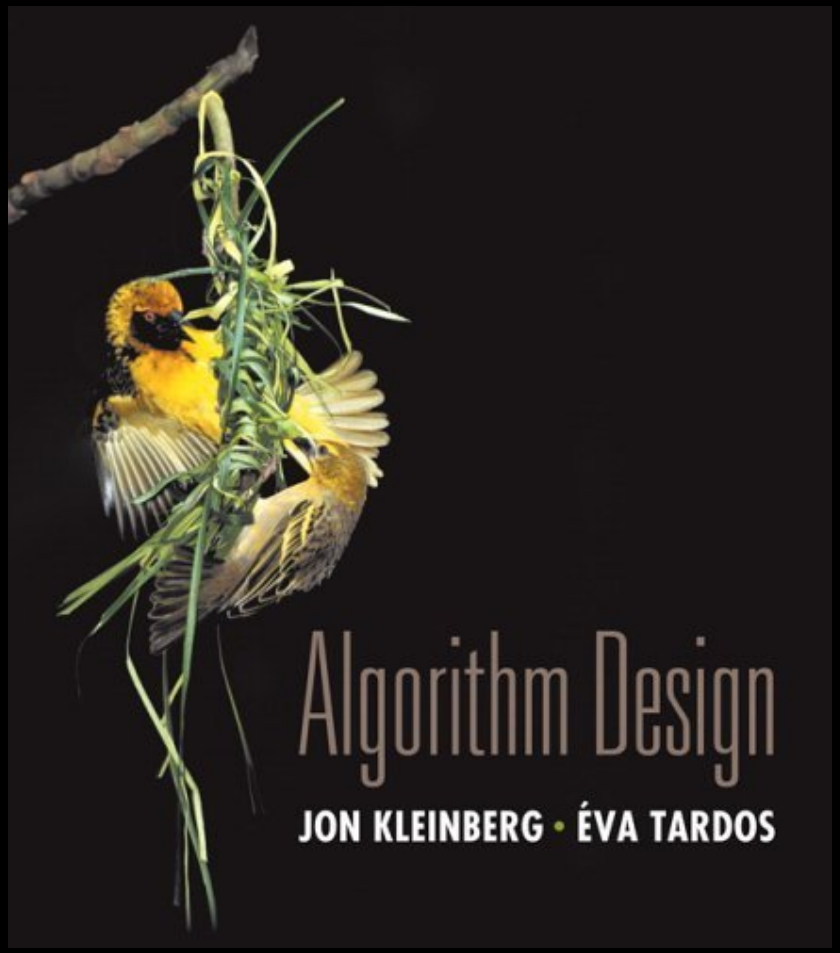


Chapitre 3

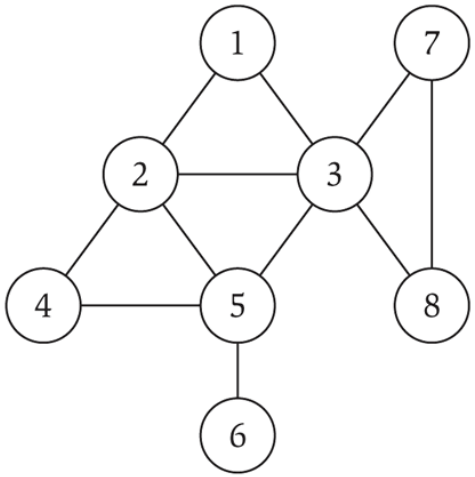
Graphes : Définitions et Algorithmes de Base

Définitions de base et applications

Graphes non orientés (undirected)

Graphe non orienté. $G = (V, E)$

- V = noeuds (nodes) ou sommets (vertices).
- E = arêtes. Une arête relie deux sommets.
- Relation binaire symétrique.
- Tailles : $n = |V|$, $m = |E|$.



V = { 1, 2, 3, 4, 5, 6, 7, 8 }

E = { 1-2, 1-3, 2-3, 2-4, 2-5, 3-5, 3-7, 3-8, 4-5, 5-6 }

n = 8

m = 11

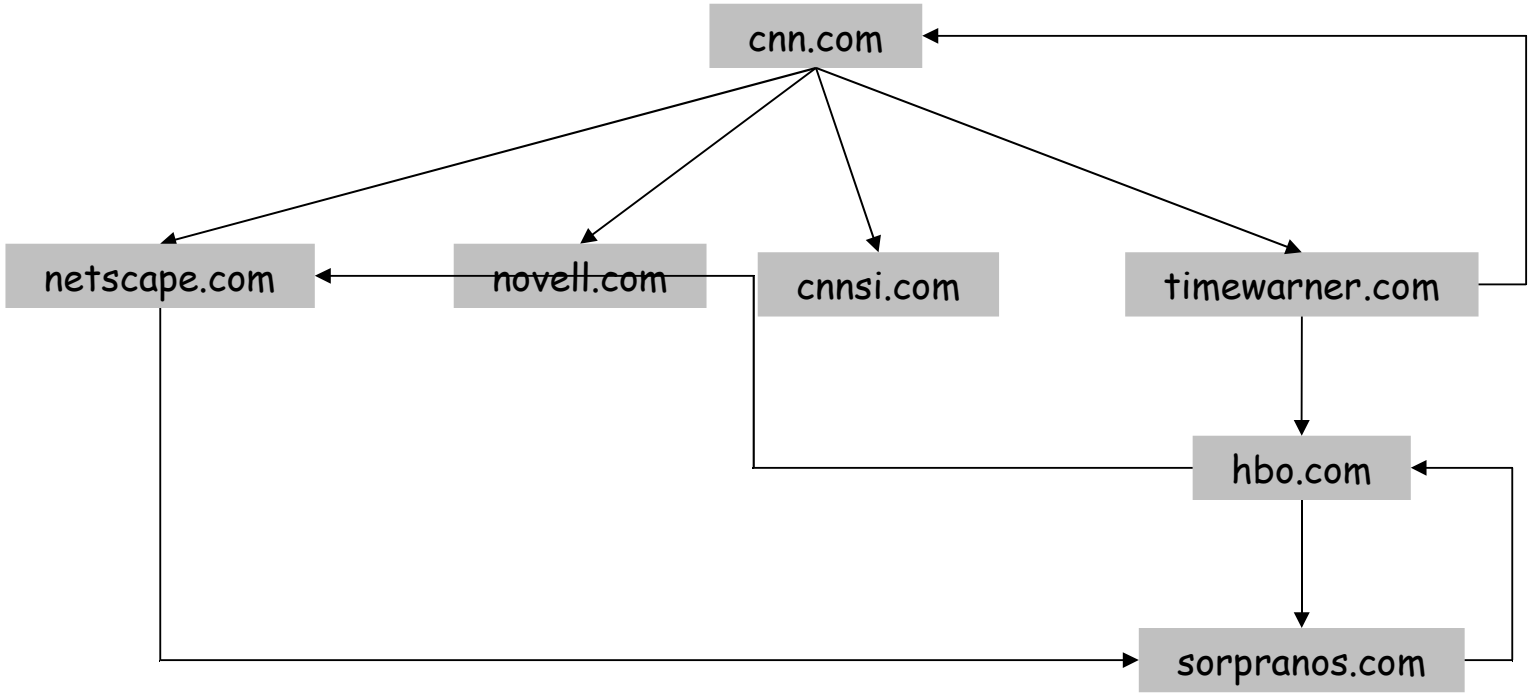
Quelques Applications

Graph	*Nodes*	*Edges*
transportation	street intersections	highways
communication	computers	fiber optic cables
World Wide Web	web pages	hyperlinks
social	people	relationships
food web	species	predator-prey
software systems	functions	function calls
scheduling	tasks	precedence constraints
circuits	gates	wires

World Wide Web

Graphe (orienté) du Web.

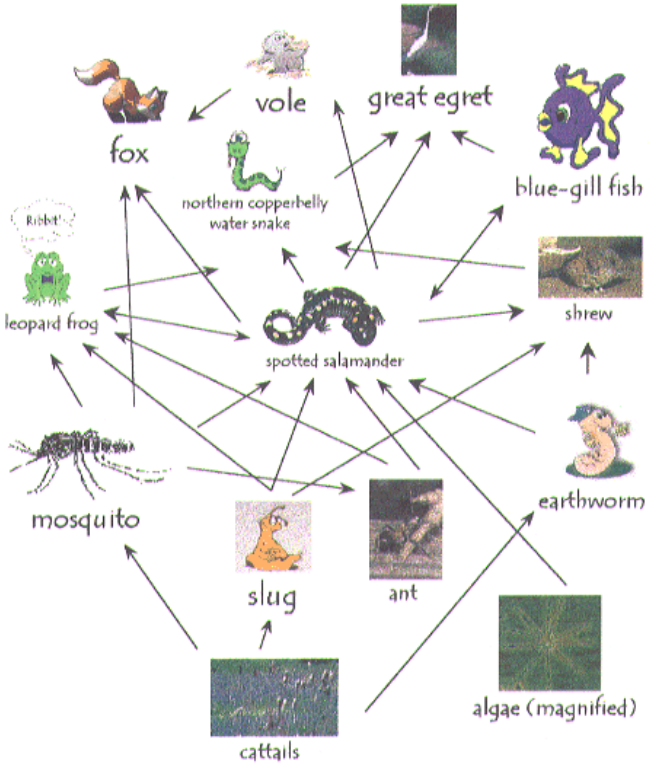
- Sommet : page web.
- Arête orientée (ou arc) : hyperlien d'une page vers une autre.



Chaîne alimentaire

Food web graph.

- Sommet = espèce.
- Arc = de la proie vers le prédateur.

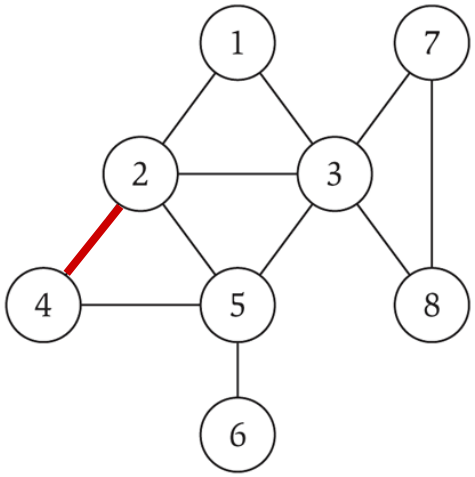


Reference: http://www.twingroves.district96.k12.il.us/Wetlands/Salamander/SalGraphics/salfoodweb.giff

Représentation d'un graphe : matrice d'adjacence

Matrice d'adjacence. matrice nxn telle que $A_{uv} = 1$ si (u, v) est une arête.

- Graphe non orienté : chaque arête est représentée deux fois.
- Espace requis par la représentation proportionnel à n^2.
- Vérifier si (u, v) est une arête prend un temps $\Theta(1)$.
- Identifier toutes les arêtes prend un temps $\Theta(n^2)$ time.



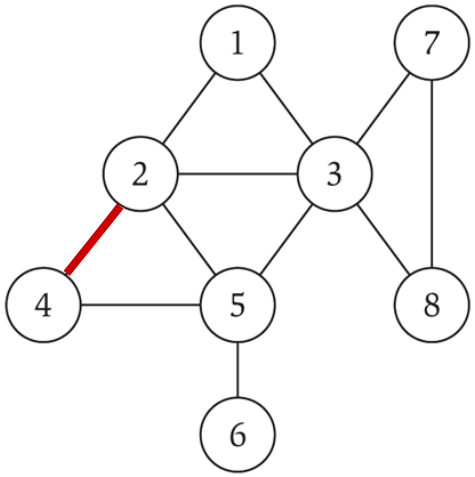
	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	1	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

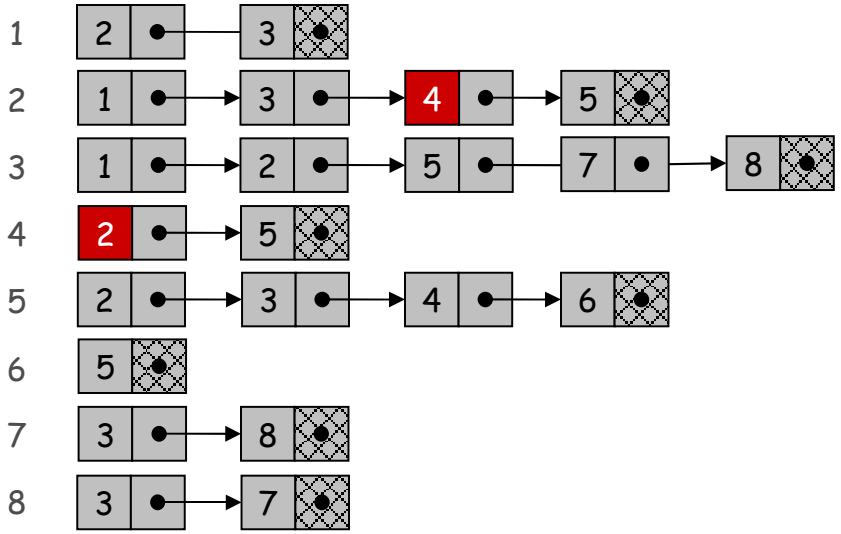
Représentation d'un graphe : listes d'adjacence

Listes d'adjacence. Tableau (indicés par les sommets) de listes.

- Graphe non orienté : chaque arête est représentée deux fois.
- Espace requis par la représentation proportionnel à m + n.
- Vérifier si (u, v) est une arête prend un temps $O(\deg(u))$.
- Identifier toutes les arêtes prend un temps $\Theta(m + n)$ time.

degré(v) = nombre de "voisins" de v



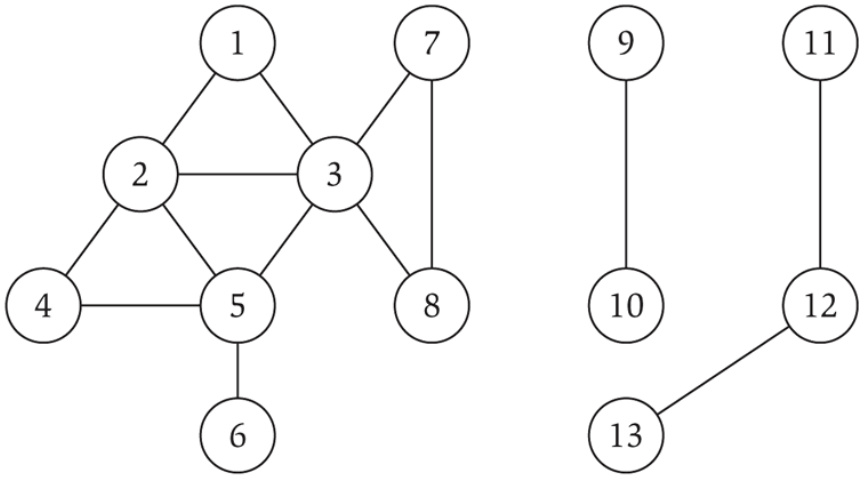


Chemins et connexité

Déf. Un chemin (path) [ou une chaîne] dans un graphe non orienté $G = (V, E)$ est une suite P de sommets $v_1, v_2, \ldots, v_{k-1}, v_k$ telle que chaque paire $\{v_i, v_{i+1}\}$ est reliée par une arête de E.

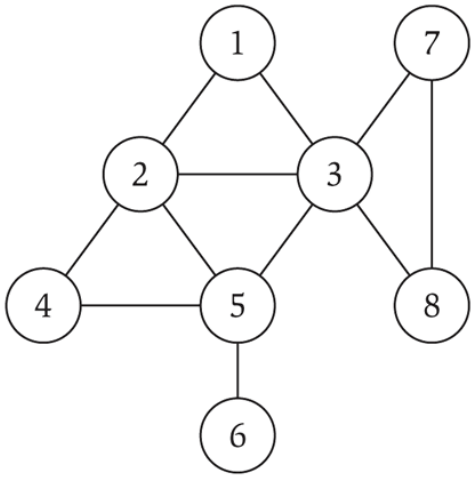
Déf. Un chemin est simple si tous les sommets sont distincts.

Déf. Un graphe orienté est connexe si pour chaque paire de sommets {u,v}, il existe une chemin reliant u à v.



Cycles

Déf. Un cycle est un chemin $v_1, v_2, \ldots, v_{k-1}, v_k$ dans lequel $v_1 = v_k$, $k > 2$, et les sommets $v_1, v_2, \ldots, v_{k-1}$ sont deux à deux distincts.



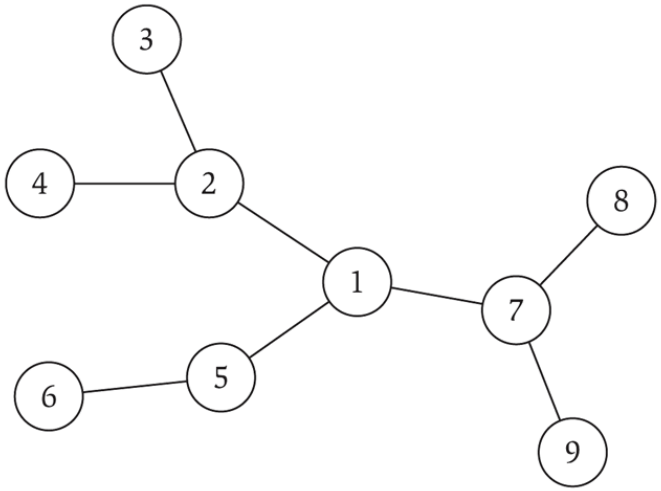
cycle C = 1-2-4-5-3-1

Arbres

Déf. Un graphe non orienté est un arbre s'il est connexe et ne contient aucun cycle.

Théorème. Soit G un graphe non orienté ayant n sommets. Chacun des énoncés suivants est impliqué par les deux autres.

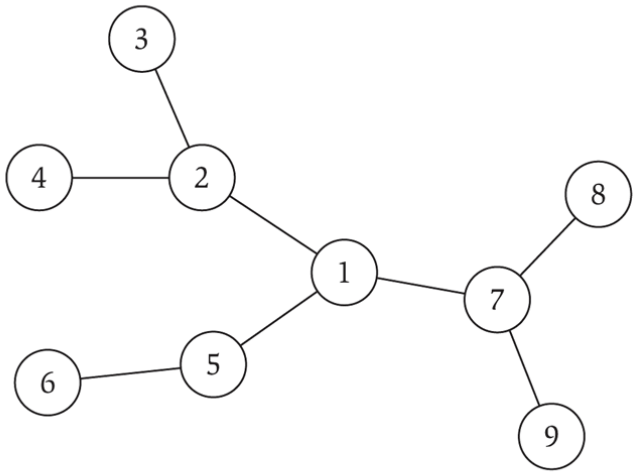
- G est connexe.
- G ne contient pas de cycle.
- G a n-1 arêtes.



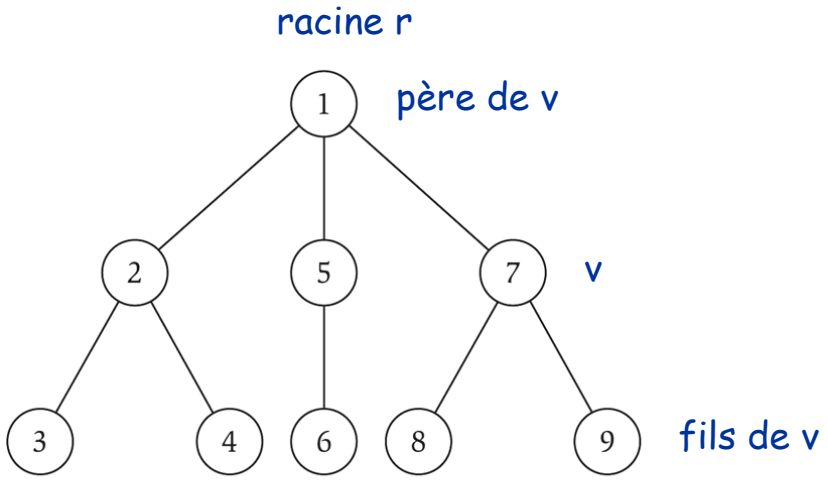
Arborescence (Rooted Trees)

Arbre enraciné. Etant donné un arbre T, on choisit un sommet comme racine et oriente chaque arête pour qu'elle s'éloigne de la racine.

Importance. Modélise des structures hiérarchiques.



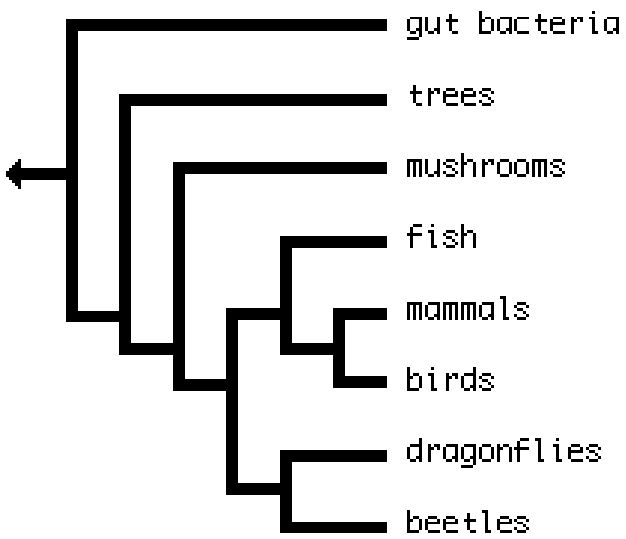
un arbre



Le même arbre, enraciné en 1

Exemple

Phylogeny trees. Décrivent certaines théories de l'évolution.



Parcours de graphes

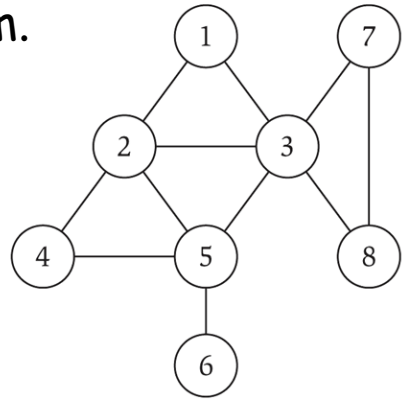
Connexité

Problème de la s-t connexité. Etant donnés deux sommets s et t, existe-t-il un chemin entre s et t ?

Problème du plus court chemin entre s et t. Etant donnés deux sommets s et t, quelle est la longueur du plus court chemin entre s et t ?

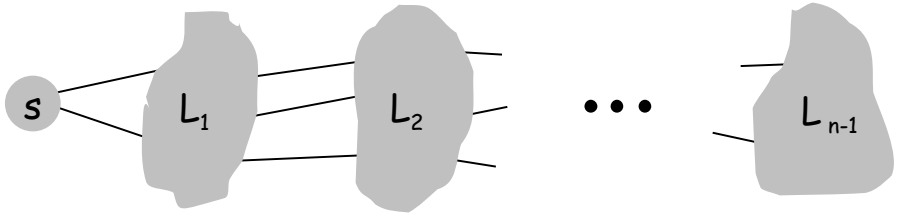
Applications.

- Friendster.
- Maze traversal.
- Kevin Bacon number.
- Nombre minimal de sauts dans un réseau de télécom.



Parcours en largeur (Breadth First Search)

Idée intuitive de BFS. Explorer toutes les directions à partir de s, en traitant les sommets "niveau par niveau".



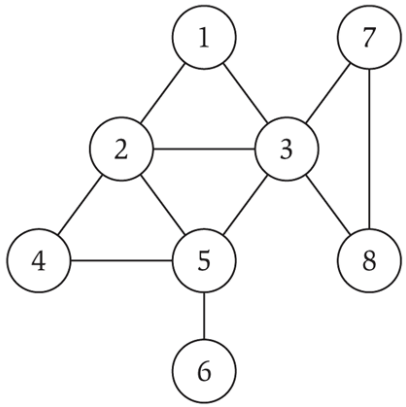
Algorithme BFS.

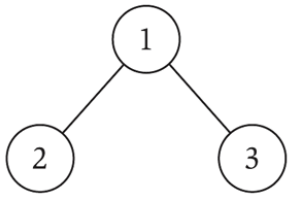
- $L_0 = \{ s \}$.
- L_1 = tous les voisins de L_0.
- L_2 = tous les sommets qui n'appartiennent ni à L_0 ni à L_1, et qui sont adjacents à un sommet de L_1.
- L_{i+1} = tous les sommets qui n'appartiennent à aucun des Lj, $j \leq i$, et qui sont adjacents à un sommet de L_i.

Théorème. Pour chaque i, L_i est composé des sommets qui sont exactement à distance i de s. Il existe un chemin de s à t ssi t apparaît dans un L_i.

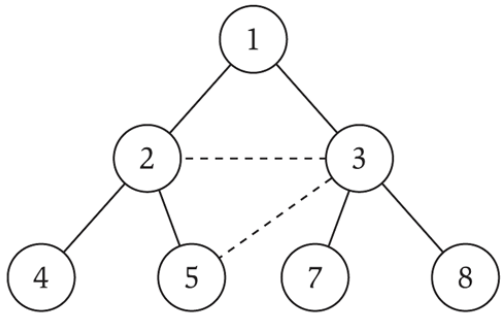
Parcours en largeur

Propriété. Soit T un arbre BFS de G = (V, E), et soit (x, y) une arête de G. Alors les niveaux respectifs de x et y diffèrent au plus de 1.

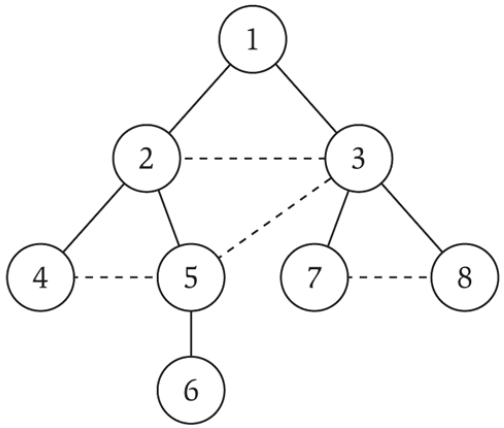




(a)



(b)



(c)

L_0

L_1

L_2

L_3

Analyse du parcours en largeur

Théorème. L'exécution de BFS coûte un temps $O(m + n)$ si le graphe est représenté par ses listes d'adjacence.

Preuve.

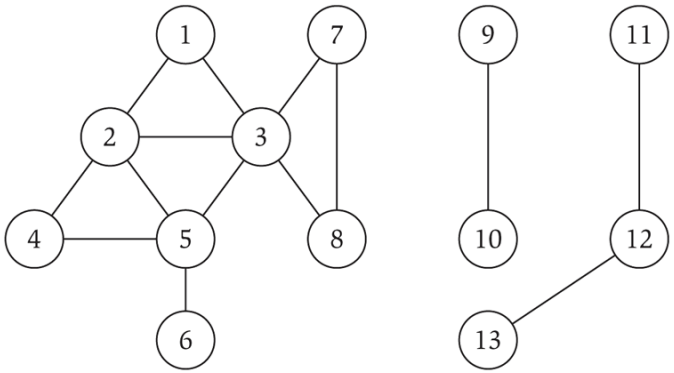
- Il est facile de montrer que le temps est $O(n^2)$:
  - il y a au plus n listes L[i]
  - chaque sommet apparaît dans au plus une liste L[i] ; la boucle **`for`** est exécutée au plus n fois
  - il y a au plus n arêtes incidentes à u
    et le traitement d'un arête coûte un temps $O(1)$.

- Pour obtenir la borne $O(m + n)$, il suffit de remarquer que :
  - il y exactement deg(u) arêtes (u,v) incidentes à u
  - le coût total du traitement des arêtes est $\Sigma_{u \in V} \deg(u) = 2m$ ▪

chaque arête (u, v) est comptée exactement deux fois
dans la somme : une fois dans deg(u) et une fois dans deg(v)

Composante connexe

Composante connexe. Trouver tous les sommets reliés à s.



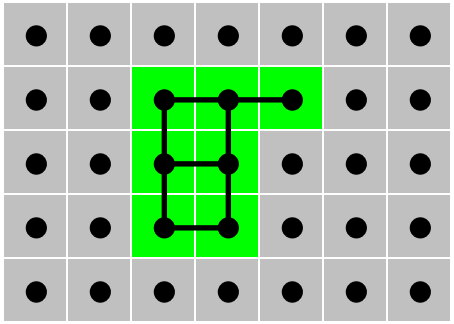
Composante connexe du sommet 1 = { 1, 2, 3, 4, 5, 6, 7, 8 }.

Remplissage (Flood Fill)

Flood fill. Etant donné un pixel vert, recolorier en bleu toute la zone de couleur verte autour du pixel.

- Sommet : pixel.
- Arête : entre deux pixels voisins de même couleur.
- Zone : composante connexe.

recolorier la zone verte en bleu

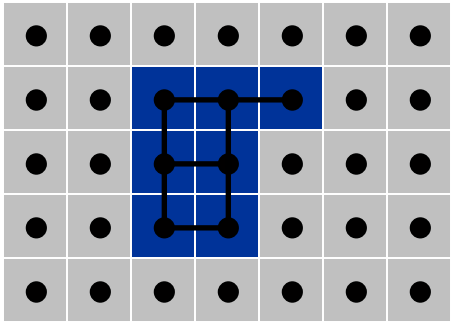


Remplissage (Flood Fill)

Flood fill. Etant donné un pixel vert, recolorier en bleu toute la zone de couleur verte autour du pixel.

- Sommet : pixel.
- Arête : entre deux pixels voisins de même couleur.
- Zone : composante connexe.

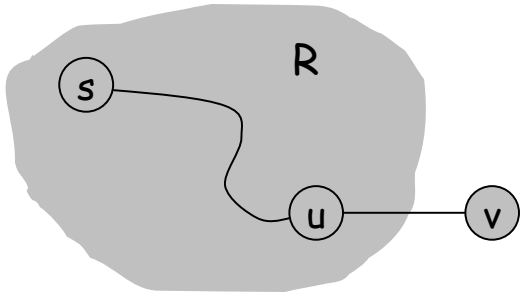
recolorier la zone verte en bleu



Composante connexe

Composante connexe. Trouver tous les sommets relier à s.

R will consist of nodes to which s has a path
Initially $R = \{s\}$
While there is an edge (u, v) where $u \in R$ and $v \notin R$
  Add v to R
Endwhile



On peut ajouter v sans danger

Théorème. A la fin de l'exécution, R est la composante connexe de s.

- BFS = exploration selon les distances croissantes depuis s.
- DFS = exploration selon une méthode différente.

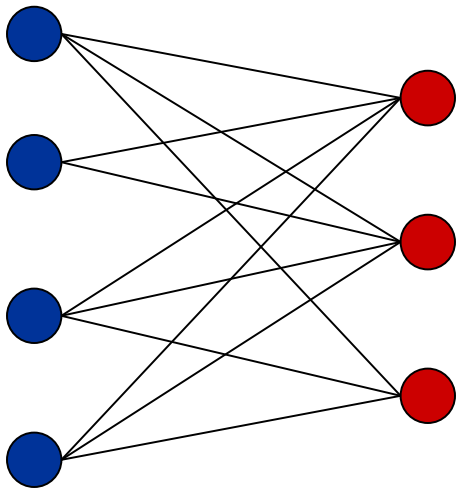
Tester la bipartionnabilité

Graphes bipartis

Déf. Un graphe non orienté $G = (V, E)$ est biparti si on peut partitionner les sommets $V = V_1 \cup V_2$ de sorte que toutes les arêtes aient une extrémité dans V_1 et l'autre dans V_2.

Applications.

- matching stable : hommes = V_1, femmes = V_2.
- Scheduling : machines = V_1, tâches = V_2.

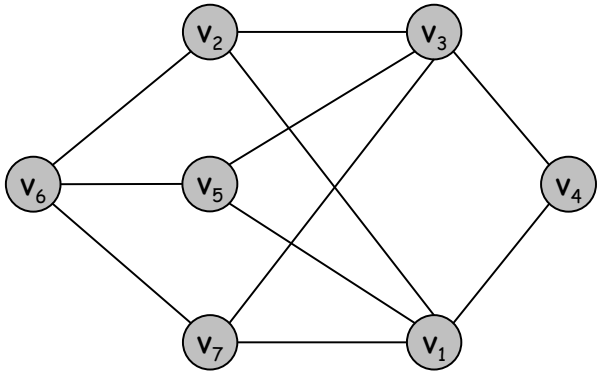


un graphe biparti

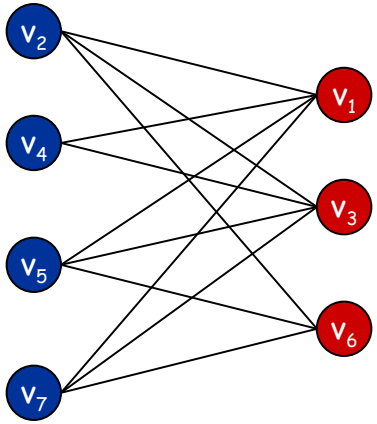
Tester la bipartitionnabilité

Tester la bipartitionnabilité. Etant donné un graphe G, est-il biparti ?

- De nombreux problèmes de graphes deviennent :
  - plus simples si le graphes sous-jacent est biparti (ex : matching)
  - traitables si le graphe sous-jacent est biparti (ex : sous-ensemble indépendant)
- Avant d'écrire un algorithme, il faut comprendre la structure des graphes bipartis.



un graphe biparti G

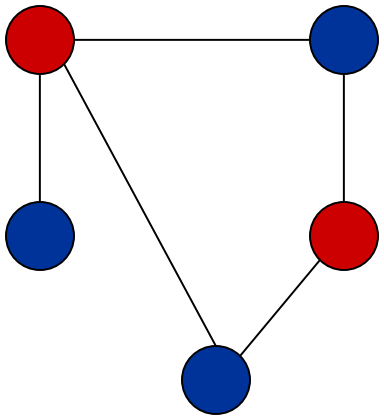


une autre représentation de G

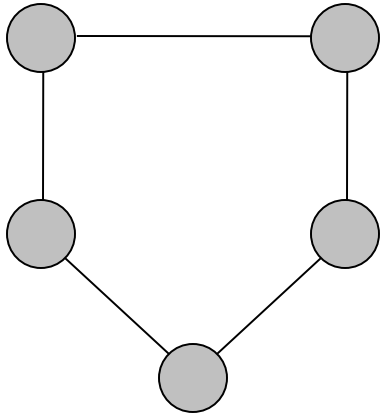
Une obstruction à la bipartitionnabilité

Lemme. Si un graphe G est biparti, alors il ne peut pas contenir un cycle de longueur impaire.

Preuve. Il est impossible de colorier un cycle de longueur impaire sans qu'une arête ait ses deux extrémités de même couleur.



biparti
(2-colorable)



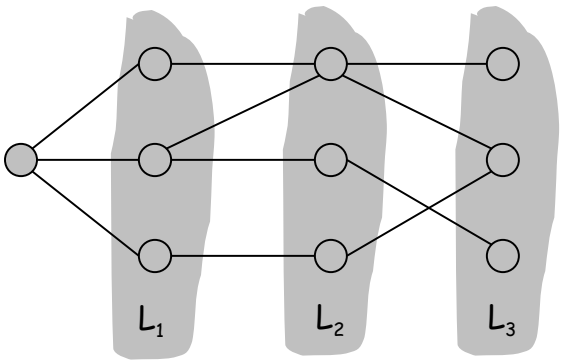
non biparti
(n'est pas 2-coloriable)

Graphes bipartis

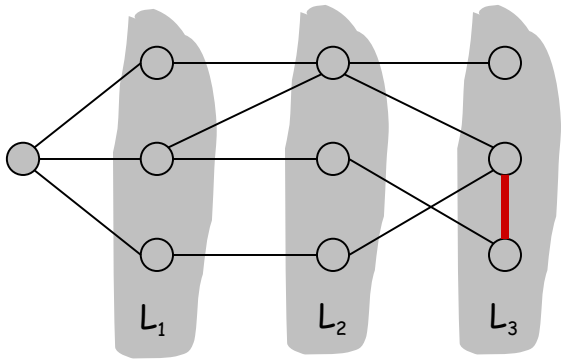
Lemme. Soit G un graphe connexe, et $L_0, \ldots, L_k$ les niveaux construits par BFS en partant du sommet s. Un et un seul des énoncés suivants est vrai.

(i) Aucune arête de G ne joint deux sommets de même niveau, et alors G est biparti.

(ii) Une arête de G joint deux sommets de même niveau, et alors G contient un cycle de longueur impaire (et n'est donc pas biparti.)



Cas (i)



Cas (ii)

Graphes bipartis

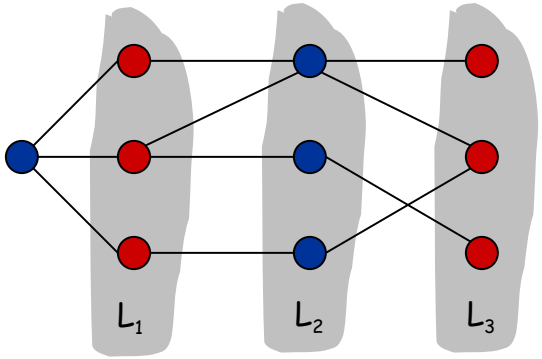
Lemme. Soit G un graphe connexe, et L_0, …, L_k les niveaux construits par BFS en partant du sommet s. Un et un seul des énoncés suivants est vrai.

(i) Aucune arête de G ne joint deux sommets de même niveau, et alors G est biparti.

(ii) Une arête de G joint deux sommets de même niveau, et alors G contient un cycle de longueur impaire (et n'est donc pas biparti.)

Preuve. (i)

- Supposons qu'aucune arête ne joint deux sommets de même niveau.
- Cela implique que toutes les arêtes relient des niveaux successifs.
- Bipartition : rouge = niveaux impairs, bleu = niveaux pairs.



Case (i)

Graphes bipartis

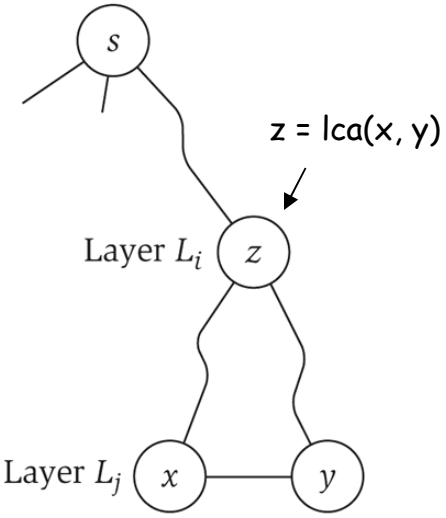
Lemme. Soit G un graphe connexe, et $L_0, \ldots, L_k$ les niveaux construits par BFS en partant du sommet s. Un et un seul des énoncés suivants est vrai.

(i) Aucune arête de G ne joint deux sommets de même niveau, et alors G est biparti.

(ii) Une arête de G joint deux sommets de même niveau, et alors G contient un cycle de longueur impaire (et n'est donc pas biparti.)

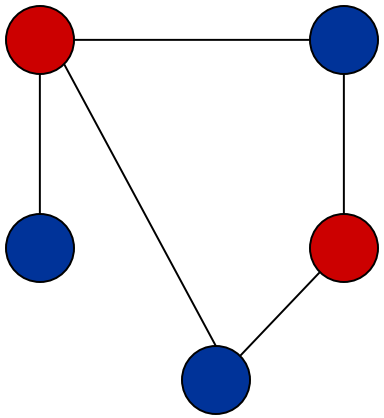
Preuve. (ii)

- Supposons que (x, y) soit une arête, avec x, y dans L_j.
- Soit z = lca(x, y) = premier ancêtre commun.
- Soit L_i le niveau de z.
- Considérons le cycle composé de l'arête x-y, du chemin de y à z, et du chemin de z à x.
- Sa longueur est $\underbrace{1}_{(x,y)} + \underbrace{(j-i)}_{\text{chemin de y à z}} + \underbrace{(j-i)}_{\text{chemin de z à x}}$, qui est impaire.

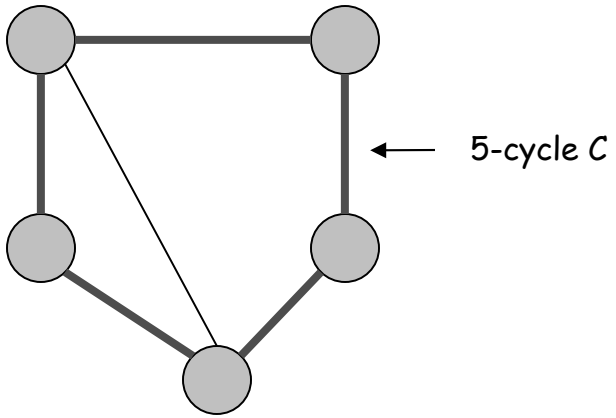


Obstruction à la bipartitionnabilité

Corollaire. Un graphe G est biparti ssi il ne contient aucun cycle de longueur impaire.



biparti
(2-coloriable)



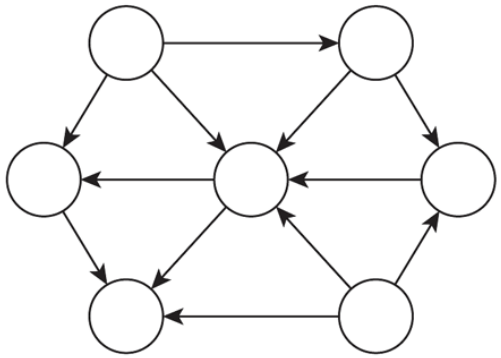
non biparti
(n'est pas 2-coloriable)

Connexité dans les graphes orientés

Graphes orientés

Graphe orienté. $G = (V, E)$

- L'arc (u, v) part de u et aboutit à v.



Ex. Graphe du Web - hyperliens d'une page web vers une autre.

- L'orientation du graphe est cruciale.
- Les moteurs de recherche exploitent la structure d'hyperliens pour classer les pages web par ordre d'importance.

Parcours de graphe

Accessibilité. Etant donné un sommet s, trouver tous les sommets accessibles à partir de s.

Plus court chemin de s à t. Etant donnés deux sommets s et t, quelle est la longueur du plus court chemin de s à t ?

Exploration. BFS s'étend naturellement aux graphes orientés.

Web crawler. On part d'une page web s. Trouver toutes les pages qui référencent s, directement ou indirectement.

Connexité forte

Déf. Eux sommets u et v sont mutuellement accessibles (l'un à partir de l'autre) s'il existe un chemin de u à v et un chemin de v à u.

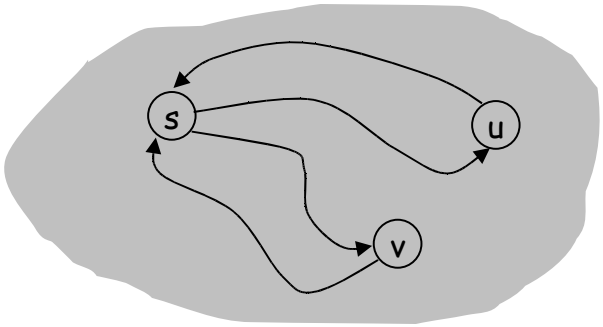
Déf. Un graphe est fortement connexe si pour toute paire de sommets {u,v}, u et v sont mutuellement accessibles.

Lemme. Soit s un sommet. G est fortement connexe ssi chaque sommet est accessible depuis s, et s est accessible depuis tout sommet.

Preuve $\Rightarrow$ Découle de la définition.

Preuve $\Leftarrow$ Chemin de u à v := chemin u-s puis chemin s-v.

Chemin de v à u := chemin v-s puis chemin s-u. ▪



Valable même si les chemins ne sont pas disjoints

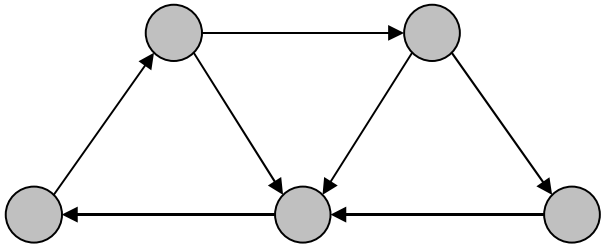
Connexité forte : algorithme

Théorème. On peut décider si G est fortement connexe en temps O(m + n).

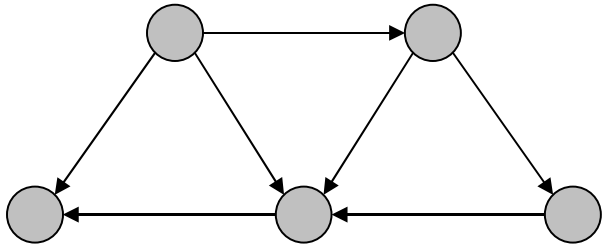
Preuve.

- Choisir un sommet arbitraire s.
- Explorer G par BFS à partir de s.
- Explorer G^{rev} par BFS à partir de s.
- Retourner "vrai" ssi tous les sommets sont atteints dans les deux appels à BFS.
- La correction découle du précédent lemme. ▪

G^{rev} est obtenu en changeant l'orientation de toutes les arêtes de G



fortement connexe



n'est pas fortement connexe

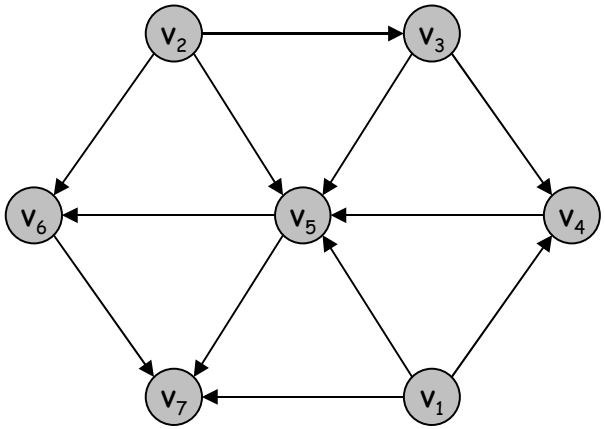
DAGs et ordre topologique

DAG (Directed Acyclic Graph)

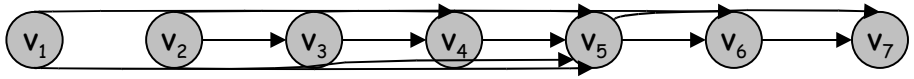
Déf. Un DAG est un graphe orienté qui ne contient pas de circuit (i.e. pas de cycle orienté).

Ex. Contraintes de précédence : un arc (v_i, v_j) si v_i doit précéder v_j.

Déf. Un ordre topologique sur un graphe orienté $G = (V, E)$ is un ordre total sur V, $v_1, v_2, \ldots, v_n$ tel que : si (v_i, v_j) alors $i < j$.



un DAG



un ordre topologique

Contraintes de précédence

Contraintes de précédence. Un arc (v_i, v_j) si la tâche v_i doit précéder la tâche v_j.

Applications.

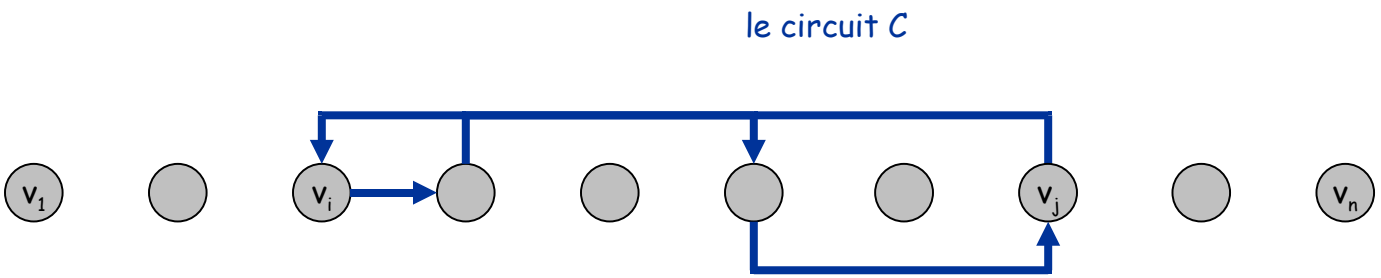
- Compilation : le module v_i doit être compilé avant le module v_j.
- Pipeline de tâches : la connaissance de l'output de la tâche v_i requise pour déterminer l'input de la tâche v_j.

DAG (Directed Acyclic Graph)

Lemme. Si G admet un ordre topologique, alors G est un DAG.

Preuve. (par l'absurde)

- Supposons que $v_1, \ldots, v_n$ soit un ordre topologique sur G et que G contienne un cycle orienté C.
- Soit v_i le premier sommet de C selon l'ordre topologique, et v_k le prédécesseur de v_i dans C ; (v_k, v_i) est donc un arc de G.
- Par définition de v_i, on a $i < k$.
- D'autre part, puisque (v_k, v_i) est un arc et $v_1, \ldots, v_n$ est un ordre topologique, on a $k < i$, d'où a contradiction. ▪



l'ordre topologique supposé : $v_1, \ldots, v_n$

DAG (Directed Acyclic Graph)

Lemme. Si G admet un ordre topologique, alors G est un DAG.

Q. Tout DAG admet-il un ordre topologique ?

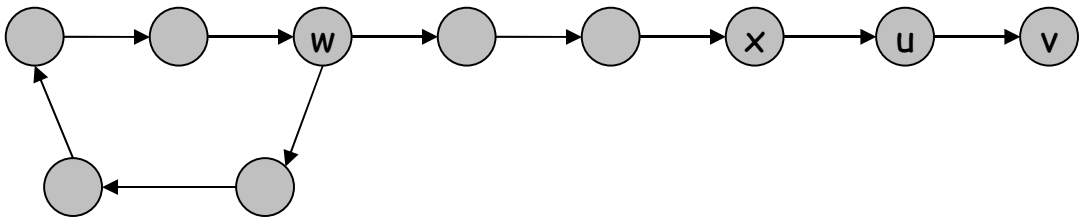
Q. Si oui, comment en calculer un ?

DAG (Directed Acyclic Graph)

Lemme. Si G est un DAG, alors G a un somet de degré entrant nul.

Preuve. (par l'absurde)

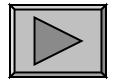
- Supposons que G soit un DAG et que chaque sommet soit l'extrémité d'au moins un arc.
- Soit v un sommet. Puisque v a au moins un arc entrant (u, v) on peut remonter de v à u.
- Ensuite, puisque u a au moins un arc entrant (x, u), on peut remonter de u à x.
- On répète le procédé jusqu'à repasser par un sommet déjà visité, soit w.
- La suite de sommets visités entre deux passages successifs par w constitue un circuit, d'où la contardiction. ▪



DAG (Directed Acyclic Graph)

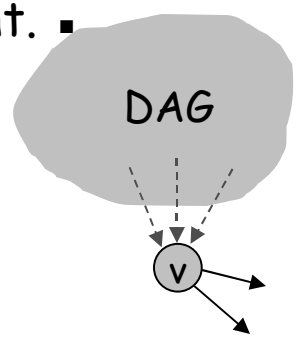
Lemme. Si G est un DAG, alors G admet un ordre topologique.

Preuve. (par induction sur le nombre n de sommets)



- Cas de base : vrai si n = 1.
- Etant donné un DAG sur n > 1 sommets, on choisit un sommet v with no sans arc entrant.
- G - { v } est un DAG, car la suppression de v ne crée pas de circuit.
- Par hypothèse d'induction, G - { v } admet un ordre topologique.
- En posant que v est plus petit que les tous les sommets de G - { v }, eux-mêmes ordonnés selon leur ordre topologique, on obtient un ordre topologique sur les sommets de G, car v n'a pas d'arc entrant. ▪

To compute a topological ordering of G:
Find a node v with no incoming edges and order it first
Delete v from G
Recursively compute a topological ordering of $G-\{v\}$
  and append this order after v



Analyse de l'algorithme de construction d'un ordre topologique

Théorème. L'algorithme ci-dessus construit un ordre topologique en temps $O(m + n)$.

Preuve.

- On "maintient" l'information suivante :
  - `count[w]` = nombre résiduel d'arcs entrants
  - S = ensemble des sommets sans arc entrant (parmi ceux qui n'ont pas été supprimés)
- Initialisation : $O(m + n)$ par simple balayage du graphe.
- Mise à jour : pour supprimer v
  - ôter v de S
  - Pour chaque arc (v,w), décrémenter `count[w]`, et ajouter w à S si `count[w]=0`
  - Cela coûte $O(\deg(v))$ par sommet. ▪