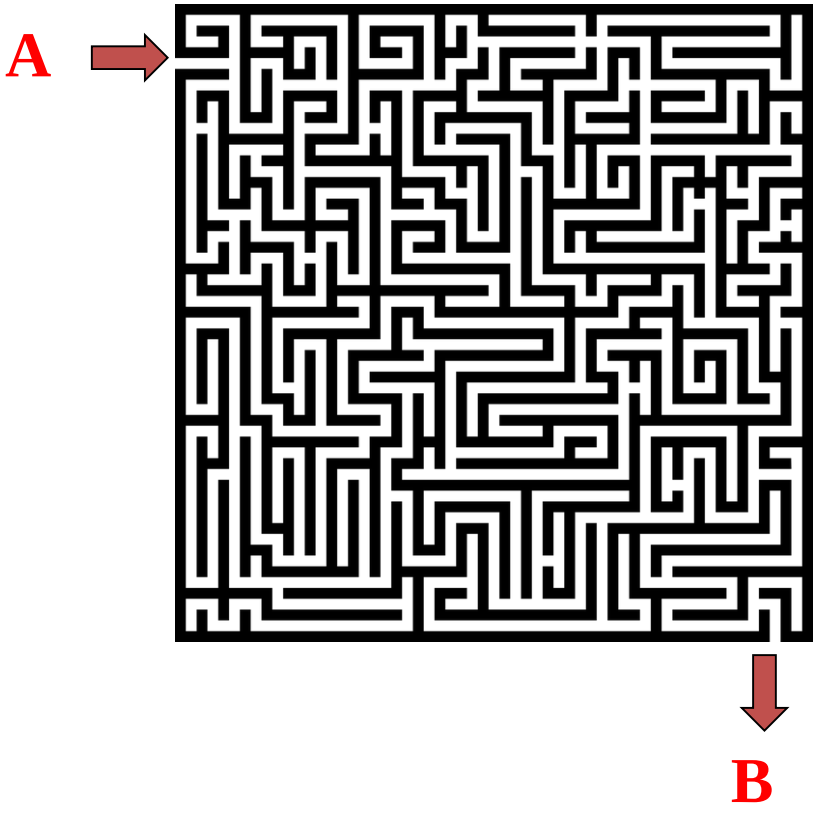
Parcours de graphes

Heike Ripphausen-Lipa - Beuth University of Applied Science - Berlin
J.M. Adam - Université de Grenoble Alpes - Grenoble

Labyrinthe



Trouver un chemin de A à B à travers le labyrinthe

Labyrinthe

A nouveau le problème peut être modélisé par un graphe :

- Le labyrinthe peut être représenté par un graphe similaire au graphe pour la représentation d'un plan
- Le problème consiste à trouver un chemin de A à B, par un parcours systématique du graphe.

Parcours de graphe

- Il existe de nombreux algorithmes de parcours systématiques de tous les sommets du graphe.
- Il y a deux stratégies de parcours différentes : partant d'un sommet, le graphe est parcouru
  - en largeur
  - en profondeur

Parcours en largeur

Breadth-First-Search (BFS)

Parcours en largeur

- Le parcours en largeur consiste à parcourir d'abord tous les voisins d'un sommet donné, puis on parcourt les voisins des voisins, etc.
- Le parcours se fait en "largeur" avant de se faire en "profondeur"
- De nombreux algorithmes sont basés sur cette stratégie, par exemple l'algorithme de Dijkstra pour trouver les chemins les plus courts

Structure de données pour le parcours en largeur

- Structure de données pour se rappeler les sommets qui n'ont pas été complètement pris en compte :

  File : puisque chaque nouveau sommet visité est positionné à la fin de la file d'attente; les sommets les premiers visités sont les premiers supprimés de la file.

- Structure de données pour représenter le graphe afin d'obtenir une mise en œuvre efficace:

  Liste d'adjacence, puisque les voisins de chaque sommet sont systématiquement visités

Parcours en largeur

Pour chaque sommet on calcul les informations suivantes :

- **dist**: la distance d'un sommet au sommet de départ (le sommet où commence la recherche)
- **pred**: le prédécesseur, c'est-à-dire le sommet depuis lequel le sommet actuel a été atteint la première fois
- **coul**: une des couleurs blanc, gris, noir

Parcours en largeur

Signification des couleurs:

- Blanc : le sommet n'a pas encore été visité
- Gris : le sommet a été visité, mais tous ses voisins ne l'ont pas encore été
- **Noir** : le sommet et tous ses voisins ont été visités

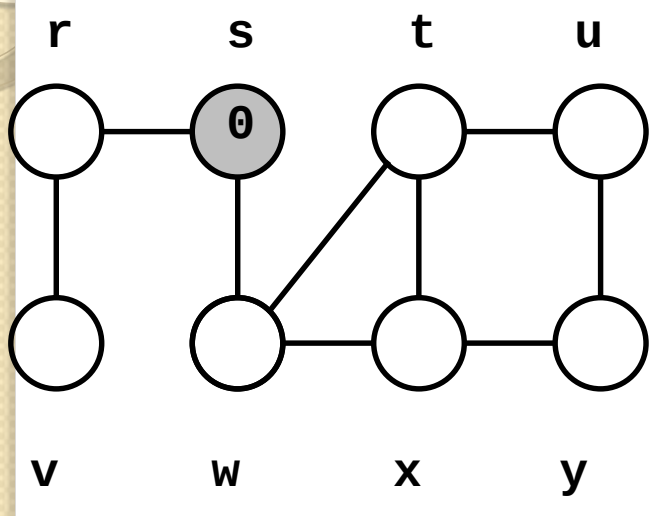
Algorithme du parcours en largeur

```
Algorithm parcours-largeur(s)
    s : le sommet de départ,
    Q : une file
     init-parcours-largeur(s)
     tantque non Q.fileVide
        Q.defiler(u)
        pour tout sommet v adjacent à u faire
           si (coul[v] = blanc)  // v pas encore visité
           alors   coul[v] ← gris
                   dist[v] ← dist[u] + 1
                   pred[v] ← u
                   Q.enfiler(v)
           fsi
        fpour
        coul[u] ← noir;
     ftq
```

Algorithme du parcours en largeur

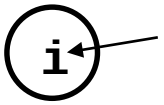
```
init-parcours-largeur(s)
 s : le sommet de départ,
 Q : une file

 coul[s] ← gris
 dist[s] ← 0
 pred[s] ← NIL
 Q.enfiler(s)
 pour tout sommet v ≠ s faire
        coul[v] ← blanc
        dist[v] ← MAXINT
        pred[v] ← NIL
 fpour
```

Parcours en largeur



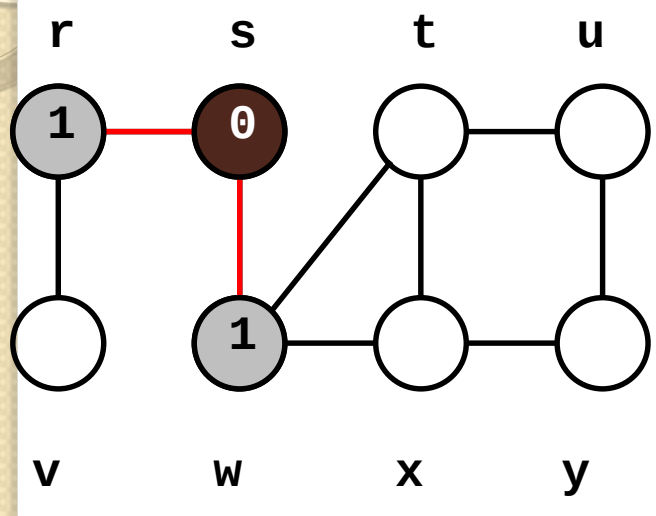
Q

s



La valeur placée dans chaque sommet est dist, pas de valeur signifie MAXINT

Parcours en largeur



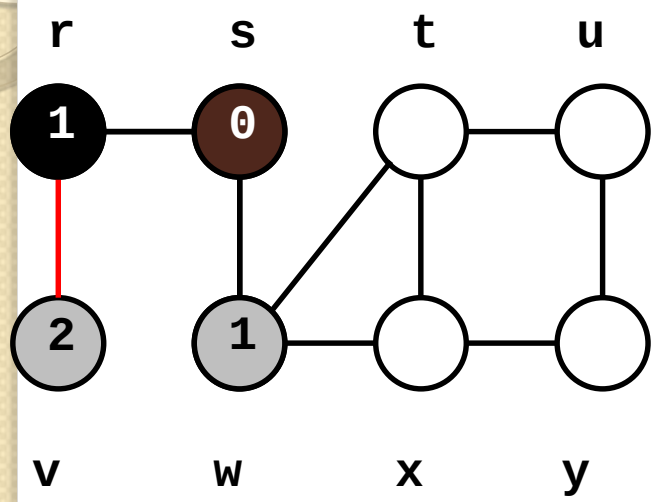
Q

r	w

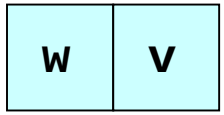
Visiter le sommet `s`:

Parcourir tous les voisins de `s`

Parcours en largeur



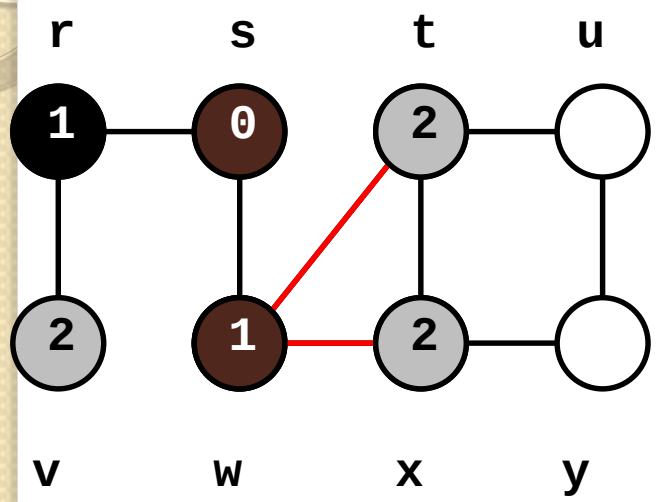
Q



Visiter le sommet `r`:

Parcourir tous les voisins de `r`

Parcours en largeur



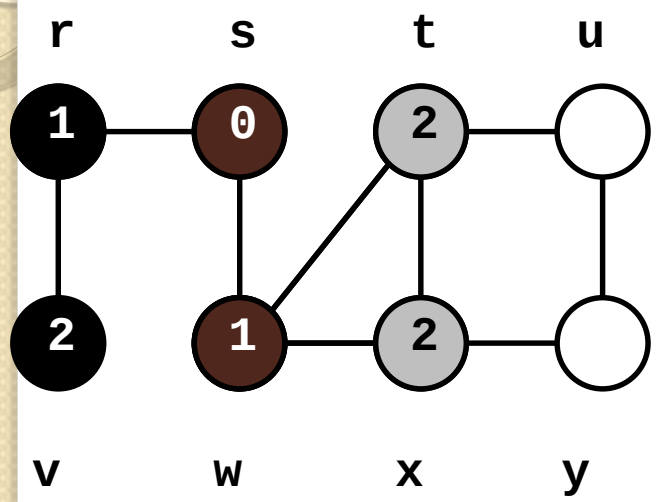
Q

v	t	x

Visiter le sommet `w`:

Parcourir tous les voisins de `w`

Parcours en largeur



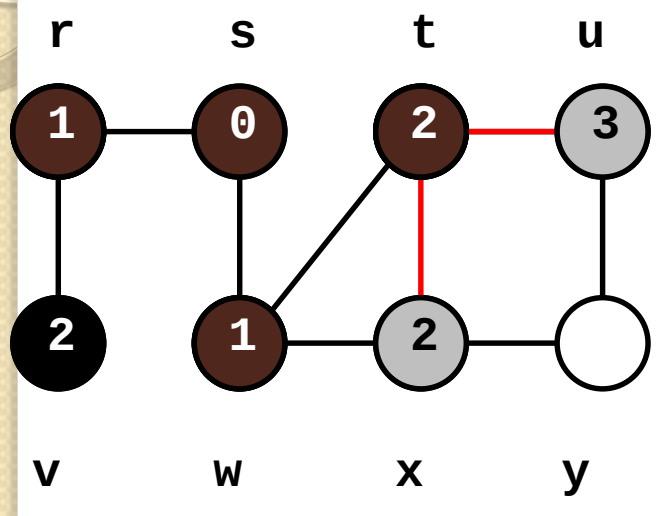
Q

t	x

Visiter le sommet v:

Parcourir tous les voisins de v

Parcours en largeur



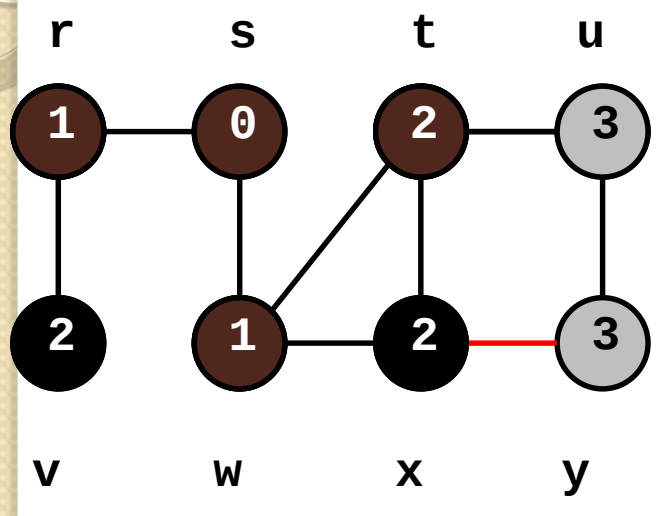
Q

x	u

Visiter le sommet `t`:

Parcourir tous les voisins de `t`

Parcours en largeur



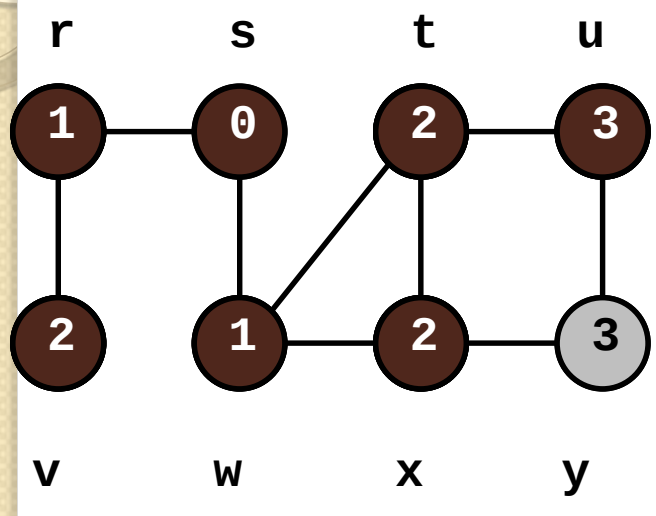
Q

u	y

Visiter le sommet x:

Parcourir tous les voisins de x

Parcours en largeur



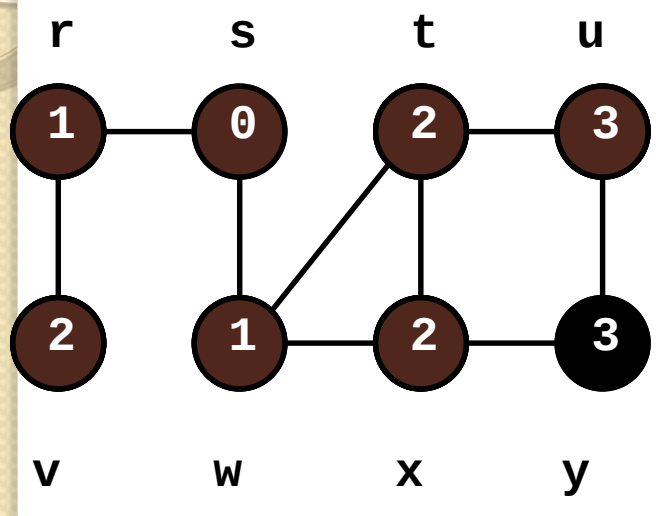
Q

y

Visiter le sommet u:

Parcourir tous les voisins de u

Parcours en largeur



Q

Visiter le sommet y:

Parcourir tous les voisins de y

Exécution de l‘algorithme dans une table

som.	r	s	t	w	u	v	x	y
pred								
dist								

Exécution de l‘algorithme dans une table

som.	r	s	t	w	u	v	x	y
pred	s	NIL	w	s	t	r	w	x
dist	1	0	2	1	3	2	2	3

Trouver un chemin de s à u:

s → w → t → u

Temps d'exécution du parcours en largeur

Pour un graphe de **`n`** sommets et **`m`** arêtes :

- Initialiser les information **`col`**, **`dist`** and **`pred`** pour tous les sommets : **`O(n)`**
- Chaque sommet est place une seule fois dans la queue; temps pour tous les sommets : **`O(n)`**
- Chaque sommet est supprimé une seule fois de la file; temps pour tous les sommets : **`O(n)`**
- Pour chaque sommet supprimé de la file, on examine tous les voisins; temps d'exécution: **`O(m)`**

Temps d'exécution pour toutes ces étapes est **`O(n+m)`**

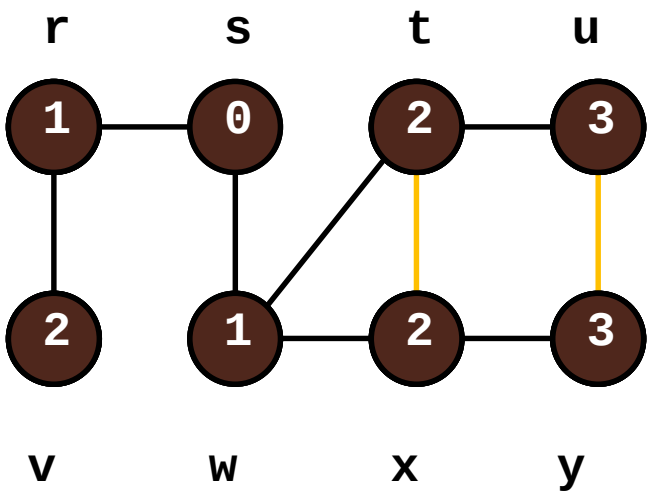
Arbre de parcours en largeur

Le parcours en largeur génère un arbre :

- La racine est le sommet de départ
- Les noeuds de l'arbre sont les sommets du graphe qui peuvent être atteints depuis le sommet de départ.
- Les arêtes de l'arbre sont celles du graphe, qui relient un sommet à son prédécesseur (**`pred`**) au cours du parcours

Remarque: tous les sommets du graphe ne doivent pas appartenir à l'arbre car il peut exister des sommets inaccessibles !

Arbre de parcours en largeur



Arêtes de l'arbre de parcours en largeur

Arcs du graphe qui n'appartiennent pas à l'arbre de parcours en largeur

Distance du plus court chemin

Definition:

La ***distance du plus court chemin*** **δ(s,v)** d'un sommet **s** à un sommet **v** est définie ainsi :

$$\delta(s,v) = \begin{cases} \textbf{min\{Dist(w)} \mid \textbf{w} \text{ est un chemin de } \textbf{s} \text{ à } \textbf{v} \text{ dans } \textbf{G} \text{ et } \textbf{Dist(w)} \text{ est le nombre d'arcs sur ce chemin}\textbf{\}} \\ \text{l'infini s'il n'existe pas de chemin de s à v} \end{cases}$$

Propriétés du parcours en largeur

Considérons l'information **`dist`** calculée pendant le parcours.

On a : **`dist(v) = δ(s,v)`**

C'est-à-dire que le parcours en largeur détermine les distances les plus courtes entre le sommet de départ et tous les autres sommets

Les chemins les plus courts peuvent être déterminés à l'aide de l'information **`pred`**

La preuve de cette propriété peut être trouvée dans le livre :

Algorithmique - 3ème édition

Thomas H. Cormen, Charles Leiserson, Ronald Rivest, Clifford Stein
Cours avec 957 exercices et 158 problèmes – Dunod juin 2010

Propriétés de l'arbre de parcours en largeur

Les chemins de **l'arbre** de parcours en largeur de **s** vers les autres sommets, sont les chemins les plus courts (en nombre d'arêtes) dans le graphe **G,** de **s** vers tous les autres sommets.

Parcours en profondeur

Parcours en profondeur

Le parcours en profondeur correspond à une autre une stratégie de parcours de graphe : en visitant un voisin **v** d'un sommet **u**, on visite d'abord les voisins de **v** avant de visiter les autres voisins de u.

Le parcours se fait d’abord en “profondeur”

Plusieurs algorithmes sont basés sur ce parcours

Structure de données pour le parcours en profondeur

- Une structure de données utile est la **pile**
- On peut cependant remplacer celle-ci par des appels récursifs.
- Structure de données pour représenter le graphe afin d'obtenir une mise en œuvre efficace:

  Liste d'adjacence, puisque les voisins de chaque sommet sont systématiquement visités

Parcours en profondeur

Dans ce parcours les sommets sont colorés de la même manière que dans le parcours en largeur :

- Blanc : le sommet n'a pas encore été visité
- Gris : le sommet a été visité, mais tous ses voisins n'ont pas encore été visités
- **Noir** : le sommet et tous ses voisins ont été visités

Parcours en profondeur

Pour chaque sommet on calcule les informations suivantes :

- **`col`**: la couleur associée au sommet
- **`first`**: un “horaire” qui indique quand un sommet a été visité en premier, dans ce cas il prend la couleur grise
- **`last`**: un “horaire” qui indique quand un sommet a été visité en dernier, dans ce cas il prend la couleur noire (on a : **`first[v] < last[v]`**)
- **`pred`**: le prédécesseur, sommet depuis lequel le sommet a été visité en premier

Algorithme de parcours en profondeur

Algorithme du parcours en profondeur de G

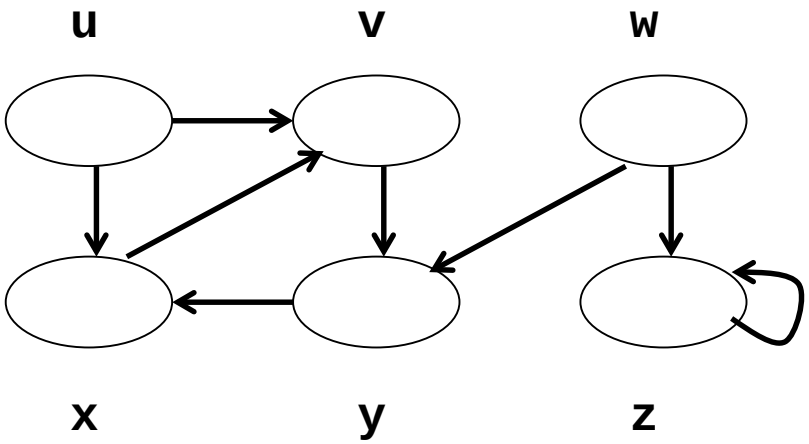
```
heure : variable globale initialisée à 0
pour chaque sommet u de G
        col[u] ← blanc
        pred[u] ← NIL
fpour
pour chaque sommet u de G
        si (col[u] = blanc)
        alors  parcoursProf (u)  //action récursive
        fsi
fpour
```

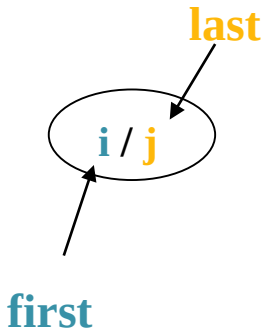
Algorithme de parcours en profondeur

Algorithme de parcoursProf (u)

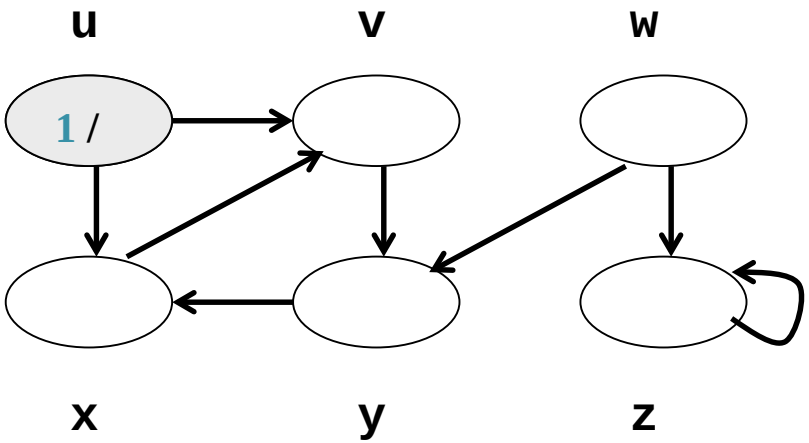
```
col[u] ← gris            // u visité pour la première fois
heure ← heure + 1
first[u] ← heure
pour tout sommet v adjacent à u faire
    si (col[v] = blanc)   // v pas encore visité
    alors    pred[v] ← u;
             parcoursProf (v);
    fsi
fpour
col[u] ← noir            // tous les voisins visités
heure ← heure + 1
last[u] ← heure
```

Exemple de de parcours en profondeur

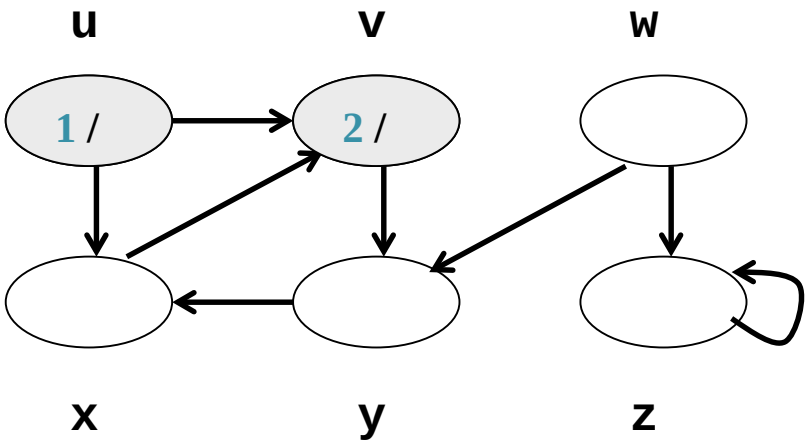




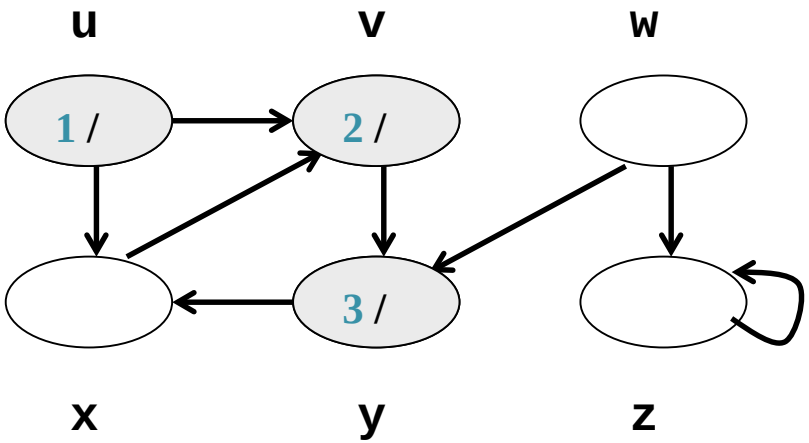
Exemple de de parcours en profondeur



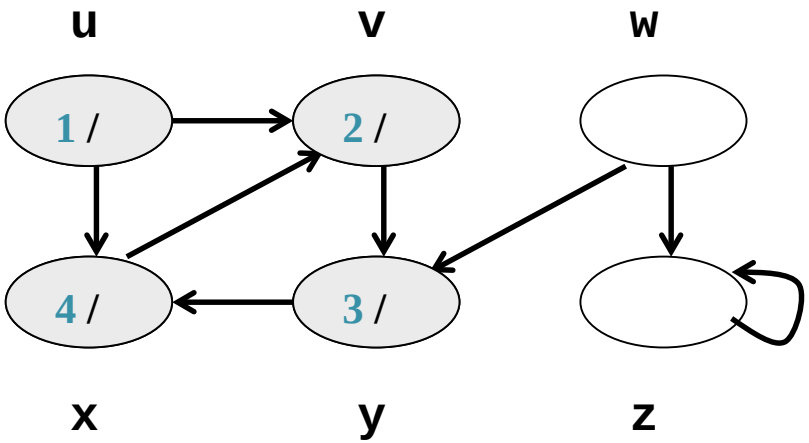
Exemple de de parcours en profondeur



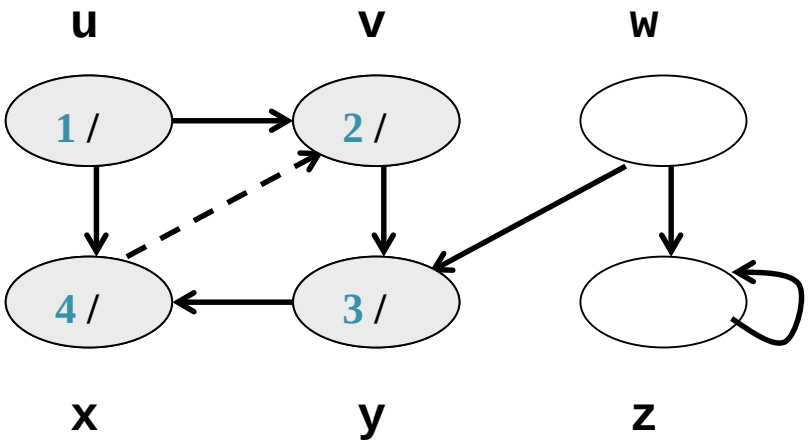
Exemple de de parcours en profondeur



Exemple de de parcours en profondeur

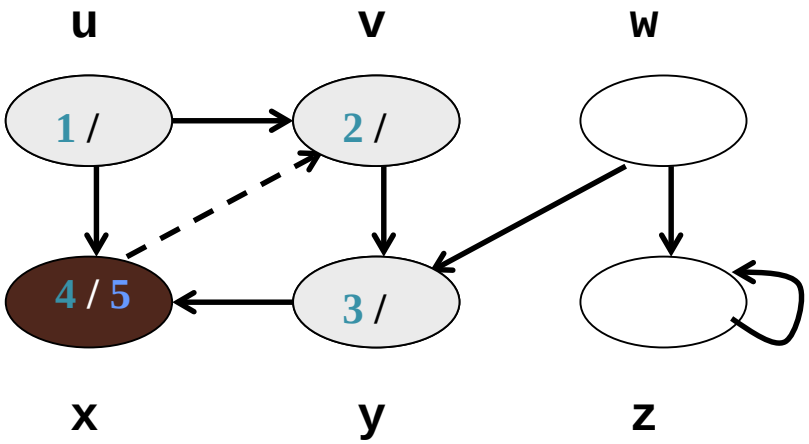


Exemple de de parcours en profondeur

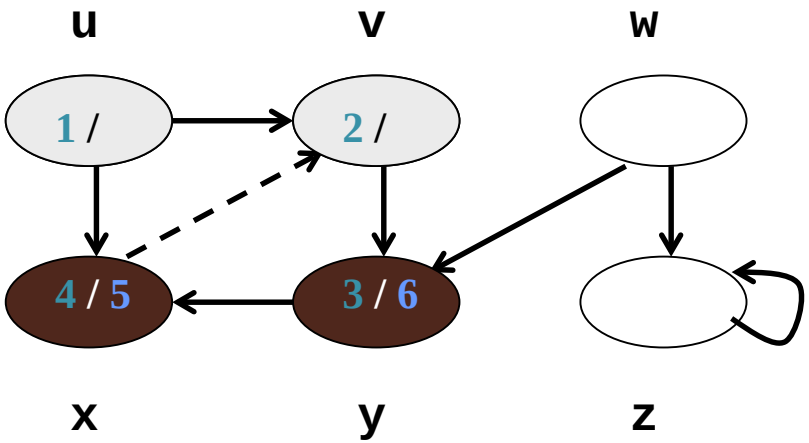


Arc visité vers un sommet déjà visité

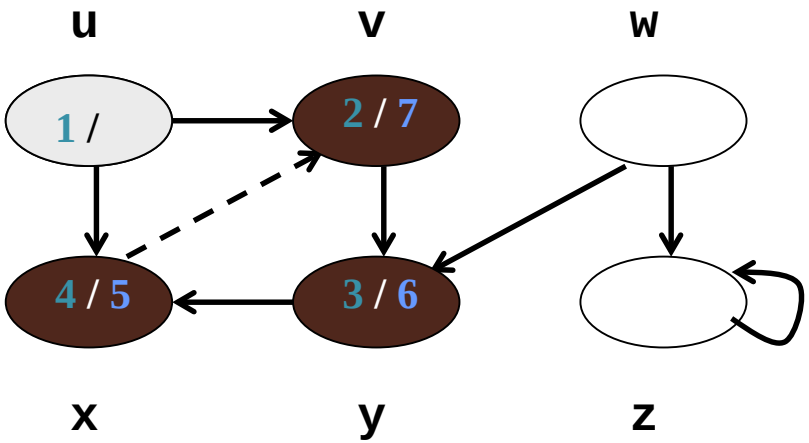
Exemple de de parcours en profondeur



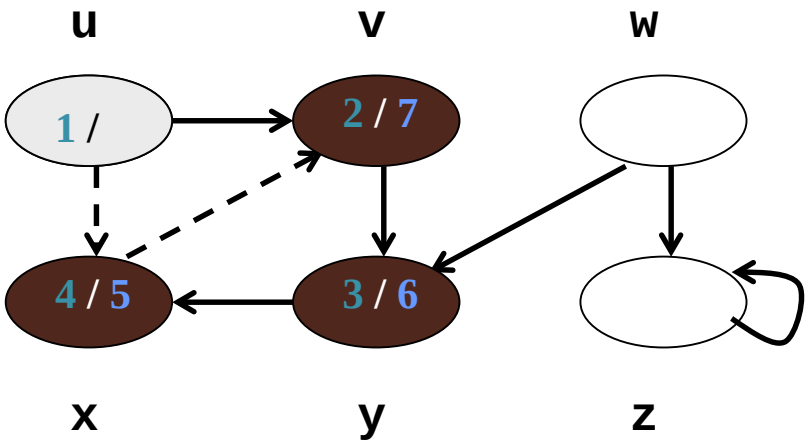
Exemple de de parcours en profondeur



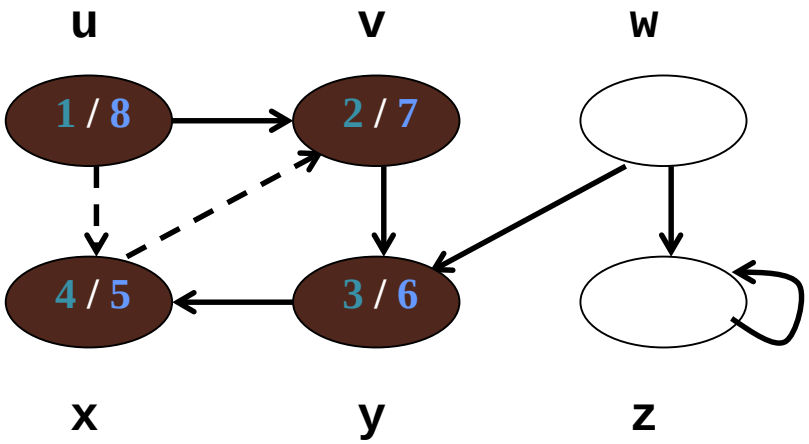
Exemple de de parcours en profondeur



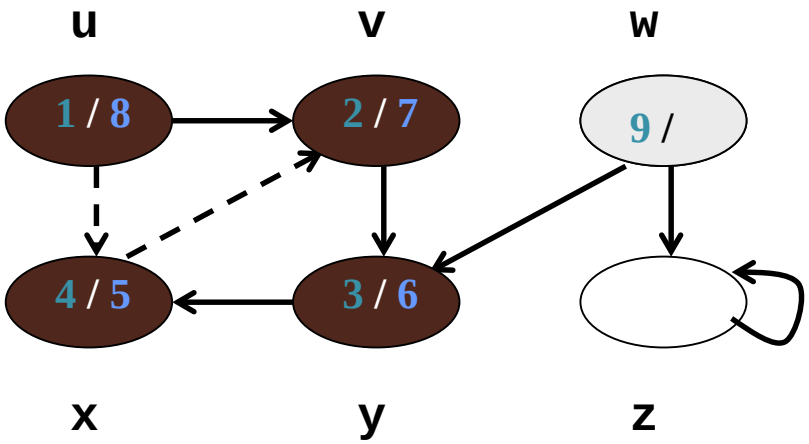
Exemple de de parcours en profondeur



Exemple de de parcours en profondeur

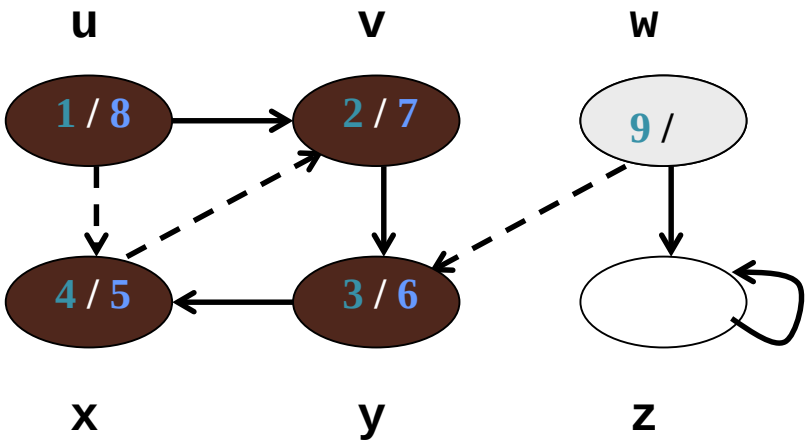


Exemple de de parcours en profondeur

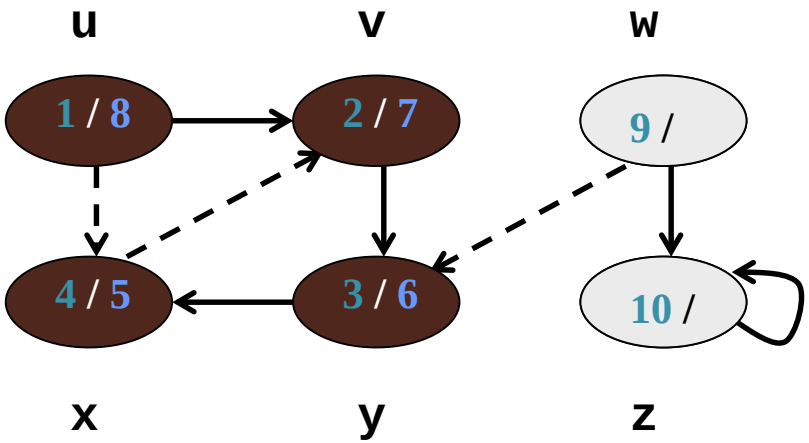


ici parcoursProf est appelée depuis l'algorithme principal

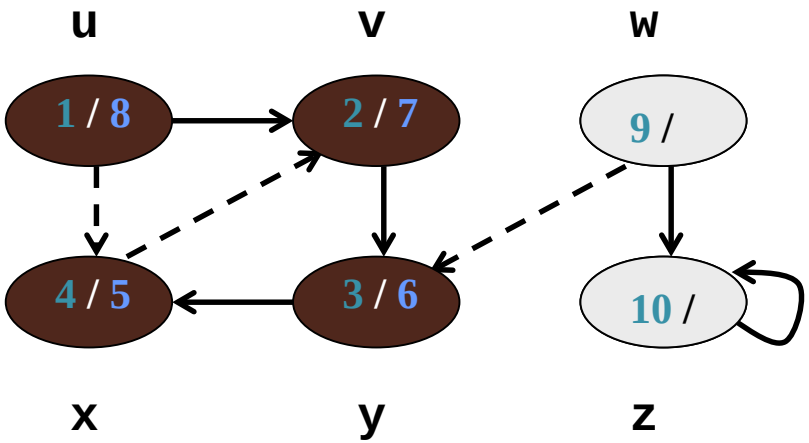
Exemple de de parcours en profondeur



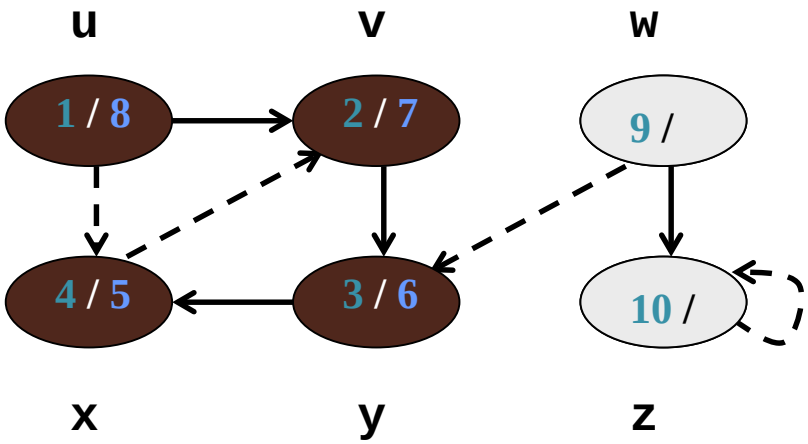
Exemple de de parcours en profondeur



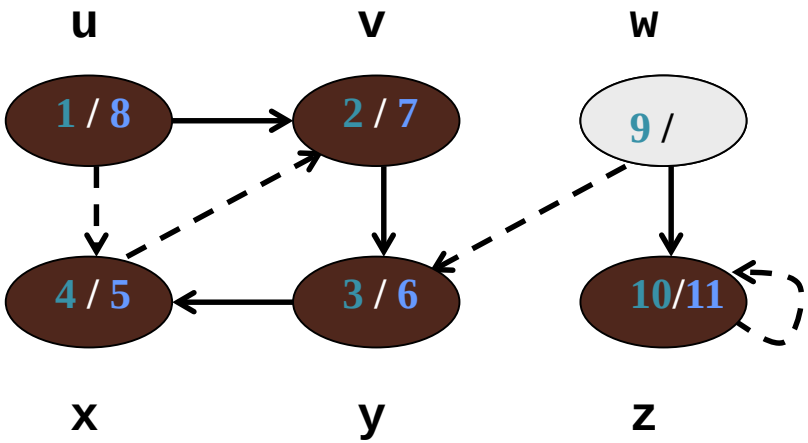
Exemple de de parcours en profondeur



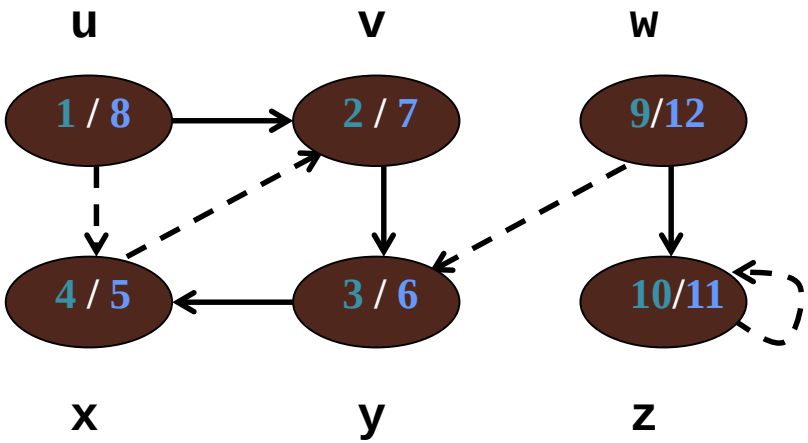
Exemple de de parcours en profondeur



Exemple de de parcours en profondeur



Exemple de de parcours en profondeur



Exécution dans une table

Som.	u	v	w	x	y	z
pred						
first						
last						

Exécution dans une table

Som.	u	v	w	x	y	z
pred	NIL	u	NIL	y	v	w
first	1	2	9	4	3	10
last	8	7	12	5	6	11

Temps d'exécution du parcours en profondeur

Pour un graphe de **n** sommets et **m** arêtes/arcs :

Algorithme principal :

- Temps d'initialisation des sommets : **O(n)**
- Nombre d'appels de parcoursProf: au plus **O(n)** puisque parcoursProf est seulement appelée pour les sommets blancs; si un sommet est visité une première fois, il obtient la couleur grise.

Algorithme récursif parcoursProf :

- Pour chaque sommet pour lequel parcoursProf est appelée, la liste d'adjacence est parcourue; en considérant tous les appels de parcoursProf: le temps de parcours est **O(m)**
- Ainsi, le temps d'exécution de l'algorithme de parcours en profondeur est **O(n+m)**

Applications du parcours en profondeur

- Tri topologique
- Composantes fortement connexes

Tri topologique

Problème du monde réel: exécuter des travaux avec des contraintes d'ordonnancement

Une personne doit effectuer **n** tâches;

Parfois il est nécessaire qu'une tâche soit faite avant une autre (les pommes de terre doivent être épluchées avant d'être frites)

Ces dépendances (ou contraintes) peuvent être modélisées par un graphe.

Tri topologique

Modélisation du problème par un graphe :

Tes tâches sont représentées par les sommets du graphe

Un arc **e** relie le sommet **v** au sommet **w** si la tâche correspondant à **v** doit être effectuée avant la tâche correspondant à **w**

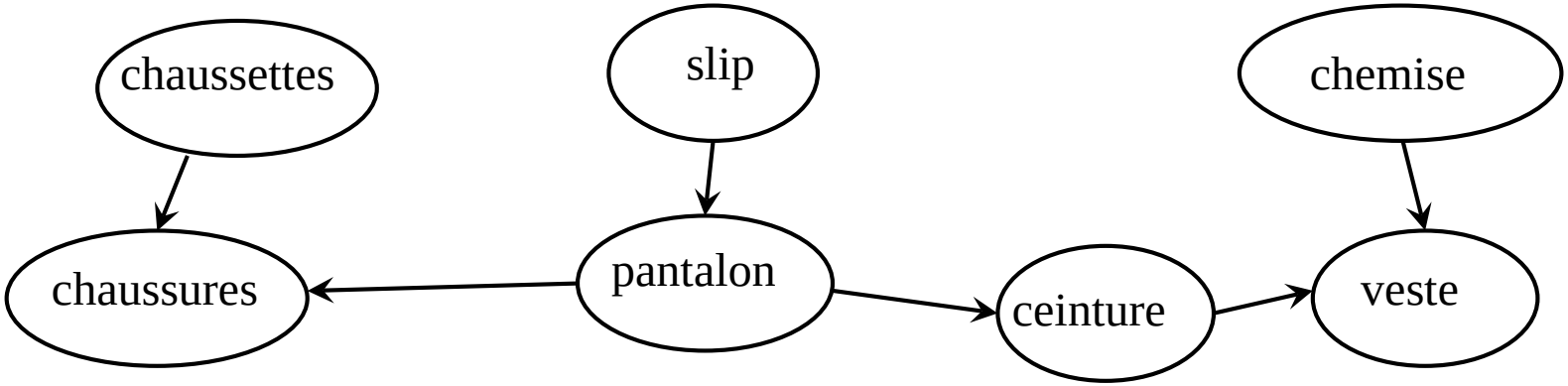
Tri topologique

Données : un graphe orienté **G = (V,E)**

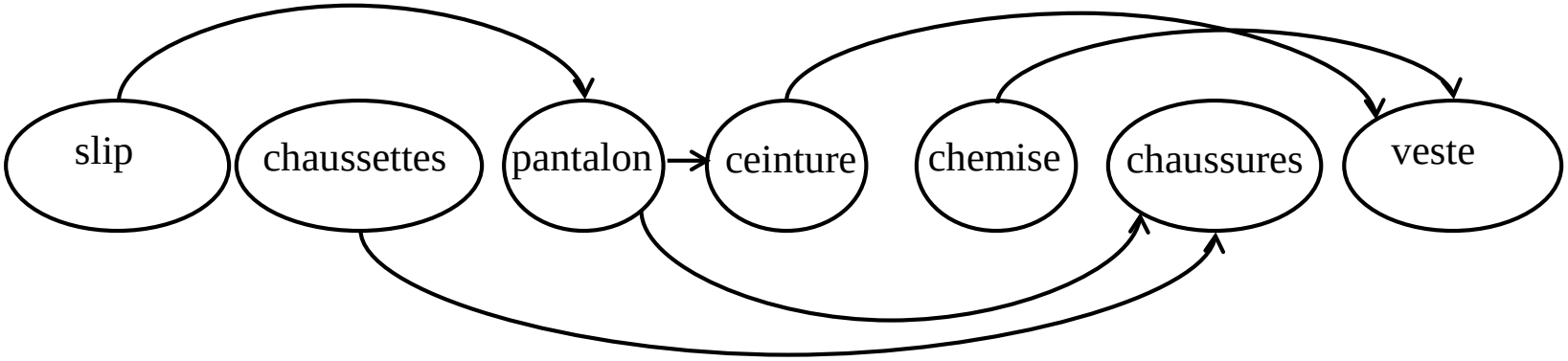
Un tri topologique est un ordonnancement linéaire des sommets de sorte que, pour chaque arc **(u,v)** de **E**, **u** apparaisse avant **v** dans l'ordonnancement.

Une tri topologique d'un graphe orienté G peut être considérée comme un ordonnancement des sommets le long d'une ligne horizontale, de sorte que tous les arcs soient dirigés de la gauche vers la droite

Exemple : “Graphe d’habillage”



Tri topologique:



Algorithme de tri topologique

Algorithme du tri topologique du graphe G:

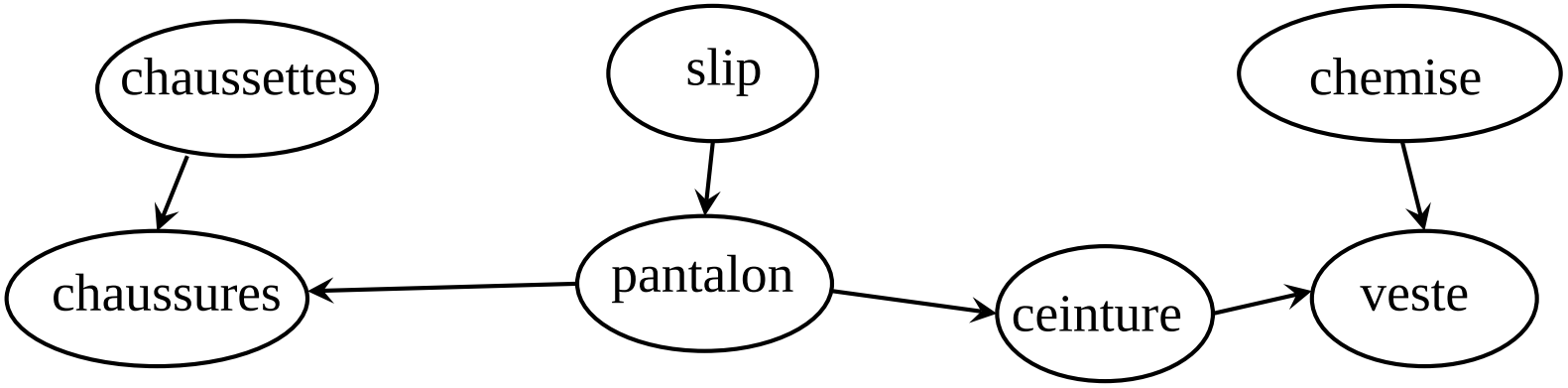
Parcours en profondeur de G

Trier les sommets en ordre décroissant selon leur valeur **`last`**

Exemple

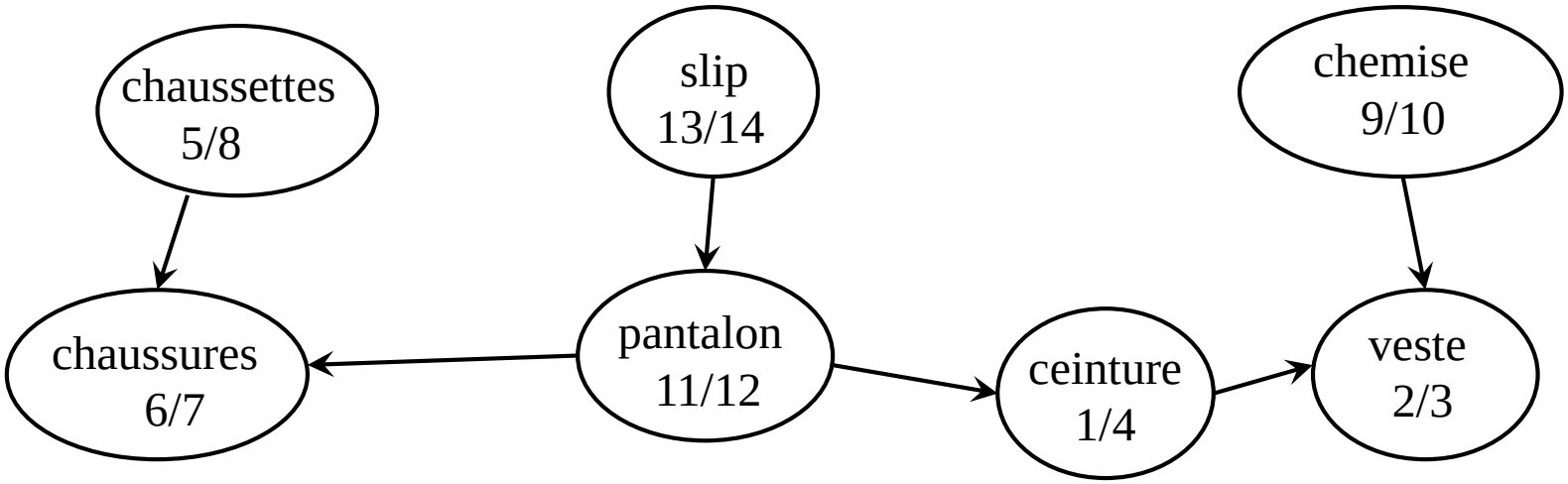
Trouver un tri topologique pour le “Graphe d’habillage”

S’il y a plusieurs choix pour un sommet, choisir le plus petite dans l’ordre lexical ; commencer par la ceinture.

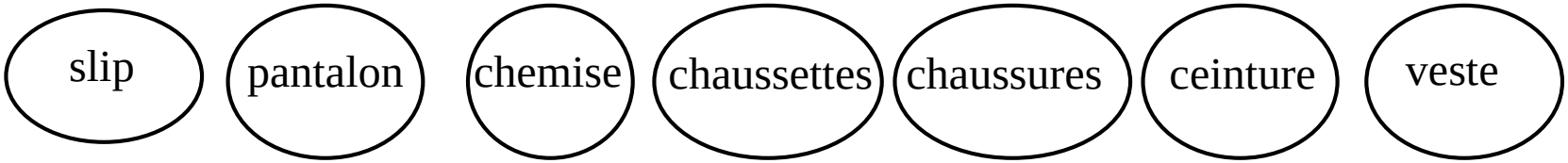


Exemple

Parcours en profondeur du “graphe d’abillage”



Tri topologique



Composantes fortement connexes

Une autre application du parcours en profondeur est de déterminer les composantes fortement connexes d'un graphe.

Pour plus de details voir le livre :

Algorithmique - 3ème édition

Thomas H. Cormen, Charles Leiserson, Ronald Rivest, Clifford Stein Cours avec 957 exercices et 158 problèmes – Dunod juin 2010