ITC – cours n°14
Introduction aux graphes

I. Définitions

1.1. Graphe

Définition : graphe non orienté

Un **graphe** non orienté $G = (S, A)$ est déterminé par la donnée de deux ensembles :
- un ensemble fini non vide S dont les éléments u sont appelés **sommets**.
- un ensemble A de paires de sommets $a = \{u, v\}$ appelées **arêtes**.

On se restreindra aux graphes **finis**, c'est-à-dire aux graphes ayant un nombre fini de sommets.

Exemple :

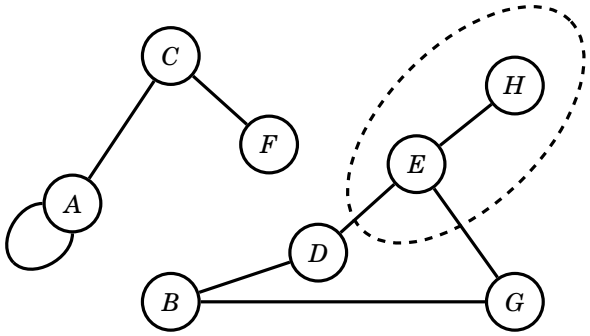


Figure 1 : Exemple de graphe non orienté

Une graphe peut servir à modéliser de nombreuses situations, par exemple :
- un réseau social : les sommets sont des individus, les arêtes marquent un lien de connaissance ;
- un réseau informatique (internet) : les sommets sont des serveurs, les arêtes sont des fibres optiques ;
- un réseau routier : les sommets sont des villes, les arêtes des routes ;
- des applications en robotique, génétique (modélisation de l'ADN), etc...

Définition : graphe orienté

Un graphe est orienté si chaque arête a un sens de parcours, c'est-à-dire un sommet départ et un sommet arrivée. Dans ce cas la on parle d'**arc** (u, v) (mathématiquement un couple et non une paire) au lieu de d'arêtes $\{u, v\}$. On appelle arc sortant d'un sommet un arc dont le départ est le sommet, et arc entrant un arc dont l'arrivée est le sommet.

Exemple :

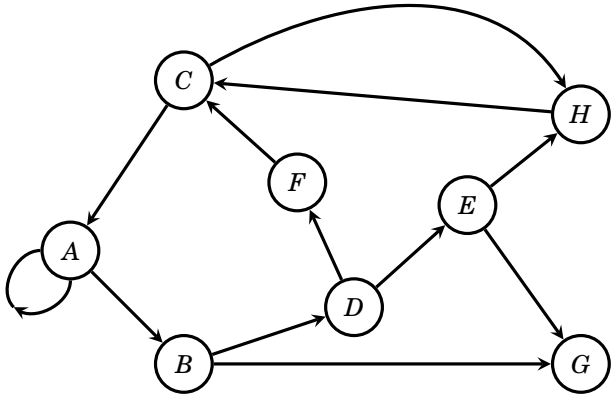


Figure 2 : Exemple de graphe orienté

1.2. Description de la topologie d'un graphe

a) Graphes non orientés

- Deux sommets u et v sont dits **adjacents** (ou voisins) dans le graphe G s'il existe une arête qui les relie : $\{u, v\} \in A$.
- Une arête $\{u, u\}$ reliant un sommet à lui-même est appelé une **boucle**.
- Un graphe sans boucle ayant au plus une arête entre deux sommets est un graphe **simple**.

Dans le cadre du programme, on ne s'intéresse qu'à des graphes ayant au plus une arête entre deux sommets. En l'absence de boucle, il s'agira donc de graphes simples.

- Le **degré** d'un sommet est le nombre d'arêtes contenant ce sommet ; par exemple, dans le graphe de la figure 1, $d(E) = 3$ et $d(A) = 2$.
- Un **chemin** ou une **chaîne** entre deux sommets u et v est une suite de sommets

$$c = (u = s_0, s_1, s_2, \dots s_{n-1}, s_n = v)$$

permettant de relier les deux sommets de proche en proche par des sommets adjacents ;

ainsi,

$$\forall j \in [\![0; n-1]\!] \quad , \quad \{s_j, s_{j+1}\} \in A$$

On dit alors que v est **accessible** depuis u, on notera $u \leftrightarrow v$ (relation symétrique). On dit aussi que les sommets u et v **communiquent**. La longueur n de la chaîne est le nombre d'arêtes contenues dans la chaîne.

Une chaîne est **simple** si elle ne passe pas deux fois par le même sommet.

Exemple : dans le graphe de la figure 1, (B, D, E, H) est une chaîne de longueur 3 reliant les sommets B et H. En revanche, A n'est pas accessible depuis D.

- Un chemin revenant au sommet de départ est appelé un **cycle** ; par exemple (B, D, E, G, D).
- Un **sous-graphe** G' de $G = (S, A)$ est un graphe $G' = (S', A')$ où :
  i. $S' \subset S$ est une partie de l'ensemble des sommets ;
  ii. $A' \subset A$ est constitué uniquement d'arêtes $\{u', v'\}$ (s'il y en a) telles que $u' \in S'$ et $v' \in S'$.

Si l'on se donne simplement une partie S' de S, le **sous-graphe induit (ou engendré)** par G sur S' est défini comme le sous-graphe $G' = (S', A')$ où A' est défini par l'ensemble des arêtes admissibles selon le point (ii). Par exemple, le sous-graphe induit par les sommets E et H est celui entouré en pointillés.

- Une composante connexe d'un graphe G est un sous-graphe $G' = (S', A')$ avec S' un singleton, ou alors vérifiant :
  i. deux sommets distincts u et v dans S' vérifient toujours $u \leftrightarrow v$;
  ii. G' est maximal pour (i), c'est-à-dire que dès qu'on rajoute un ou plusieurs sommet(s) à S', on perd la propriété i.

Un graphe est dit connexe si et seulement s'il possède une unique composante connexe. Dans l'exemple, il y a deux composantes connexes, G n'est donc pas un graphe connexe.

Application : On se donne un ensemble de sept sommets A, B, ...G. Dessiner :

- un graphe G possédant 6 arêtes, 2 composantes connexes : une à 4 sommets, une autre à 3 sommets ;
- un graphe ayant une composante connexe à 2 sommets et une autre à 5 ;
- un graphe connexe à 6 arêtes ; quel est le degré de chaque sommet ?
- un graphe connexe à 10 arêtes ; quel est le degré de chaque sommet ?
- un graphe possédant 7 composantes connexes ;
- combien peut-on avoir d'arêtes possibles dans un graphe simple à 7 sommets ?

b) Graphes orientés

Dans le cas des graphes orientés, les définitions précédentes s'adaptent :

- On parle d'**arc** plutôt que d'arête.
- La notion de degré d'un sommet doit être précisée : on parle de **degré entrant** (noté d_-) pour un arc arrivant sur le sommet, et de **degré sortant** (noté d_+) pour un arc partant du sommet. Par exemple, sur le graphe de la figure 2, $d_-(D) = 1$ et $d_+(D) = 2$.
- Le terme **chaîne** ou **chemin** désigne dans ce cas un ensemble de sommets reliés par des arcs ; la relation d'accessibilité n'est donc plus nécessairement symétrique. On note $u \to v$ l'existence d'une chaîne allant de u à v, et on dit que v est un **descendant** de u. Si la relation est effectivement symétrique, on peut réutiliser la notation $u \leftrightarrow v$.
- Un chemin dont les deux extrémités sont égales est appelé **circuit**.
- La notion de connexité est remplacée par celle de composante **fortement connexe** : une composante fortement connexe d'un graphe orienté est un sous-graphe dans lequel chaque sommet est le descendant de chacun des autres. Autrement dit, on peut atteindre n'importe quel sommet d'une composante fortement connexe depuis n'importe quel autre sommet de cette même composante. Par exemple, le graphe de la figure 2 contient deux composantes fortement connexes, $\{G\}$ et $S \setminus \{G\}$.

Application : On se donne un ensemble de sept sommets A, B, ...G. Dessiner :

- un graphe G possédant 8 arcs, 2 composantes fortement connexes ;
- un graphe ayant une composante fortement connexe à 2 sommets et une autre à 5 ;
- un graphe connexe à 7 arcs ; quel est le degré entrant/sortant de chaque sommet ?
- un graphe connexe à 10 arcs ; quel est le degré entrant/sortant de chaque sommet ?

1.3. Pondération, informations supplémentaires

On peut ajouter une ou des informations supplémentaires à un sommet et à une arête. Le premier exemple (et de loin le principal) est celui de la **pondération** des arêtes : on associe un poids à chaque arête, de façon similaire à une distance ou un temps ; ceci est par exemple utilisé pour représenter un temps de trajet ou un kilométrage pour un réseau routier / de transports en commun. Pour un arc (u, v), on notera $w_{(u,v)}$[1] son poids.

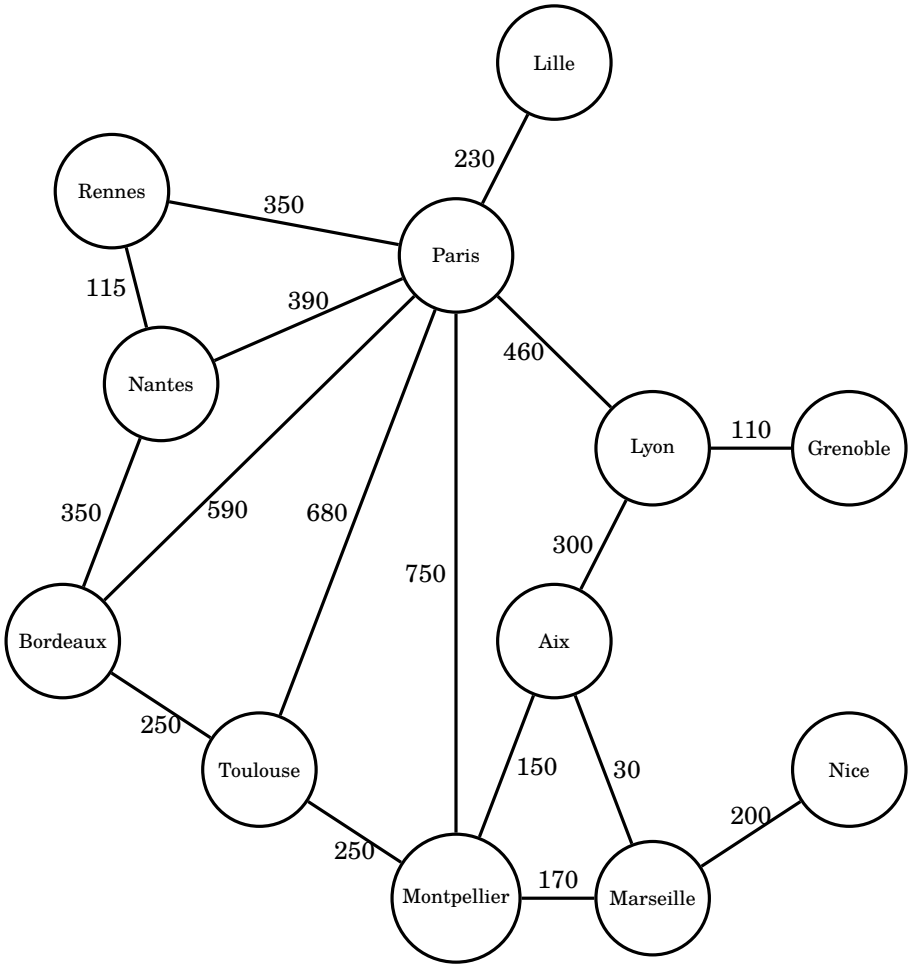


Figure 3 : Graphe simplifié du réseau autoroutier

D'une manière plus générale, il peut être intéressant d'ajouter des informations supplémentaires sur les sommets ou les arêtes/arcs. Dans le cas du réseau autoroutier ci-dessus, on pourrait ajouter un temps de traversée des villes, une vitesse moyenne sur les routes...

Pour proposer une utilisation très complète, prenons l'exemple d'un graphe représentant les relations sur un réseau social (type Mastodon) :

- les sommets seraient les utilisateurs ;
- on utiliserait un graphe orienté, les arcs représentant le fait de suivre quelqu'un ; le degré entrant est donc le nombre de followers d'un sommet ;
- on peut caractériser la « force » du lien en mettant sur chaque arc le nombre de like, de réponses ou de retoot, donc trois informations supplémentaires pour chaque arc ;
- on peut également ajouter à chaque arc la date à laquelle le follow a eu lieu, et éventuellement celle à laquelle il a été rompu ;
- on peut ajouter sur chaque sommet la date de création du compte, le nombre de posts etc...

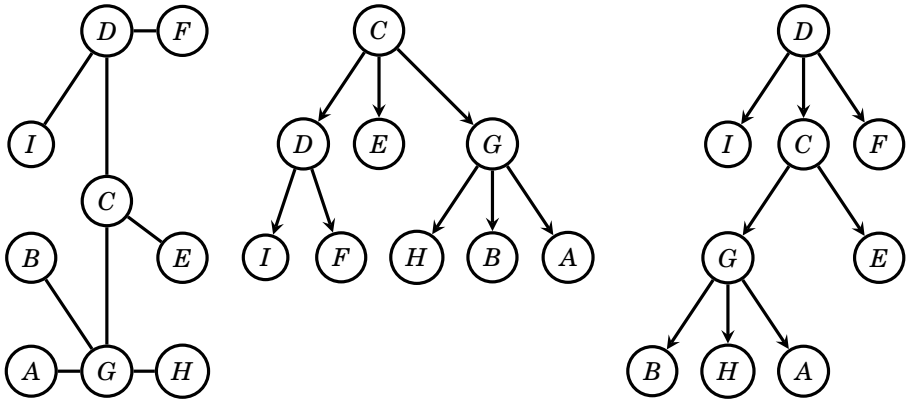
Finalement, on peut se concentrer sur la topologie d'un graphe (sa description, son parcours, éventuellement avec des arêtes pondérées), et c'est ce sur quoi porteront les algorithmes que nous allons étudier ; cependant, en ajoutant des informations supplémentaires, on peut interroger de très nombreuses problématiques différentes.

1.4. Arbres

Les arbres sont un cas particulier de graphes : ce sont des **graphes simples non orientés connexes et acycliques**. Si le graphe n'est pas connexe mais reste acyclique, on parle de **forêt**, chaque composante connexe étant un arbre.

L'absence de cycle conduit à des représentations assez cohérentes avec la notion d'arbre, avec des branches. On peut **enraciner** l'arbre en choisissant un sommet comme racine, et on obtient alors une relation orientée entre un sommet parent (relié par un chemin simple « vers la racine ») et un sommet enfant ; on peut donc orienter l'arbre (ce n'est pas obligatoire). On appelle feuille un sommet n'ayant pas de descendant.

[1] w pour « weight »



Arbre simple Arbre enraciné en C Arbre enraciné en D

Figure 4 : Exemple d'arbre

Les arbres ont de nombreuses applications algorithmiques, et il en existe de très nombreuses formes selon les propriétés recherchées (arbres binaires de recherche, arbres rouge-noir...) ; dans le cadre de notre programme, nous nous contenterons de les voir comme un cas particulier de graphes.

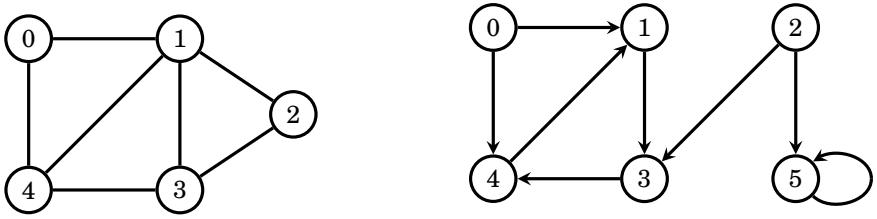
II. Représentation informatique

2.1. Matrice d'adjacence

Une première représentation possible est celle en matrice : si on numérote les sommets de 0 à $n_S - 1$ et on définit une matrice g telle que

$$g_{ij} = \begin{cases} 1 & \text{si } \{s_i, s_j\} \in A \text{ ;} \\ 0 & \text{sinon} \end{cases}$$

Si le graphe est non orienté, la matrice est nécessairement symmétrique. Par exemple, on considère les deux cas de la figure 5 :



Graphe non orienté G_1 Graphe orienté G_2

Figure 5 : Représentation de graphes : exemples

leurs matrices d'adjacence sont

$$M_1 = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix} \quad \text{et} \quad M_2 = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Écrivons les matrices d'adjacence des graphes représentés en figures 1, 2 et 4

$$\begin{pmatrix} - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - \end{pmatrix} \quad \begin{pmatrix} - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - \end{pmatrix} \quad \begin{pmatrix} - & - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - & - \end{pmatrix}$$

Si le graphe est pondéré, on peut choisir d'écrire le poids de l'arête a_{ij} (si elle existe) dans g_{ij}.

Le principal avantage de la représentation par matrice d'adjacence est de donner accès à l'existence ou non d'une arête en $\mathcal{O}(1)$; en revanche, elle occupe une place $\mathcal{O}(n_s^2)$ en mémoire, ce qui pose des contraintes pour les très gros graphes[2] ; en particulier, si le graphe est très « creux », c'est-à-dire si $n_A \ll n_S^2$, la matrice contient essentiellement des zéros. Par ailleurs, la matrice d'adjacence n'est pas la représentation la plus naturelle pour les algorithmes de parcours.

[2] Pour un graphe non orienté, on peut diviser par deux la taille puisque la matrice est symétrique, mais cela reste un problème

2.2. Listes d'adjacence

La représentation en listes d'adjacence consiste à fournir, pour chaque sommet, la liste des sommets qu'il peut atteindre : pour les deux graphes de la figure 5, cela donne

$$G_1 = \begin{cases} 0 : & [1,4] \\ 1 : & [0,2,3,4] \\ 2 : & [1,3] \\ 3 : & [1,2,4] \\ 4 : & [0,1,3] \end{cases} \quad \text{et} \quad G_2 = \begin{cases} 0 : & [1,4] \\ 1 : & [3] \\ 2 : & [3,5] \\ 3 : & [4] \\ 4 : & [1] \\ 5 : & [5] \end{cases}$$

Écrivons l'ensemble des listes d'adjacence pour les graphes représentés en figures 1, 2 et 4 :

$$\begin{cases} A : \\ B : \\ C : \\ D : \\ E : \\ F : \\ G : \\ H : \end{cases} \qquad \begin{cases} A : \\ B : \\ C : \\ D : \\ E : \\ F : \\ G : \\ H : \end{cases} \qquad \begin{cases} A : \\ B : \\ C : \\ D : \\ E : \\ F : \\ G : \\ H : \\ I : \end{cases}$$

En pratique, on peut soit utiliser des tableaux contigüs (listes Python), soit des listes chaînées ; les algorithmes les plus courants doivent de toutes façon passer en revue tous les sommets adjacents, donc on peut trouver avantageux la liste chaînée pour ajouter ou retirer des sommets. On peut également utiliser des dictionnaires pour tester la présence d'une clé en $\mathcal{O}(1)$ et stocker aisément une éventuelle pondération.

L'avantage principal de la liste d'adjacence est de prendre moins de place, en l'occurrence $\mathcal{O}(n_S + n_A)$; son inconvénient principal est de demander un temps $\mathcal{O}(n_A)$ pour déterminer si un arc (u,v) se trouve dans le graphe : il faut parcourir toute la liste d'adjacence de u. La représentation avec des dictionnaires d'adjacence évite cet éceuil.

2.3. Choix pratique dans ce cours

Dans ce cours et les TD associés, nous utiliserons peu les matrices d'adjacence ; mais si on souhaite les utiliser, nous les représenterons avec des listes Python de listes Python, comme nous avons généralement fait pour simuler les matrices.

Dans la plupart des cas, nous représenterons un graphe comme un **dictionnaire de dictionnaires d'adjacence** ; les sommets sont identifiés par un `str` qui servent de clé. Si une arête (u,v) existe le dictionnaire $g[u]$ contient la clé v, et la valeur $g[u][v]$ vaut 1 (ou le poids de l'arête si le graphe est pondéré). Pour créer « à la main » le graphe de la figure 1, on pourrait donc écrire le code suivant :

```
graphe = {
  'A': {'A': 1, 'C': 1},
  'B': {'D': 1, 'G': 1},
  'C': {'A': 1, 'F': 1},
  'D': {'B': 1, 'E': 1},
  'E': {'D': 1, 'G': 1, 'H': 1},
  'F': {'C': 1},
  'G': {'B': 1, 'E': 1},
  'H': {'E': 1}
}
```

La liste des sommets est donc la liste des clés dans le dictionnaire.

Nous utiliserons donc la même représentation pour les graphes orientés ou non orientés. On pourra déterminer si un graphe est orienté en vérifiant si tout sommet u dans la liste d'adjacence de v est tel que v soit dans la liste d'adjacence de u.

On utilisera également des dictionnaires pour les informations supplémentaires : les informations sur les sommets auront comme clés les sommets, et celles sur les arêtes auront les tuples représentant une arête ; par exemple, si on ajoute une priorité au graphe de de la figure 1 :

```
prio = {
  'A': 4,
  'B': 2,
  'C': 13,
  'D': 7,
  'E': 2,
  ...
}
```