$n_3 : (n_0, n_1, n_2)$
$n_4 : ()$
}

Parcours de graphe

Un parcours de graphe est une énumération de nœuds visités en suivant les différents liens à partir d'un nœud source. Dans un parcours aucun nœud ne doit être visité deux fois. Si le graphe est connexe, la longueur d'un parcours est donc égale à l'ordre du graphe. Deux algorithmes classiques de parcours de graphes sont les algorithmes de *parcours en largeur* et de *parcours en profondeur*. Selon Wikipédia : "Un parcours en largeur débute à partir d'un nœud source. Puis il liste tous les voisins de la source, pour ensuite les explorer un par un. Ce mode de fonctionnement utilise donc une file dans laquelle il prend le premier sommet et place en dernier ses voisins non encore explorés. Les nœuds déjà visités sont marqués afin d'éviter qu'un même nœud soit exploré plusieurs fois. Dans le cas particulier d'un arbre, le marquage n'est pas nécessaire. Voici les étapes de l'algorithme :

- mettre le nœud source dans la file ;
- retirer le nœud du début de la file pour le traiter ;
- mettre tous ses voisins non explorés dans la file (à la fin) ;
- si la file n'est pas vide reprendre à l'étape 2.

L'algorithme de parcours en profondeur diffère du parcours en largeur car il continue l'exploration jusqu'à arriver un cul-de-sac. C'est seulement à ce moment-là qu'il revient en arrière pour explorer depuis un autre nœud voisin. L'utilisation d'une pile au lieu d'une file transforme l'algorithme du parcours en largeur en l'algorithme de parcours en profondeur."

A titre d'exemple, un parcours en largeur du graphe illustré à la figure 2 et à partir du nœud source 1 donnera la parcours suivant : 1,2,3,4.

Exercices

1. Proposer une **classe abstraite** `Graph` qui modélise un graphe quelconque. Les nœuds du graphe sont à modéliser par une classe `Node`. Chaque nœud doit posséder un identifiant unique dans l'espace du graphe. La classe doit fournir les méthodes de services suivantes (vous précisez pour chaque méthode si elle est abstraite on concrète) :

   - `addNode` qui ajoute un nœud au graphe.
   - `addNodes(n)` : qui permet d'ajouter n nœuds au graphe où n est un entier positif.
   - `addEdge(x,y)` : qui permet d'ajouter un lien entre les deux nœuds x et y.
   - `removeNode(n)` : qui permet de supprimer le nœud n
   - `removeEdge(x,y)` : qui supprime un lien entre les nœuds x, y s'il existe.
   - `getNodeById(id)` : qui retourne le nœud ayant l'identifiant est id s'il existe.
   - `getNode(n)` : qui retourne le nième nœud dans le graphe s'il existe.
   - `order()` qui retourne l'ordre du graphe.
   - `size()` : qui retourne la taille du graphe.
   - `density()` : qui retourne la densité du graphe
   - `getNeighbors(x)` : qui retourne la liste de voisins du nœud x.
   - `degree(x)` : qui retourne le degré du nœud x.
   - `degreeDistribution()` : qui retourne la distribution de degrés du graphe.

2. proposer une classe `UndirectedGraph` qui modélise un graphe simple non dirigé. En plus des méthodes citées ci-avant, cette classe offre les méthodes suivantes :

   - `dfs` qui effectue un parcours en profondeur d'abords du graphe
   - `IsConnected()` : qui teste si le graphe est connexe ou non.

3. **Bonus** : proposer une méthode `DFS` qui effectue un parcours en largeur d'abord du graphe. Puis proposer un eméthode pour retourner la plus grande composante connexe d'un graphe.