INFORMATIQUE
Sciences
Université Paris Cité

# Arbres couvrants

## M2


## Lélia Blin

lelia.blin@irif.fr


2023-2024

# Rappels théorie des graphes

# Distance, diamètre, Rayon

**Distance** : La distance entre deux noeuds $u$ et $v$ dans un graphe non orienté $G$ est le nombre de sauts d'un **chemin minimum** entre $u$ et $v$.

**Rayon d'un nœud**: Le rayon d'un nœud $u$ est la **distance maximale** entre $u$ et tout autre nœud du graphe.

**Rayon:** Le rayon d'un graphe est le rayon minimal de tout nœud du graphe.

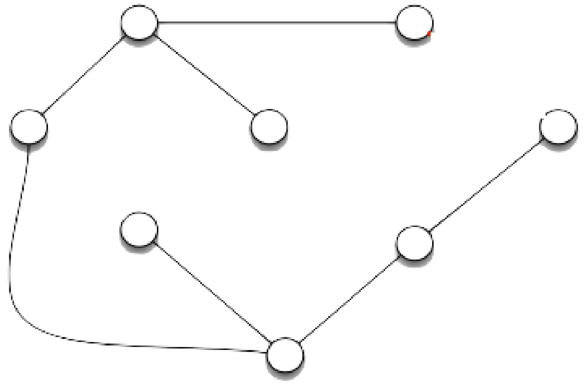
**Le diamètre**: Le diamètre d'un graphe est la distance maximale entre deux nœuds arbitraires.

# Remarque

- Il est clair qu'il existe une relation étroite entre le rayon $R$ et le diamètre $D$ d'un graphe, telle que $R \leq D \leq 2R$

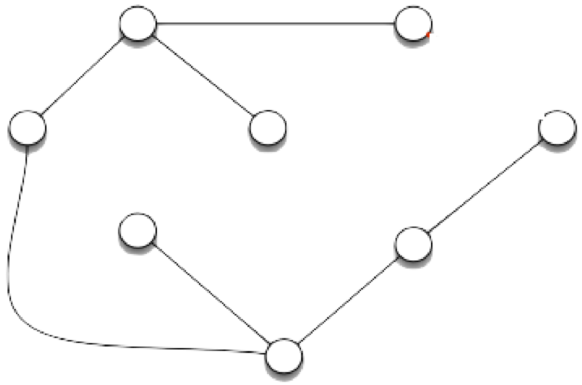
# Définitions d'un arbre (connexe)

- Pour un arbre $T$ a $n$ sommets il y a équivalence entre les définitions suivantes :
  - $T$ est un arbre
  - $T$ est un graphe connexe à $n - 1$ arêtes
  - $T$ est un graphe connexe et la suppression de toute arête le déconnecte



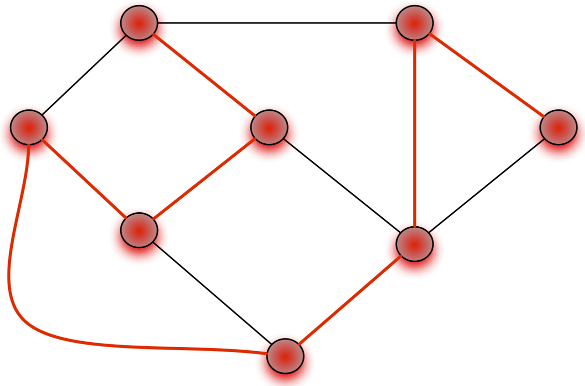
# Définitions d'un arbre (cycle)

- Pour un arbre $T$ a n sommets il y a équivalence entre les définitions suivantes :
  - $T$ est un arbre
  - $T$ est un graphe acyclique à $n-1$ arêtes
  - $T$ est un graphe acyclique et l'ajout de toute arête le rend cyclique.



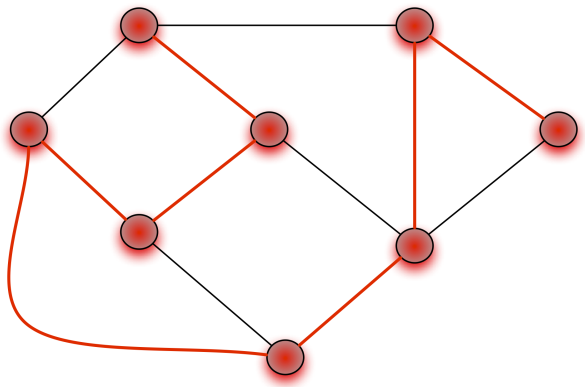
# Graphe partiel

- Un graphe partiel $G'(V, E')$ d'un graphe $G(V, E)$ est:
  - Un graphe qui a les mêmes sommets que $G$.
  - Un graphe dont l'ensemble des arêtes $E'$ est inclus dans $E$.



# Arbres couvrants

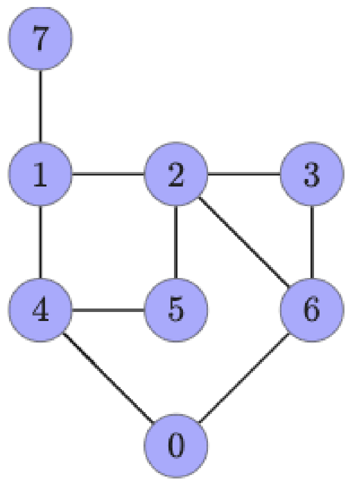
- Un arbre couvrant T d'un graphe G(V,E) est:
  - Un graphe partiel, sans cycle.



# BFS (*Breadth-First Search*)

## Arbre couvrant en Largeur d'abord

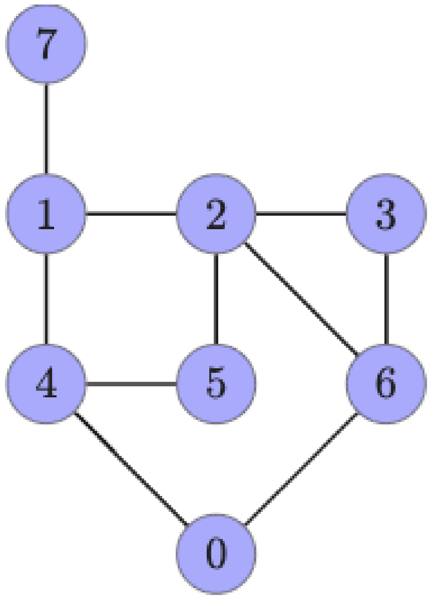
- L'algorithme calculer les distances de tous les nœuds depuis un nœud source dans un graphe non pondéré (orienté ou non orienté).
- Construit un arbre couvrant en largeur



# DFS (Depth-First Search)

## Arbre en profondeur d'abord

- Parcours de graphe récursif
- Construit un arbre couvrant en profondeur



# La diffusion (ou inondation)

# Diffusion

- Une opération de diffusion est initiée par un seul nœud, la **source**.
- La source souhaite envoyer un message à tous les autres nœuds du système.

# Complexité en nombre de message de la diffusion

**Théorème** : (limite inférieure de la diffusion). La complexité en nombre de message de la diffusion est d'au moins $n-1$. Le rayon de la source est une borne inférieure pour la complexité temporelle.

**Théorème** (Limite inférieure de diffusion). Pour un réseau, le nombre d'arêtes $m$ est une borne inférieure pour la complexité du message de diffusion.

# Algorithme d'inondation

```
Initialisation:
        La source v envoie <M> à tous ses voisins
A la réception de <M>
        si <M> n est pas connu
                Delivrer <M>
                Envoie de <M> à tous ses voisins
```

> Remarque: en synchrone cela construit un BFS

# Convergecast

# Convergecast

- La diffusion convergente est identique à la diffusion, mais inversée.
- Au lieu qu'une racine envoie un message à tous les autres nœuds, tous les autres nœuds envoient des informations à une racine
  - en commençant par les feuilles, (autrement dit sur une structure d'arbre $T$ connu).

# Algorithme Echo

```
Initialisation:
        Si feuille envoie <M> à parent
A la récéption de <M> envoyé par u
        envoie <M> au parent
```

# Remarques

- L'algorithme d'écho est généralement associé à l'algorithme d'inondation, qui est utilisé pour indiquer aux feuilles qu'elles doivent commencer le processus d'écho ; c'est ce que l'on appelle l'inondation/écho.
- On peut utiliser convergecast pour la détection de terminaison, par exemple. Si une racine veut savoir si tous les nœuds du système ont terminé une tâche, elle lance un inondation/echo ; le message de l'algorithme d'écho signifie alors "Ce sous-arbre a terminé la tâche".

# Remarques

- La complexité du message de l'algorithme d'écho est $n - 1$, mais avec l'inondation, elle est $O(m)$, où $m = |E|$ est le nombre d'arêtes dans le graphe.
- La complexité temporelle de l'algorithme de l'écho est déterminée par la profondeur de l'arbre couvrant (c'est-à-dire le rayon de la racine dans l'arbre) généré par l'algorithme d'inondation.

# BFS synchrone vs asynchrone

- Dans les systèmes synchrones, l'algorithme d'inondation est une méthode simple mais efficace pour construire un arbre de recouvrement par recherche en largeur (BFS).
- Toutefois, dans les systèmes asynchrones, l'arbre de recouvrement construit par l'algorithme d'inondation peut être loin d'être BFS.
- Nous allons voir deux constructions BFS classiques en asynchrone
  - Dijkstra
  - Bellman-Ford

# Variables

- $p$ phase de recrutement
- $d$ distance à la racine
- $enfant$ ensemble des identifiants des enfants du noeud
- $rep\_attendue$ ensemble des identifiants dont le noeud attend des réponse
- $continue \in \{0,1\}$, $0$ le BFS est fini de mon coté, $1$ le BFS n'est pas fini le noeuds a ajouter des enfants.

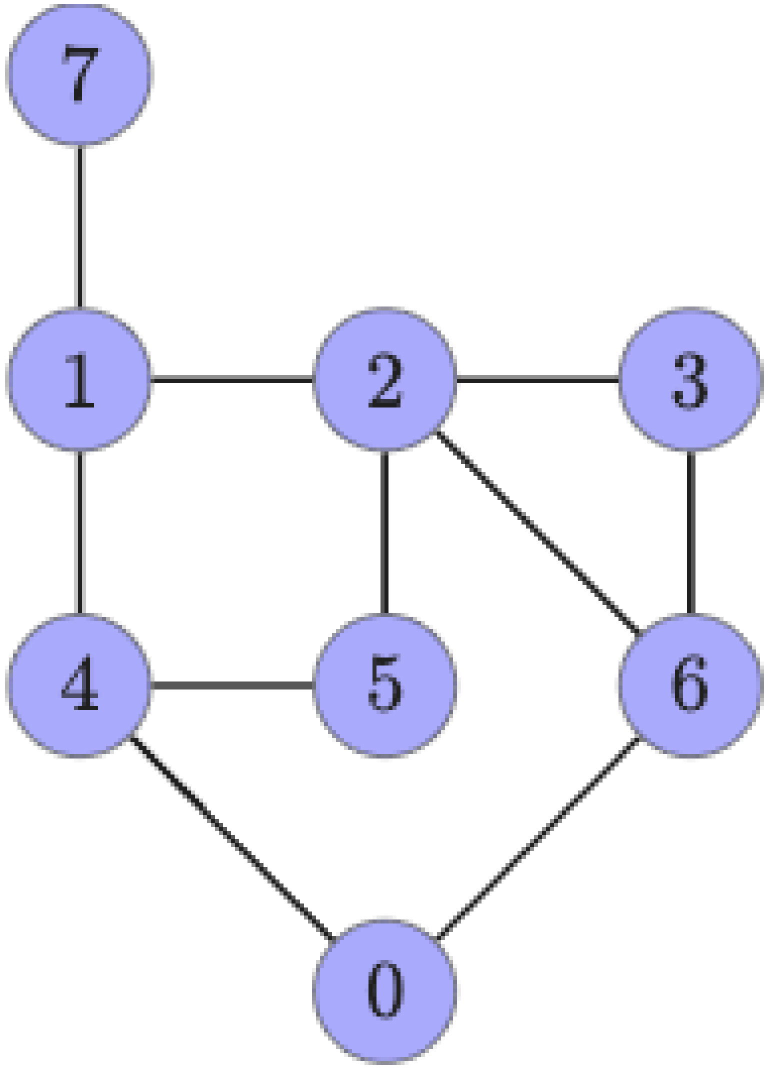
# Messages

- `<start,p>` lance le recrutement des descendants à distance $p$ de la racine (vague)
- `<ACK_ok,p>` je deviens ton enfant
- `<ACK_non,p>` j'ai reçu ton message mais je deviens pas ton enfant
- `<ACK,p,i>` Retour de vague, la construction n'est pas fini i=1 la construction est fini i=0

```
Réveil de la racine
        p:=1
        d:=0
        enfants:=vide
        envoie < start,p> à tous les voisins
        rep_attendue:=voisins
        continue=0

A la réception de <start,p> envoyer par u
        if parent=vide
                d:=p
                parent:=u
                enfants:=vide
                envoie <ACK_ok,p> à parent
                continue=0
        else
                if p=d
                        envoie <ACK_non,p> à u
                if p=d+1
                        envoie < start,p> à tous les voisins sauf parent
                        rep_attendue:=N(v)\{parent}
                if phase>d+1
                        envoie < start,p> à tous les enfants
                        rep_attendue:=enfants
```

```
A la réception <ACK_ok,p> envoyé par u
        if p=d+1:
                enfants:=enfants ∪ {u}
        rep_attendue:=rep_attendue\{u}
        if rep_attendue= vide:
                if racine
                        p:=p+1
                        envoie < start,p> à tous les enfants
                        rep_attendue:=enfant
                else
                        if enfants=vide: envoie <ACK,p,0> au parent
                        else envoie <ACK,p,1> au parent

A la réception <ACK_non,p> envoyé par u
        rep_attendue:=rep_attendue\{u}
        if rep_attendue= vide
                if enfants=vide: envoie <ACK,p,0> au parent; continue:=False
                else envoie <ACK,p,1> au parent

A la réception <ACK,p,i> envoyé par u
        rep_attendue:=rep_attendue\{u}
        if i=1: continue:=1
        if rep_attendue= vide
                if racine
                        if continue=1
                                p=p+1
                                envoie < start,p> à tous les enfants
                                rep_attendue:=enfant
                else
                        envoie <ACK,p,continue> au parent
```

# Exemple



**Théorème** La complexité de l'algorithme de Dijkstra pour le BFS est

- temps: $O(D^2)$ étapes,
- la complexité en message: $O(m + nD)$ message
  - $O(n^2 + nD)$ messages
- la complexité en taille des messages: $O(\log_2 D)$ bits
- La complexité en mémoire de chaque noeud: $O(\Delta \log_2 n \log + \log_2 D)$ bits

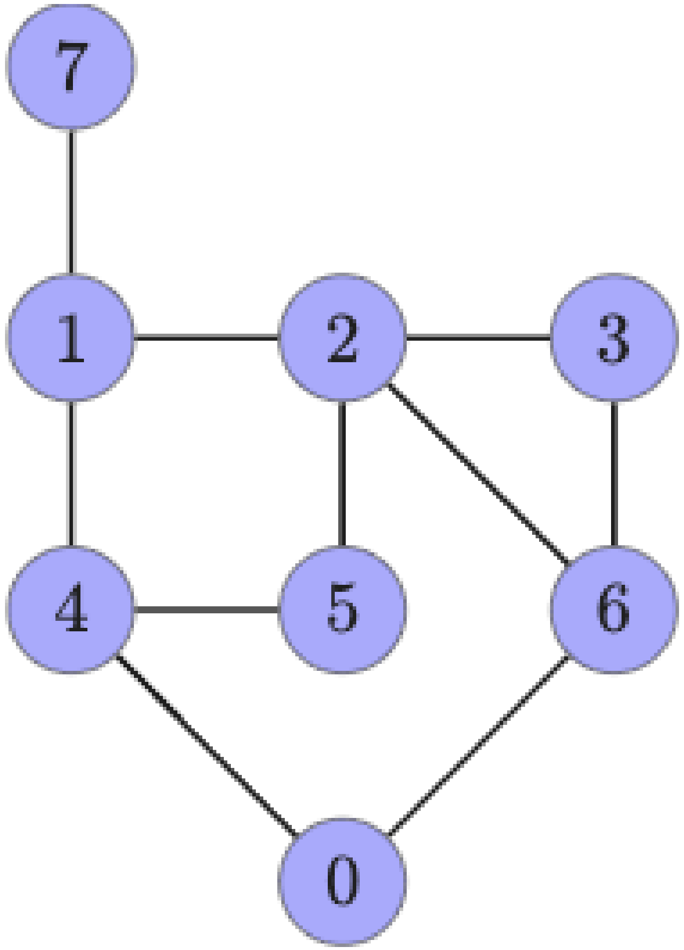
**Conclusion**: peut-on faire mieux?

# Algorithme de Bellman-Ford

- L'idée de base de Bellman-Ford est encore plus simple et très utilisée sur l'internet, puisqu'il s'agit d'une version de base de l'omniprésent protocole BGP (border gateway protocol).
- L'idée est simplement de maintenir la distance à la racine exacte. Si un voisin a trouvé une meilleure route vers la racine, un nœud peut également avoir besoin de mettre à jour sa distance.

```
Intialisation
         d= infini
         parent=vide
Réveil de la racine
        d=0
        envoie <1> à tous les voisins
Reception de <D> envoyé par u
        if d>D
                d:=D; parent:=u;
                envoie <D+1> à tous les voisin sauf parent
```

```
Intialisation
         d= infini
         parent=vide
Réveil de la racine
        d=0
        envoie <1> à tous les voisins
Reception de <D> envoyé par u
        if d>D
                d:=D+1; parent:=u;
                envoie <D> à tous les voisin sauf parent
```



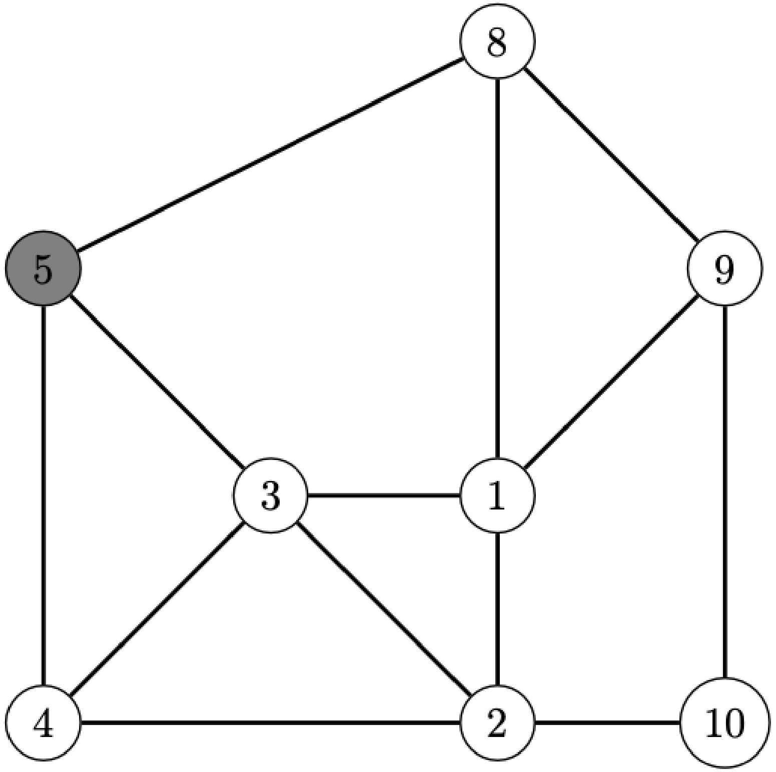
**Théorème** La complexité de l'algorithme de Bellman-Ford pour le BFS est

- temps: $O(D)$ étapes,
- la complexité en message: $O(mn)$ message
  - $O(n^3)$ messages
- la complexité en taille des messages: $O(\log_2 D)$ bits
- La complexité en mémoire de chaque noeud: $O(\log_2 D)$ bits

**Remarque**: en l'état les noeuds ne connaissent pas leurs enfants.

# DFS

```
Procédure chercher
if (il existe un k tel que mark(k)=nonvisite
        Envoyer(<TOKEN>) sur le lien k
        mark(k):=enfant
else if initiateur alors stop;
        else Envoyer(<TOKEN>) sur le lien k tel que mark(k)=parent

Initialisation de la racine
if etat=idle
        etat:=decouvert
        chercher
        Pour tout (k: mark(k)=visite ou mark(k)=nonvisite)
                Envoyer (<VISITED>) sur le lien k

Lors de la réception de <VISITED> sur le lien j
if mark(j)=nonvisite ou mark(j)=enfan
        mark(j):=visite

Lors de la réception de <TOKEN> sur le lien j
if etat=idle
        mark(j):= parent
        etat:=decouvert
        chercher
        Pour tout (k: mark(k)=visite ou mark(k)=nonvisite)
                Envoyer (<VISITED>) sur le lien k
else
        if (mark(j)= enfant ou mark(j)=visite) alors chercher
        if mark(j)=non_visite
                mark(j):=visite; envoye <TOKEN> à j
```


8
5
9
3
1
4
2
10

**Théorème** La complexité de l'algorithme de DFS est

- temps: $O(n^2)$ étapes,
- la complexité en message: $O(n^2)$ message
- la complexité en taille des messages: $O(1)$ bits
- La complexité en mémoire de chaque noeud: $O(\log_2 \Delta)$ bits

# Conclusion

- On a vu des algorithmes pour la construction de BFS et DFS
- Il existe de nombreuse construction d'arbres couvrants sous contraintes
- Donc de nombreux sans racine désigné
  - arbre couvrant de diamètre minimum
  - arbre couvrant de degré minimum
  - **arbre couvrant de poids minimum**