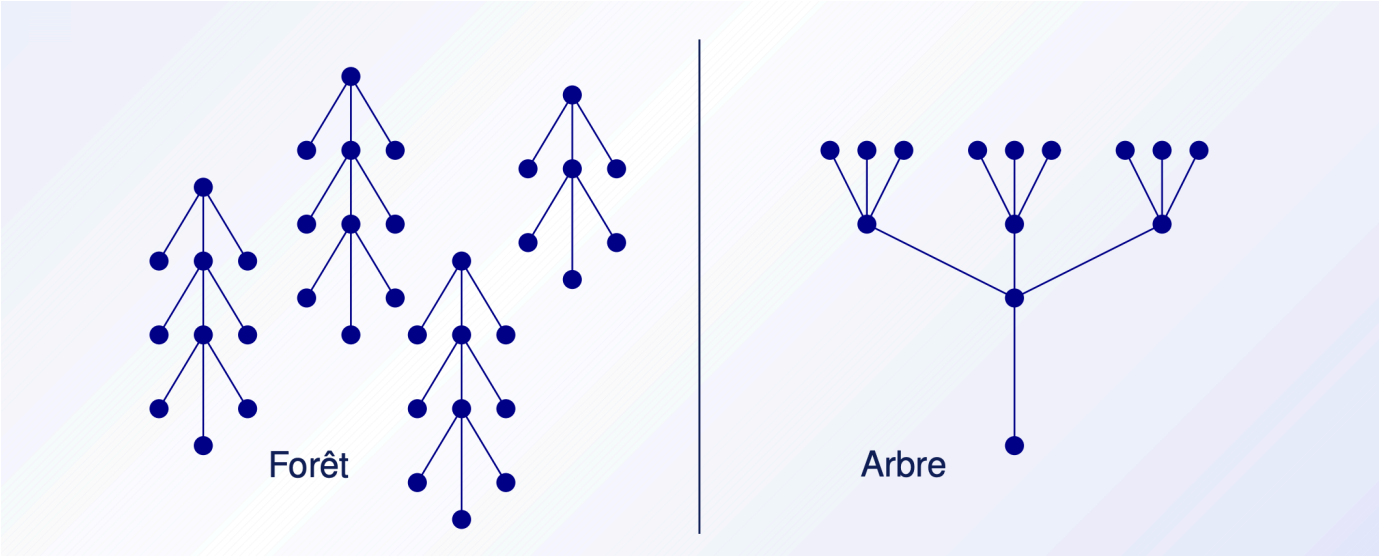
Arbres et Forets

GRE
4 - Arbres et Forets

Abstract
Definition

Table des matières

1. Définition



1.1. Propriétés

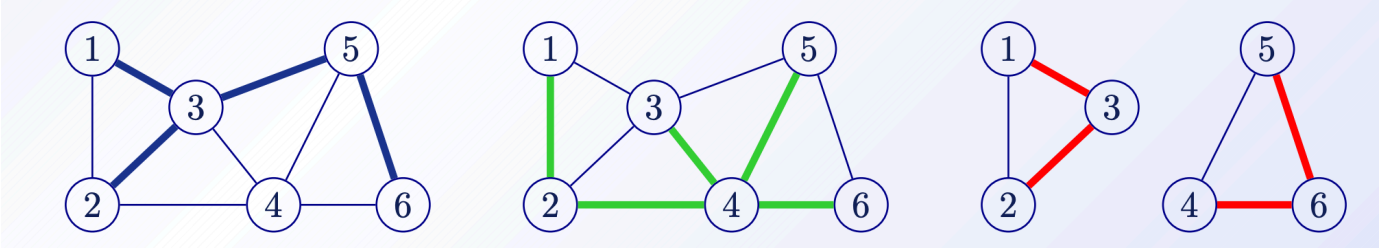
Les propriétés des arbres et forêts sont les suivantes :

- Un arbre est une forêt connexe.
- Chaque composante connexe d'une forêt est un arbre.
- Forêts et arbres sont des graphes simples ! En effet, toute boucle ou toute paire d'arêtes parallèles crée un cycle simple.
- Dans un arbre (ou une forêt) les sommets pendants (sommet de degré 1) sont souvent appelés des feuilles.
- Tout arbre comptant au moins 2 sommets possède au moins 2 feuilles.

2. Arbres et forêts

2.1. Recouvrant

Un arbre recouvrant d'un graphe non orienté connexe est un arbre qui contient tous les sommets du graphe initial. Cela est pareil pour une forêt recouvrante.



- En bleu: un arbre non recouvrant
- En vert: un arbre recouvrant
- En rouge: une forêt recouvrante

2.2. Arbre recouvrant minimal

Un arbre recouvrant minimal d'un graphe non orienté connexe est un arbre recouvrant de poids minimal. L'objectif est de trouver un arbe permettant de relier tous les sommets du graphe avec le plus petit poids possible.

2.3. Algorithme de Kruskal (1956)

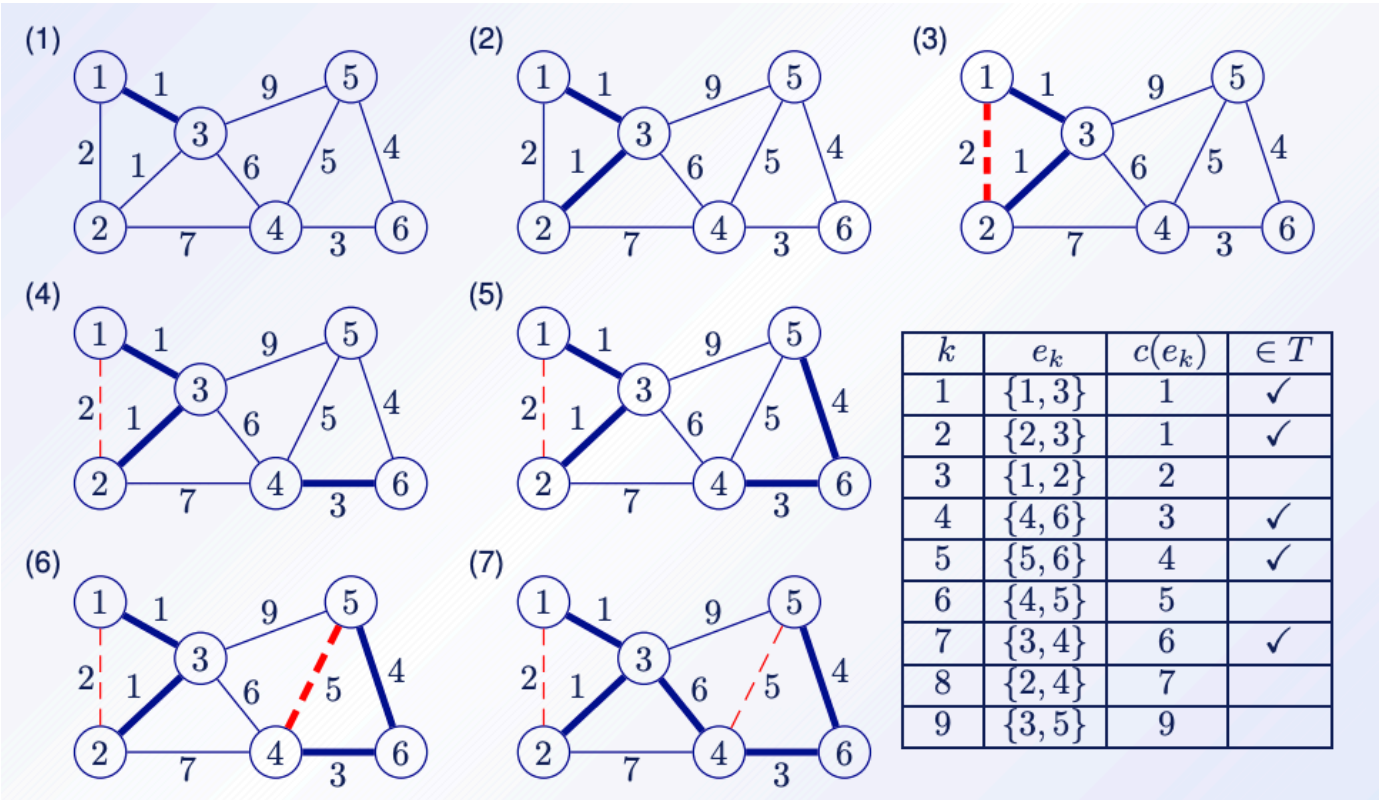
L'algorithme de Kruskal consisite à partir d'une forêt vide ne contenant que les sommets du graphe puis diminuer progressivement le nombre de composantes connexes en reliant à chaque fois de la façon la plus économique possible deux composantes de la forêt actuelle.

2.3.1. Application

Pour appliquer cet algorithme, il faut:

- construire un tableau contenant toutes les arêtes du graphe avec leurs poids respectifs
- trier ce tableau par ordre croissant de poids
- sélectionner les plus petites arêtes du tableau pour relier tous les sommets du graphe

2.3.2. Exemple



k	e_k	$c(e_k)$	$\in T$
1	$\{1,3\}$	1	✓
2	$\{2,3\}$	1	✓
3	$\{1,2\}$	2	
4	$\{4,6\}$	3	✓
5	$\{5,6\}$	4	✓
6	$\{4,5\}$	5	
7	$\{3,4\}$	6	✓
8	$\{2,4\}$	7	
9	$\{3,5\}$	9	

2.3.3. Complexité

$$O(m \log n + mn) == (mn)$$

2.4. Algorithme de Prim (1957)

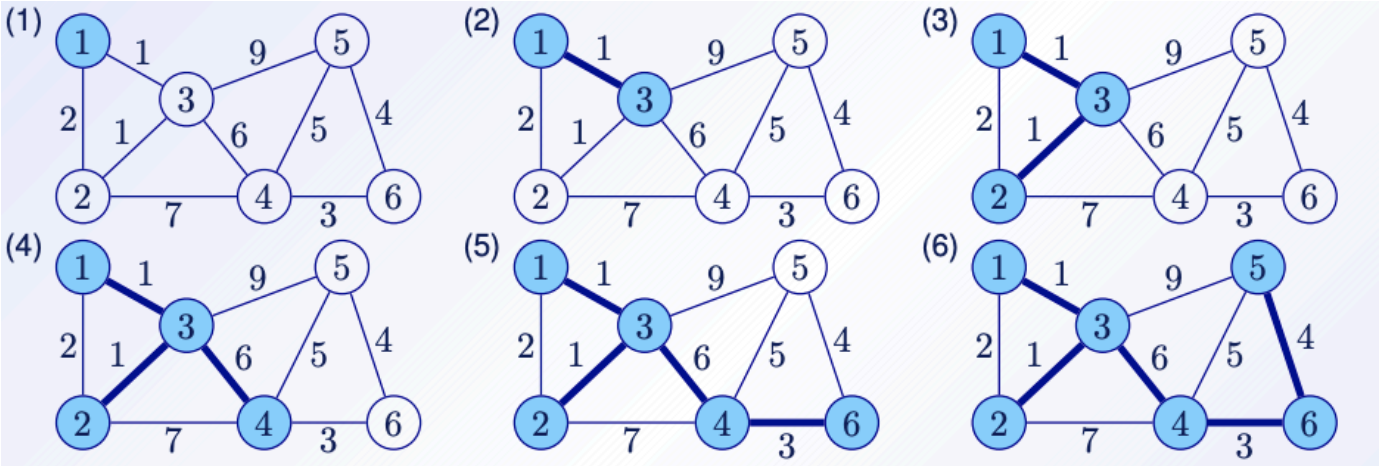
L'algorithme de Prim consiste à partir d'un arbre vide ne contenant qu'un sommet du graphe puis ajouter progressivement des sommets au graphe en choisissant à chaque fois le sommet le plus proche de l'arbre actuel.

2.4.1. Application

Pour appliquer cet algorithme, il faut:

- construire un tableau ayant comme colonne
  - le n°d'itération
  - le sommet en cours de traitement
  - tous les sommets du graphe
- on définit des couples (distance, sommet) en commançant par l'itération 0
- on choisit le sommet le plus proche de l'arbre actuel puis mets à jour le tableau

2.4.2. Exemple



Itér.	Sommet retiré de L	Couples $(\lambda[u], p[u])$ pour les sommets encore dans L, en fin d'itér.					
		1	2	3	4	5	6
0		**(0,—)**	(∞,—)	(∞,—)	(∞,—)	(∞,—)	(∞,—)
1	1		(2,1)	**(1,1)**	(∞,—)	(∞,—)	(∞,—)
2	3		**(1,3)**		(6,3)	(9,3)	(∞,—)
3	2				**(6,3)**	(9,3)	(∞,—)
4	4					(5,4)	**(3,4)**
5	6					**(4,6)**	
6	5						

2.4.3. Complexité

En fonction de la structure de données utilisée pour représenter le graphe, la complexité de l'algorithme de Prim est:

	Tableau	**Tas binaire**	**Tas de Fibonacci**
Insert	$O(1)$	$O(\log n)$	$O(1)$
FindMin	$O(n)$	$O(1)$	$O(1)$
ExrtactMin	$O(n)^{[a]}$	$O(\log n)$	$O(\log n)^{[b]}$
DecreaseKey	$O(1)$	$O(\log n)$	$O(1)^{[b]}$
Prim	$O(n^2)$	$O(m \log n)$	$O(m + n \log n)$

Note
Les algorithmes de Kruskal, de Prim et de Boruvka permettent également de déterminer des arbres recouvrants de poids total **maximum**.