Cours de la théorie des graphes
Pour les étudiants de la première année Master Mathématiques A+F
Département de mathématiques
Centre Universitaire Abdelhafid Boussouf, Mila
Anné universitaire 2023/2024

Chapitre 3, Arbres et arborescences

Table des matières

1

Arbres et arborescences

1.1 Arbre

Un arbre est un graphe connexe sans cycle.
D'autres définitions équivalentes sont possibles pour qu'un graphe G d'ordre n soit un arbre :

- G est connexe et possède $n-1$ arêtes.
- G est sans cycle et possède $n-1$ arêtes.
- G est connexe et minimal pour cette propriété.
- G est sans cycle et maximal pour cette propriété.
- Entre toute paire de sommets, il existe une unique chaîne les reliant.

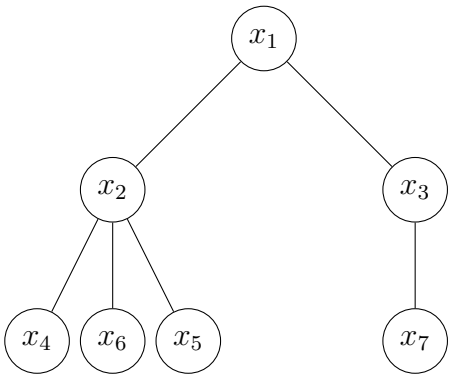

FIGURE 1.1 – Un arbre T où x_1 est sa racine, x_2 et x_3 sont des noeuds internes et x_4, x_5, x_6 et x_7 sont des feuilles.

1.1.1 Arbre n-aire complet

Pour tous entiers k ≥ 1 et t ≥ 2, on appelle arbre t-aire complet de profondeur k, le graphe $A_{k,t}$ défini inductivement comme suit :

- $A_{1,t}$ est le graphe biparti complet $K_{1,t}$.
- Pour k ≥ 2, $A_{k,t}$ est obtenu à partir de t copies disjointes de $A_{k-1,t}$ et d'un sommet relié par une arête à l'unique sommet de degré t de chacune des t copies de $A_{k-1,t}$.

L'unique sommet r de $A_{k,t}$ de degré t est la racine de $A_{k,t}$.
Soient k, $t \in \mathbf{N}^*$. Pour tout entier $0 \leq$ i $\leq$ k, on définit le i^{eme} *niveau* de $A_{k,t}$ par $V_i(A_{k,t}) = \{u \in V(A_{k,t}), d(u,r) = i\}$. En particulier, $V_k(A_{k,t})$ est l'ensemble des sommets pendants de $A_{k,t}$.
Pour un sommet $v \in V_i(A_{k,t})$, on écrit $l(v) = i$ et on dit que i est la profondeur de v. Enfin, pour tout sommet $u \in V(A_{k,t})$, on note par $A_{k,t}(u)$ l'unique sous-arbre binaire complet de $A_{k,t}$ de racine u et de profondeur $k - l(v)$.
Si t = 2, $A_{k,2}$ est appelé arbre binaire de profondeur k.

1.1.2 Arbre binaire complet

un arbre binaire enraciné est un arbre binaire complet ssi chaque sommet d'arbre possède exactement deux fils sauf les feuilles .

Exercice 1.1 *Calculer le nombre de sommets d'un arbre binaire complet en fonction de son profondeur k.*

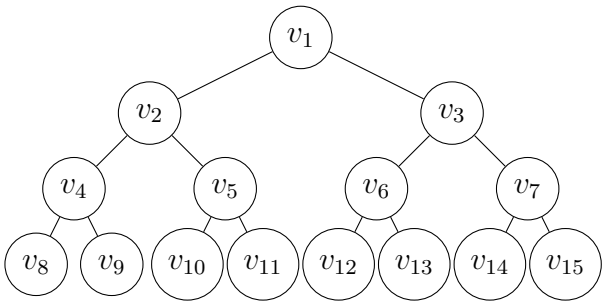

FIGURE 1.2 – Arbre binaire complet

1.2 Forêt

est un graphe non connexe et sans cycle.

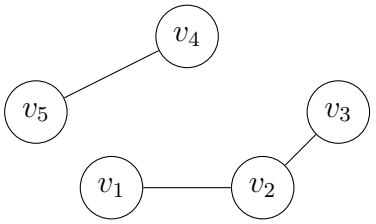

FIGURE 1.3 – Forêt

Dans cette section, nous présentons quelques algorithmes de cheminement dans les graphes orientés et les graphes non orientés. Nous commençons par le recherche d'un arbre couvrant de poids minimum dans un graphe non orienté.

1.3 Arbre couvrant de poids minimum

1.3.1 Arbre couvrant :

Un arbre couvrant (aussi appelé arbre maximal) est un graphe partiel qui est aussi un arbre. On distingue trois types de sommets dans un arbre enraciné :

- La racine.
- Les feuilles : ce sont les sommets de degré égal à 1.
- Les noeuds internes : ce sont les sommets de degré supérieur à 1.

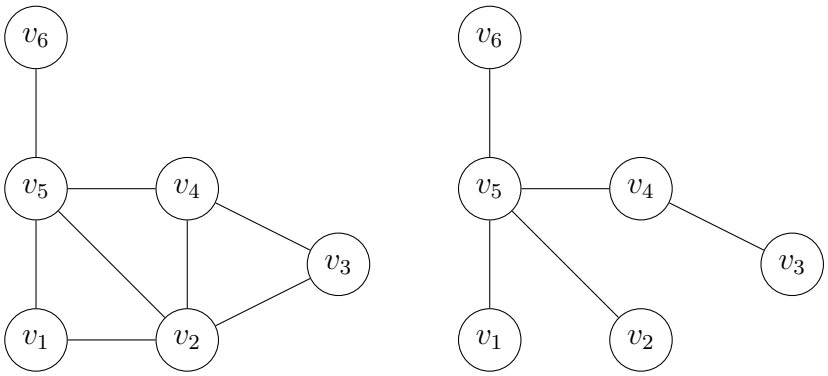

FIGURE 1.4 – Graphe G et arbre couvrant sur le graphe G

1.3.2 Arbre couvrant de poids minimum :

Soit le graphe G = (V,E) avec un poids associé à chacune de ses arêtes. On veut trouver, dans G, un arbre maximal A = (V,F) de poids total minimum.

Algorithme de Kruskal (1956) :

Données :

- Graphe G = (V,E) (|V| = n, |E| = m)
- Pour chaque arête e de E , son poids c(e).

Résultat : Arbre ou forêt maximale A = (V,F) de poids minimum.

- Trier et renuméroter les arêtes de G dans l'ordre croissant de leur poids : $c(e_1) \leq c(e_2) \leq c(e_3)$ ... $\leq c(e_m)$.
- Poser F := $\emptyset$, k := 0
- Tant que k < m et |F| < n - 1 faire
  Début
  si e_{k+1} ne forme pas de cycle avec F alors F := F $\bigcup \{e_{k+1}\}$
  k := k + 1
  Fin

Exemple 1.1 *Appliquer l'algorithme de Kruskal pour déterminer un arbre couvrant de poids minimum sur le graphes G suivant :*

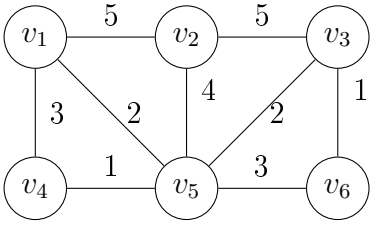

FIGURE 1.5 – G

1.4 Arborescences

En mathématiques, plus précisément dans la théorie des graphes, une arborescence est un arbre comportant un sommet particulier r, nommé racine de l'arborescence, à partir duquel il existe un chemin unique vers tous les autres sommets.
Structure arborescente de fichiers informatique En informatique, cette notion désigne souvent celle d'arbre de la théorie des graphes. Une arborescence désigne alors généralement une organisation des données en mémoire, de manière logique et hiérarchisée, utilisant une structure algorithmique d'arbre. Cette organisation rend plus efficace la consultation et la manipulation des données stockées. Les usages les plus courants en sont : l'arborescence de fichiers, qui est l'organisation hiérarchique des fichiers sur une partition, et dans certains cas de partitions entre elles – par exemple : partitions virtuelles (« lecteurs logiques ») dans des partitions réelles ; le tri arborescent en mémoire ; les fichiers en mode séquentiel indexé.
La logique générale de l'arborescence coïncide avec le modèle relationnel du SQL : 1 vers N et réciproquement 1 vers 1. Un nœud peut posséder N feuilles, mais chaque feuille n'est possédée que par un seul nœud.

En informatique, elle désigne aussi un composant d'interface graphique qui présente une vue hiérarchique de l'information. Chaque élément (souvent appelé branche ou nœud) peut avoir un certain nombre de sous-éléments. Ceci est souvent représenté sous forme d'une liste indentée. Un élément peut être déplié pour révéler des sous-éléments, s'ils existent, et replié pour cacher des sous-éléments. La vue en arborescence apparaît souvent dans les applications de gestion de fichiers, où elle permet à l'utilisateur de naviguer dans les répertoires du système de fichiers. Elle est également utilisée pour présenter les données hiérarchiques, comme un document XML.
Usage pour la gestion des disques Structure arborescente de fichiers informatique à la base d'une arborescence se trouve un répertoire appelé la racine. Ce répertoire peut contenir des fichiers et des répertoires, qui eux-mêmes peuvent contenir la même chose. Si les fichiers et les répertoires sont placés de manière cohérente, la recherche de fichier est relativement aisée et rapide.

Forêt : Une forêt est un graphe qui contient plusieurs arbres ou arborescences distinctes.

Définition 1.1 *Étant donné un graphe orienté G=(V,E). Une arborescence est un graphe oriente sans circuit admettant une racine r telle que pour tout autre sommet si $v \in V$, il existe un chemin unique allant de r vers u. Si l'arborescence comporte n sommets, alors elle comporte exactement n - 1 arcs.*

Arborescence couvrante : On parcourt un graphe a partir d'un sommet donné r. Cette arborescence contient un arc (v_i, v_j) si et seulement si le sommet v_j a eté decouvert a partir du sommet v_i. L'arborescence associée a un parcours de graphe sera memorisée dans un tableau Π tel que $\Pi(v_j) = v_i$. Si v_j a été découvert a partir de v_i, et $\Pi(v_k) = nul$ si v_k est la racine, ou s'il n'existe pas de chemin de la racine vers v_k.
Parcours en largeur (Breadth First Search = BFS) :

Le parcours en largeur est obtenu en gérant la liste d'attente au coloriage comme une file d'attente (FIFO = First In First Out). Autrement dit, on enlève a chaque fois le plus vieux sommet gris dans la file d'attente, et on introduit tous les successeurs blancs de ce sommet dans la file d'attente, en les coloriant en gris.
Structures de donnees utilisees : On utilise une file F, pour laquelle on suppose definies les operations :
$init - file(F)$ qui initialise la file F a vide,
$ajoute - fin - file(F, u)$ qui ajoute le sommet u a la fin de la file F,
$est - vide(F)$ qui retourne vrai si la file F est vide et faux sinon,
et $enleve - debut - file(F, u)$ qui enleve le sommet u au debut de la file F.
On utilise un tableau Π qui associe a chaque sommet le sommet qui a fait entrer dans la file, et un tableau *couleur* qui associe a chaque sommet sa couleur (blanc, gris ou noir).
On va en plus utiliser un tableau d qui associe a chaque sommet son niveau de profondeur par rapport au sommet de depart r (autrement dit, $d[v_i]$ est la longueur du chemin dans l'arborescence Π de la racine r jusqu'au sommet v_i). Ce tableau sera utilisé plus tard.
Algorithme
$init - file(F)$
pour tout sommet $v_i \in V$ faire
$\Pi(v_i) \longleftarrow nil$
$d[v_i] \longleftarrow \infty$
couleur$(v_i) \longleftarrow$ blanche
fin pour
$d[r] \longleftarrow 0$
$ajoute - fin - file(F, r)$
$couleur[r] \longleftarrow$ gris
tant que $est - vide(F) =$ faux faire
$enleve - debut - file(F, v_i)$
pour tout $v_i \in succ(v_i)$ faire
si $couleur[v_i] =$ blanche alors

$ajoute - fin - file(F, v_i)$
$couleur[v_i] \longleftarrow$ gris
$\Pi(v_j) \longleftarrow v_i$
$d[v_i] \longleftarrow d[v_i] + 1$
fin si
fin pour
$couleur[v_i] \longleftarrow$ noir
fin tant
fin BFS

Applications du parcours en largeur

depuis r. À la fin de l'exécution de $BFS(V; E; r)$, chaque sommet est soit noir soit blanc. Le parcours en largeur peut être utilisé pour rechercher l'ensemble des sommets accessibles sommets noirs sont ceux accessibles depuis r ; les sommets blancs sont ceux pour lesquels il n'existe pas de chemin/chaîne à partir de r. D'une façon plus générale, le parcours en largeur permet de déterminer les composantes connexes d'un graphe non orienté. Pour cela, il suffit d'appliquer l'algorithme de parcours en largeur à partir d'un sommet blanc quelconque. À la suite de quoi, tous les sommets en noirs appartiennent à la première composante connexe. S'il reste des sommets blancs, cela implique qu'il y a d'autres composantes connexes. Il faut alors relancer le parcours en largeur sur le sous-graphe induit par les sommets blancs, pour découvrir une autre composante connexe. Le nombre de fois où l'algorithme de parcours en largeur a été lancé correspond au nombre de composantes connexes. Le parcours en largeur peut aussi être utilisé pour chercher le plus court chemin (en nombre d'arcs ou arêtes) entre la racine r et chacun des autres sommets du graphe accessibles depuis r. Pour cela, il suffit de remonter dans l'arborescence Π du sommet concerné jusqu'à la racine r. L'algorithme 2 (récursif) affiche le plus court chemin pour aller de r à v_j.

Algorithme 2 : plus-court-chemin(r, v_j, Π)
Algorithme 2 : plus-court-chemin(r ; v_j ; Π)
Entrées : r sommet de départ, v_j sommet d'arrivée, Π arborescence couvrante. 1 si r = v_j alors
2 afficher(r)
3 sinon si $\Pi(v_j)$ = nil alors
4 afficher("pas de chemin")
5 sinon
6 plus-court-chemin(r, $\Pi(v_j)$, Π)
7 afficher v_j

1.5 Graphe orienté

En donnant un sens aux arêtes d'un graphe, on obtient un digraphe (ou graphe orienté). Le mot « digraphe » est la contraction de l'expression anglaise « directed graph ».
Un digraphe fini G = (V,E) est défni par l'ensemble V = $\{v_1, v_2, ..., v_n\}$ dont les éléments sont appelés sommets, et par l'ensemble E = $\{e_1, e_2, ..., e_m\}$ dont les éléments sont appelés arcs.
Un arc e de l'ensemble E est défini par une paire ordonnée de sommets. Lorsque e = (u,v), on dit que l'arc e va de u à v. On dit aussi que u est l'extrémité initiale et v l'extrémité finale de e.

1.5.1 Degré d'un sommet d'un digraphe

Soit v un sommet d'un graphe orienté.
On note $d^+(v)$ le degré extérieur du sommet v, c'est-à-dire le nombre d'arcs ayant v comme extrémité initiale.
On note $d^-(v)$ le degré intérieur du sommet v, c'est-à-dire le nombre d'arcs ayant v comme extrémité finale.
On définit le degré : $d(v) = d^+(v) + d^-(v)$

1.5.2 Chemins et circuits

Un chemin conduisant du sommet a au sommet b est une suite ayant pour éléments alternativement des sommets et des arcs, commençant et se terminant par un sommet, et telle que chaque arc est encadré à gauche par son sommet origine et à droite par son sommet destination. On ne peut donc pas prendre les arc à rebours. Sur le digraphe ci-après, on peut voir par exemple le chemin $(v_3, e_2, v_2, e_1, v_1)$. Par convention, tout chemin comporte au moins un arc.
On appelle distance entre deux sommets d'un digraphe la longueur du plus petit chemin les reliant. S'il n'existe pas de chemin entre les sommets x et y, on pose d(x,y) = ∞. Par exemple, sur le digraphe ci-dessous, $d(v_5, v_4) = 2$, $d(v_4, v_5) = \infty$, $d(v_3, v_1) = 1$.
Un circuit est un chemin dont les sommets de départ et de fin sont les mêmes. Le digraphe ci-dessus ne contient pas de circuit.
Les notions de chemins et de circuits sont analogues à celles des chaînes et des cycles pour les graphes non orientés. **Digraphe fortement connexe** Un digraphe est fortement connexe, si toute paire ordonnée (a,b) de sommets distincts du graphe est reliée par au moins un chemin. En d'autres termes, tout sommet est atteignable depuis tous les autres sommets par au moins un chemin.

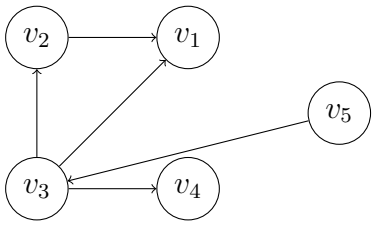

FIGURE 1.6 – G

On appelle composante fortement connexe tout sous-graphe induit maximal fortement connexe (maximal signifie qu'il n'y a pas de sous-graphe induit connexe plus grand contenant les sommets de la composante).

1.5.3 Représentations non graphiques des digraphes

Matrice d'adjacences On peut représenter un digraphe par une matrice d'adjacences. Une matrice (n×m) est un tableau de n lignes et m colonnes. (i, j) désigne l'intersection de la ligne i et de la colonne j.
Dans une matrice d'adjacences, les lignes et les colonnes représentent les sommets du graphe. Un « 1 » à la position (i, j) signifie qu'un arc part de i pour rejoindre j.

Cette matrice a plusieurs caractéristiques :

1. Elle est carrée : il y a autant de lignes que de colonnes.
2. Il n'y a que des zéros sur la diagonale. Un « 1 » sur la diagonale indiquerait une boucle.
3. Contrairement à celle d'un graphe non orienté, elle n'est pas symétrique.
4. Une fois que l'on fixe l'ordre des sommets, il existe une matrice d'adjacences unique pour chaque digraphe. Celle-ci n'est la matrice d'adjacences d'aucun autre digraphe.

1.5.4 Algorithme de Dijkstra

Edgser Wybe Dijkstra (1930-2002) a proposé en 1959 un algorithme qui permet de calculer le plus court chemin entre un sommet particulier et tous les autres. Le résultat est une arborescence, c'est-à-dire un arbre avec un sommet particulier appelé racine. Numérotons les sommets du graphe G = (V,E) de 1 à n. Supposons que l'on s'intéresse aux chemins partant du sommet 1. On construit un vecteur $\lambda = (\lambda(1), \lambda(2), \lambda(3), ..., \lambda(n)$ ayant n composantes tel que $\lambda(j)$ soit égal à la longueur du plus court chemin allant de 1 au sommet j.

On initialise ce vecteur à c_{1j} , c'est-à-dire à la première ligne de la matrice des coûts du graphe, définie comme indiqué ci-dessous :

$$c_{ij} = \begin{cases} 0 & si \quad i = j, \\ \infty & si \quad i \neq j \quad et \quad (i,j) \quad n'appartient \quad E, \\ \delta(i,j) & si \quad i \neq j \quad et \quad (i,j) \in E. \end{cases}$$

où $\delta(i,j) > 0$ est le poids de l'arc (i,j).
On construit un autre vecteur p pour mémoriser le chemin pour aller du sommet 1 au sommet voulu. La valeur $p(i)$ donne le sommet qui précède i dans le chemin.
On considère ensuite deux ensembles de sommets, S initialisé à $\{1\}$ et T initialisé à $\{2, 3, ..., n\}$. À chaque pas de l'algorithme, on ajoute à S un sommet jusqu'à ce que S = V de telle sorte que le vecteur λ donne à chaque étape la longueur minimale des chemins de 1 aux sommets de S.

Résumé de l'algorithme de Dijkstra

On suppose que le sommet de départ (qui sera la racine de l'arborescence) est le sommet numéroté 1. Notons qu'on peut toujours renuméroter les sommets pour que ce soit le cas.
Initialisations
$\lambda(j) = c_1$ et $p(j)$ = NIL, pour $1 \leq j \leq$ n.
Pour $2 \leq j \leq$ n faire
Si $c_{1j} < \infty$ alors $p(j) = 1$.
S=$\{1\}$, T = $\{2, 3, ..., n\}$.
Itérations
Tant que T n'est pas vide faire
Choisir i dans T tel que $\lambda(i)$ est minimum
Retirer i de T et l'ajouter à S
Pour chaque successeur j de i, avec j dans T , faire
Si $\lambda(j) > \lambda(i) + \delta(i,j)$ alors $\lambda(j) = \lambda(i) + \delta(i,j)$
$p(j) = i$

Exerice

Appliquez l'algorithme de Dijkstra au graphe ci-dessous pour trouver tous les plus courts chemins en partant du sommet 5.