Cours de la matière : Théorie des graphes
Pour les étudiants de la première année Master Mathématiques Appliquée et Fondamentales
Département de mathématiques
Centre Universitaire Abdelhafid Boussouf, Mila
Anné universitaire 2025/2026

Cours N : 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.

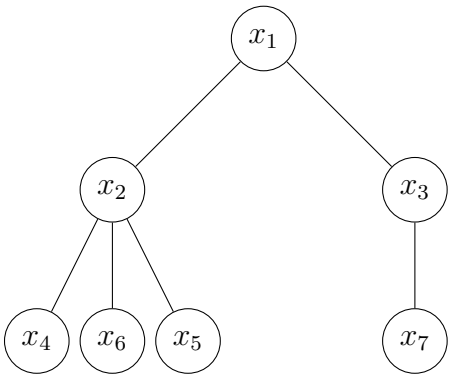


FIGURE 1.1 – Un arbre T où x_1 est sa racine, x_2 et x_3 sont des noeuds internes et x_4, x_5, x_6 et x_7 sont des feuilles.

1.1.1 Arbre n-aire complet

Pour tous entiers k $\geq$ 1 et t $\geq$ 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 $\geq$ 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 sa profondeur k.*

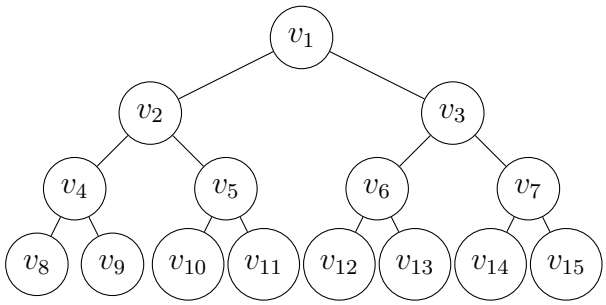


FIGURE 1.2 – Arbre binaire complet

1.2 Forêt

est un graphe non connexe et sans cycle.

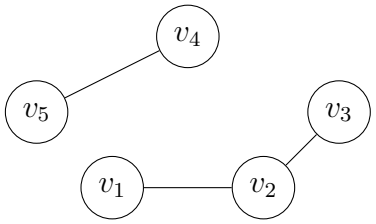


FIGURE 1.3 – Forêt

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.

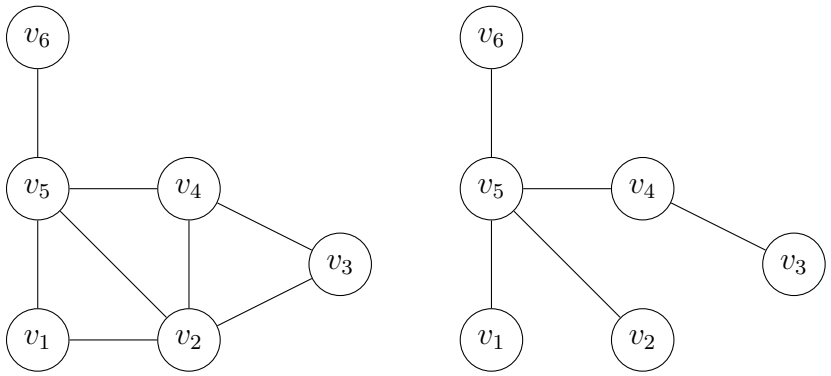


FIGURE 1.4 – Graphe G et arbre couvrant sur le graphe G

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 :*

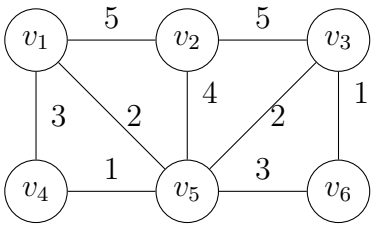


FIGURE 1.5 – G

1.4 Arborescences

En théorie des graphes, une **arborescence** est un graphe orienté sans circuit possédant une racine r telle que, pour tout sommet $v \in V$, il existe un chemin orienté unique de r vers v. Une arborescence à n sommets contient exactement $n - 1$ arcs.

Forêt : Une forêt est un graphe composé de plusieurs arborescences disjointes.

Arborescence couvrante

Soit $G = (V, E)$ un graphe orienté. Une arborescence couvrante issue d'un sommet r est obtenue lors d'un parcours du graphe à partir de r. Elle contient un arc (v_i, v_j) si v_j est découvert à partir de v_i.

Elle est représentée par un tableau Π tel que :

$$\Pi(v_j) = v_i \quad \text{et} \quad \Pi(r) = \texttt{nil}.$$

Parcours en largeur (BFS)

Le parcours en largeur explore les sommets par niveaux en utilisant une file FIFO.

Structures de données :

- une file F définie par les opérations :
  - `init-file(F)` : initialise la file vide,
  - `ajoute-fin-file(F, u)` : insère u en fin de file,
  - `enleve-debut-file(F)` : retire et retourne le premier élément,
  - `est-vide(F)` : teste si la file est vide.
- un tableau Π (prédécesseurs),

— un tableau d (distance à la racine),
— un tableau `couleur` (blanc, gris, noir).

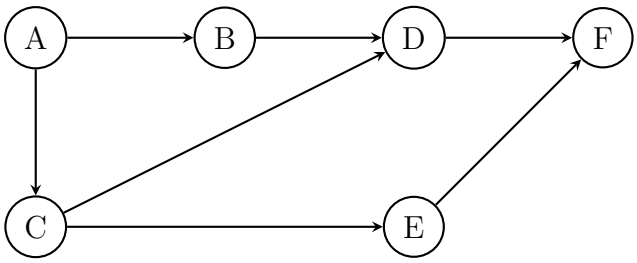
Algorithme BFS :

```
init-file(F)
pour tout sommet v dans V faire
    Pi[v] <- nil
    d[v] <- infini
    couleur[v] <- blanc
fin pour

d[r] <- 0
ajoute-fin-file(F, r)
couleur[r] <- gris

tant que est-vide(F) = faux faire
    v <- enleve-debut-file(F)
    pour tout u dans successeurs(v) faire
        si couleur[u] = blanc alors
            ajoute-fin-file(F, u)
            couleur[u] <- gris
            Pi[u] <- v
            d[u] <- d[v] + 1
        fin si
    fin pour
    couleur[v] <- noir
fin tant
```

Exemple d'application

Considérons le graphe orienté :

$$V = \{A, B, C, D, E, F\}, \quad E = \{(A, B), (A, C), (B, D), (C, D), (C, E), (D, F), (E, F)\}.$$



On applique BFS à partir de A.

Étapes :

Étape	File F	Π	d
Initial	A	$\Pi(A) = nil$	$d(A) = 0$
1	B, C	$\Pi(B) = A$, $\Pi(C) = A$	$d = 1$
2	C, D	$\Pi(D) = B$	$d(D) = 2$
3	D, E	$\Pi(E) = C$	$d(E) = 2$
4	E, F	$\Pi(F) = D$	$d(F) = 3$
5	F	–	–

Arborescence obtenue :

$$\{(A, B), (A, C), (B, D), (C, E), (D, F)\}.$$

Applications

Le BFS permet :

- de déterminer les sommets accessibles,
- de calculer les plus courts chemins (en nombre d'arcs),
- de déterminer les composantes connexes.

Plus court chemin

Pour reconstruire le chemin de A à F :

$$F \leftarrow D \leftarrow B \leftarrow A \quad \Rightarrow \quad A \rightarrow B \rightarrow D \rightarrow F.$$

Algorithme :

```
Procédure PlusCourtChemin(r, v, Pi)
    si r = v alors
        afficher(r)
    sinon si Pi[v] = nil alors
        afficher("pas de chemin")
    sinon
        PlusCourtChemin(r, Pi[v], Pi)
        afficher(v)
    fin si
fin procédure
```