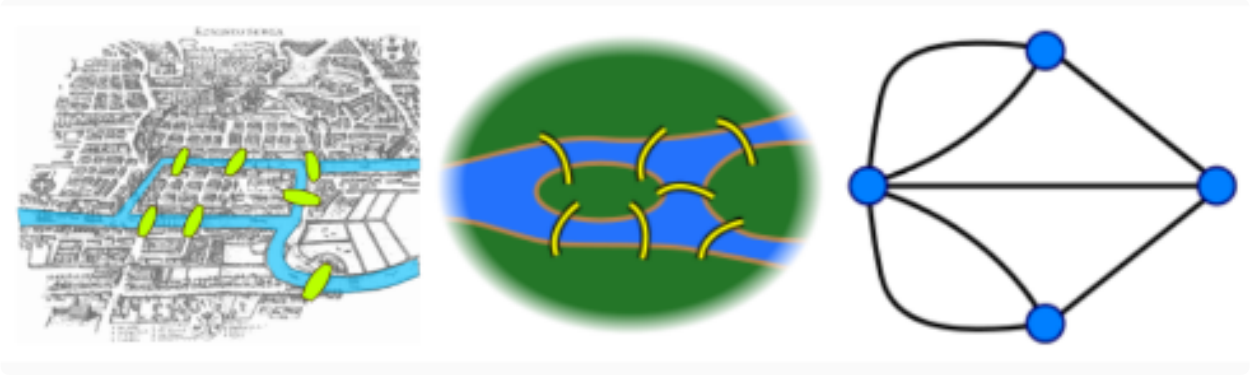
Chapitre 12 - Graphes

**Les graphes sont des structures de données relationnelles.

Les bases de la théorie des graphes remontent au $XVIII^{ème}$ siècle, où Euler a résolu le problème des 7 ponts de Königsberg en le ramenant à la recherche d'un chemin passant une unique fois par chaque arc d'un graphe (appelé ultérieurement chemin *eulérien*) :



Question : Peut-on trouver un chemin dans le graphe qui passe une et une seule fois par chaque arête ?

Aujourd'hui, les graphes ont de très nombreuses applications :

- Réseaux sociaux
- GPS
- Internet
- Graphes de flot de contrôle
- réseaux

I. Définition des graphes orientés et non orientés

1. Graphes non orientés

Définition :

Un graphe non orienté G est un couple (S, A) avec S un ensemble non vide dont les éléments sont appelés des sommets. A un ensemble de paires non ordonnés $\{x, y\}$ avec $x \neq y$ où $(x, y) \in S$ dont les éléments sont appelés des arêtes.

Exemple de graphe non orienté :

$S = \{0, 1, 2, 3, 4, 5, 6\}$ et $A = \{\{0, 6\}, \{1, 3\}, \{1, 2\}, \{1, 4\}, \{3, 4\}, \{2, 4\}, \{5, 4\}\}$

Modifications possibles de la définition :

Boucles : Liaison d'un sommet x à lui même (on n'impose plus $x \neq y$)

Multi-arêtes : plusieurs arêtes entre deux sommets

On parle à ce moment là de multigraphes. Souvent, on ne considère que les graphes finis (S est fini).

Vocabulaire :

- Deux sommets reliés par une arête sont adjacents
- Les sommets adjacents avec un sommet x sont les voisins de x.
- une arête $\{x, y\}$ est incidente aux sommets x et y.

Notation :

Arête $\{x, y\} \iff \{y, x\} \iff xy \iff yx$

Propriété : nombre maximal d'arêtes d'un graphe non orienté.

$|A| \leq \frac{|S|(|S|-1)}{2}$

Preuve :

$\binom{|S|}{2} = \frac{|S|(|S|-1)}{2}$

- On retire le premier sommet aux $n-1$ autres
- On retire le deuxième sommet aux $n-2$ autres
- $\dots$
- On retire le dernier sommet aux 0 autres $\sum_{i=0}^{|S|-1} i$

Exemples concrets de graphes non orientés :

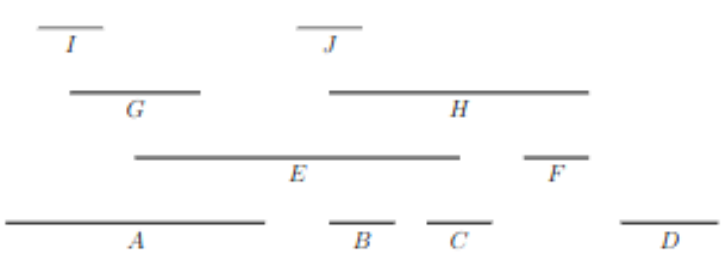
- Graphe de Facebook :

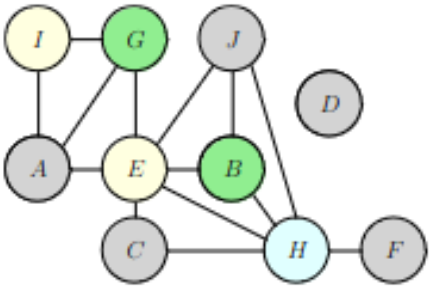
Sommets S = utilisateurs

Arête A = 2 utilisateurs “amis”

$|A| \simeq 10^{11}$ arêtes

- Graphes d'intervalles :





Utile pour les problèmes d’ordonnances.

$|A|$ = nombre d’arêtes

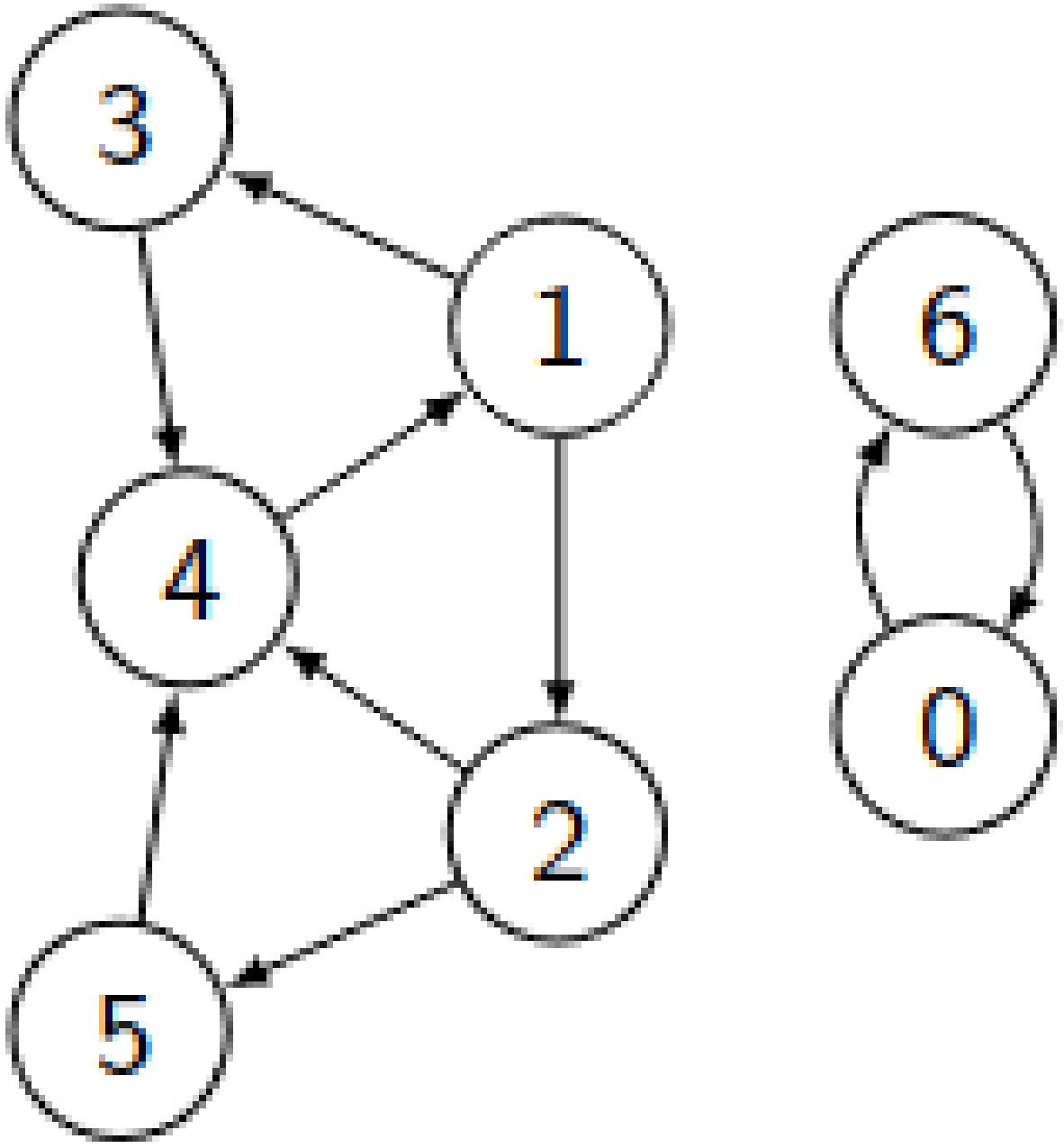
$|S|$ = nombre de sommets

2. Graphes orientés

Définition :

Un graphe orienté G est un couple (S, A) avec S l’ensemble non vide de sommets et A un ensemble de couples (x, y) ordonnés (avec $x \neq y$) avec x et y éléments de S dont les éléments sont des arcs.

Exemple de graphe orienté :



$S = \{0, 1, 2, 3, 4, 5, 6\}$ et $A = \{\{1, 3\}, \{3, 4\}, \{4, 1\}, \{4, 2\}, \{2, 4\}, \{1, 2\}, \{5, 2\}\{0, 6\}\{6, 0\}\}$

On peut éventuellement modifier cette définition pour autoriser les boucles et les multi-arcs.

Notation :

(x, y) un arc $\iff x \to y$

Vocabulaire :

Si on a un arc $x \to y$:

- x est le prédécesseur de y
- y est le successeur de x

Propriété : nombre maximal d'arcs d'un graphe orienté

$|A| \leq |S| \times (|S| - 1)$

Exemples concrets de graphes orientés :

- Graphe d'Instagram :

Sommets S = utilisateurs

arc $x \to y$ = x est abonné à y

- Graphes de flot de contrôle :

Arc $x \to y$ = si le bloc d'instructions y peut s'exécuter juste après x.

II. Vocabulaire de la théorie des graphes

1. Degrés dans un graphe

- Dans un graphe non orienté :

Définition :

Dans un graphe non orienté, le degré d'un sommet x noté $d(x)$ est le nombre d'arêtes incidentes à x (= le nombre de voisins de x).

On a $0 \leq d(x) < |S|$

Le degré d'un graphe est le degré maximal d'un de ses sommets.

Propriété : Graphe $G = (S, A)$

$\sum_{x \in d(x)} d(x) = 2|A|$

- Dans un graphe orienté :

Définition :

Pour un sommet x d'un graphe :

- Le degré sortant de x, noté d_+x est est le nombre de successeurs de x.
- Le degré entrant de x, noté d_-x est est le nombre de prédécesseurs de x

Propriété : Graphe $G = (S, A)$

$\sum_{x \in S)} d_-(x) = \sum_{x \in d(x)} d_+(x) = |A|$

2. Notion de sous-graphe

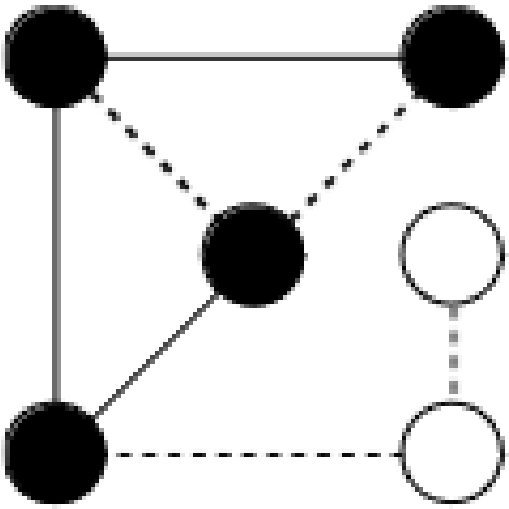
Définition :

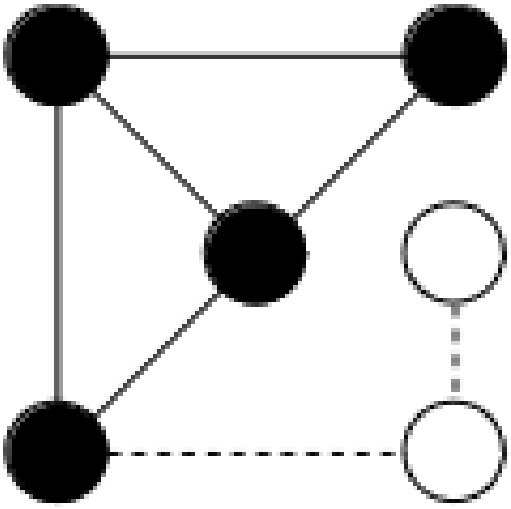
Un sous-graphe d'un graphe $G = (S, A)$ est un graphe $G' = (S', A')$ avec $S' \subset S$ et $A' \subset A$

Un sous-graphe induit est un graphe $G' = (S', A')$, avec $S' \subset S$ et A' est exactement l'ensemble des arêtes/arcs qui relient deux sommets de S' dans G.

Exemple :

Gauche : un sous graphe (sommets noirs et arêtes pleines) ; Droite : un sous-graphe induit





Propriété :

Il y a $2^{|S|} - 1$ sous-graphes induits d'un graphe $G = (S, A)$

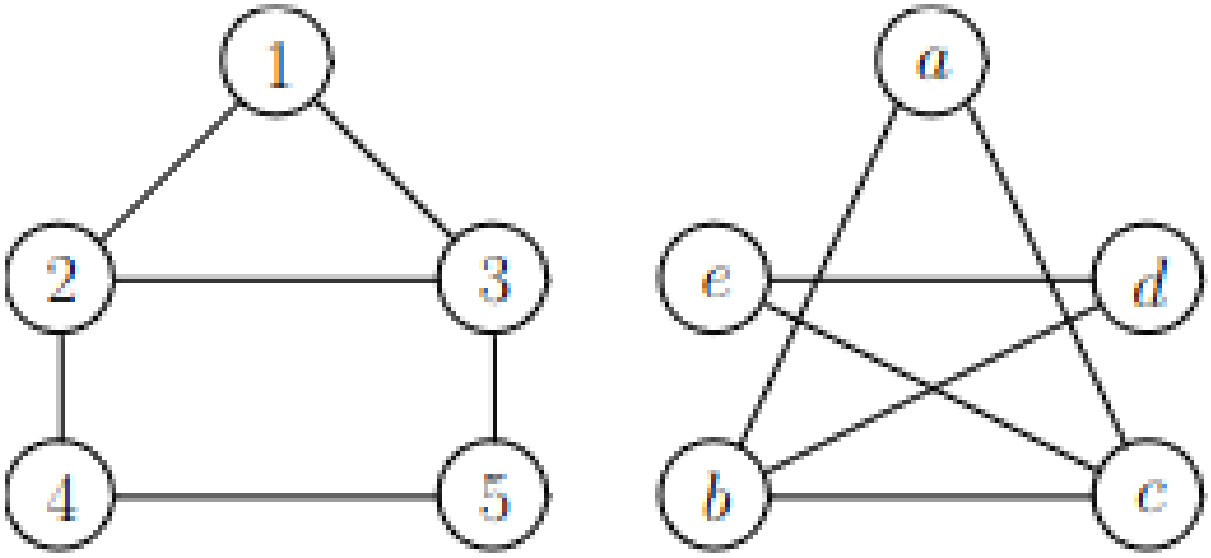
3. Graphes isomorphes

Définition :

Un isomorphisme entre 2 graphes $G = (S, A)$ et $G' = (S', A')$ est une bijection $\varphi : S \to S'$ telle que :

$$\forall (x, y) \in S = \begin{cases} x, y \in A \iff \varphi(x) \times \varphi(y) \in A' \\ x \to y \in A \iff \varphi(x) \to \varphi(y) \in A' \end{cases}$$

Exemple :



4. Chemins dans un graphe

Définition :

Dans un graphe $G = (S, A)$, un chemin de longueur n est une suite $S_0, S_1, \ldots, S_n$ de sommets tels que

$$\forall i \in [|0, n-1|] = \begin{cases} (S_i, S_{i+1}) \in A \\ \{S_i, S_{i+1}\} \in A \end{cases}$$

Un chemin de longueur n possède n arêtes/arcs et n+1 sommets. On accepte les chemins de longueur 0.

"Excalidraw/Image/Drawing 2024-04-10 10.18.08.excalidraw" could not be found.

Chemins :

- $1, 2, 3, 4, 6$
- $3, 7, 6, 8, 9, 6, 5$

- $6, 5, 4, 6, 5$
- $3, 4, 3$
- 6
- $\ldots$

Définition :

Un chemin est dit élémentaire si il ne comporte pas deux fois le même sommet (les S_i sont distincts).

Un chemin est dit simple s'il ne passe pas deux fois par le même arc/la même arête (les $(S_i, S_{i+1}) et \{S_i, S_{i+1}\}$ sont distinctes)

Un chemin élémentaire est forcément simple, la réciproque est fausse

Définition (cycle/circuit) :

(Un cycle est dans les graphes non orientés et le circuit est son équivalent dans un graphe orienté.)

Un cycle est un chemin simple de longueur supérieure à 1 dont les deux extrémités sont le même sommet.

Cycles (en reprenant le schéma précédent) :

- $3, 4, 6, 7, 3$
- $6, 9, 8, 6$
- $3, 7, 6, 4, 3$
- $6, 4, 5, 6$
- $3, 7, 6, 8, 9, 6, 4, 3$

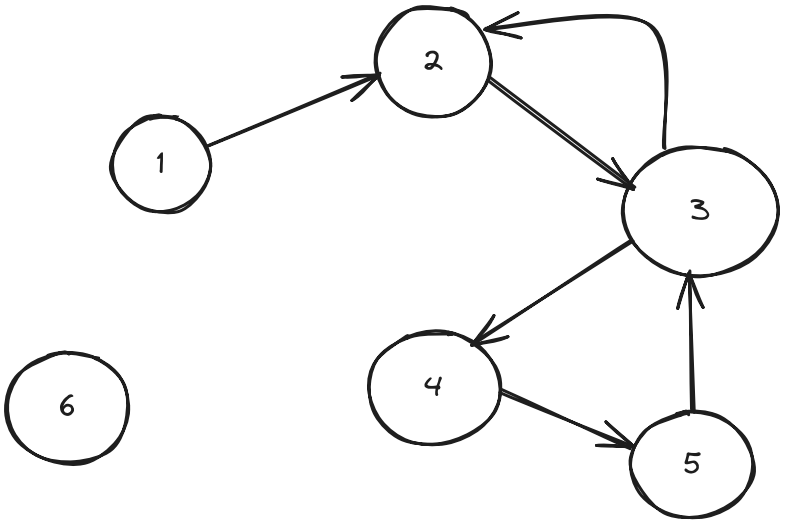
Définition :

Un graphe est dit cyclique s'il comporte au minimum un cycle et est dit acyclique sinon.

5. Notion d'accessibilité dans un graphe

Définition :

Le sommet y est dit accessible depuis le sommet x s'il existe un chemin allant de x à y.



Le sommet 5 est accessible depuis $1, 2, 3, 4, 5$. Le sommet 1 n'est pas accessible depuis 3.

Propriété :

La relation d'accessibilité dans un graphe non orienté est une relation d'équivalence.

Preuve :

- réflexif : un sommet est accessible depuis lui-même (chemin vide).
- symétrie : vrai car dans un graphe non orienté, si on a une arête xy, on a aussi $yx \in A$ donc si y accessible depuis x, le même chemin "à l'envers" donne x accessible depuis y.
- transitivité : si on a un chemin $x = S_0, \ldots, S_n = y$ et un chemin $y = p_0, . , p_m = 2$ alors le chemin $x = S_0, \ldots, S_n, p_1, \ldots, p_m = 2$ relie x à 2 donc 2 est accessible depuis x.

Dans les graphes orientés, la symétrie n'est plus vérifiée donc ce n'est pas une relation d'équivalence.

Les propriétés suivantes sont équivalentes :

- il existe un chemin allant du sommet x au sommet y.
- il existe un chemin simple allant du sommet x au sommet y.
- il existe un chemin élémentaire allant du sommet x au sommet y.

On les numérotent de 1 à 3 pour la preuve suivante.

Preuve :

On a déjà $(3) \Rightarrow (2)$ et $(2) \Rightarrow (1)$. Montrons que $(1) \Rightarrow (3)$.

Soit $x = S_0, \ldots, S_n = y$ de longueur minimale. Par l'absurde, si on a deux sommets de chemin $S_i = S_j$ avec $i < j$. Alors $x = S_0, \ldots, S_i, S_{j+1}, \ldots, S_n = y$ est un chemin plus petit ce qui contredit la minimalité de la longueur.

Donc le chemin $x = S_0, \ldots, S_n = y$ est élémentaire.

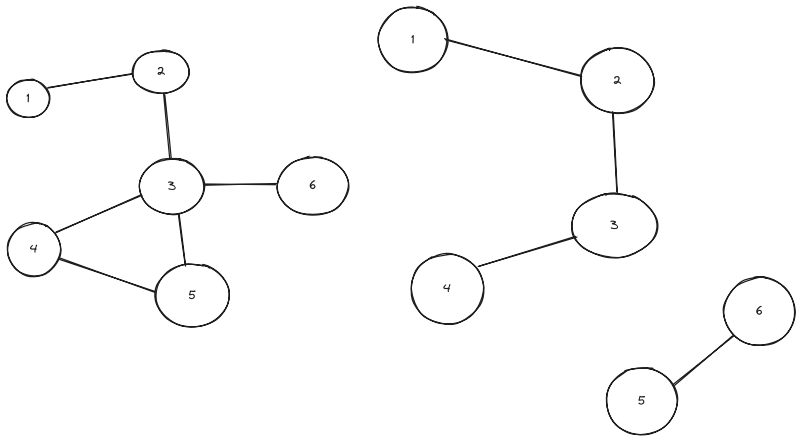
6. Connexité

Dans les graphes non orientés :

Définition :

Un graphe non orienté $G = (S, A)$ est dit connexe s'il existe un chemin de x à y pour tous sommets $x, y \in S$.

Exemple :



Le graphe de gauche est connexe, le graphe de droite ne l'est pas.

Définition :

Les composantes connexes d'un graphe non orienté sont les sous-graphes induits par les classes d'équivalences de la relation d'accessibilité.

En reformulant :

- la composante connexe C_x d'un sommet x contient tous les sommets accessibles depuis x.
- $y \in C_x \Rightarrow C_x = C_y$
- les composantes d'un graphe sont les sous-graphes induits connexes et maximaux (en nombre de sommets).
- Un graphe connexe possède 1 composante connexe.

Propriété :

$G = (S, A)$. Soient x et y tels que $\{x, y\} \notin A$. On ajoute une arête xy dans G.

2 cas possibles :

- x était accessible depuis y : le nombre de composantes connexes ne change pas, et il existe un cycle passant par xy.
- x et y n'appartenaient pas à la même composante connexe : le graphe a une composante connexe en moins et il n'y a aucun cycle passant par l'arête xy.

Dans un graphe orienté :

Définition :

Un graphe orienté, $G = (S, A)$ est dit fortement connexe s'il existe un chemin reliant x à y pour tout sommets $x, y \in S$

Définition :

Un graphe orienté est dit faiblement connexe quand l'oubli de l'orientation donne un graphe non orienté connexe.

Propriété :

La relation R définie sur les graphes orientés par

$xRy \iff x$ accessible depuis y et y accessible depuis x

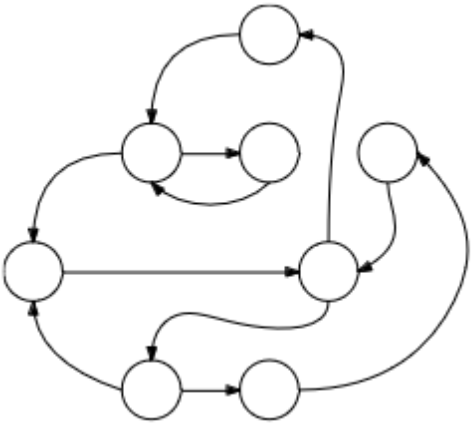
est une relation d'équivalence.

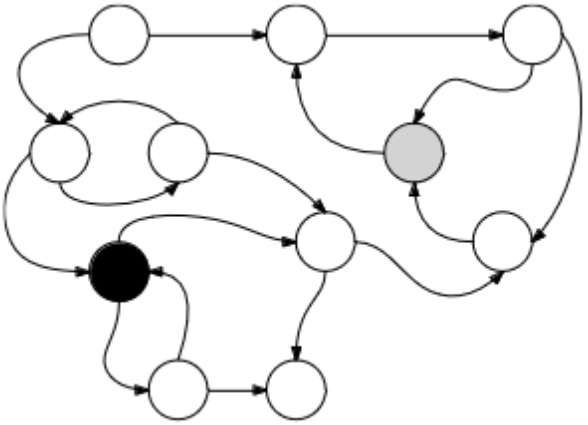
Définition :

Les composantes fortement connexes d'un graphe orienté sont les sous-graphes induits par les classes d'équivalences de cette relation R.

Exemples :

(un graphe fortement connexe et un non fortement connexe (sommet noir non accessible depuis le sommet gris))





7. Coloration de graphes

Définition :

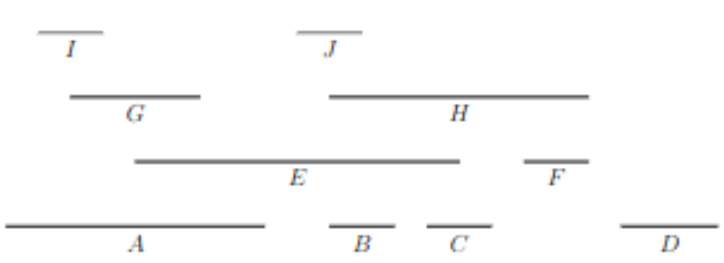
Une k-coloration d'un graphe $G = (S, A)$ est une application $\varphi : S \to [\![0, k-1]\!]$ telle que $\{x, y\} \in A \Rightarrow \varphi(x) \neq \varphi(y)$, la valeur $\varphi(x)$ est appelé couleur de x

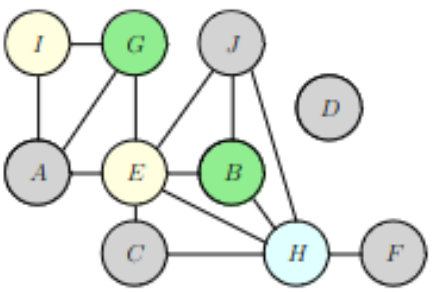
Définition :

Un graphe G est dit k-colorable s'il existe une k-coloration. Le nombre chromatique $X(G)$ d'un graphe G est le plus petit k tel que G est k-colorable.

Exemple d'application aux graphes d'intervalles :

- Sommets = intervalles
- Voisins = intervalles incompatibles
- Coloration : 2 intervalles incompatibles ont des couleurs différentes.





III. Graphes particuliers

1. Graphes non orientés ayant une forme particulière

- Graphe entièrement déconnecté :

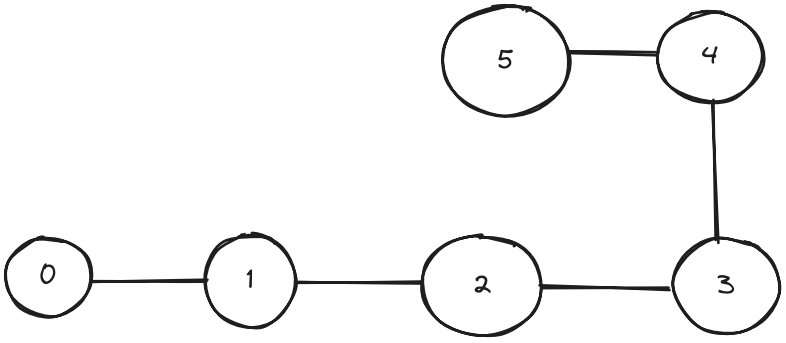
Graphe composé entièrement de sommets et d'aucune arête. Il possède $|S|$ composantes connexes.

- Graphe complet :

Possède une arête entre chaque couple de sommets possible. On le note K_n avec n le nombre de sommets. Il possède donc $\frac{n(n-1)}{2}$ arêtes, il est connexe.

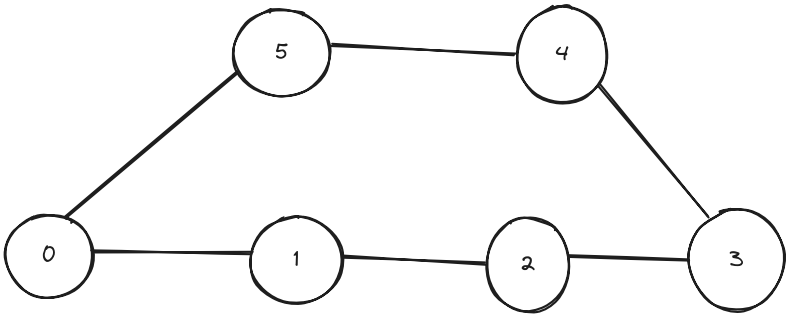
- Graphe chemin :

Ne possède que un "chemin". On note P_n le graphe chemin à n sommets.
$\{i,j\} \in A \iff i = j+1$.



- Graphe cycle :

On note C_n le graphe cycle à n sommets numérotés dans $[\![0, n-1]\!]$.
$\{i,j\} \in A \iff i = j+1$ (modulo n)



On peut donner des définitions équivalentes pour les graphes orientés.

2. Graphes ayant des propriétés particulières

- Graphes creux / denses :

Si peu d'arêtes/d'arcs, on dit que le graphe est creux, sinon il est dense.

- Graphes bipartis :

Un graphe $G = (S, A)$ est dit biparti s'il existe une partition de S en deux ensemble S_1, S_2 telle que pour toute arête $\{x, y\} \in A, x \in S_1$ et $y \in S_2$ (ou l'inverse).

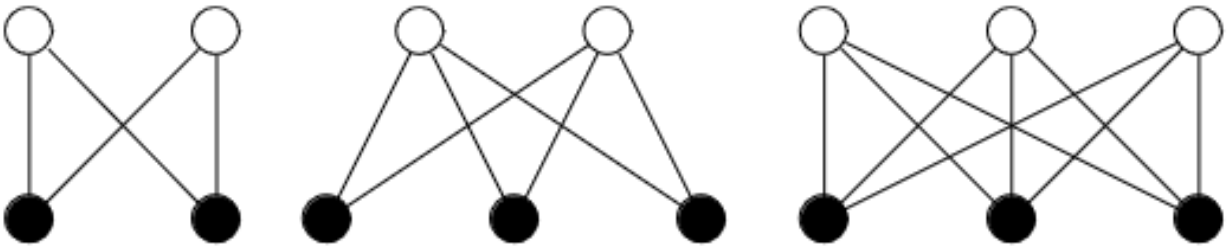
Propriété :

Un graphe est biparti si et seulement si il est 2-colorable.

- Graphes bipartis complets :

$K_{n,p}$ est le graphe biparti à $n + p$ sommets avec n le nombre de sommets de S_1 et p le nombre de sommets de S_2. Chacun des n sommets de S_1 est relié par une arête à chacun des p sommets de S_2.

Exemple :



$K_{2,2}$ $K_{3,2}$ $K_{3,3}$

- Graphes planaires :

On peut représenter le graphe dans le plan sans qu'aucune arête ne se croise. Le plus petit graphe complet non planaire est K_5, biparti complet est $K_{3,3}$.

- Graphes eulériens :

Un chemin eulérien de G est un chemin simple qui contient toutes les arêtes de G.

Un cycle (circuit pour un graphe orienté) eulérien est un chemin eulérien dont les extrémités sont identiques.

Un graphe est eulérien s'il possède au moins un cycle eulérien.

- Graphes hamiltoniens :

Un chemin hamiltoniens de G est un chemin élémentaire qui contient tous les sommets de G.

Un cycle (circuit pour un graphe orienté) hamiltonien est un chemin hamiltonien élémentaire sauf aux extrémités.

Un graphe est hamiltonien s'il possède au moins un cycle hamiltonien.

3. Arbres dans la théorie des graphes

Définition :

Dans la théorie des graphes, un arbre est défini comme étant un graphe connexe et acyclique (non positionnel).

Propriété :

Un graphe non orienté acyclique à n sommets possède au plus $n-1$ arêtes.

Lemme :

Tout graphe non orienté acyclique possède un sommet qui a un degré inférieur ou égal à 1.

Preuve du lemme :

On raisonne par l'absurde. Supposons que tous les sommets d'un graphe non orienté acyclique sont de degré au moins 2. Construisons un chemin depuis n'importe quel sommet du graphe, en se déplaçant à chaque fois vers un sommet différent de celui dont on venait (possible car de degré supérieur ou égal à 2).

Comme le nombre de sommets est fini, on finit forcément par tomber sur un sommet déjà dans le chemin. Donc il y a un cycle, absurde.

Preuve de la propriété :

Par récurrence, montrons H_n : “Un graphe non orienté acyclique à n sommets possède au plus $n-1$ arêtes.”

Initialisation : Un graphe à 1 sommet possède 0 arête.

Hérédité : Supposons H_n.

D’après le lemme, un graphe non orienté acyclique à $n+1$ sommets possède un sommet de degré inférieur ou égal à 1.

Le graphe induit sur $S\backslash\{x\}$ avec x un sommet de degré inférieur ou égal à 1 est aussi acyclique et a n sommets et par hypothèse de récurrence a donc au plus $n-1$ arêtes.

Donc G a au plus : $n-1+d(x)$ arête (avec $d(x)$ le degré du sommet x) donc G a au plus n arêtes.

Propriété :

Un graphe non orienté connexe à n sommets possède au moins $n-1$ arêtes.

Preuve par récurrence sur n :

Initialisation : Un graphe connexe à 1 sommet possède 0 arête.

Hérédité : Soit $G=(S,A)$ un graphe connexe à $n \geq 2$ sommets. Tout sommet est de degré non nul.

- Si G a un sommet de degré 1, noté x. Le sous-graphe induit par $S\backslash\{x\}$ est connexe à $n-1$ sommets. Donc par hypothèse de récurrence, il a au moins $n-2$ arêtes donc G a au moins $n-2+d(x)=n-1$ arêtes.
- Sinon, tous les sommets sont de degré supérieur ou égal à 2.

$$|A| = \frac{1}{2} \times \sum_{x \in S} d(x) \geq \frac{1}{2} \times 2n \geq n$$

Caractérisation des arbres :

Les propriétés suivantes sont équivalentes :

- G est un arbre.
- G est acyclique et a $|S| - 1$ arêtes.
- G est connexe et a $|S| - 1$ arêtes.
- G est minimalement connexe (impossible d'enlever une arête en conservant la connexité).
- G est maximalement acyclique (impossible d'ajouter une arête sans créer de cycle).
- Pour tous sommets x, y de G, il existe un unique chemin élémentaire reliant x et y.

Définition :

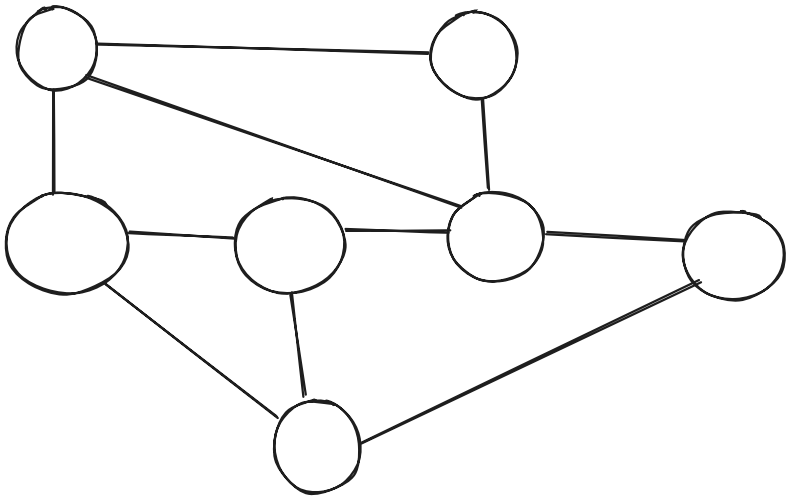
Un graphe acyclique est une forêt. Les composantes connexes d'une forêt sont les arbres.

Définition :

Les arbres enracinés sont les arbres pour lesquels on a distingué un sommet particulier appelé la racine.

Définition :

Un arbre couvrant d'un graphe $G = (S, A)$ est un arbre dont l'ensemble des sommets est S.



Propriété :

Tout graphe connexe admet un arbre couvrant.

Preuve :

Considérons l'algorithme suivant :

- Entrée : graphe $G = (S, A)$ connexe
- Algorithme :

```
G' = (S,A'=A)
Tant que G' n'est pas minimalement connexe :
        Choisir une arête de xy appartenant à A' telle que le graphe
(S,A'\xy)
        est connexe
        A' <- A'\xy
Renvoyer G'
```

Correction de l'algorithme :

Invariant : G' est connexe

- Avant la boucle, vrai car G est connexe par précondition.
- Si l'invariant est vrai en entrée d'une itération, il est vrai en sortie car S' n'a pas changé et on a retiré une arête choisie telle que G' reste connexe.
- Si la boucle termine, on a bien en sortie G' connexe.

Variant : $|A'|$

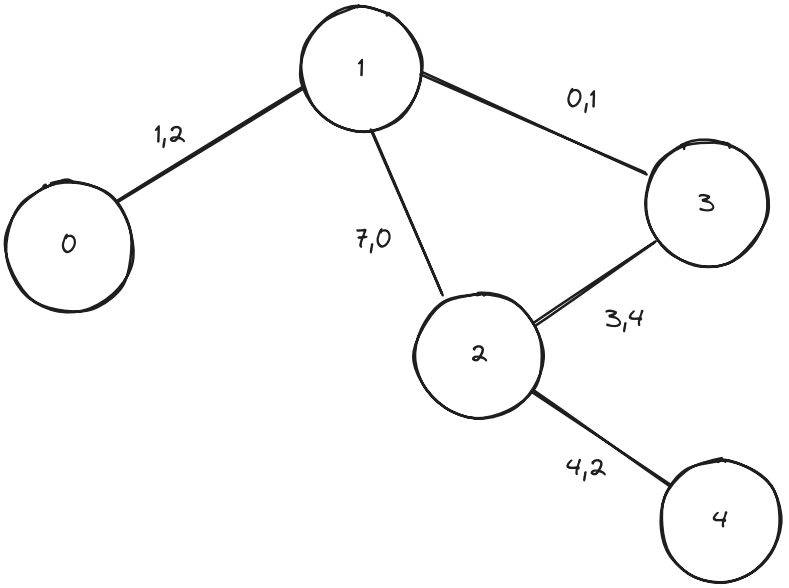
- positif car c'est un cardinal
- strictement décroissant car on retire un élément à chaque itération

De plus, en sortie de boucle, G' est minimalement connexe donc c'est un arbre et il a le même ensemble de sommets que G donc c'est un arbre couvrant de G.

L'existence et la correction de cet algorithme montre la propriété.

4. Graphes pondérés

Données sur les arcs/arêtes :



Définition :

Un graphe non orienté/orienté pondéré est un triplet (S, A, ω) avec S l'ensemble des sommets, A l'ensemble des arêtes/arcs et ω est une application de A dans $\mathbb{R}$ qui à une arête/arc associe son poids ω est appelée fonction de pondération. Généralement, on étend ω en définissant

$$\begin{cases} \omega(xy) \\ \omega(x \to y) \end{cases} = +\infty$$

si

$$\begin{cases} xy \\ x \to y \end{cases} \notin A$$

Exemples :

- Graphes probabilistes
- Graphes routiers
- Internet

Définition :

Soit c le chemin $S_0, S_{1,} \ldots, S_n$ le poids du chemin c est défini comme
ω(c) = $\sum_{i=0}^{n-1} \omega(S_i, S_{i+1})$.

IV. Représentation des graphes

1. Matrice d'adjacence

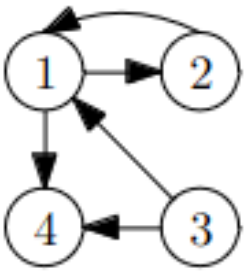
Définition :

Soit $G = (S, A)$ un graphe. La matrice d'adjacence de G est une matrice $M = (m_{i,j})$ avec $0 \leq i < |S|$ et $0 \leq j < |S|$ telle que $(m_{i,j}) = 1$ si

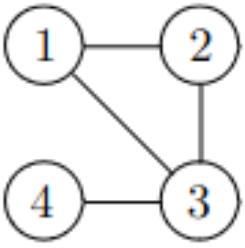
$$\begin{cases} S_i \to S_j \\ S_i \times S_j \end{cases} \in A$$

et 0 sinon.

Exemples :



$$\begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$



$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Propriété :

La matrice adjacente d'un graphe non orienté est symétrique.

Propriété :

Il n'y a que des 0 sur la diagonale (puisque les boucles ne sont pas autorisées dans la définition simple des graphes non orienté/graphe orienté).

Complexité spatiale :

Matrice d'adjacence d'un graphe $G = (S, A)$ a une complexité spatiale de $|S|^2$

Plus adaptée pour les graphes denses.

Complexité temporelle :

- déterminer si 2 sommets sont voisins/successeur $\mathcal{O}(1)$
- Trouver tous les voisins/successeurs d'un sommet : $\mathcal{O}(|S|)$
- trouver tous ces prédécesseurs $\mathcal{O}(|S|)$

Avantage : même complexité

Inconvénient : pas terrible

Implémentation :

- OCaml :

```
type matrice_adjacente = int array array
```

- C :

Tableau statique de taille fixée (nombre maximum de sommets) linéarisé.

Exemple : Nombre maximal de sommets est 20

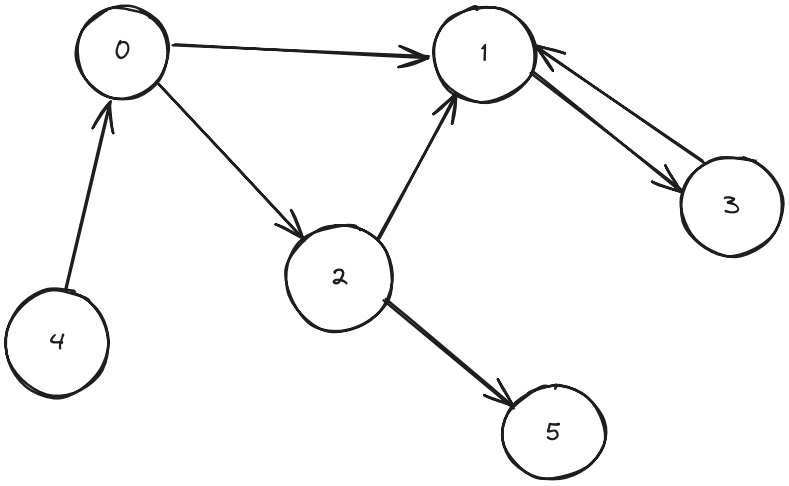
```
typedef int mat_adj[400];

struct graphe_s {
        int nb_sommets;
        mat_adj matrice;
};
```

2. Liste d'adjacence

Définition :

Cette représentation consiste à stocker, pour chaque sommets du graphe, la liste de ses voisins/successeurs.



Complexité spatiale :

$\mathcal{O}(|S| + |A|)$

Plus appropriée pour les graphes creux.

Complexité temporelle :

- Déterminer si S_j voisins /successeur de S_i : $\mathcal{O}(d(S_i))$
- Trouver les successeurs/voisins d'un sommet S_i : $\mathcal{O}(d(S_i))$
- Trouver les prédécesseurs : $\mathcal{O}(|S| + |A|)$

Implémentation :

- Ocaml :

```
type liste_adj = int list array
```

- C :

```
//max 20 sommets

typedef int lst_adj[20][21];

struct graphe_s {
        int nb_sommets;
        lst_adj adj;
};
```

Le 21 vient soit d'une sentinelle à la fin, soit du degré au début.

3. Sérialisation

- Première ligne : $|S|$
- Les lignes suivantes sont des x, y représentant $x \to y \in A$.

V. Parcours de graphes

- Si aucune contrainte, on peut regarder les sommets dans l'ordre de numérotation.

  Mais cela n'apporte aucune information sur les arêtes.
- Généralement, on parcourt un graphe de voisins en voisins.

  2 manières de faire :
  - Le parcours en profondeur
  - Le parcours en largeur

Pour ne pas tourner en rond, on stocke les sommets déjà vus.

Cette structure doit permettre un test d'appartenance efficace :

- Tableau associatif
- Tableau de taille $|S|$ que l'on rempli de true/false.

Mesures de la complexité : notions d'ordre d'un graphe et de taille d'un graphe.

Parcours général d'un graphe :

Si les étiquettes des sommets ne sont pas des entiers naturels consécutifs.

- On attribue aux sommets de nouvelles étiquettes $0, 1, 2, 3, \ldots, |S| - 1$
- On garde dans une structure de données (tableau/tableau associatif) les correspondances étiquette/numérotation.

1. Parcours en profondeur

Algorithme :

- Entrée : $G = (S, A)$ et sommet $dep \in S$
  vus $\leftarrow$ { } (tableau associatif vide, tableau de taille $|S|$ rempli de false)

```
Fonction explorer (sommet s)
        Si S n'appartient pas à vus : (Pas d'association de clés/indice
s=false)
                traitement de s
                vus <- ajouter s (ajouter association de clés/indice s =
true)
                Pour chaque voisin/successeur v de s :
                        explorer(v)
explorer(dep)
```

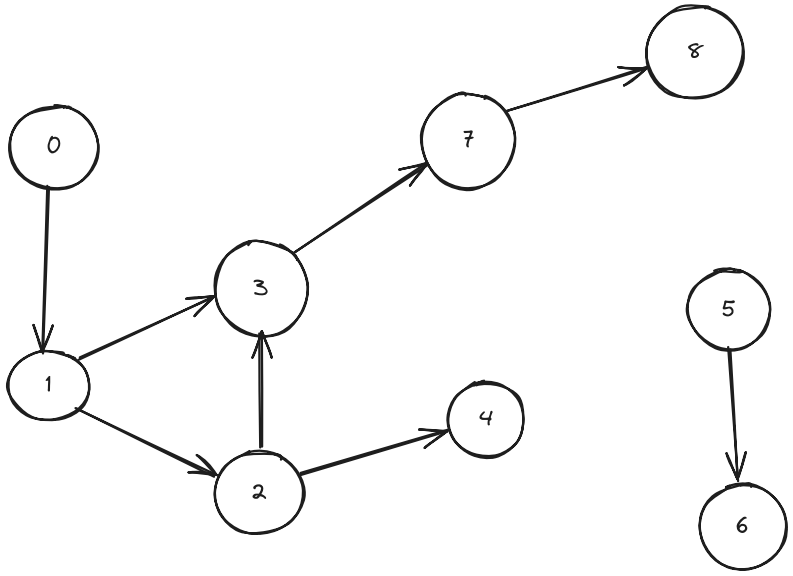
Complexité :

- Création de vus : $\mathcal{O}(|S|)$
- Fonction explorer :
  $\rightarrow$ Chaque sommet est vu au plus 1 fois.
  $\rightarrow$ Test d'appartenance à vus est en $\mathcal{O}(1)$.
  $\rightarrow$ Pour un sommet s fixé, le nombre d'itérations de la boucle (= nombre d'appels récursifs lancés) est en $\mathcal{O}(d(s))$ en supposant avoir utilisé la liste d'adjacence de G.
  $\rightarrow$ L'ajout de s à vus est en $\mathcal{O}(1)$

Total :

$\mathcal{O}(|S|) + \sum_{s \in S}(\mathcal{O}(d(s)) + \mathcal{O}(1)) = \mathcal{O}(|S| + |A|)$ si traitement en $\mathcal{O}(1)$.

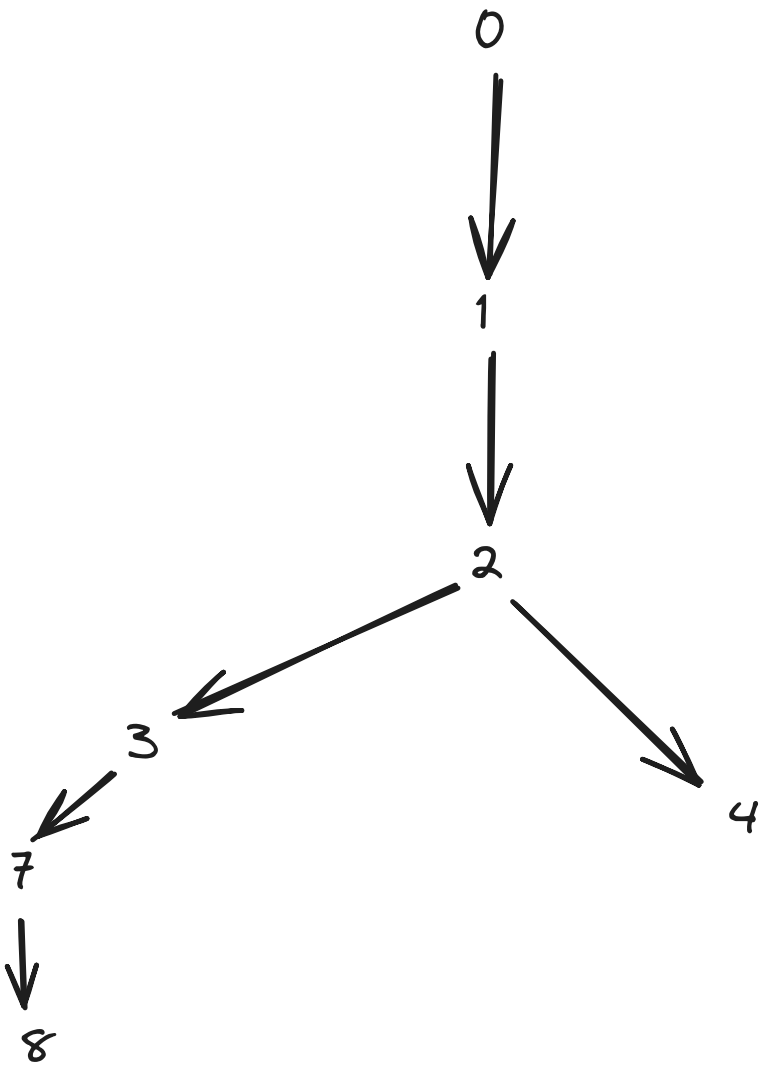
Exemple :



Parcours en profondeur depuis 0 : $0, 1, 2, 3, 7, 8, 4$

Seuls les sommets accessibles depuis le départ sont traités.

Arborescence du parcours :



2. Parcours en largeur

Algorithme : graphe $G = (S, A)$ et sommet $dep \in S$.

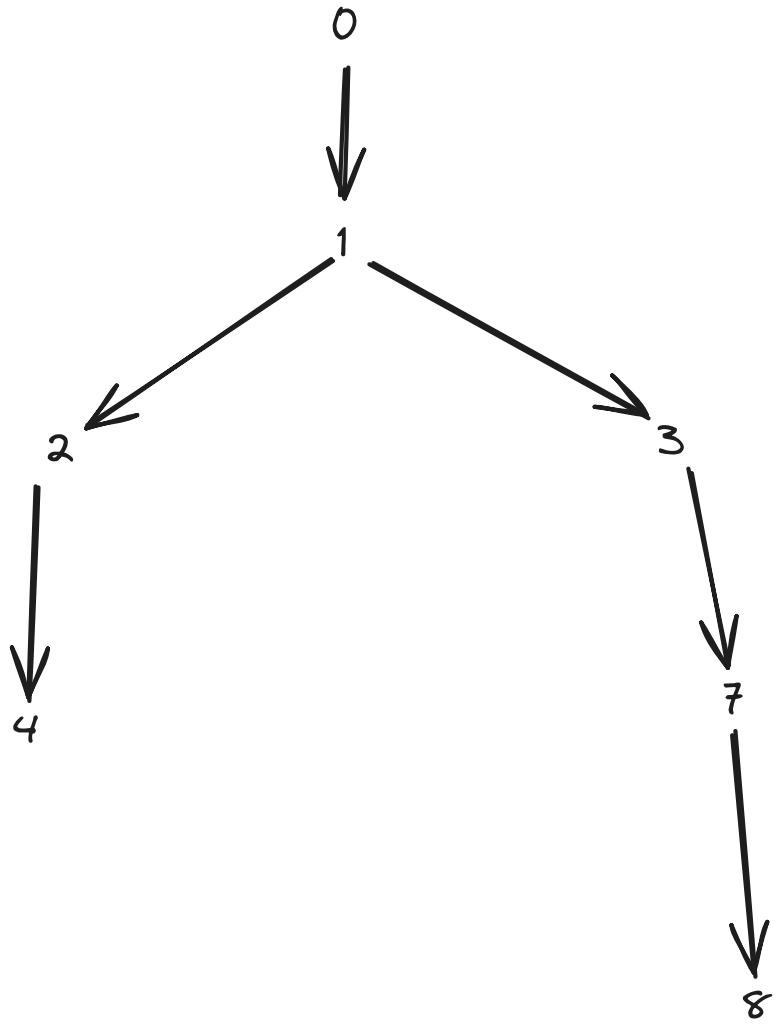
```
vus <- { }
f <- file vide
f <- enfiler(dep)
vus <- ajouter(dep)
Tant que f non vide :
        s <- defiler(f)
        traitement de s
        Pour chaque voisin/successeur de s :
                si v n'appartient pas à vus :
```

```
                    f <- enfiler(v)
                    vus <- ajouter(v)
```

Complexité : $\mathcal{O}(|S| + |A|)$ (même calcul que pour le parcours en profondeur)

En reprenant le même graphe que pour le parcours précédent, le parcours en largeur donne : $0, 1, 2, 3, 4, 7, 8$

Arborescence :



Définition :

Soit $G = (S, A)$ un graphe. La distance $D(x, y)$ d'un sommet $x \in S$ à un sommet $y \in S$ est la longueur minimale en nombres d'arêtes/arcs d'un chemin allant de x à y. Si un tel chemin n'existe pas, $D(x, y) = +\infty$

Propriété :

S'il existe un arc $y \to y'$ alors pour tout sommet x.

$D(x, y') \leq D(x, y) + 1$

Propriété du parcours en largeur depuis sommet dep :

Si $D(dep, x) < D(dep, y) < \infty$

alors x sera traité avant y.

Preuve :

Il faut montrer l'invariant suivant :

Un sommet est dit :

- ouvert si s est dans file f.
- fermé si s est dans vus mais pas f.
- vierge si s n'est pas dans vus.

L'invariant, valable à la fermeture d'un sommet s_k, en ayant numéroté les sommets dans l'ordre de leur ouverture $s_0, \ldots, s_n$.

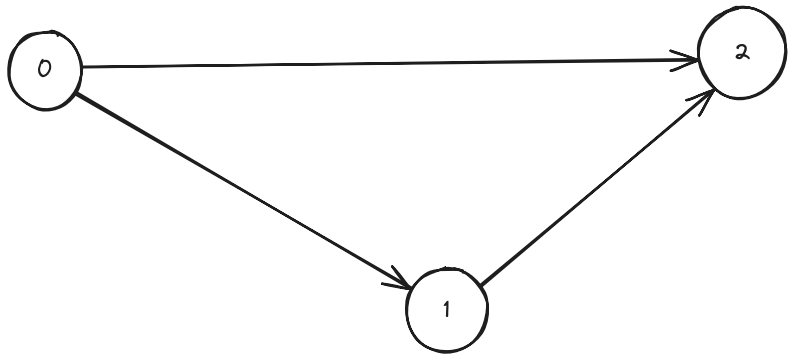
(1) La file a la forme $s_{k+1}, \ldots, s_{k+p}$ avec
$D(dep, s_k) \leq D(dep, s_{k+1}) \leq \cdots \leq D(dep, s_{k+p}) \leq D(dep, s_k) + 1$

(2) Si $D(dep, s_i) \leq D(dep, s_k)$, alors s_i est fermé.

(3) Si $D(dep, s_i) = D(dep, s_k)$, alors s_i n'est pas vierge.

3. Algorithme générique

Si on remplace la file par une pile.



Il faut autoriser le fait d'avoir plusieurs fois le même sommet dans la pile pour retrouver le parcours en profondeur.

Algorithme générique de parcours de $G = (S, A)$ depuis sommet $dep \in S$:

```
PARCOURS (G, depart) :
    a_traiter <- structure vide
         a_traiter <- ajouter(dep)
    vus <- {}
    Tant que a_traiter est non vide :
                s <- retirer un sommet de a_traiter
                Si s n'appartient pas à vus :
                        vus <- ajouter s
                        traitement de s
                        POUR chaque voisin v de s :
                                a_traiter <- ajouter(v)
```

Si `a_traiter` est une pile, le parcours est en profondeur.
Une file, le parcours est en largeur.
D'une stratégie aléatoire, le parcours est quelconque.

4. Applications des parcours

Déterminer si un graphe non orienté est connexe :

→ vérifier à la fin du parcours si tous les sommets sont dans vus.

Trouver les composantes connexes :

→ les sommets dans vus à l'issue d'un parcours sont dans la même composante.

→ on recommence le parcours avec d'autres sommets tant que nécessaire.

Distance d'un sommet x à y :

→ on garde un tableau de distances, au début rempli de $+\infty$ sauf distance de x à $x = 0$.

→ au moment où on regarde les voisins v d'un sommet s, on met à jour $distance[v] = distance[s] + 1$.

→ largeur

Détecter si G a un cycle impliquant dep :

→ On regarde si au moment d'explorer les voisins d'un sommet s, un de ces voisins est dep, en conservant le sommet d'où l'exploration de s a été lancée (si c'est dep ce n'est pas un cycle).

Biparti :

Au départ, on place dep dans une partition quelconque.

Quand on explore les voisins v de s :

- Si v est déjà dans une partition, on vérifie que ce n'est pas celle de s.
- Sinon, on attribue à v la partition opposée.

Tri topologique :

Définition :

Un ordre topologique sur un graphe orienté G est une relation d'ordre totale $\prec$ sur les sommets de G telle que si $x \to y$ est un arc, alors $x \prec y$.

Propriété :

Un graphe orienté admet un ordre topologique si et seulement s'il est acyclique.

Définition :

Faire un tri topologique d'un graphe orienté acyclique consiste à trouver une énumération des sommets qui respecte l'ordre topologique.

Exemple :

Chaussettes $\prec$ pantalon $\prec$ chaussures $\prec$ chemise $\prec$ montre $\prec$ cravate $\prec$ veste $\prec$ ceinture