

## Chapitre 2

# Arbres et arborescences

### 2.1 Introduction

Les arbres représentent une classe de graphes largement utilisée pour la représentation des données et la résolution des problèmes. Ils permettent de représenter des relations de hiérarchie et s'apprêtent bien aux traitements récursifs et les traitements efficaces. Par exemple, pour le problème de tri d'un tableau, le tri par tas offre une complexité très intéressante par rapport à d'autres algorithmes de tri. Cet algorithme place les éléments dans un arbre binaire avant de les trier. Ce chapitre introduit le concept des arbres d'un point de vue de la théorie des graphes et présentent la méthode de couvrir un graphe par un arbre.

### 2.2 Concepts généraux

Étant donné que les arbres font partie des graphes simples, on ne considère que les graphes simples dans ce chapitre.

#### 2.2.1 Définitions générales et propriétés

Un *arbre* est un graphe non-orienté connexe et sans cycle. Si le graphe est d'ordre  $n$ , alors il contient  $n - 1$  arêtes (comment peut-on montrer cela?). La figure 2.1 donne un exemple d'un arbre ayant cinq sommets (avec 4 arêtes).

Pour un arbre  $G$  d'ordre  $n$  ( $n \geq 2$ ), les propriétés suivantes sont équivalentes :

1.  $G$  est connexe et sans cycle (on dit qu'il est acyclique) ;
2.  $G$  est connexe et minimal, en d'autres termes, si on enlève une arête alors il ne sera plus connexe ;
3.  $G$  est sans cycle et maximal, en d'autres termes, si on lui rajoute encore une arête alors il contiendra un cycle.

4. Il existe une chaîne unique entre deux sommets quelconques de  $G$ .

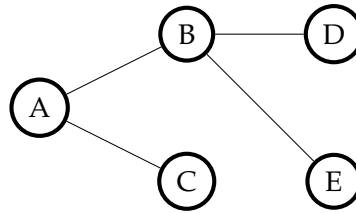


Figure 2.1 : exemple d'un arbre d'ordre 5

Il est fort instructif de procéder à la démonstration de ces équivalences pour les étudiants intéressés.

Tout sommet pendant d'un arbre s'appelle *feuille*. Dans le graphe considéré ici, les feuilles sont : C, D, E.

On peut également énoncer les propriétés suivantes (dont la démonstration peut constituer également un bon exercice) :

- Dans un arbre d'ordre supérieur à 1, il existe au moins deux sommets pendants (donc deux feuilles);
- Un arbre peut être colorié avec deux couleurs, il est donc biparti. Attention, un graphe biparti n'est pas forcément un arbre.

### 2.2.2 Forêts

Une forêt est un graphe dont chaque composante connexe est un arbre éventuellement réduit à un sommet unique.

La forêt est obtenue en relaxant (en laissant tomber) la contrainte de connexité. En d'autres termes, une forêt est un graphe sans cycle. D'ailleurs, tout graphe partiel d'un arbre est une forêt. Dans l'arbre de la figure 2.1, on peut enlever l'arête  $(B, D)$ , on obtient la forêt de la figure 2.2 avec les composantes connexes :  $\{A, B, C, E\}$  et  $\{D\}$ .

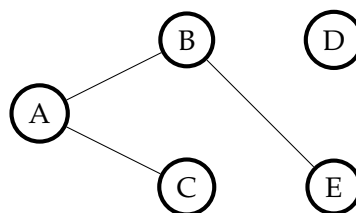


Figure 2.2 : exemple d'une forêt

### 2.2.3 Racine et anti-racine

Dans un graphe orienté, on peut définir un sommet particulier, appelé *racine*, comme étant un

sommet permettant de rejoindre n'importe quel autre sommet. En d'autres termes, si  $x$  est une racine, alors il existe un chemin joignant entre  $x$  et n'importe quel autre sommet du graphe.

Une *anti-racine* est le concept opposé de la racine. Un sommet  $x$  est dit anti-racine s'il existe un chemin joignant tout sommet du graphe à  $x$ .

Dans le graphe orienté de la figure 2.3, le sommet A est une racine, alors que le sommet C est une anti-racine.

Pour un graphe ne contenant pas de circuit, si on applique l'algorithme de classement (vu au premier chapitre), alors il est clair que la racine sera classée au premier rang, alors que l'anti-racine sera classée au dernier rang.

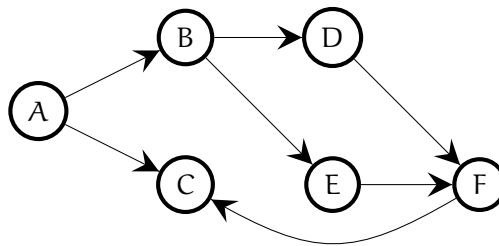


Figure 2.3 : exemple de racine et anti-racine

## 2.2.4 Arborescence

Une *arborescence* est un graphe orienté  $G$  ayant les propriétés suivantes :

1. Si on transforme les arcs de  $G$  en arêtes, alors le graphe orienté obtenu est un arbre (en d'autres termes,  $G$  est connexe et sans cycle);
2.  $G$  possède une racine  $S$ ;
3. Pour tout sommet  $x$  différent de  $S$ , alors  $d^-(x) = 1$  (il a un seul prédécesseur).
4. La troisième propriété peut également être exprimée comme suit : il existe un seul chemin entre la racine de l'arbre et tout sommet d'une arborescence.

Le graphe orienté de la figure 2.4 est une arborescence, sa racine est le sommet A. Si on inverse le sens des arcs d'une arborescence, on obtient une anti-arborescence.

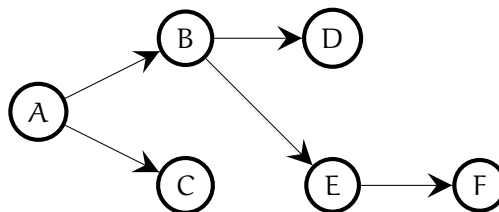


Figure 2.4 : exemple d'une arborescence

Les arborescences ont beaucoup d'applications, parmi lesquelles on peut citer :



L’algorithme proposé par Kruskal (prononcez KRASKAL) permet de construire l’arbre de couverture de poids minimal pour un graphe valué quelconque.

**Algorithme de Kruskal pour construire l’arbre de couverture minimal d’un graphe G**

```
1 n = ordre(G)
2 m = taille(G)
3 trier les arêtes de G dans l’ordre croissant des poids : a1, a2, ..., am
4 F = ∅, k = 1
5 tant que k ≤ m et card(F) < n - 1 faire
6   si l’ajout de ak ne crée pas de cycle alors
7     F = F ∪ {ak}
8   k = k + 1
```

On applique l’algorithme de Kruskal au graphe de la figure 2.6. On commence alors par trier les poids :

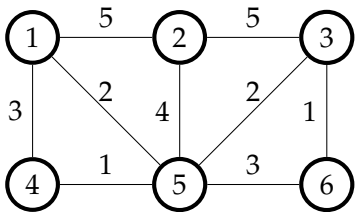


Figure 2.6 : exemple d’un graphe valué

Arête	(4,5)	(3,6)	(1,5)	(3,5)	(1,4)	(5,6)	(2,5)	(1,2)	(2,3)
Poids	1	1	2	2	3	3	4	5	5
Indice	1	2	3	4	5	6	7	8	9

Le déroulement se fait alors selon les valeurs k :

k	Décision	Graphe
1	Rajouter l’arête (4,5)	
2	Rajouter l’arête (3,6)	

3	Rajouter l'arête (1,5)	
4	Rajouter l'arête (3,5)	
5	Ne pas rajouter l'arête (1,4)	Même graphe
6	Ne pas rajouter l'arête (5,6)	Même graphe
7	Rajouter l'arête (2,5) (fin)	

L'arbre est donné par le dernier graphe du tableau de déroulement. Son poids minimal est de 10.

La construction de l'arbre de couverture de poids minimal a plusieurs applications. Parmi lesquelles on peut donner la suivante : dans un système de communication par message, si un nœud souhaite diffuser un message, il peut construire un arbre de couverture de poids minimal pour faire parvenir le message à tous les autres nœuds avec un coût minimal.

### 2.3.3 Application de l'algorithme de Kruskal sur un graphe non-connexe

L'algorithme de Kruskal s'applique aussi si le graphe n'est pas connexe. Cependant, il produit une forêt et non pas un arbre. Prenons l'exemple du graphe de la figure 2.7 qui comporte deux composantes connexes. Nous devons, cependant, changer la condition d'arrêt de l'algorithme précédent car, dans une forêt, le nombre d'arêtes dépend du nombre de composantes connexes.

On trie les arêtes selon l'ordre croissant de leurs poids :

Arête	(3,6)	(4,5)	(3,7)	(1,5)	(6,7)	(1,4)	(4,8)	(2,7)	(1,8)	(2,3)
Poids	1	1	2	2	3	3	4	5	5	6
Indice	1	2	3	4	5	6	7	8	9	10

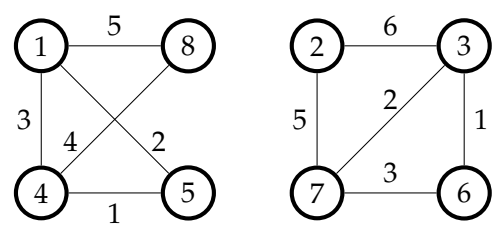
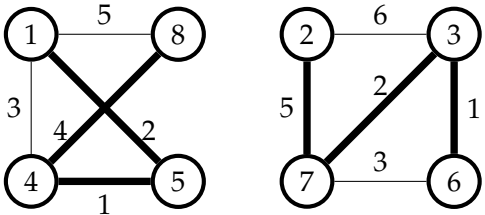


Figure 2.7 : exemple d'un graphe valué non-connexe

Le déroulement se fait comme suit :

k	Décision	Graphe	
1	Rajouter l'arête (4,5)		
2	Rajouter l'arête (3,6)		
3	Rajouter l'arête (3,7)		
4	Rajouter l'arc (1,5)		
5	Ne pas rajouter l'arête (6,7)		
6	Ne pas rajouter l'arête (1,4)		
7	Rajouter l'arête (4,8)		

8	Rajouter l'arête (2,7)	
9	Ne pas rajouter l'arête (1,8)	
10	Ne pas rajouter l'arête (2,3)	

La forêt de couverture est celle que l'on trouve à l'itération où  $k = 8$ . Il existe plusieurs autres algorithmes pour construire l'arbre de couverture minimal (comme l'algorithme de Sollin), mais ils ne seront pas couverts par ce cours.