



# Chapitre 1

## Arbres et Arborescences

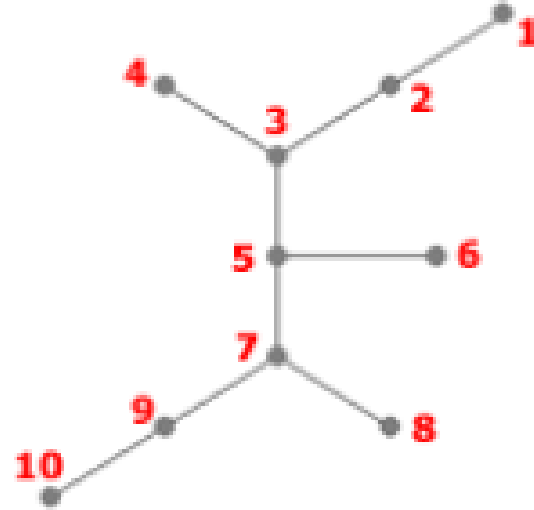


# Plan

- Définitions
- Codage de Prüfer
- Algorithme union-find ou gestion des partitions
- Arbres couvrants
- Application : Méthode de compression statistique de données (Codage de Huffman)

# Définitions

Un **arbre** est un graphe connexe, sans cycles.



Le théorème fondamental suivant donne les caractérisations alternatives des arbres :

## Théorème

Soit  $G$  un graphe non orienté à  $n$  sommets. Les propositions suivantes sont équivalentes :

1.  $G$  est connexe et sans cycle.
2.  $G$  est connexe et a  $n - 1$  arêtes.
3.  $G$  est connexe et la suppression de n'importe quelle arête le déconnecte.
4.  $G$  est sans cycle et a  $n - 1$  arêtes.
5.  $G$  est sans cycle et l'ajout de n'importe quelle arête crée un cycle.
6. Entre toute paire de sommets de  $G$  il existe une unique chaîne élémentaire.



# Codage de Prüfer

# Codage de Prüfer

- Le **codage de Prüfer** est une manière très compacte de décrire un arbre.
- Pour un arbre de  $n$  nœuds, ce code correspond à une suite  $S$  de  **$n-2$  termes** employant (éventuellement plusieurs fois) des nombres choisis parmi  $\{1, \dots, n\}$ .

## Algorithme codage

**Données:**  $T = (V, E)$  t.q  $V = \{1, 2, \dots, n\}$  (cet étiquetage est fait aléatoirement).

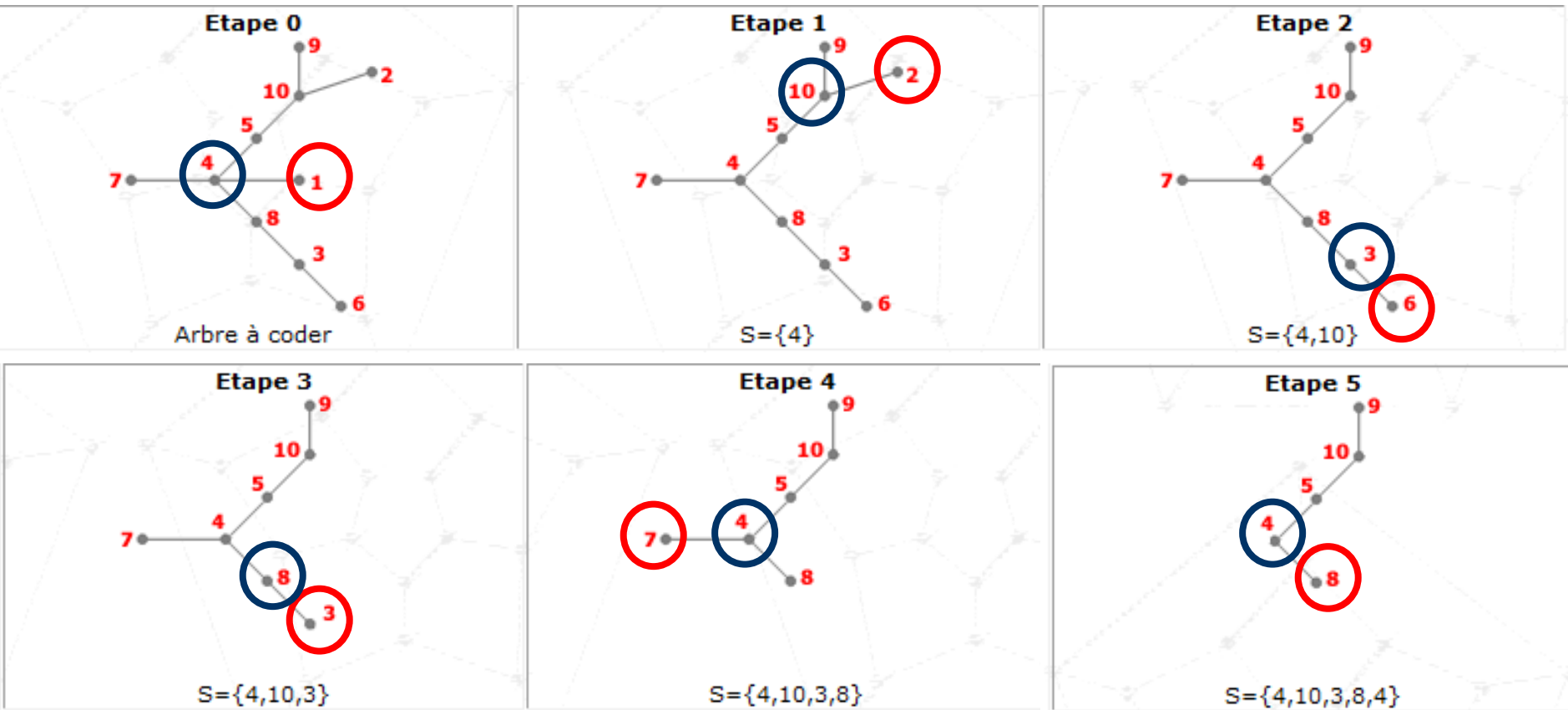
**Résultat:**  $S$ : un codage de Prüfer de  $T$

$S \leftarrow \emptyset$

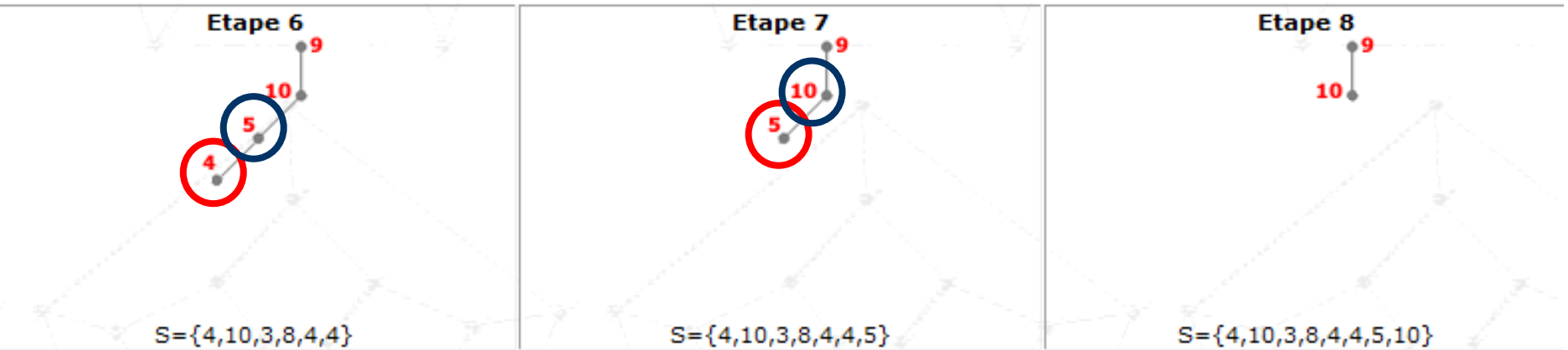
Tant qu'il reste plus de deux sommets dans l'arbre courant  $T$

- Identifier la feuille  $v$  de l'arbre courant ayant le numéro minimum ;
- Ajouter à la suite  $S$  le seul sommet  $s$  adjacent à  $v$  dans l'arbre  $T$  courant ;
- Enlever de l'arbre  $T$  courant le sommet  $v$  et l'arête incidente à  $v$ .

# Exemple Codage de Prüfer (1)



# Exemple Codage de Prüfer (2)



$S = \{4, 10, 3, 8, 4, 4, 5, 10\}$  est le codage de Prüfer de l'arbre initial.



# Décodage

## Algorithme Décodage

**Données:**  $S$  = un codage de Prüfer de  $T$ ,

**Résultat:**  $T = (V, E)$  correspondant à  $S$

$I \leftarrow \{1, \dots, n\}$

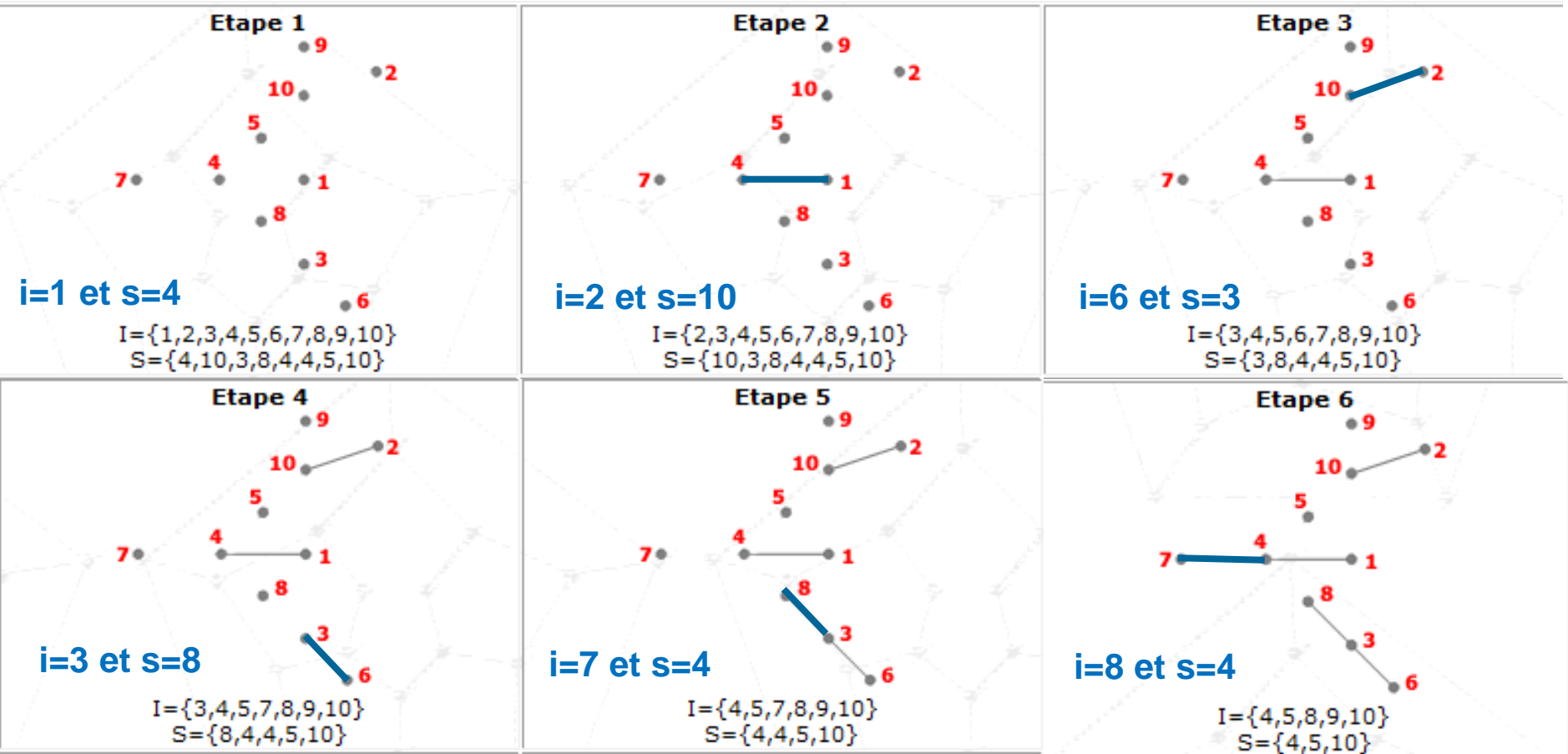
Tant qu'il reste des éléments dans  $S$  et plus de deux éléments dans  $I$

1. Identifier le plus petit élément  $i$  de  $I$  n'apparaissant pas dans la suite  $S$
2. Relier par une arête de  $T$  le sommet  $i$  avec le sommet  $s$  correspondant au premier élément de la suite  $S$
3. Enlever  $i$  de  $I$  et  $s$  de  $S$

Les deux éléments qui restent (car  $S$  contient  $n-2$  termes alors que  $I$  contient  $n$  termes) dans  $I$  à la fin de l'algorithme constituent les extrémités de la dernière arête à ajouter à  $T$

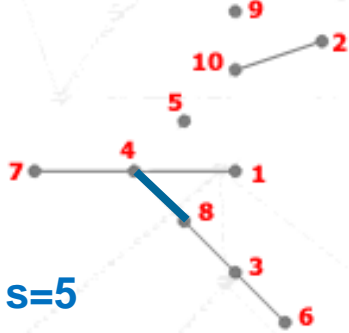


# Exemple de décodage (1)



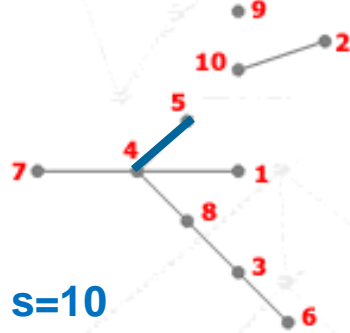
# Exemple de décodage (2)

Etape 7



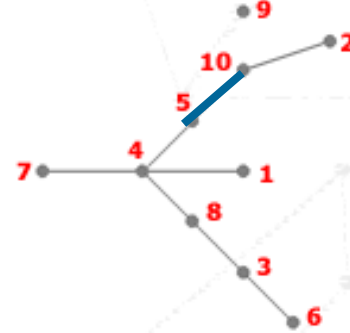
$I=\{4,5,9,10\}$   
 $S=\{5,10\}$

Etape 8



$I=\{5,9,10\}$   
 $S=\{10\}$

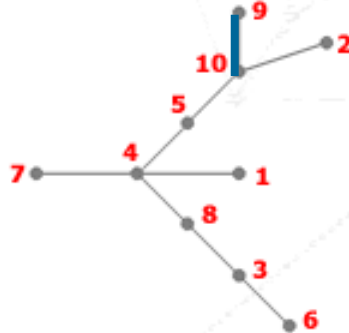
Etape 9



$I=\{9,10\}$   
 $S=\{\}$

Dernière arête 9-10

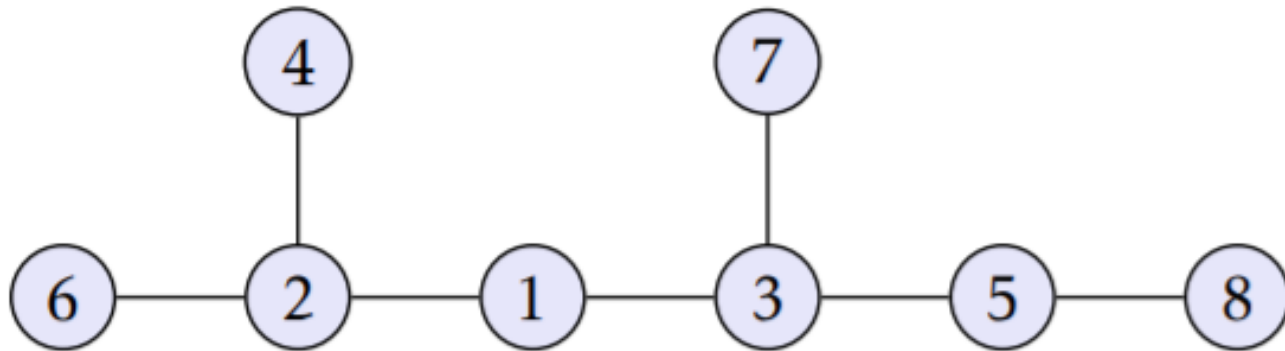
Etape 10

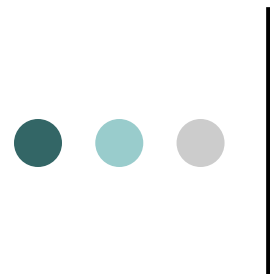


$I=\{\}$   
 $S=\{\}$

# Applications

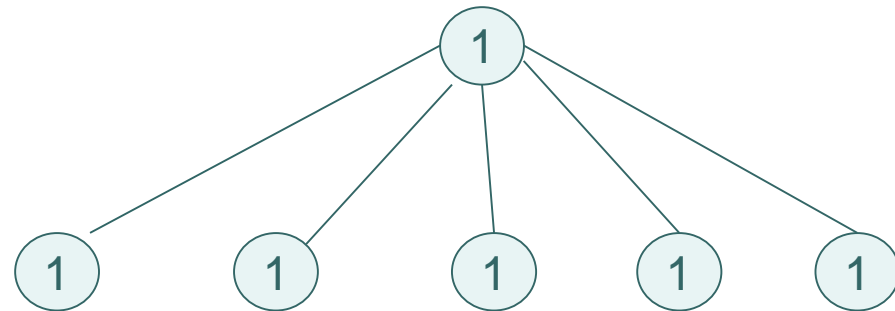
1. Donner le codage et le décodage de Prüfer de l'arbre suivant





- Solution:  $S=\{2,2,1,3,3,5\}$

2. Donner la suite de codage de l'arbre suivant:





# **Algorithme union-find ou gestion des partitions**



# Algorithme union-find (1)

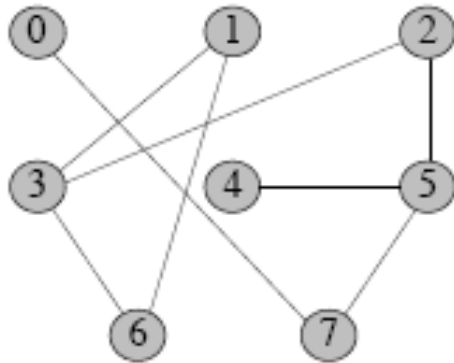
- Comme premier exemple de l'emploi des arbres et des forêts, nous considérons un problème célèbre, et à ce jour pas encore entièrement résolu, appelé le problème *Union-Find*.
- Rappelons qu'une *partition* d'un ensemble  $E$  est un ensemble de parties non vides de  $E$  disjointes et dont la réunion est  $E$ .
- Étant donné une partition de l'ensemble  $\{0, \dots, n-1\}$ , on veut résoudre les deux problèmes :
  - trouver la classe d'un élément (*find*)
  - faire l'union de deux classes (*union*).
- Plusieurs applications
  - Vérification des composantes d'un graphe dynamique
  - Est ce qu'un composant  $p$  communique avec un composant  $q$ ?

# Algorithme union-find (2)

Deux points d'un réseau sont-ils connectés ?

Les points (1, 3), (2, 3), (5, 4), (6, 3), (7, 5), (1, 6) et (7, 0) sont connectés.

Est-ce que 4 et 6 sont connectés ?



Union-Find résout ce problème efficacement !

→ Il existe plusieurs implémentations possibles avec des complexités différentes

# Première solution : tableau

- Données : une partition de l'ensemble  $\{0, 1, \dots, n-1\}$
- Trouver la classe d'un élément (find).
- Faire l'union de deux classes (union).

## *Première solution*

- On représente la partition par un tableau classe tel que `classe[i]` soit la classe de l'élément  $i$ .
- Exemple:
- $P = \{\{2, 5, 8\}, \{0, 6\}, \{1\}, \{3, 4, 7, 9\}\}$   

<i>classe</i>	0	1	2	3
---------------	---	---	---	---

	0	1	2	3	4	5	6	7	8	9
classe	1	2	0	3	3	0	1	3	0	3



# Deuxième solution : forêts

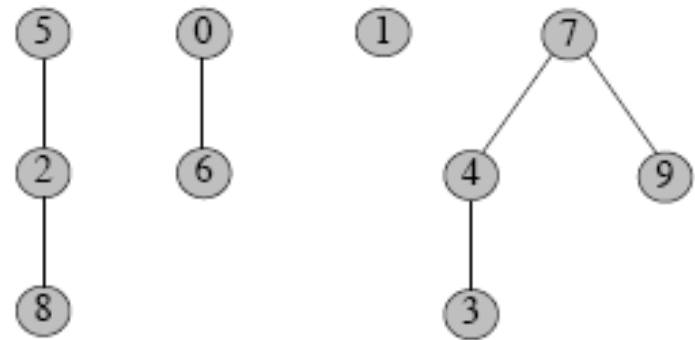
union rapide

On représente la partition par une forêt.

- Chaque arbre correspond à une classe d'équivalence.
- La racine de chaque arbre donne un représentant canonique (unique) de la classe.

On représente la forêt par un tableau **pere** tel que **pere[s]** est le **père** de l'élément **s** (si **s** est une racine, **pere[s] = s**)

$$P = \{\{2, 5, 8\}, \{0, 6\}, \{1\}, \{3, 4, 7, 9\}\}$$



	0	1	2	3	4	5	6	7	8	9
pere	0	1	5	4	7	5	0	7	2	7

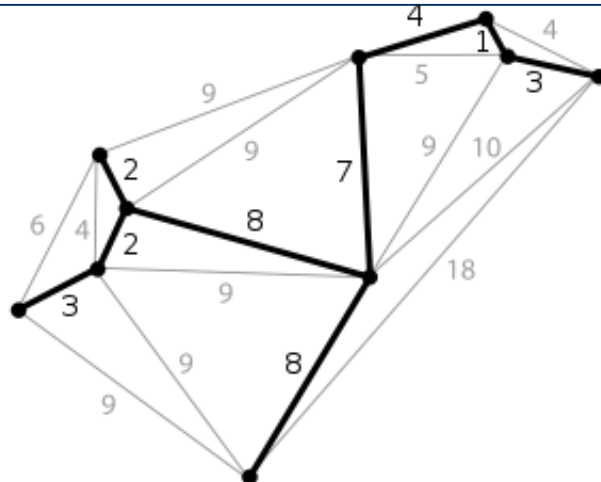


# Arbres couvrants

# Arbre couvrant minimal (1)

Soit  $G$  un graphe valué non orienté connexe.

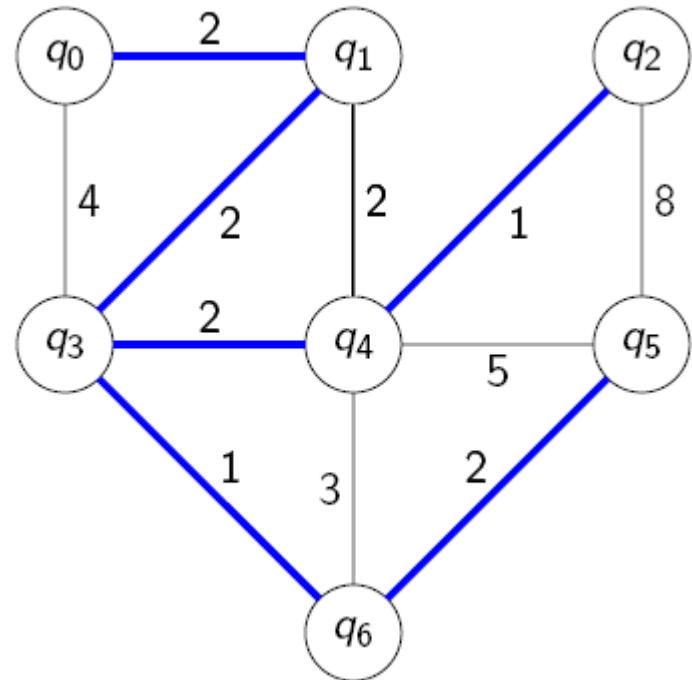
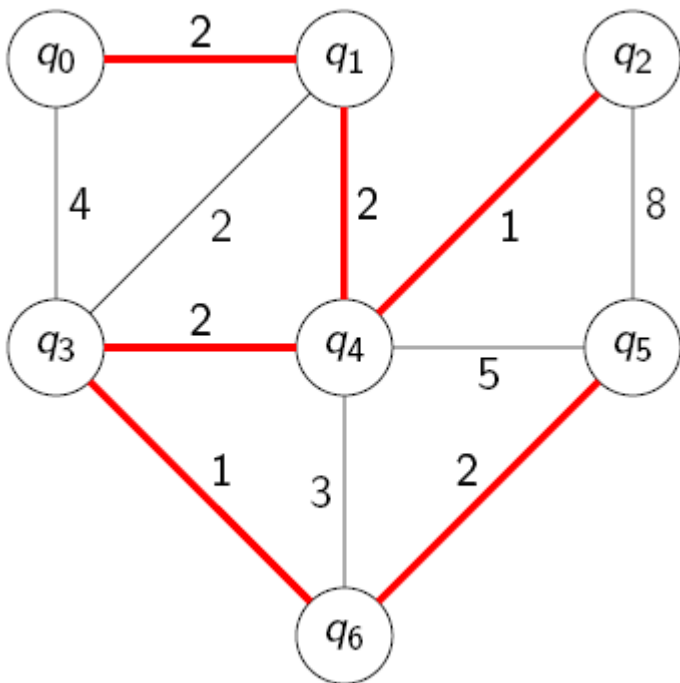
- Un **arbre couvrant** est un sous-graphe de  $G$ , **contenant tous les sommets, connexe et sans cycle**. Son poids est la somme des valuations de ses arêtes.
- Un **arbre couvrant de poids minimum** (ACM) (en anglais minimum spanning tree (MST)) est un arbre couvrant dont le poids est le plus petit possible parmi les arbres couvrants de  $G$ . Si toutes les arêtes ont des valuations positives, son poids est le plus petit possible parmi tous les sous-graphes connexes couvrants de  $G$ .



## Arbre couvrant minimal (2)

### Propriété :

Tout graphe non-orienté, valué et connexe admet un ou plusieurs ACM.





# Arbres couvrants (1)

- **Application 1:** Ce problème se pose, par exemple, lorsqu'on désire relier  $n$  villes par un réseau routier de coût minimum. Les sommets du graphe représentent les villes, les arêtes, les tronçons qu'il est possible de construire et les poids des arêtes correspondent aux coûts de construction du tronçon correspondant.
- **Application 2:** Imaginons que chaque nœud soit un ordinateur et que le poids de l'arête entre  $x$  et  $y$  indique la longueur de câble nécessaire pour relier les ordinateurs  $x$  et  $y$ . Alors, l'ACM donne la plus courte longueur de câble nécessaire pour relier tous les ordinateurs.

## Arbres couvrants (2)



Otakar Boruvka  
1899 -1995

*C'est un problème qui est plus vieux que l'informatique elle même. En effet en 1926, Otakar Boruvka fut un des premiers (si ce n'est le premier) à fournir un algorithme pour trouver « efficacement » un arbre couvrant de poids minimum. La motivation d'alors était de concevoir un réseau de distribution électrique pour la Moravie du Sud (République tchèque) avec la contrainte que cela coûte le moins possible.*

- Ce n'est qu'au milieu des années 50 et l'avènement de l'informatique que ce problème est revenu à la mode. C'est à Joseph B. **Kruskal** (1956) et à Robert **Prim** (1957) que nous devons les algorithmes que nous allons étudier (ce sont les plus connus).
- Les deux algorithmes (Kruskal et Prim) sont des algorithmes gloutons.



# Arbre maximal de poids minimum: Algorithme de Kruskal

**Principe** : L'algorithme consiste à d'abord ranger par ordre de poids croissant les arêtes d'un graphe, puis à retirer une à une les arêtes selon cet ordre et à les ajouter à l'ACM cherché tant que cet ajout ne fait pas apparaître un cycle dans l'ACM.

>> L'algorithme s'arrête quand on aura incorporé  $n-1$  arêtes.

# Algorithme de Kruskal

- Afin d'optimiser l'implémentation, on utilise des fonctions **Trouver** qui retourne un élément représentatif d'un ensemble et **union** qui combine les arbres obtenus au fur et à mesure.

## Algorithme Recherche-ACM-Kruskal

Données: Graphe  $G(S, A)$

Résultat: ACM  $A'$

$A' \leftarrow \emptyset$

Pour chaque  $s \in S$  faire

**CréerEnsemble(s)**

*%Chaque sommet est placé dans un ensemble (une partition)*

Fin pour

**Trier**  $A$  par poids croissant

Pour chaque  $(x,y) \in A$  faire *% les arêtes sont choisies de  $A$  trié*

Si **Trouver(x)  $\neq$  Trouver(y)**

$A' \leftarrow A' \cup \{(x,y)\}$

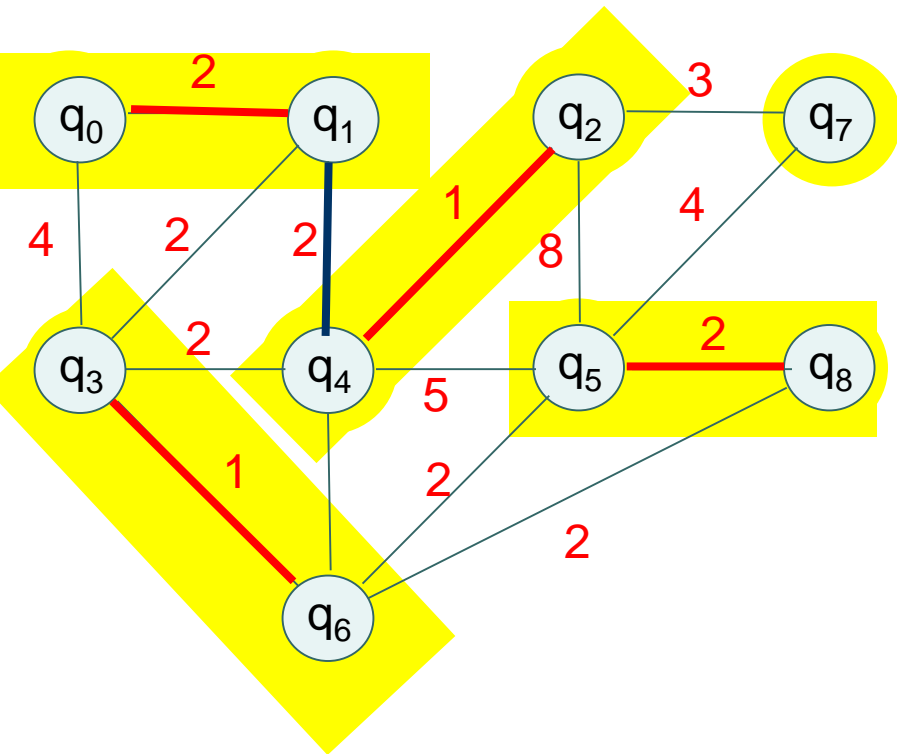
**Union(x,y)**

Fin Si

Fin pour



# Exemple (1)



- Au départ chaque nœud appartient à un ensemble cad on a 8 ensembles

-Trier A:

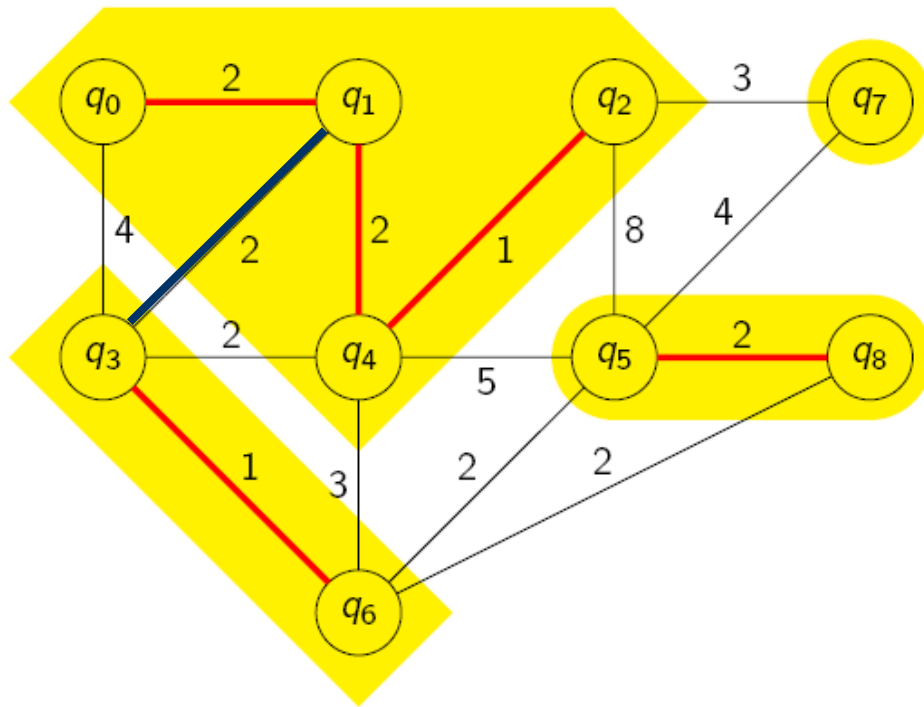
$A = [q_3 - q_6, q_4 - q_2,$   
 $q_0 - q_1, q_5 - q_8, q_1 - q_4, q_1 - q_3, q_6 - q_8, q_5 - q_6, q_3 - q_4,$   
 $q_2 - q_7$   
 $q_5 - q_7, q_0 - q_3,$   
 $q_4 - q_5$   
 $q_2 - q_5]$

- Trouver( $q_3$ )  $\neq$  Trouver( $q_6$ )  $\rightarrow$  Union  $q_3$  et  $q_6$
- Trouver( $q_4$ )  $\neq$  Trouver( $q_2$ )  $\rightarrow$  Union  $q_4$  et  $q_2$
- Trouver( $q_0$ )  $\neq$  Trouver( $q_1$ )  $\rightarrow$  Union  $q_0$  et  $q_1$
- Trouver( $q_5$ )  $\neq$  Trouver( $q_8$ )  $\rightarrow$  Union  $q_5$  et  $q_8$

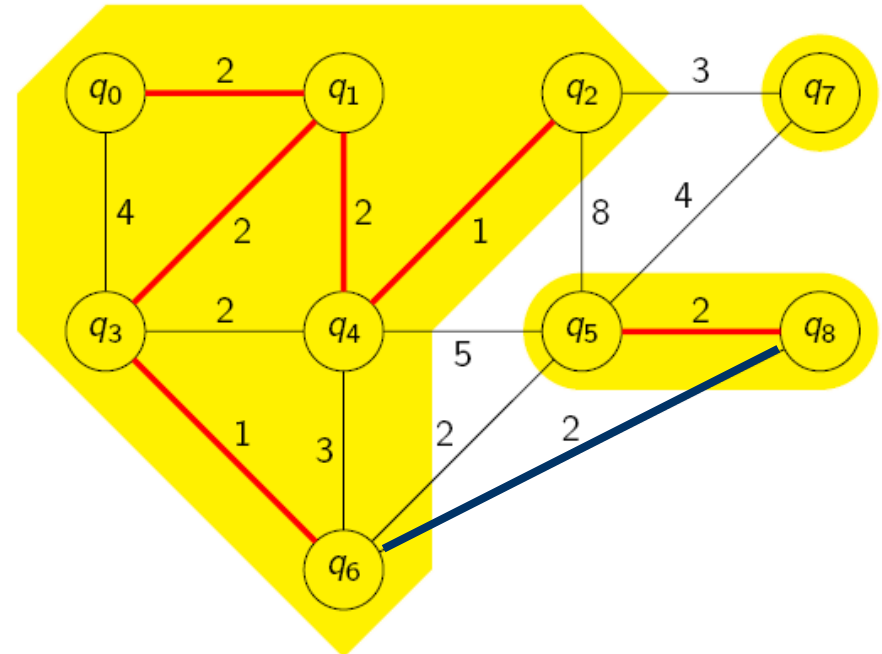
A ce stade on a cinq partitions

- Trouver( $q_1$ )  $\neq$  Trouver( $q_4$ )  $\rightarrow$  Union  $q_0, q_1$  et  $q_4, q_2$

## Exemple (2)

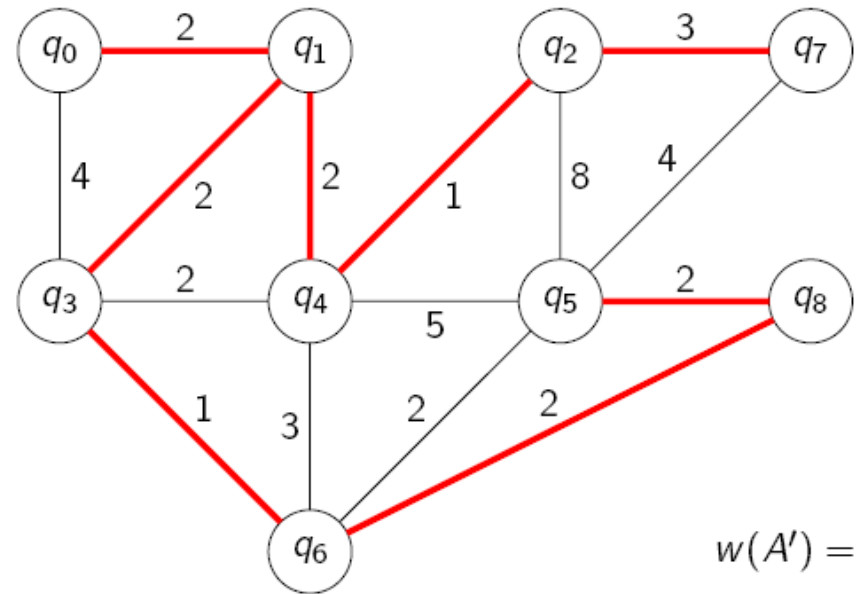
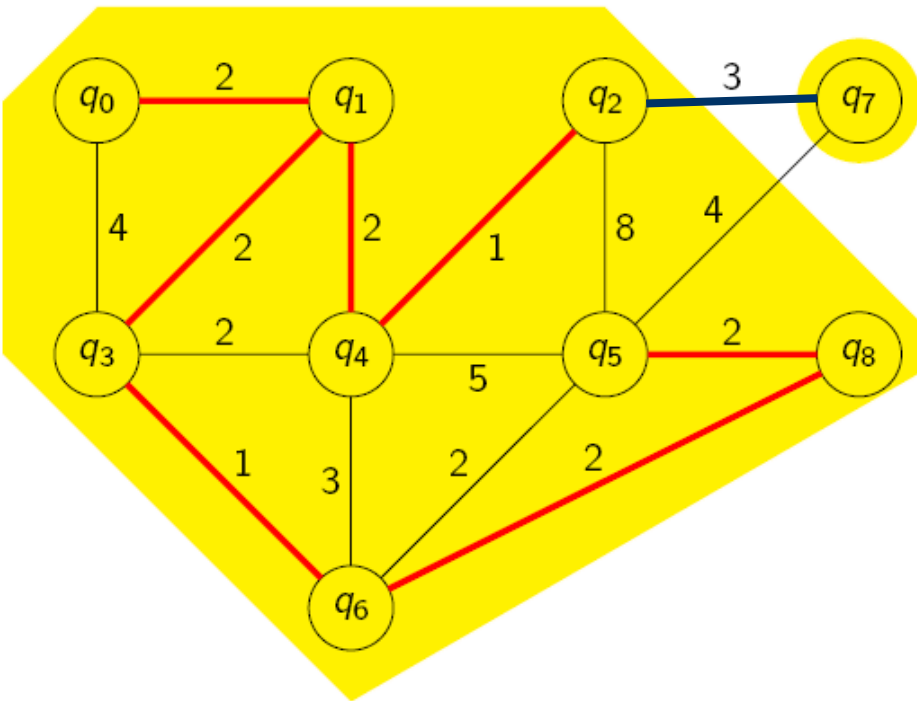


Trouver( $q_1$ )  $\neq$  Trouver( $q_3$ )  
 $\rightarrow$  Union  $q_0, q_1, q_4, q_2$  et  $q_3, q_6$



Trouver( $q_6$ )  $\neq$  Trouver( $q_8$ )  
 $\rightarrow$  Union  $q_0, q_1, q_4, q_2, q_3, q_6$  et  $q_5, q_8$

# Exemple (3)



$$w(A') = 15$$

- Trouver( $q_5$ ) = Trouver( $q_6$ )
  - Trouver( $q_3$ ) = Trouver( $q_4$ )
  - Trouver( $q_2$ )  $\neq$  Trouver( $q_7$ )
- Union  $q_0, q_1, q_4, q_2, q_3, q_6, q_5, q_8$  et  $q_7$

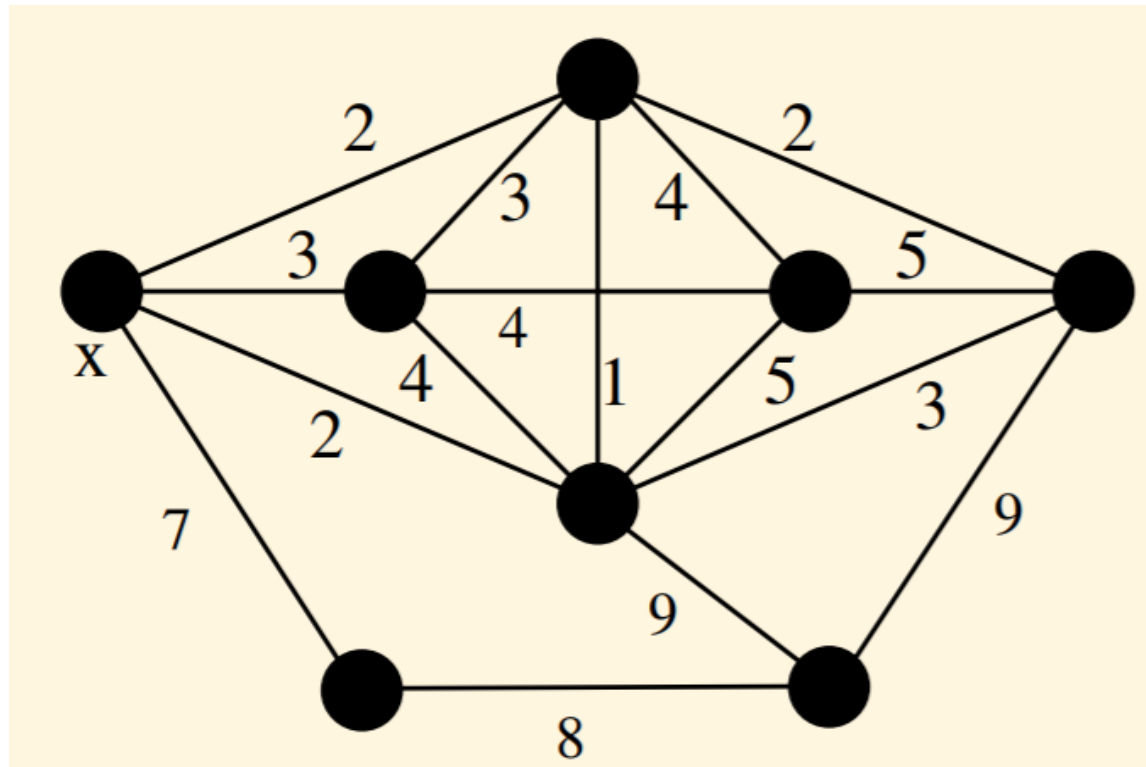


# Remarque

- La recherche d'un ACM peut être utilisée pour approximer le problème du voyageur de commerce.
- C-à-d approximer le circuit hamiltonien minimal.

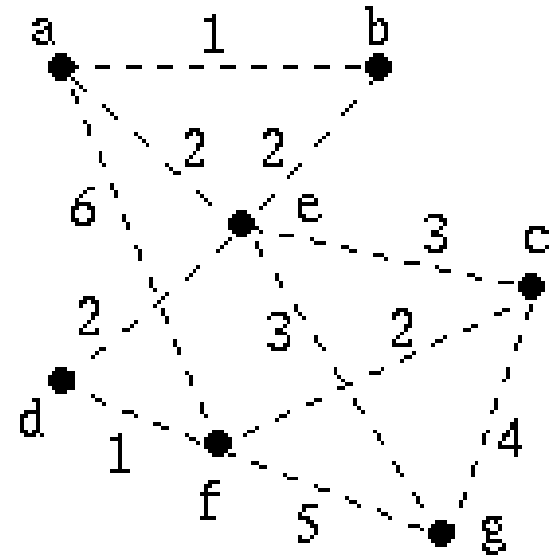
# Exercice

Appliquer l'algorithme de Kruskal pour obtenir le ACM:

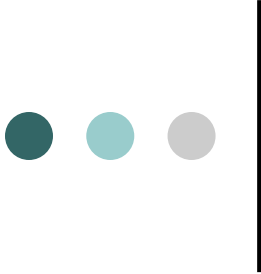


# Travail à rendre (19/11)

1. Trouver l'ACM du graphe suivant:



1. Implémenter l'algorithme de Kruskal (en c/java) et exécuter le afin de trouver l'ACM relatif à la carte de la Tunisie.
2. Etudier l'algorithme de Prim et déceler les différences avec l'algorithme de Kruskal.

- 
- Le rendu doit comprendre:
    - L'exécution des algorithmes Kruskal et Prim. Préciser la différence entre les deux algorithmes.
    - Un CD comportant le programme de l'algorithme de Kruskal



# **Application : Méthode de compression statistique de données (Codage de Huffman)**





# Codage de Huffman (1)

- Le **codage de Huffman** est une méthode de **compression statistique de données** qui permet de réduire la longueur du codage d'un alphabet.
- Le code de Huffman (1952) est un code de longueur variable optimal, c'est-à-dire tel que la longueur moyenne d'un texte codé soit minimale. On observe ainsi des réductions de taille de l'ordre de 20 à 90%.
- Le principe de l'algorithme de Huffman consiste à recoder les octets rencontrés dans un ensemble de données source avec des valeurs de longueur binaire variable.
- Rappelons que le code ASCII étendu code les octets avec des valeurs de longueur binaire constante= 8 ("a" est associé à "01100001" et "A" à "01000001"... )



# Codage de Huffman (2)

- Huffman propose de:
  - recoder les données qui ont une occurrence très faible sur une longueur binaire supérieure à la moyenne, et
  - recoder les données très fréquentes sur une longueur binaire très courte.
- Pour les données rares, nous perdons quelques bits regagnés pour les données répétitives.
- L'algorithme est de type *glouton* : il choisit à chaque étape les deux arbres d'étiquettes minimales, soit **a** et **b**, et les remplace par l'arbre formé de a et b et ayant comme étiquette la somme de l'étiquette de a et de l'étiquette de b.



# Codage de Huffman : principe

**Données:** Alphabet : qui est l'alphabet de symboles de taille  $n$

**Résultat:** Code

## Initialisation

1. Classer les symboles à compresser par ordre décroissant de fréquence d'apparition dans l'alphabet
2. Pour chaque symbole, former un nœud  $X$  tel que  $POIDS(X) :=$  probabilité de son apparition (fréquence /  $n$ )

## Construction de l'arbre

Tant que tous les nœuds ne sont pas reliés

3. Soit  $X$  et  $Y$  deux nœuds ayant les poids les plus faibles (en résolvant les égalités de manière arbitraire)
4. Relier  $X$  et  $Y$  en créant un nœud  $Z$  tel que  $POIDS(Z) := POIDS(X) + POIDS(Y)$
5. Affecter à l'une des branches  $Z-X$  et  $Z-Y$  le label '0' et à l'autre le label '1' (l'ordre n'a pas d'importance)

Fin Tant que

## Détermination des codes

6. Pour chaque symbole de l'alphabet initial, le code de Huffman est constitué par la suite des bits portés par les branches entre la racine de l'arbre et l'octet en question

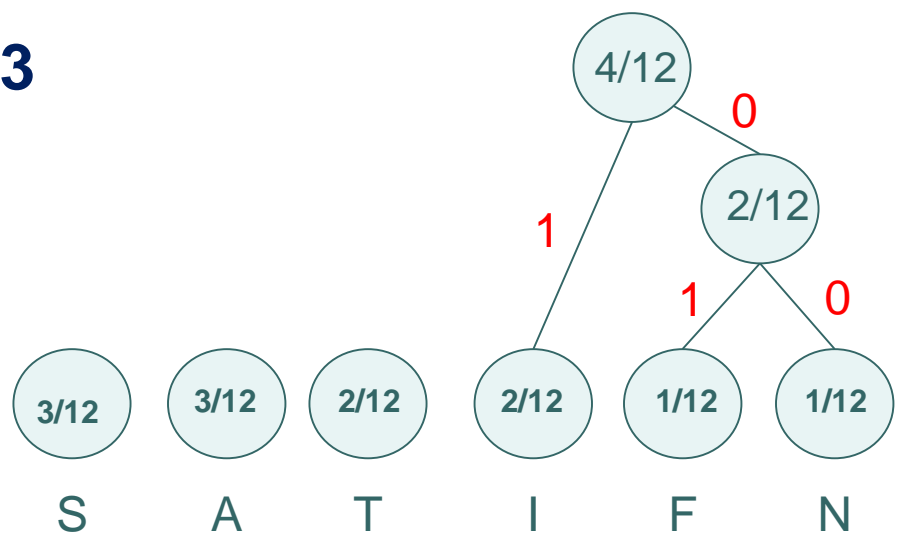
# Exemple (1)

1

On veut compresser : SATISFAISANT



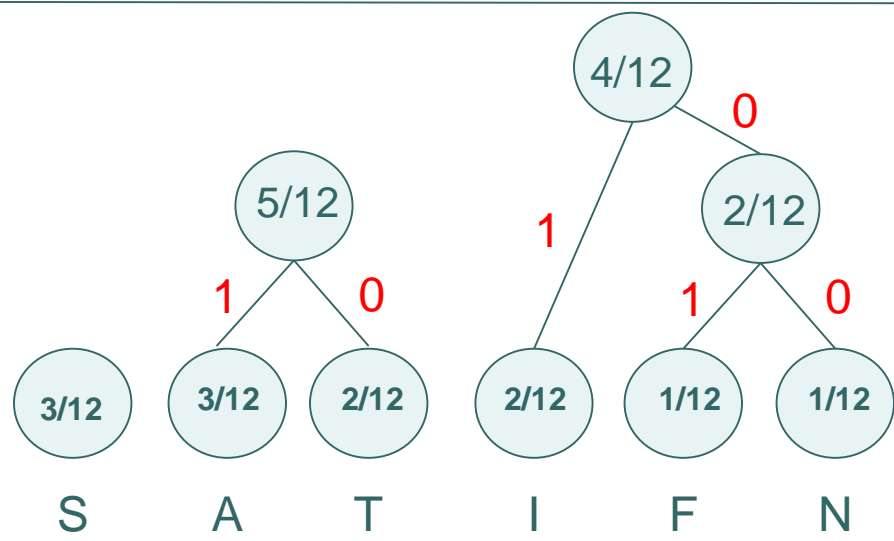
3



2



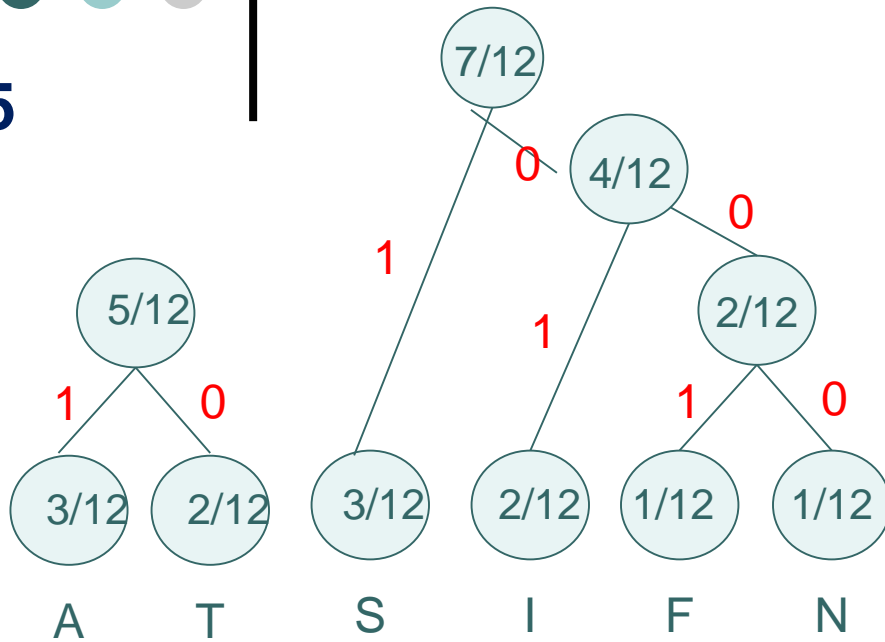
4



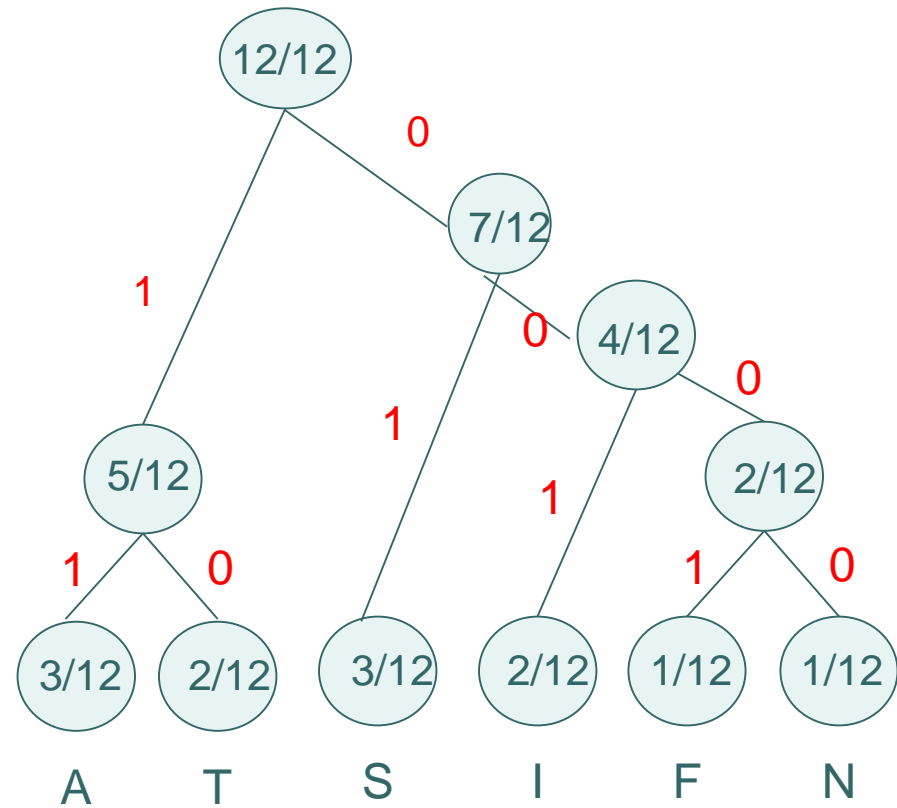
# Exemple(2)



5



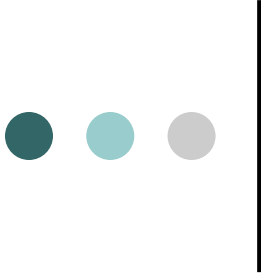
6



Code: 11 10 01 001 0001 0000 AT S I F N

Le texte compressé relatif à SATISFAISANT est donc :

01	11	10	001	01	0001	11	001	01	11	0000	10
S	A	T	I	S	F	A	I	S	A	N	T

- 
- La réduction par rapport au code ASCII est de combien en pourcentage?



# Exercice

- On veut compresser « aabbbbcddef »  
Appliquer l'algorithme de Huffman pour obtenir le codage de ce mot.