DIU EIL bloc 5

Graphes - Définitions, algorithmes élémentaires

Laure Gonnord

http://laure.gonnord.org/pro/

Laure.Gonnord@univ-lyon1.fr

DIU EIL, Dpt Info UCBL

2019-2020

Crédits : O. Bournez pour Polytechnique : définition, parcours ; avec son aimable autorisation.

Extraits du programme SNT Seconde

- **Les algorithmes et les programmes**

De très nombreux algorithmes sont mis en œuvre par les applications de réseautage social.

Toutes les applications s'appuient sur des services de mise en relation avec des internautes membres du réseau, relations ou amis communs : des algorithmes opérant sur les **graphes** et sur les bases de données sont au cœur de ces services.

À l'aide d'algorithmes de recommandation, les réseaux sociaux suggèrent aux utilisateurs des amis, des contenus, des annonces promotionnelles. Ils permettent aussi aux plateformes sociales d'étudier les comportements de leurs utilisateurs à des fins commerciales, politiques ou d'amélioration du service.

Rayon, diamètre et centre d'un graphe	Déterminer ces caractéristiques sur des graphes simples.

Exemples d'activités
- Construire ou utiliser une représentation du graphe des relations d'un utilisateur. S'appuyer sur la densité des liens pour identifier des groupes, des communautés. - Sur des exemples de graphes simples, en informatique débranchée, étudier les notions de rayon, diamètre et centre d'un graphe, de manière à illustrer la notion de « petit monde ».

FIGURE – Extraits du programme de SNT

Extrait du programme de NSI Terminale

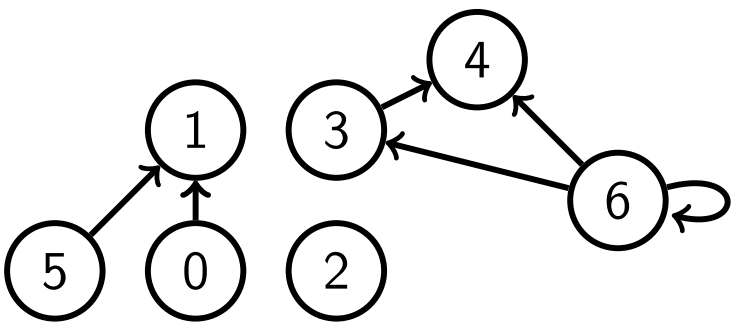
Graphes : structures relationnelles. Sommets, arcs, arêtes, graphes orientés ou non orientés.	Modéliser des situations sous forme de graphes. Écrire les implémentations correspondantes d'un graphe : matrice d'adjacence, liste de successeurs/de prédécesseurs. Passer d'une représentation à une autre.	On s'appuie sur des exemples comme le réseau routier, le réseau électrique, Internet, les réseaux sociaux. Le choix de la représentation dépend du traitement qu'on veut mettre en place : on fait le lien avec la rubrique « algorithmique ».
Algorithmes sur les graphes.	Parcourir un graphe en profondeur d'abord, en largeur d'abord. Repérer la présence d'un cycle dans un graphe. Chercher un chemin dans un graphe.	Le parcours d'un labyrinthe et le routage dans Internet sont des exemples d'algorithme sur les graphes. L'exemple des graphes permet d'illustrer l'utilisation des classes en programmation.

FIGURE – Extraits du programme de NSI Terminale sur les graphes

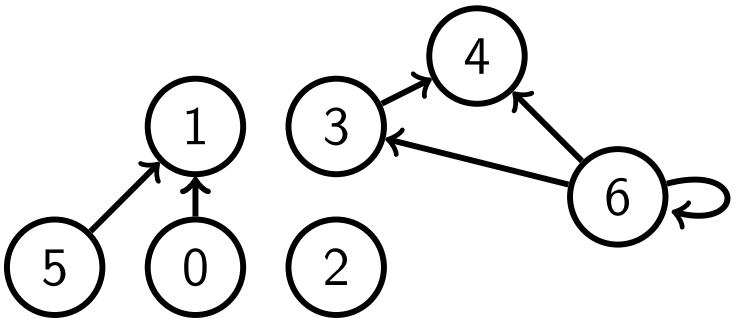
Un graphe orienté

- Un *graphe orienté* (digraph) est donné par un couple $G = (V, E)$, où
  - V est un ensemble.
  - $E \subset V \times V$.

- Exemple:
  - $V = \{0, 1, \ldots, 6\}$.
  - $E = \{(0,1), (3,4), (5,1), (6,3), (6,4), (6,6)\}$.



Vocabulaire

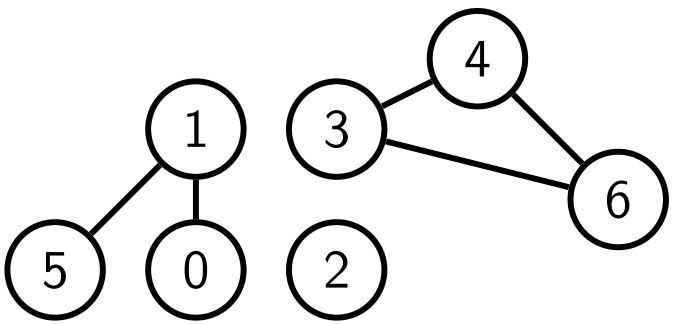


- Les éléments de V sont appelés des *sommets* (parfois aussi des *nœuds*).
- Les éléments e de E sont appelés des *arcs*.
- Si $e = (u, v)$, u est appelé *la source* de e, v est appelé *la destination* de e.
- Remarque:
  - Les boucles (les arcs (u, u)) sont autorisées.

Un graphe

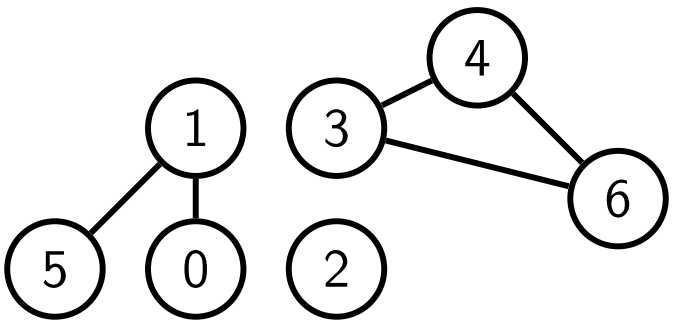
- Un *graphe*[1] est donné par un couple $G = (V, E)$, où
  - V est un ensemble.
  - E est un ensemble de paires $\{u, v\}$ avec $u, v \in V$.
- On convient de représenter une paire $\{u, v\}$ par (u, v) ou (v, u).
- Autrement dit, (u, v) et (v, u) dénotent la même arête.

- Exemple:
  - $V = \{0, 1, \ldots, 6\}$
  - $E = \{(0, 1), (3, 4), (5, 1), (6, 3), (6, 4)\}$.



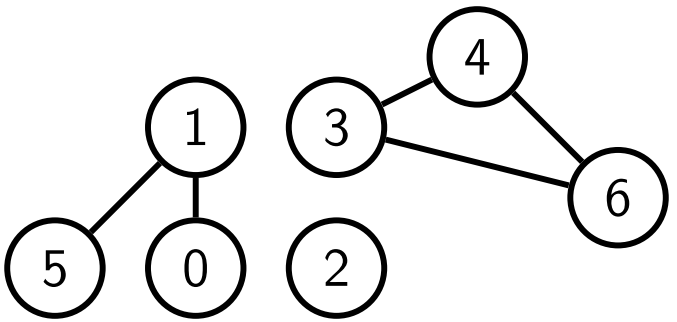
[1] Lorsqu'on ne précise pas, par défaut, un graphe est non-orienté.

Vocabulaire



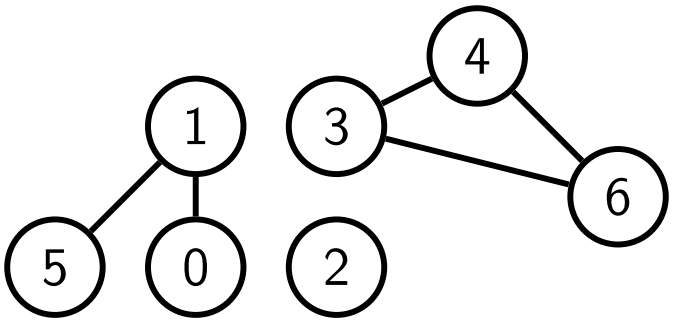
- Les éléments e de E sont appelés des *arêtes*. Si $e = (u, v)$, u et v sont appelés les *extrémités* de e.
- Remarque: (sauf autre convention explicite)
  - Les boucles ne sont pas autorisées.

Vocabulaire (cas non-orienté)



- u et v sont dits *voisins* s'il y a une arête entre u et v.
- Le degré de u est le nombre de voisins de u.

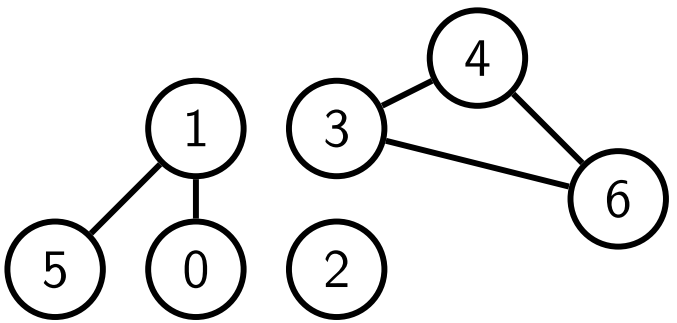
Vocabulaire (cas non-orienté)



- Un *chemin du sommet s vers le sommet t* est une suite $e_0, e_1, \cdots, e_n$ de sommets telle que $e_0 = s$, $e_n = t$, $(e_{i-1}, e_i) \in E$, pour tout $1 \leq i \leq n$.
  - n est appelé la *longueur* du chemin, et on dit que t *est joignable* à partir de s.
  - Le chemin est dit *simple* si les e_i sont distincts deux-à-deux.
  - Un *cycle* est un chemin de longueur non-nulle avec $e_0 = e_n$.
- Le sommet s est dit *à distance n* de t s'il existe un chemin de longueur n entre s et t, mais aucun chemin de longueur inférieure.

Composantes connexes (cas non-orienté)

- Prop. La relation "être joignable" est une relation d'équivalence.
- Les classes d'équivalence sont appelées les *composantes connexes*.



- Un graphe est dit *connexe* s'il n'y a qu'une seule classe d'équivalence.
  - Autrement dit, tout sommet est joignable à partir de tout sommet.

Les graphes sont partout!

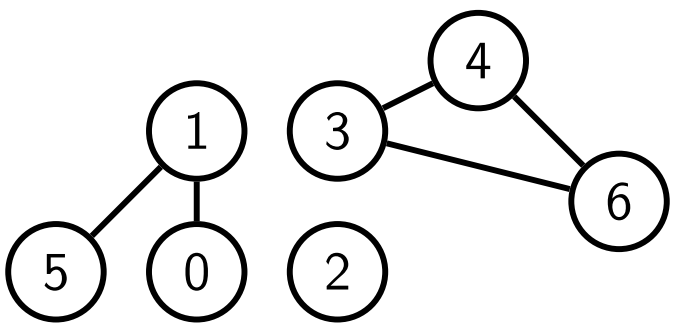
- Beaucoup de problèmes se modélisent par des objets et des relations entre objets.
- Exemples:
  - Le graphe routier.
  - Les réseaux informatiques.
  - Le graphe du web.
- Beaucoup de problèmes se ramènent à des problèmes sur les graphes.
- Théorie des graphes:
  - Euler, Hamilton, Kirchhoff, König, Edmonds, Berge, Lovász, Seymour,...
- Les graphes sont omniprésents en informatique.

Représentation des graphes: matrice d'adjacence

- Matrice d'ajacence. Pour $G = (V, E)$, $V = \{1, 2, \ldots, n\}$, on représente G par une matrice M $n \times n$.

$$M_{i,j} = \begin{cases} 1 & \text{si } (i,j) \in E \\ 0 & \text{sinon} \end{cases}$$

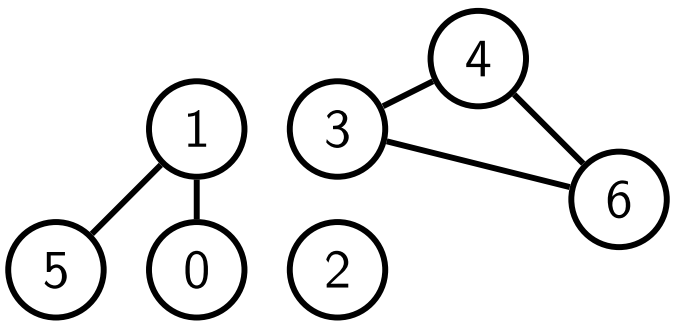
  - Si graphe valué: $M_{i,j} = v(i,j)$.



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

Représentation des graphes: liste de successeurs

- On associe à chaque sommet i, la liste des sommets j tels que $(i, j) \in E$.



- $L[0] = (1)$
- $L[1] = (0, 5)$
- $L[2] = ()$
- $L[3] = (4, 6)$
- $L[4] = (3, 6)$
- $L[5] = (1)$
- $L[6] = (3, 4)$

Meilleure représentation?

- Matrice: mémoire $O(n^2)$ (mieux pour graphes denses).
- Listes: mémoire $O(n + m)$ (mieux pour graphes creux).

- où n nombre de sommets, m nombre d'arêtes.

- Le mieux?: cela dépend du contexte.

Matrice vs. liste d'adjacence

Pour $G = (V, E)$:

- Liste d'adjacence : chacun connaît ses voisins.
- Matrice d'adjacence : toutes les transitions/distances directes.
- Complexités spatiales différentes.

Travail d'analyse essentiel !

Quelques trucs pour analyser

Lien entre $|E|$ et $|V|$?

- Toujours :

$$|E| \leqslant |V|^2$$

- Connexes :

$$|V| - 1 \leqslant |E|$$

- Planaires :

$$|E| \leqslant 3\,|V| - 6$$

Python, liste d'adjacence

Liste d'adjacence implémentée par un dictionnaire :

```
g = { "a" : ["d"],
      "b" : ["c"],
      "c" : ["b", "d", "e"],
      "d" : ["a", "c"],
      "e" : ["c"],
      "f" : []
}
```

Accéder à ma liste de voisins : `g["c"]`. ici les étiquettes (*label*) sont des chaînes, ce qui n'est pas forcément une bonne idée

Python, liste d'adjacence 2/2

Attention à l'ajout d'une arête dans le cas non-orienté :

```
if vertex1 in self.__graph_dict:
    self.__graph_dict[vertex1].append(vertex2)
else:
    self.__graph_dict[vertex1] = [vertex2]
if vertex2 in self.__graph_dict:
    self.__graph_dict[vertex2].append(vertex1)
else:
    self.__graph_dict[vertex2] = [vertex1]
```

Passage d'une représentation à une autre

À Implémenter (c'est au programme de Terminale !)

Parcours générique

ParcoursGenerique(G,s):

- $L := \{s\}$
- $B := Voisins(s)$
- Tant que $B \neq \emptyset$
  - choisir u dans B
  - $L := L \cup \{u\}$.
  - $B := B - \{u\}$.
  - $B := B \cup (Voisins(u) - L)$

Application 1: numérotation

ParcoursNumerotation(G,s):

- $num := 0$
- $numero[s] := num$; $num := num + 1$.
- $L := \{s\}$
- $B := Voisins(s)$
- Tant que $B \neq \emptyset$
  - choisir u dans B
  - $numero[s] := num$; $num := num + 1$.
  - $L := L \cup \{u\}$.
  - $B := B - \{u\}$.
  - $B := B \cup (Voisins(u) - L)$

Application 2: composantes connexes

UneComposante(G,s):

- $L := \{s\}$
- $B := Voisins(s)$
- Tant que $B \neq \emptyset$
  - choisir u dans B
  - $L := L \cup \{u\}$.
  - $B := B - \{u\}$.
  - $B := B \cup (Voisins(u) - L)$
- Retourner L

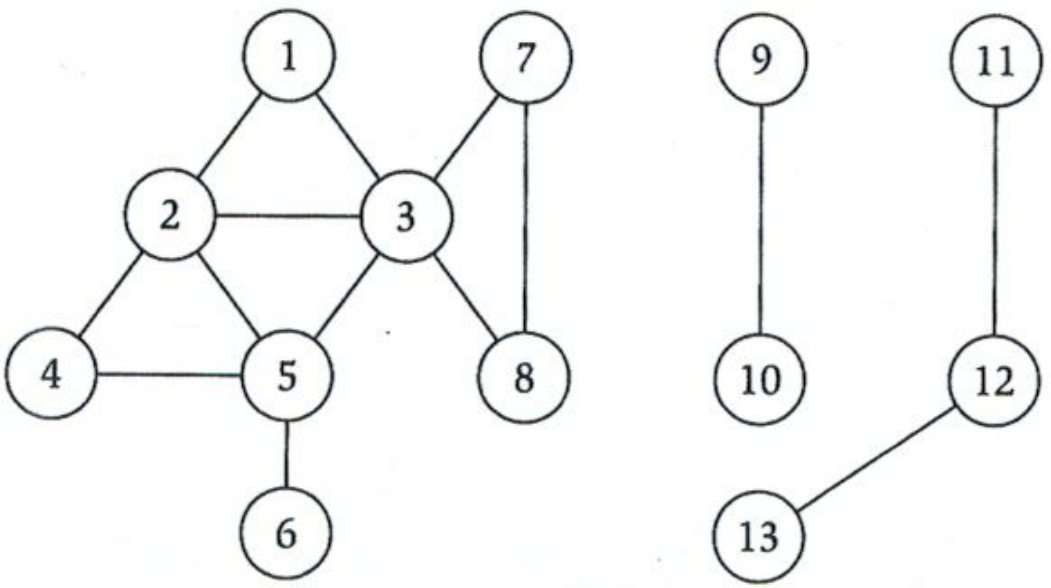
ComposantesConnexes(G)

- Tant que $V \neq \emptyset$
  - Choisir s dans V
  - $L := UneComposante(G, s)$
  - Afficher L
  - $V := V - L$

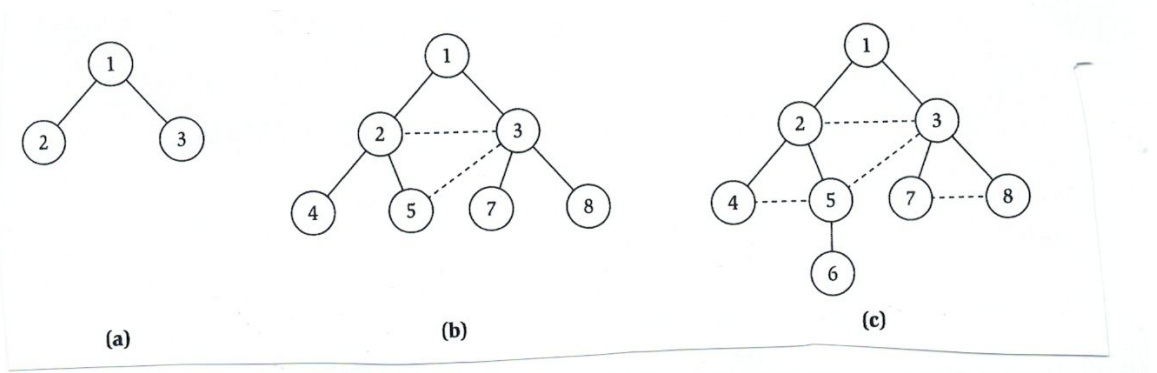
Plan

Parcours en largeur BFS

- B est implémenté par une file (FIFO = First In First Out).
- On explore les sommets par niveau: le niveau k correspond aux sommets à distance k du sommet s.
- Exemple:



BFS sur l'exemple précédent: (a), (b) et (c) montre les niveaux successifs qui sont considérés. Les sommets sont visités dans l'ordre $1, 2, 3, 4, 5, 7, 8, 6$.



- Pour chaque $k \geq 1$, le niveau k correspond aux sommets à distance exactement k de s.
- Il y a un chemin de s vers t si et seulement si t apparaît dans un certain niveau. Si t apparaît au niveau k, alors t est à distance k de s.

- Soit x et y deux sommets aux niveaux L_i et L_j avec $(x, y) \in E$. Alors i et j diffèrent d'au plus 1.

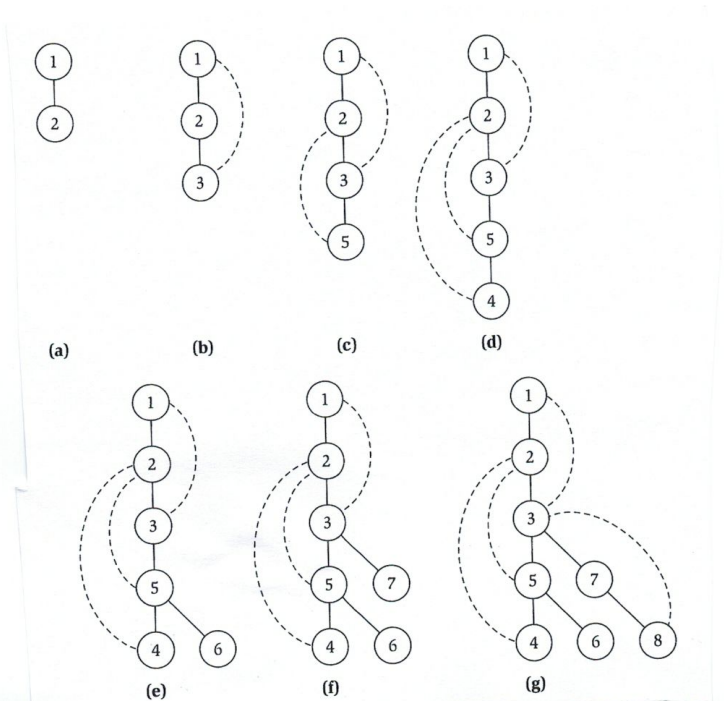
Plan

Parcours en profondeur DFS

DFS(s)

- Marquer s comme "exploré" et ajouter s à L.
- Pour chaque arête $(s, v) \in E$
  - si v n'est pas marqué "exploré" alors
    - appeler récursivement DFS(v)

Sur le graphe précédent, les sommets sont visités cette fois dans l'ordre $1, 2, 3, 5, 4, 6, 7, 8$. Le dessin plus bas montre les sommets dans l'ordre où ils sont découverts.



Si on appelle T l'arbre couvrant produit (celui qui contient "l'histoire du parcours": on décrète que v a u comme père si l'arête (u, v) a permis de découvrir v), on a:

- Pour chaque appel récursif DFS(u), tous les sommets qui sont marqués "explorés" entre l'invocation de l'appel et son retour sont des descendants de u dans l'arbre T produit.
- Soit T un arbre DFS, et x et y deux sommets dans T, avec $(x, y) \in E$ qui n'est pas une arête de T. Alors soit x est un ancêtre de y, soit le contraire.

Implémenter BFS(s)

- $Discovered[s] := true$; $Discovered[v] := false$ pour $v \neq s$.
- $L[0] = \{s\}$; i:=0 // Layer counter
- Set $T = \emptyset$ // Arbre
- Tant que $L[i] \neq \emptyset$
  - $L[i+1] := \emptyset$
  - Pour chaque $u \in L[i]$
    - Pour chaque arête $(u, v) \in E$
    - Si $Discovered[v] == false$ alors
    - Discovered[v]:=true;
    - Ajouter (u, v) à l'arbre T ($\pi(v) = u$).
    - Ajouter v à $L[i+1]$
    - Fin si
  - Fin pour
  - i:=i+1
- Fin tant que

Temps $O(n+m)$, où n est le nombre de sommets, et m le nombre d'arêtes avec une représentation par liste de successeurs.

Implémenter DFS(s)

- $Discovered[s] := true$; $Discovered[v] := false$ pour $v \neq s$.
- Créer S comme la pile contenant uniquement l'élément s
- Tant que $S \neq \emptyset$
  - Prendre le sommet u de S
  - Si $Discovered[u] == false$ alors
    - $Discovered[u] := true$
    - Pour chaque arête $(u, v) \in E$
    - Ajouter u à la pile S
    - Fin pour
  - Fin si
- Fin while

Temps $O(n + m)$, où n est le nombre de sommets, et m le nombre d'arêtes, avec une représentation par liste de successeurs.

Applications des parcours

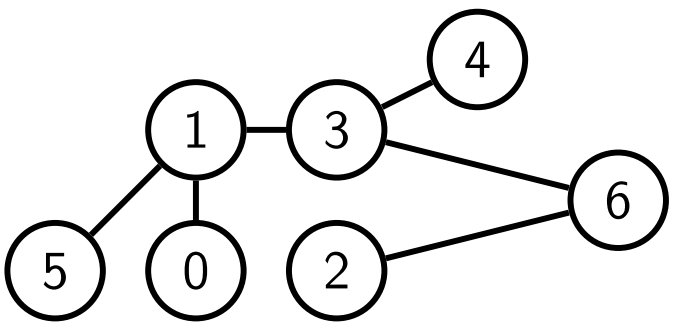
- Numérotation (vue plus haut).
- Découverte de cycles.
- Numérotation en niveaux ...

Remarque en passant

La définition inductive des arbres (cf slides de N. Pronost), est clairement plus efficace.

Les arbres

- Un graphe[2] connexe sans cycle est appelé un *arbre* (libre).
- Un graphe[2] sans-cycle est appelé une *forêt*:
  - chacune de ses composantes connexes est un arbre.

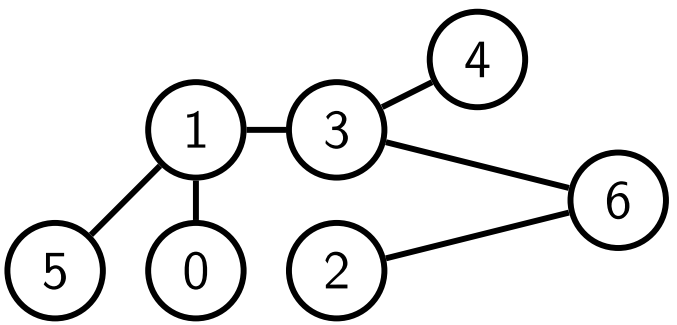


[2] non-orienté.

Les arbres sont partout!

- Un graphe[2] connexe sans cycle est appelé un *arbre* (libre).
- Un graphe[2] sans-cycle est appelé une *forêt*:
  - chacune de ses composantes connexes est un arbre.

- Dès qu'on a des objets, des relations entre objets, et pas de cycle, on a donc un arbre ou une forêt.

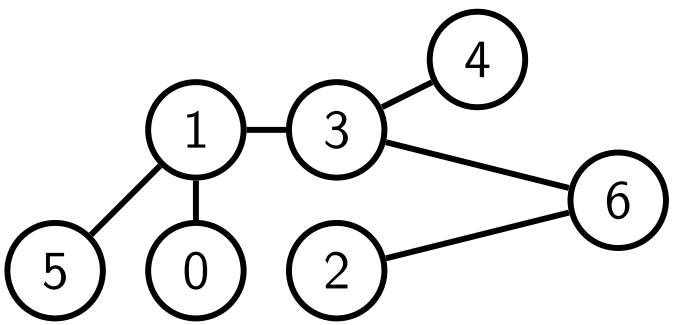


[2] non-orienté.

Les arbres sont partout!

- Un graphe[2] connexe sans cycle est appelé un *arbre* (libre).
- Un graphe[2] sans-cycle est appelé une *forêt*:
  - chacune de ses composantes connexes est un arbre.

- Dès qu'on a des objets, des relations entre objets, et pas de cycle, on a donc un arbre ou une forêt.



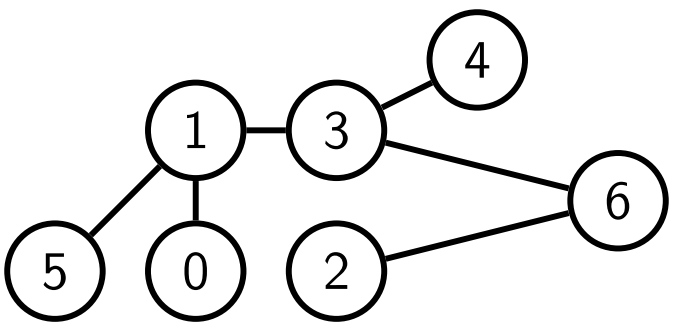
- Les arbres sont omniprésents en informatique.

[2] non-orienté.

Une caractérisation & Quelques propriétés

Soit $G = (V, E)$ un graphe[3]. Les propriétés suivantes sont équivalentes:

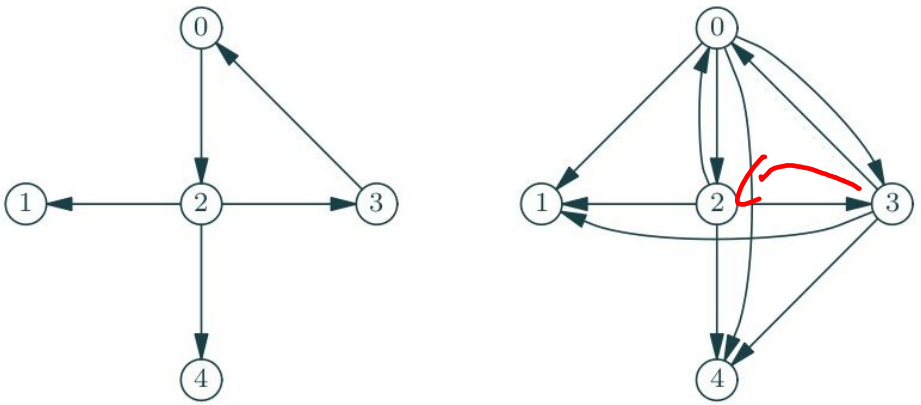
- G est un arbre (libre).
- Deux sommets quelconques de V sont connectés par un unique chemin simple.
- G est connexe, mais ne l'est plus si on enlève n'importe laquelle de ses arêtes.
- G est connexe, et $|E| = |V| - 1$.
- G est sans cycle, et $|E| = |V| - 1$.
- G est sans cycle, mais ne l'est plus si l'on ajoute n'importe quelle arête.



[3] non-orienté.

Clôture transitive.

Attention il manque une flèche sur le dessin de droite !



Le graphe $G^+ = (V, E^+)$ de la clôture transitive du graphe orienté $G = (V, E)$ est tel que E^+ est le plus petit ensemble d'arêtes satisfaisant :

- $E \subseteq E^+$
- $(x, y) \in E^+$ and $(y, z) \in E^+ \Leftrightarrow (x, z) \in E^+$

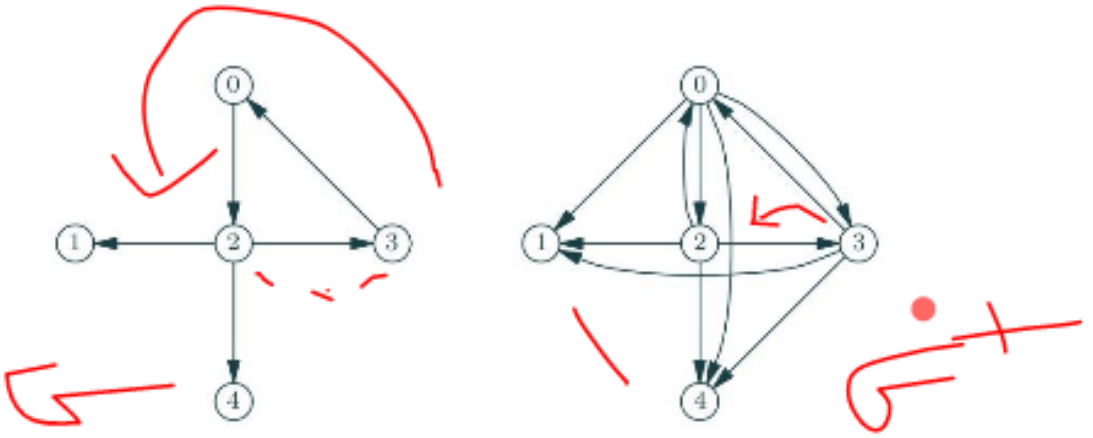
Si on fait la clôture transitive d'un graphe orienté connexe, on obtient le graphe complet.

La clôture transitive permet de savoir si un sommet est accessible à partir d'un autre par un chemin

↳ Intéressant à calculer si on doit souvent répondre à ce type de question.

Clôture transitive.

Attention il manque une flèche sur le dessin de droite !



Le graphe $G^+ = (V, E^+)$ de la clôture transitive du graphe orienté $G = (V, E)$ est tel que E^+ est le plus petit ensemble d'arêtes satisfaisant :

- $E \subset E^+$
- $(x, y) \in E^+$ and $(y, z) \in E^+ \Leftrightarrow (x, z) \in E^+$

Graphe

Matrice d'adjacence

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

1 accessible à partir de 2 en 1 coup

$$M^2 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

1 accessible depuis 0 en 2 coups

On a multiplié $M_{0,2}$ avec $M_{2,1}$ pour obtenir le chemin $0 \to 2 \to 1$

Attention on fait des calculs booléens

$1 + 1 = 1$!

$E^+ = M + M^2$ = clôture transitive

$M + M^2$: carré dans les booléens (intersection)

$+$ dans les booléens (réunion).

→ matrice booléenne avec des 0 et des 1

Un premier algorithme pour la clôture

Attention ici la multiplication n'est pas forcément celle que l'on croit

Si E est la matrice d'adjacence du graphe initial :

$E^+ = E + E^2 + \dots E^{n-1}$.

- Cela donne un algo en $O(n^4)$.

$n \times n^3$ coût d'une multiplication de matrice

- Mieux ? oui ; en $O(n^3)$, avec Floyd Warshall, voir plus tard !

.

Essentiel sur les graphes :

- Parties 1, 2, 3 et 4 du diaporama sur les généralités

dont : les defs élémentaires ; la notion de parcours ; les deux pacours classiques BFS et DFS, leurs applications notamment pour les composantes connexes, et savoir les implémenter.