Algorithme 2

Algorithmes élémentaires pour les graphes

Louis-Claude Canon
louis-claude.canon@univ-fcomte.fr

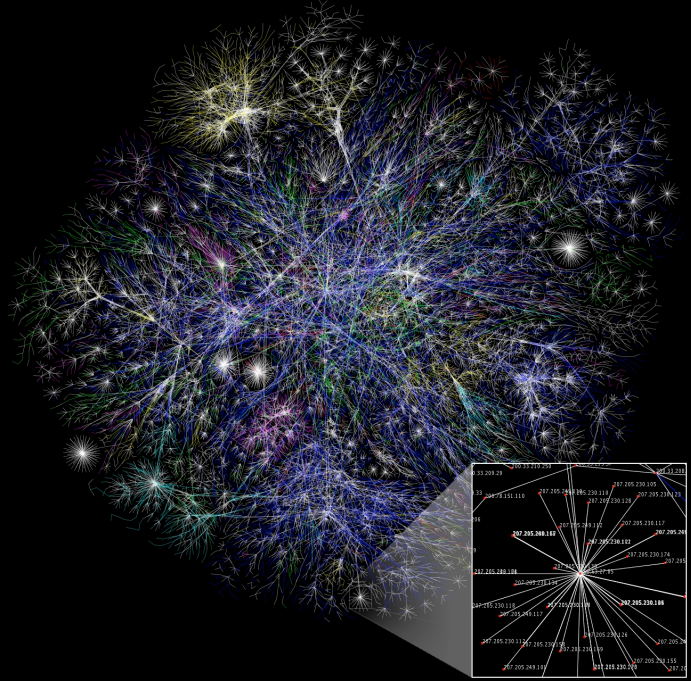
Licence 2 Informatique – Semestre 4

Plan

Plan

Motivation

- De nombreux problèmes se modélisent avec des graphes :
  - cartes routières ;
  - relations sur un réseau social ;
  - réseaux informatique ;
  - dépendances entre cibles dans un Makefile ;
  - automates à états finis ;
  - etc.
- Il existe des centaines d'algorithmes de graphe connus.
- Les *parcours* consistent à considérer successivement chaque sommet et ils sont utilisés comme opérations élémentaires par de nombreux algorithmes.


207.205.230.109
207.205.230.118
207.205.230.128
207.205.230.117
207.205.230.174
207.205.230.186
207.205.230.122
207.205.230.186

Wikipedia

Paul Butler

Graphe

- Un *graphe* $G = (V, E)$ est un ensemble de *sommets* V connectés par des *arêtes* E.
- Un *chemin* est une séquence de sommets connectés pas des arêtes.
- Un *cycle* est un chemin dont le premier et dernier sommet sont les mêmes.
- Le *degré* d'un sommet est le nombre d'arêtes qui y sont connectées.

Graphe

- Un *graphe* $G = (V, E)$ est un ensemble de *sommets* V connectés par des *arêtes* E.
- Un *chemin* est une séquence de sommets connectés pas des arêtes.
- Un *cycle* est un chemin dont le premier et dernier sommet sont les mêmes.
- Le *degré* d'un sommet est le nombre d'arêtes qui y sont connectées.

Graphe

- Un *graphe* $G = (V, E)$ est un ensemble de *sommets* V connectés par des *arêtes* E.
- Un *chemin* est une séquence de sommets connectés pas des arêtes.
- Un *cycle* est un chemin dont le premier et dernier sommet sont les mêmes.
- Le *degré* d'un sommet est le nombre d'arêtes qui y sont connectées.

Graphe

- Un *graphe* $G = (V, E)$ est un ensemble de *sommets* V connectés par des *arêtes* E.
- Un *chemin* est une séquence de sommets connectés pas des arêtes.
- Un *cycle* est un chemin dont le premier et dernier sommet sont les mêmes.
- Le *degré* d'un sommet est le nombre d'arêtes qui y sont connectées.

Exemples de modélisation

Application	sommet	arête
Communication	téléphone, ordinateur	câble, fibre optique
Circuit	porte, registre, processeur	connexion
Mobilité	intersection, gare	route, rails
Jeu	plateau de jeu	déplacement valide
Réseau de neurone	neurone	synapse
Réseau social	personne	connexion

Exemple de problèmes

Problème	difficulté [1]
Trouver un chemin entre 2 sommets	

1. https://xkcd.com/1425/

Exemple de problèmes

Problème	difficulté [1]
Trouver un chemin entre 2 sommets	Licence-
Trouver un plus court chemin entre 2 sommets	

1. https://xkcd.com/1425/

Exemple de problèmes

Problème	difficulté [1]
Trouver un chemin entre 2 sommets	Licence-
Trouver un plus court chemin entre 2 sommets	Licence
Déterminer si tous les sommets sont connectés	

1. https://xkcd.com/1425/

Exemple de problèmes

Problème	difficulté [1]
Trouver un chemin entre 2 sommets	Licence-
Trouver un plus court chemin entre 2 sommets	Licence
Déterminer si tous les sommets sont connectés	Licence
Trouver un cycle	

1. https://xkcd.com/1425/

Exemple de problèmes

Problème	difficulté [1]
Trouver un chemin entre 2 sommets	Licence-
Trouver un plus court chemin entre 2 sommets	Licence
Déterminer si tous les sommets sont connectés	Licence
Trouver un cycle	Licence-
Trouver un cycle qui passe par chaque arête	

1. https://xkcd.com/1425/

Exemple de problèmes

Problème	difficulté [1]
Trouver un chemin entre 2 sommets	Licence-
Trouver un plus court chemin entre 2 sommets	Licence
Déterminer si tous les sommets sont connectés	Licence
Trouver un cycle	Licence-
Trouver un cycle qui passe par chaque arête	Master-
Trouver un cycle qui passe par chaque sommet	

1. https://xkcd.com/1425/

Exemple de problèmes

Problème	difficulté [1]
Trouver un chemin entre 2 sommets	Licence-
Trouver un plus court chemin entre 2 sommets	Licence
Déterminer si tous les sommets sont connectés	Licence
Trouver un cycle	Licence-
Trouver un cycle qui passe par chaque arête	Master-
Trouver un cycle qui passe par chaque sommet	Master ?
Dessiner le graphe sans arêtes qui se croisent	

1. https://xkcd.com/1425/

Exemple de problèmes

Problème	difficulté [1]
Trouver un chemin entre 2 sommets	Licence-
Trouver un plus court chemin entre 2 sommets	Licence
Déterminer si tous les sommets sont connectés	Licence
Trouver un cycle	Licence-
Trouver un cycle qui passe par chaque arête	Master-
Trouver un cycle qui passe par chaque sommet	Master ?
Dessiner le graphe sans arêtes qui se croisent	Master++
Déterminer si 2 graphes sont les mêmes	

1. https://xkcd.com/1425/

Exemple de problèmes

Problème	difficulté [1]
Trouver un chemin entre 2 sommets	Licence-
Trouver un plus court chemin entre 2 sommets	Licence
Déterminer si tous les sommets sont connectés	Licence
Trouver un cycle	Licence-
Trouver un cycle qui passe par chaque arête	Master-
Trouver un cycle qui passe par chaque sommet	Master ?
Dessiner le graphe sans arêtes qui se croisent	Master++
Déterminer si 2 graphes sont les mêmes	?

1. https://xkcd.com/1425/

Problème de l'isomorphisme de graphes

Curiosité

Problème : deux graphes sont-ils les mêmes, c'est-à-dire peut-on obtenir le premier graphe en renommant les sommets du second ?
Question ouverte : problème dont on ne sait ni s'il est NP-Complet, ni s'il est résoluble en temps polynomial.
Même situation qu'avec les problèmes de la plus courte distance rotationnelle et de la factorisation en facteurs premiers.

Plan

Listes d'adjacences

- Un graphe peut se représenter par une *liste d'adjacences* associée à chaque sommet : $G.adj[v]$ pour le sommet v.
- $G.adj[v]$ donne la liste des sommets voisins accessibles à partir de v, c'est-à-dire que $v \in G.adj[u]$ et $u \in G.adj[v]$ pour toute arête $(u, v) \in E$.
- C'est une représentation adaptée pour des graphes *peu denses* : la mémoire utilisée est en $O(V + E)$.
- Inconvénient : déterminer si une arête (u, v) est présente dans E nécessite de parcourir la liste d'adjacences de u ou de v, ce qui est en $O(E)$.

Listes d'adjacences

- Un graphe peut se représenter par une *liste d'adjacences* associée à chaque sommet : $G.adj[v]$ pour le sommet v.
- $G.adj[v]$ donne la liste des sommets voisins accessibles à partir de v, c'est-à-dire que $v \in G.adj[u]$ et $u \in G.adj[v]$ pour toute arête $(u, v) \in E$.
- C'est une représentation adaptée pour des graphes *peu denses* : la mémoire utilisée est en $O(V + E)$.
- Inconvénient : déterminer si une arête (u, v) est présente dans E nécessite de parcourir la liste d'adjacences de u ou de v, ce qui est en $O(E)$.

Listes d'adjacences

- Un graphe peut se représenter par une *liste d'adjacences* associée à chaque sommet : $G.adj[v]$ pour le sommet v.
- $G.adj[v]$ donne la liste des sommets voisins accessibles à partir de v, c'est-à-dire que $v \in G.adj[u]$ et $u \in G.adj[v]$ pour toute arête $(u, v) \in E$.
- C'est une représentation adaptée pour des graphes *peu denses* : la mémoire utilisée est en $O(V + E)$.
- Inconvénient : déterminer si une arête (u, v) est présente dans E nécessite de parcourir la liste d'adjacences de u ou de v, ce qui est en $O(E)$.

Listes d'adjacences

- Un graphe peut se représenter par une *liste d'adjacences* associée à chaque sommet : $G.adj[v]$ pour le sommet v.
- $G.adj[v]$ donne la liste des sommets voisins accessibles à partir de v, c'est-à-dire que $v \in G.adj[u]$ et $u \in G.adj[v]$ pour toute arête $(u, v) \in E$.
- C'est une représentation adaptée pour des graphes *peu denses* : la mémoire utilisée est en $O(V + E)$.
- Inconvénient : déterminer si une arête (u, v) est présente dans E nécessite de parcourir la liste d'adjacences de u ou de v, ce qui est en $O(E)$.

Notation ensembliste

Notion mathématique

$V = \{u, v, w\}$ Ensemble V contenant 3 éléments : u, v et w.

$|V|$ Nombre d'éléments, ou cardinalité, de l'ensemble V (ici nombre de sommets). Pour les complexités cependant, on simplifiera $O(|V|) = O(V)$ et $O(|E|) = O(E)$.

$\emptyset$ Ensemble vide.

$A \setminus B$ Opération de différence sur les ensembles (ce qui est dans A mais pas dans B).

$A \cup B$ Opération d'union sur les ensembles (ce qui est dans A ou/et dans B).

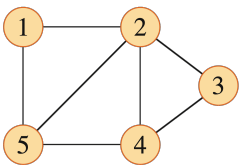
Matrice d'adjacences

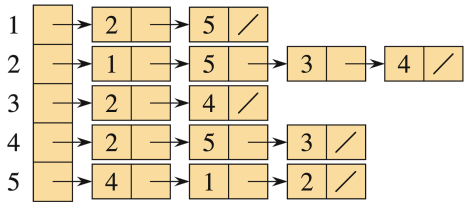
- Un graphe peut aussi se représenter par une *matrice d'adjacences*.
- On a alors une matrice $V \times V$, $A = (a_{ij})$ telle que :

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

- La mémoire utilisée est en $O(V^2)$.
- Déterminer si une arête est présente dans E est en $\Theta(1)$.
- Il s'agit aussi d'une représentation qui facilite le codage.

Exemple



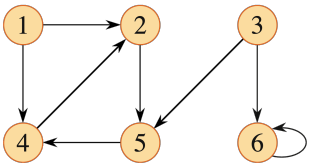


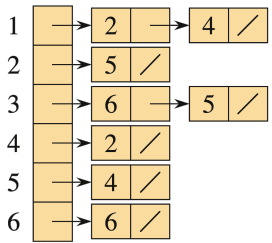
	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Graphe orienté

- Dans un graphe orienté, les arêtes sont dirigées d'un sommet à l'autre et se nomment *arcs*.
- Les cycles se nomment des *circuits*.
- Les deux représentations précédentes sont possibles pour des graphes orientés : chaque liste d'adjacences contiendra la liste des sommets destinations et la matrice aura les sommets sources sur les lignes et les sommets destinations sur les colonnes.

Exemple





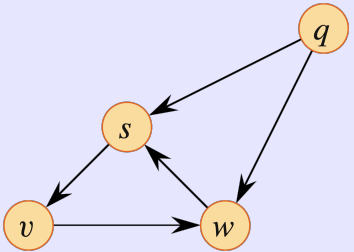
	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

Pondérations et attributs

- On peut rajouter des poids ou des attributs arbitraires (données satellites) sur les sommets et les arêtes.
- Avec les listes d'adjacences, on ajoute ces valeurs dans la liste (ou on crée un tableau séparé pour chaque sommet).
- Avec la matrice d'adjacences, on les ajoute dans la matrice (ou on crée une matrice séparée).
- On notera $v.d$ l'attribut d du sommet v et $(u, v).d$ l'attribut d de l'arête (u, v).

Question

Quelle est la matrice d'adjacences du graphe suivant ?



1.

	w	s	v	q
w	0	1	0	0
s	0	0	1	0
v	0	0	0	1
q	0	1	1	0

2.

	q	v	s	w
q	0	0	1	1
v	0	0	1	0
s	0	1	0	0
w	0	0	1	0

3.

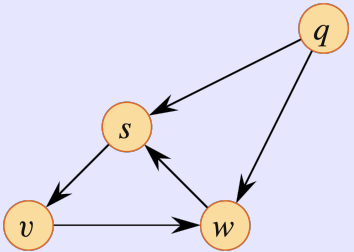
	v	s	q	w
v	0	0	0	1
s	1	0	0	0
q	0	1	0	1
w	0	0	1	0

4.

	q	s	v	w
q	0	1	0	1
s	0	0	1	0
v	0	0	0	1
w	0	1	0	0

Question

Quelle est la matrice d'adjacences du graphe suivant ?



1.

	w	s	v	q
w	0	1	0	0
s	0	0	1	0
v	0	0	0	1
q	0	1	1	0

2.

	q	v	s	w
q	0	0	1	1
v	0	0	1	0
s	0	1	0	0
w	0	0	1	0

3.

	v	s	q	w
v	0	0	0	1
s	1	0	0	0
q	0	1	0	1
w	0	0	1	0

✓

	q	s	v	w
q	0	1	0	1
s	0	0	1	0
v	0	0	0	1
w	0	1	0	0

Plan

Contexte

- Le *parcours en largeur* (ou *Breadth-First Search*, BFS) sert de base à deux algorithmes essentiels vus par la suite (Prim et Dijkstra).
- L'objectif consiste à explorer les sommets depuis une *origine* en commençant par les plus proches, puis en élargissant progressivement jusqu'aux plus distants.
- On s'en sert surtout lorsqu'il est question de trouver un plus court chemin à partir d'une origine.

Explication de PL

- On va considérer une frontière de sommets Q qui va s'élargir petit à petit.
- Les sommets sont colorés (ils sont marqués) pour éviter que l'on tourne en rond : blanc pour tous les sommets inexplorés ; gris pour les sommets de la frontière Q ; noir pour les sommets explorés.
- Initialement, il n'y a que l'origine qui est grise.
- À chaque fois qu'un sommet gris est exploré, on rajoute tous ses voisins blancs à la frontière Q en les coloriant en gris, puis on l'enlève de la frontière en le coloriant en noir.

Pseudo-code de PL-Simplifié

PL-Simplifié(G, s)

pour chaque sommet $u \in G.V \setminus \{s\}$ **alors**
  $u.couleur \leftarrow$ BLANC
$s.couleur \leftarrow$ GRIS
$Q \leftarrow \emptyset$
Enfile(Q, s) `// La frontière contient l'origine`
tant que $Q \neq \emptyset$ **faire**
  $u \leftarrow$ Defile(Q)
  pour chaque $v \in G.adj[u]$ **faire** `// On parcourt les voisins de` u
    si $v.couleur =$ BLANC **alors** `// Si` v `n'a jamais été considéré`
      $v.couleur \leftarrow$ GRIS
      Enfile(Q, v) `// ` v `est désormais dans la frontière`
  $u.couleur \leftarrow$ NOIR `// ` v `est désormais derrière la frontière`

Analyse de complexité

- Chaque sommet n'est jamais colorié en blanc plus d'une fois.
- Chaque sommet blanc est enfilé une fois, puis colorié en gris. Il ne sera donc pas enfilé une seconde fois et sera défilé au plus une fois.
- Enfiler et défiler prend un temps $\Theta(1)$ pour chaque sommet, donc l'ensemble des opérations liées à Q prend un temps $O(V)$.
- Les voisins de chaque sommet sont parcourus au plus une fois, ce qui prend un temps $O(E)$.
- Le coût d'initialisation est $O(V)$.
- Le coût total est donc $O(V + E)$.

Plus court chemin et arbre de parcours en largeur

- L'algorithme calcule également le *plus court chemin* entre chaque sommet et l'origine : il s'agit du chemin dont le nombre d'arêtes qui les composent est minimal.
- L'attribut d de chaque sommet mesure cette distance.
- L'algorithme génère aussi un *arbre de parcours en largeur* : chaque chemin depuis l'origine vers chaque sommet est un chemin minimal.
- L'attribut π de chaque sommet représente son *parent* dans cet arbre.

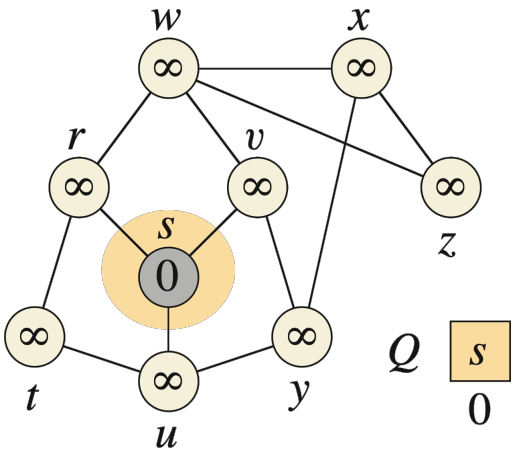
Pseudo-code de PL

PL(G, s)

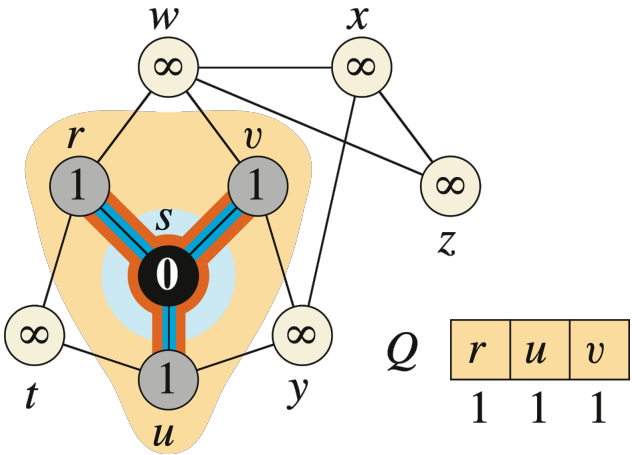
pour chaque sommet $u \in G.V \setminus \{s\}$ **alors**
  $u.couleur \leftarrow$ BLANC
  $u.d \leftarrow \infty$
  $u.\pi \leftarrow NIL$
$s.couleur \leftarrow$ GRIS
$s.d \leftarrow 0$
$s.\pi \leftarrow NIL$
$Q \leftarrow \emptyset$
ENFILE(Q, s)

tant que $Q \neq \emptyset$ **faire**
  $u \leftarrow$ DEFILE(Q)
  pour chaque $v \in G.adj[u]$ **faire**
    si $v.couleur =$ BLANC **alors**
      $v.couleur \leftarrow$ GRIS
      $v.d \leftarrow u.d + 1$
      $v.\pi \leftarrow u$
      ENFILE(Q, v)
  $u.couleur \leftarrow$ NOIR

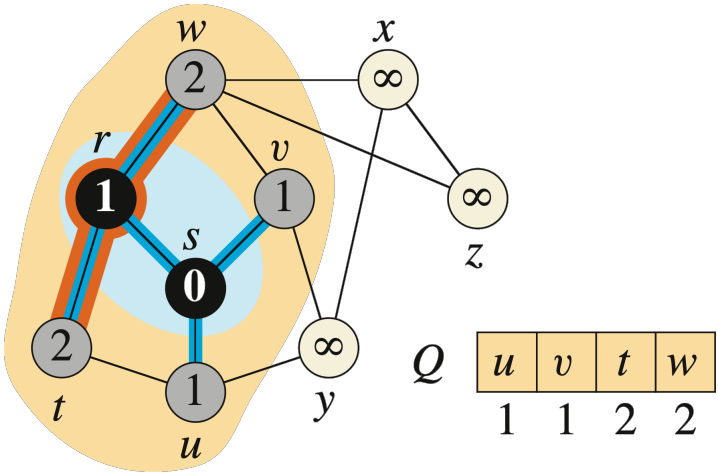
Exemple



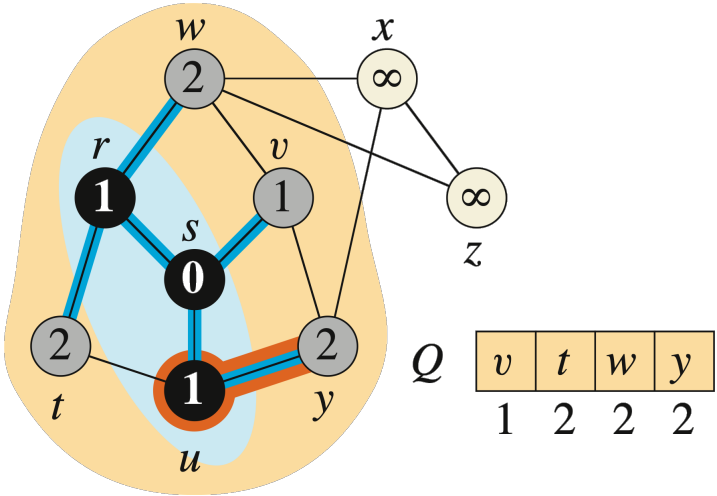
Exemple



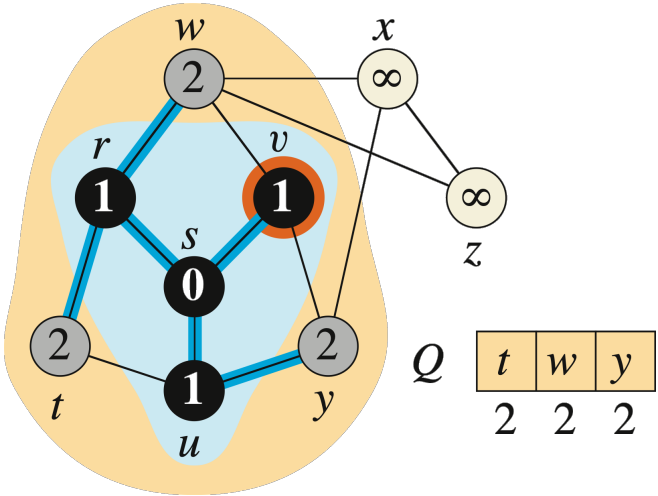
Exemple



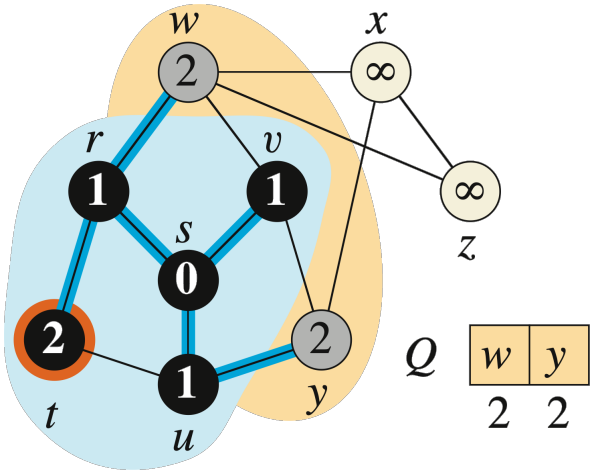
Exemple



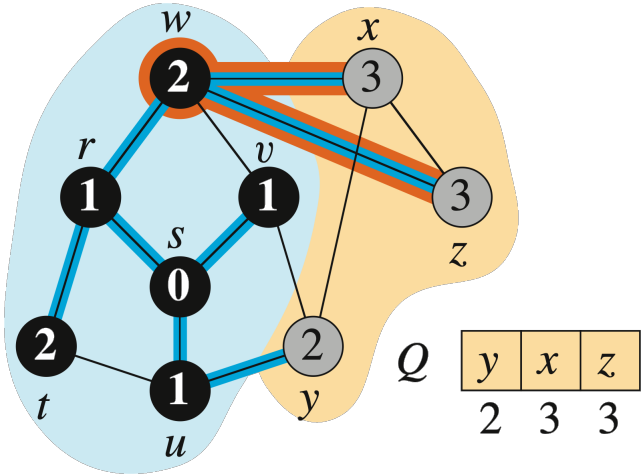
Exemple



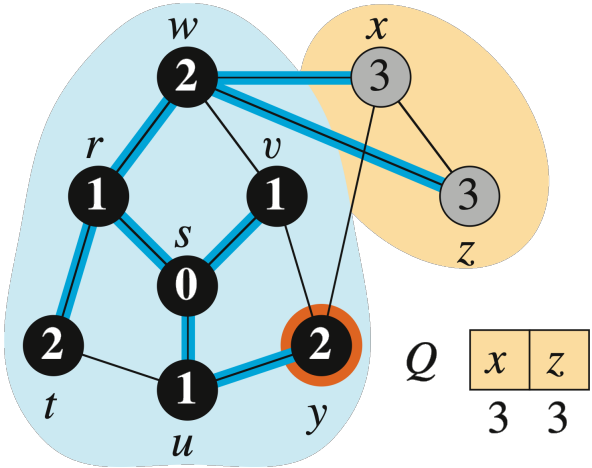
Exemple



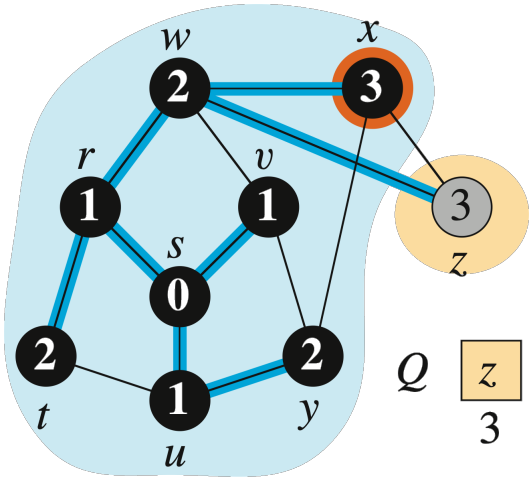
Exemple



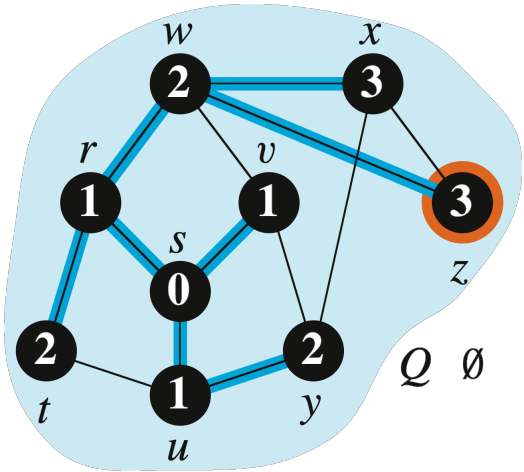
Exemple



Exemple



Exemple



Validité du parcours en largeur

Théorème

Soit $G = (V, E)$ un graphe orienté ou non ; supposons que PL est exécuté sur G à partir d'un certain sommet origine $s \in V$. Alors :

- *pendant son exécution, PL découvre chaque sommet $v \in V$ accessible à partir de l'origine s ;*
- *à la fin, $v.d$ est la distance du plus court chemin depuis l'origine pour tout $v \in V$;*
- *pour tout sommet $v \neq s$ accessible à partir de s, l'un des plus courts chemins de s à v est le plus court chemin de s à $v.\pi$ complété par l'arête $(v.\pi, v)$.*

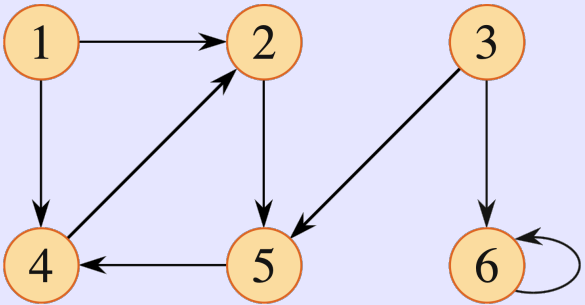
Preuve de validité (invariant)

À chaque début d'itération de la boucle principale :

- Tous les sommets gris sont dans la file et celle-ci ne contient que des sommets gris.
- La file contient au moins un sommet gris dont la distance à partir de s est k, suivi de zéro ou plus sommets gris de distance $k+1$.
- La distance de tout sommet noir v à partir de s est $v.d$, $v.d \leq k$ et l'un des plus courts chemins de s à v est le plus court chemin de s à $v.\pi$ complété par l'arête $(v.\pi, v)$.
- La distance de tout sommet blanc v à partir de s est supérieure ou égale à $k+1$.

Question

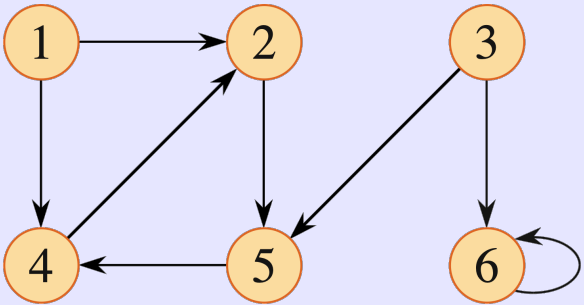
Quelles sont les valeurs de d d'un parcours en largeur sur le graphe orienté suivant en prenant pour origine le sommet 3 ?



1. $[3, 2, 0, 2, 1, 1]$
2. $[\infty, 3, 0, 2, 1, 1]$
3. $[3, 2, 0, 2, 1, \infty]$
4. $[\infty, 3, 0, 2, 1, \infty]$

Question

Quelles sont les valeurs de d d'un parcours en largeur sur le graphe orienté suivant en prenant pour origine le sommet 3 ?



1. $[3, 2, 0, 2, 1, 1]$
2. $[\infty, 3, 0, 2, 1, 1]$ ✓(l'ordre suit les labels)
3. $[3, 2, 0, 2, 1, \infty]$
4. $[\infty, 3, 0, 2, 1, \infty]$

Plan

Contexte

- Le *parcours en profondeur* (ou *Depth-First Search*, DFS) sert de base aux algorithmes vus dans un prochain cours (tri topologique et composantes connexes).
- L'objectif consiste à explorer les sommets en favorisant l'exploration des sommets les plus récemment découverts.
- On s'en sert surtout comme sous-procédure dans d'autres algorithmes.

Explication de PP-Simplifié

- On colorie à nouveau les sommets de la même façon : initialement blanc, gris quand il est découvert, puis noir en fin de traitement.
- Comme il n'y a pas d'origine, on démarrera de chaque sommet, ce qui permettra de parcourir des graphes non connexes.
- Lorsqu'un nouveau sommet est découvert, sa couleur passe de blanc à gris et tous ses voisins sont alors considérés.
- Lorsque ces voisins sont traités, le sommet est alors noirci et le parcours continue avec le sommet le plus récemment découvert qui est encore gris.
- Cela consiste à parcourir le plus profondément le graphe puis à revenir à arrière pour continuer l'exploration.

Pseudo-code de PP-Simplifié

PP-Simplifié(G)

pour chaque sommet $u \in G.V$ **alors**
  $u.couleur \leftarrow$ BLANC
pour chaque sommet $u \in G.V$ **alors**
  si $u.couleur =$ BLANC **alors**
    Visiter-PP-Simplifié(G, u)

Visiter-PP-Simplifié(u)

$u.couleur \leftarrow$ GRIS
pour chaque $v \in G.adj[u]$ **faire**
  si $v.couleur =$ BLANC **alors**
    Visiter-PP-Simplifié(G, v)
$u.couleur \leftarrow$ NOIR

Analyse de complexité

- Les 2 boucles de PP-Simplifié requièrent un temps $\Theta(V)$.
- Visiter-PP-Simplifié est appelé une seule fois par sommet car la procédure ne s'applique que sur des sommets blancs qu'elle grise en première étape.
- La boucle de Visiter-PP-Simplifié s'exécute $|G.adj[u]|$ fois.
- Comme $\sum_{v \in V} |G.adj[v]| = E$ et que la procédure est appelée une seule fois par sommet, le coût total de Visiter-PP-Simplifié est $\Theta(V + E)$.
- Le coût du parcours en profondeur est donc $\Theta(V + E)$.

Forêt de parcours en profondeur et dates

- Comme pour le parcours en largeur, on conserve le parent de chaque sommet (attribut π).
- On obtient donc un ou plusieurs arbres que l'on appelle une *forêt de parcours en profondeur* (une forêt est un ensemble d'arbres).
- La structure de cette forêt reflète la structure des appels récursifs à VISITER-PP-SIMPLIFIÉ.
- D'autre part, pour chaque sommet, on note sa date de découverte (attribut d) et sa date de fin de traitement (attribut f).
- La *date* est juste un entier que l'on incrémente à chaque traitement.

Parcours en profondeur

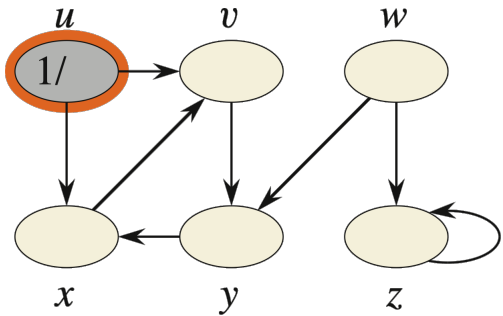
PP(G)

pour chaque sommet $u \in G.V$ **alors**
  $u.couleur \leftarrow$ BLANC
  $u.\pi \leftarrow NIL$
$date \leftarrow 0$
pour chaque sommet $u \in G.V$ **alors**
  si $u.couleur =$ BLANC **alors**
    VISITER-PP(G, u)

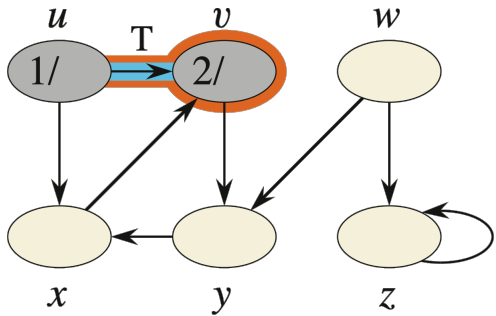
VISITER-PP(u)

$date \leftarrow date + 1$
$u.d \leftarrow date$
$u.couleur \leftarrow$ GRIS
pour chaque $v \in G.adj[u]$ **faire**
  si $v.couleur =$ BLANC **alors**
    $v.\pi \leftarrow u$
    VISITER-PP(G, v)
$date \leftarrow date + 1$
$u.f \leftarrow date$
$u.couleur \leftarrow$ NOIR

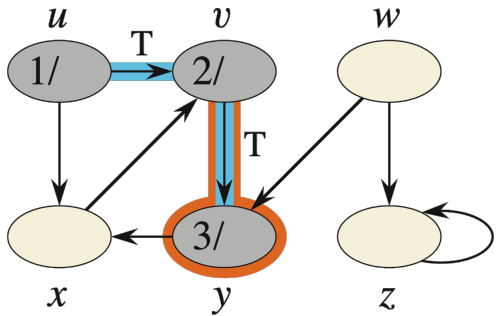
Exemple



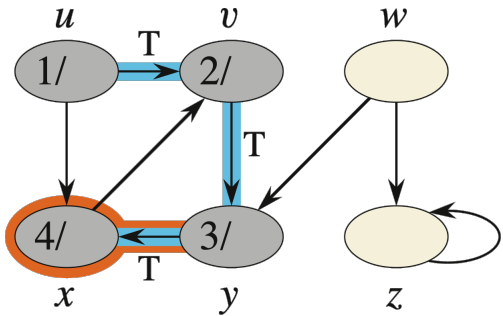
Exemple



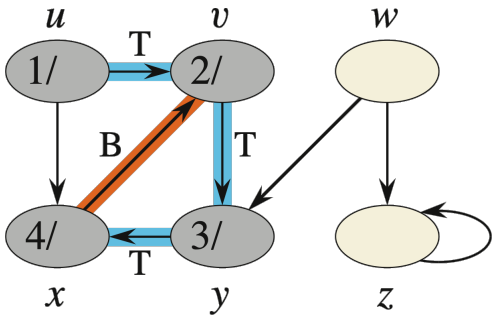
Exemple



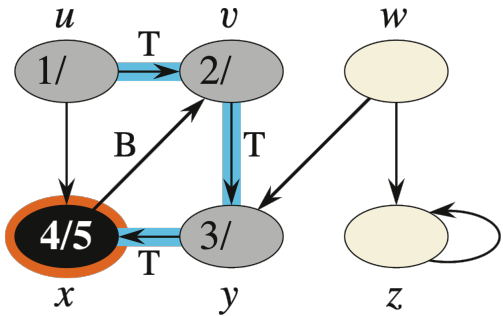
Exemple



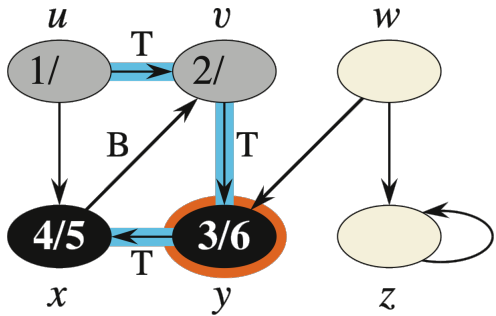
Exemple



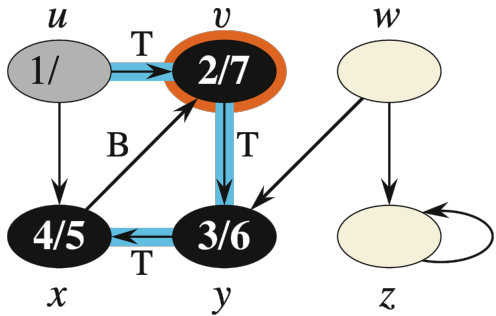
Exemple



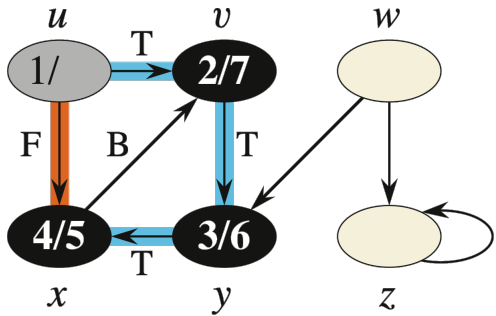
Exemple



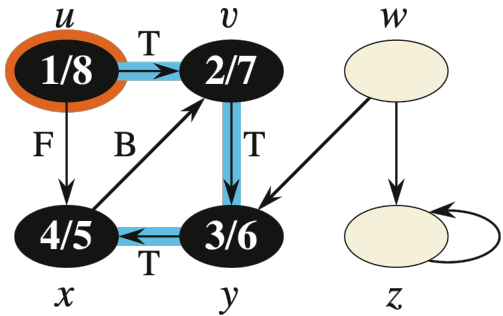
Exemple



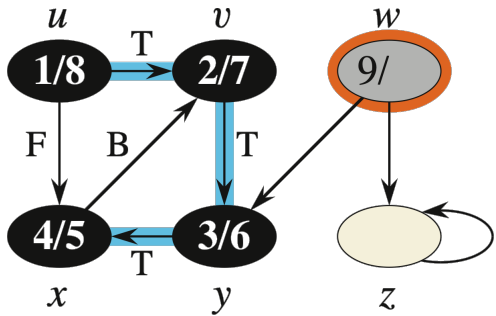
Exemple



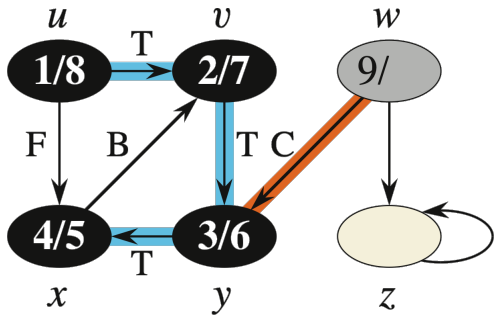
Exemple



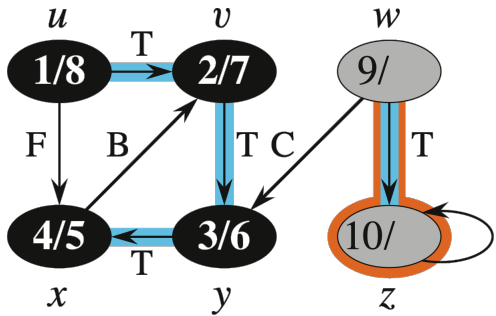
Exemple



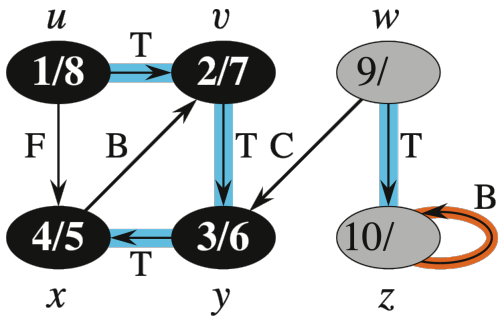
Exemple



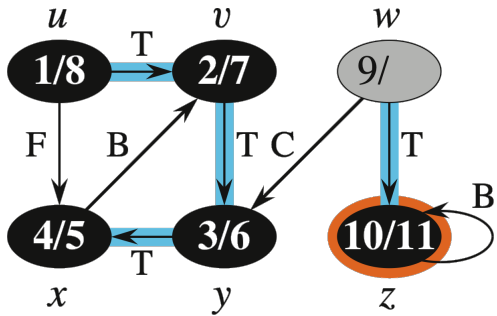
Exemple



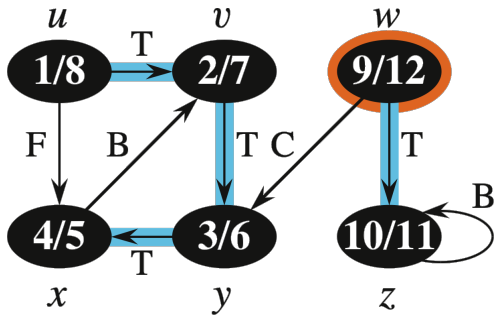
Exemple



Exemple

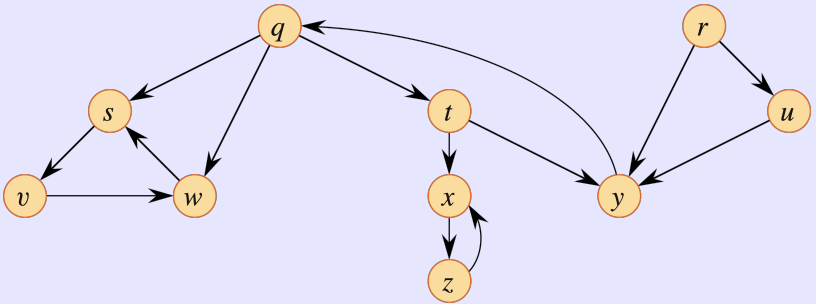


Exemple



Question

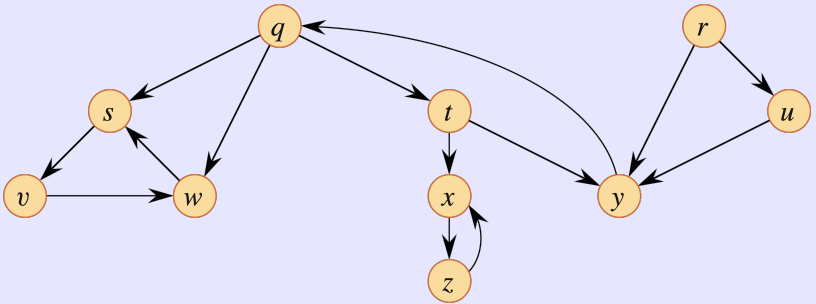
Quel ordre ne peut pas être obtenu par un parcours en profondeur sur le graphe suivant (quelque soit l'ordre considéré sur les sommets) ?



1. $t, x, z, y, q, w, s, v, u, r$
2. $r, u, y, q, t, x, z, s, v, w$
3. $v, w, s, x, z, t, u, y, q, r$
4. $w, s, v, x, z, y, q, t, r, u$

Question

Quel ordre ne peut pas être obtenu par un parcours en profondeur sur le graphe suivant (quelque soit l'ordre considéré sur les sommets) ?



1. $t, x, z, y, q, w, s, v, u, r$
2. $r, u, y, q, t, x, z, s, v, w$
3. $v, w, s, x, z, t, u, y, q, r$ ✓(u est trop tôt)
4. $w, s, v, x, z, y, q, t, r, u$

Plan

Résumé

Contenu

- Représentations des graphes : liste d'adjacences ou matrice d'adjacences.
- Le parcours en largeur ordonne les sommets en fonction de leur proximité à l'origine.
- Le parcours en profondeur priorise le dernier sommet exploré.
- https://xkcd.com/761/

Prochaines échéances

- QCM à la prochaine séance de CM.
- DS 1 : semaine du 24/3.
- Projet-tournoi : début la semaine du 14/4, à rendre la semaine du 26/5.

Résumé

Contenu

- Représentations des graphes : liste d'adjacences ou matrice d'adjacences.
- Le parcours en largeur ordonne les sommets en fonction de leur proximité à l'origine.
- Le parcours en profondeur priorise le dernier sommet exploré.
- https://xkcd.com/761/

Prochaines échéances

- QCM à la prochaine séance de CM.
- DS 1 : semaine du 24/3.
- Projet-tournoi : début la semaine du 14/4, à rendre la semaine du 26/5.