Graphes

Stéphane Grandcolas

Aix-Marseille Université

2023-2024

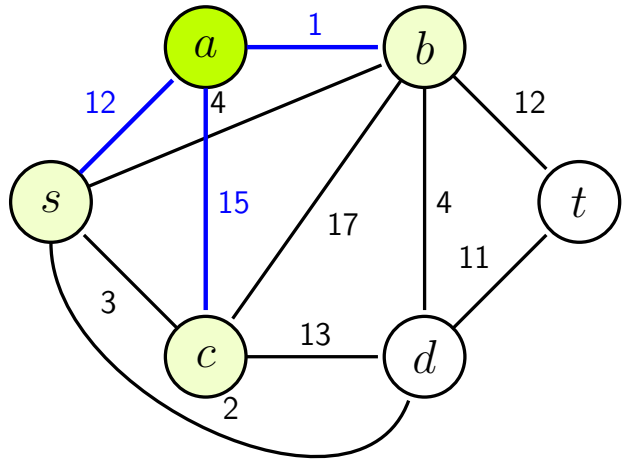
Graphes

Plan du cours :

- exemples, définitions,
- parcours en profondeur,
- plus courts chemins : Dijkstra,
- plus courts chemins : Bellman-Ford,
- arbres couvrants de poids minimal (ACM) : (Prim, Kruskal),

Graphes

Représentent une relation, une distance.



Souvent utilisés pour modéliser des problèmes

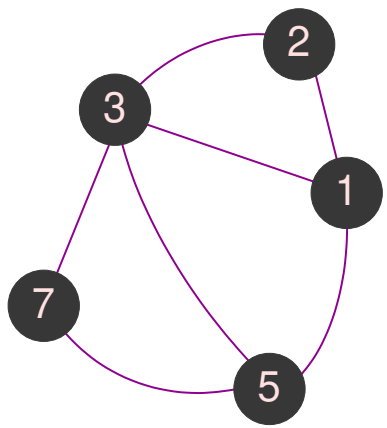
- calcul de plus courts chemins,
- calcul de flots minimaux,
- calcul d'arbres couvrants de poids minimal,
- coloriage,
- ...

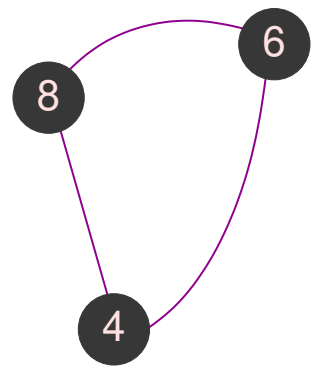
Graphe (non orienté)

Un graphe est défini par ses **sommets** et ses **arêtes**

Graphe $G = (S, A)$

- S : ensemble des sommets,
- A : ensemble des arêtes, $A = \{\{u, v\} | u, v \in S\}$





(1,2) (2,3) (1,3) (1,5) (3,7) (4,6) (8,4) (7,5) (3,5) (8,6)

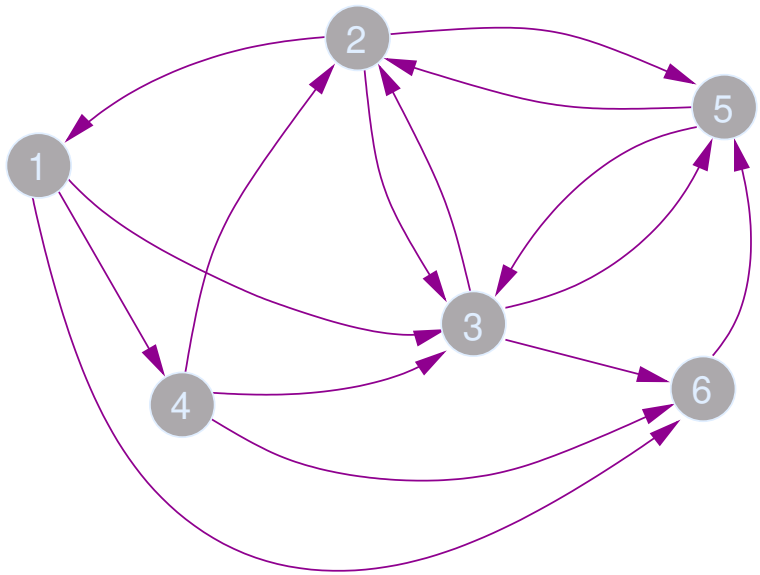
La représentation graphique d'un graphe aide à le visualiser.

Chaque arête est schématisée par une ligne entre ses deux sommets.

Graphe orienté

Défini par ses sommets et ses arcs

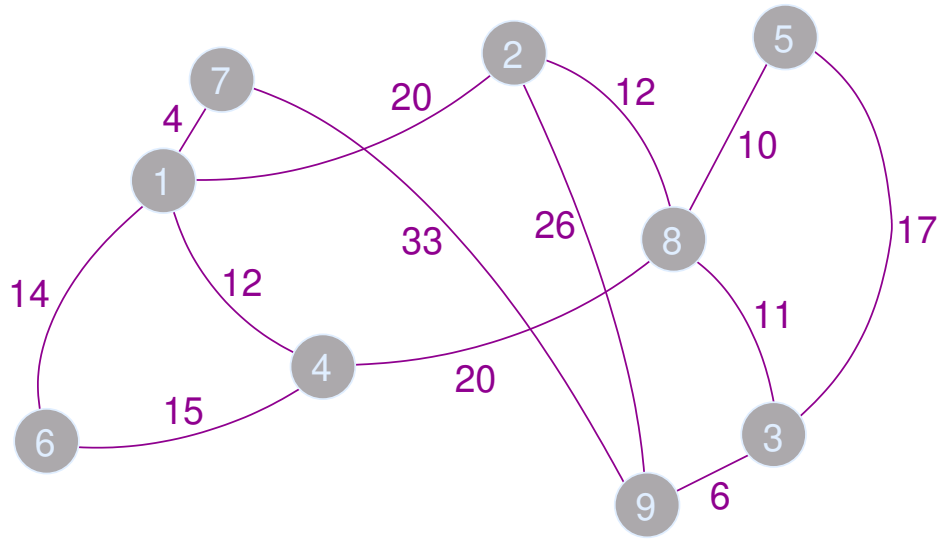
$G = (S, A)$, avec $A = \{(u, v) | u, v \in S\}$ un ensemble d'*arcs*



Graphe pondéré

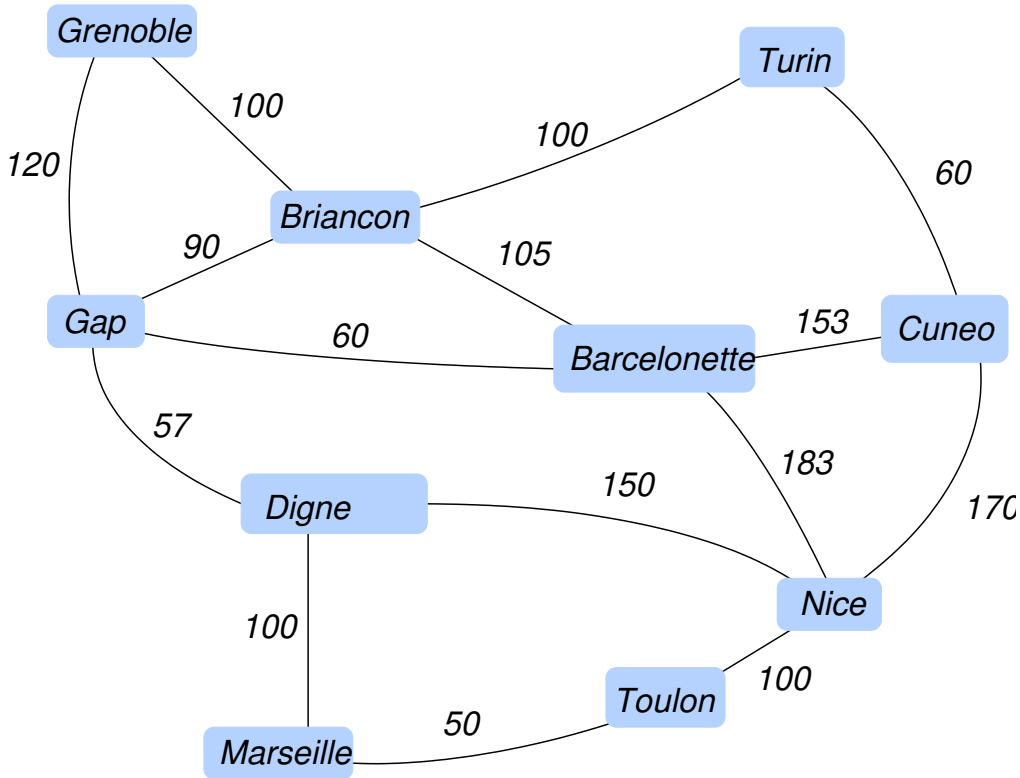
Chaque arête (resp. arc) a un **poids**

$G = (S, A, w)$: $w(u, v)$ est le poids de l'arête (u, v).



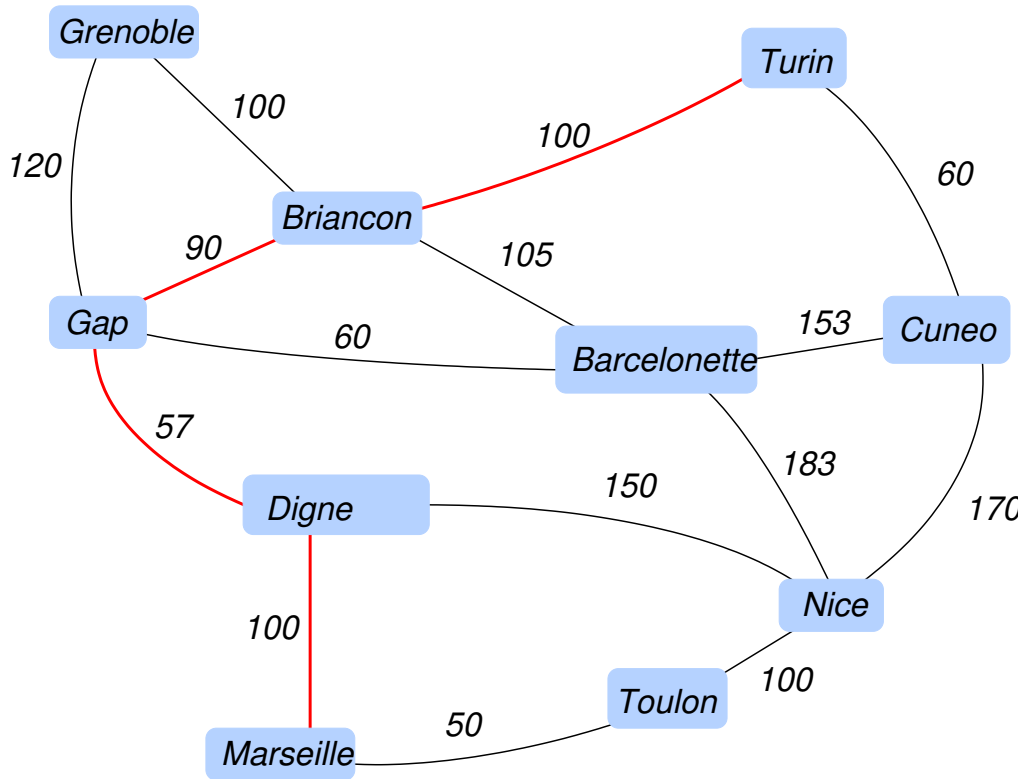
Dans le cas d'un graphe orienté chaque arc est pondéré.
$w(u, v)$ est le poids de l'arc (u, v), $w(v, u)$ est le poids de l'arc (v, u).

Exemple : trajet le plus court entre deux villes



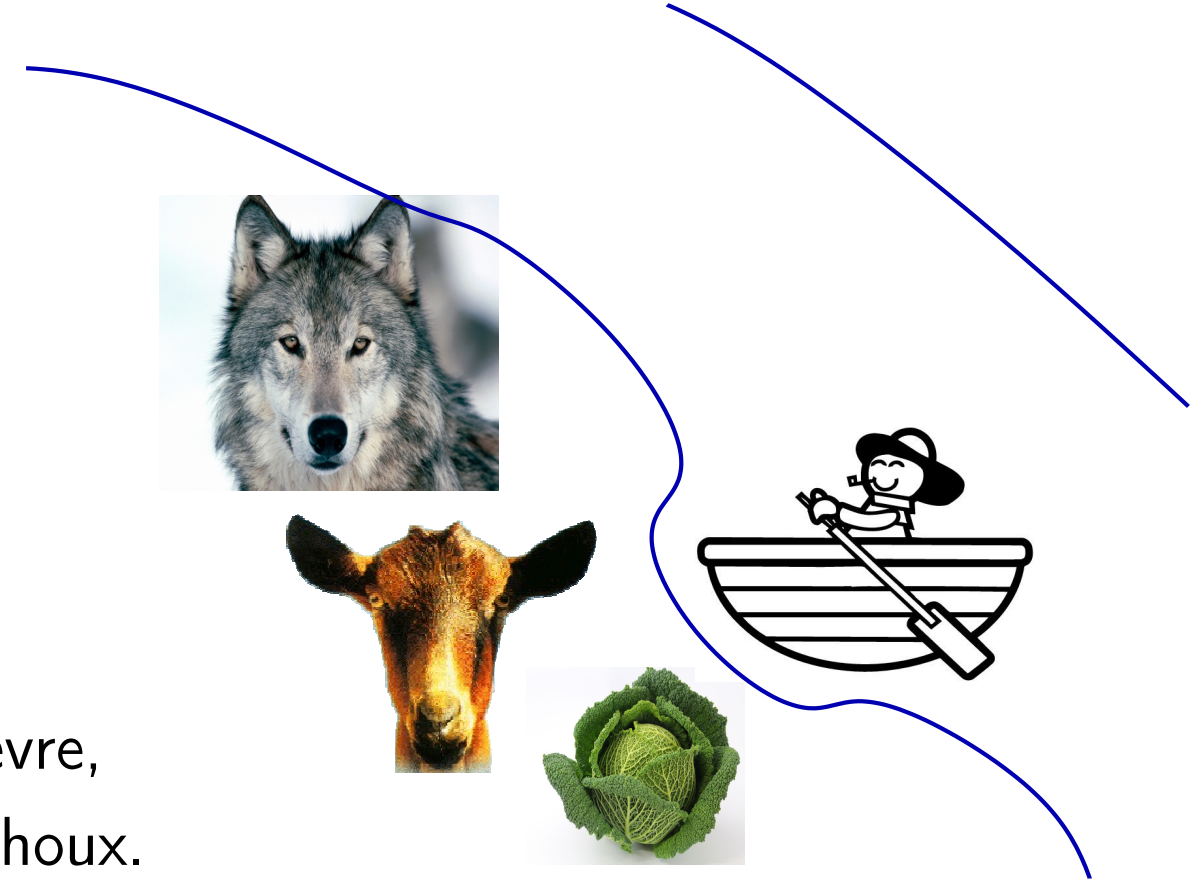
Graphe représentant les liaisons routières et les durées des trajets

Exemple : trajet le plus court entre deux villes



Calcul d'un **plus court chemin** entre Marseille et Turin

Exemple : le choux, la chèvre et le loup



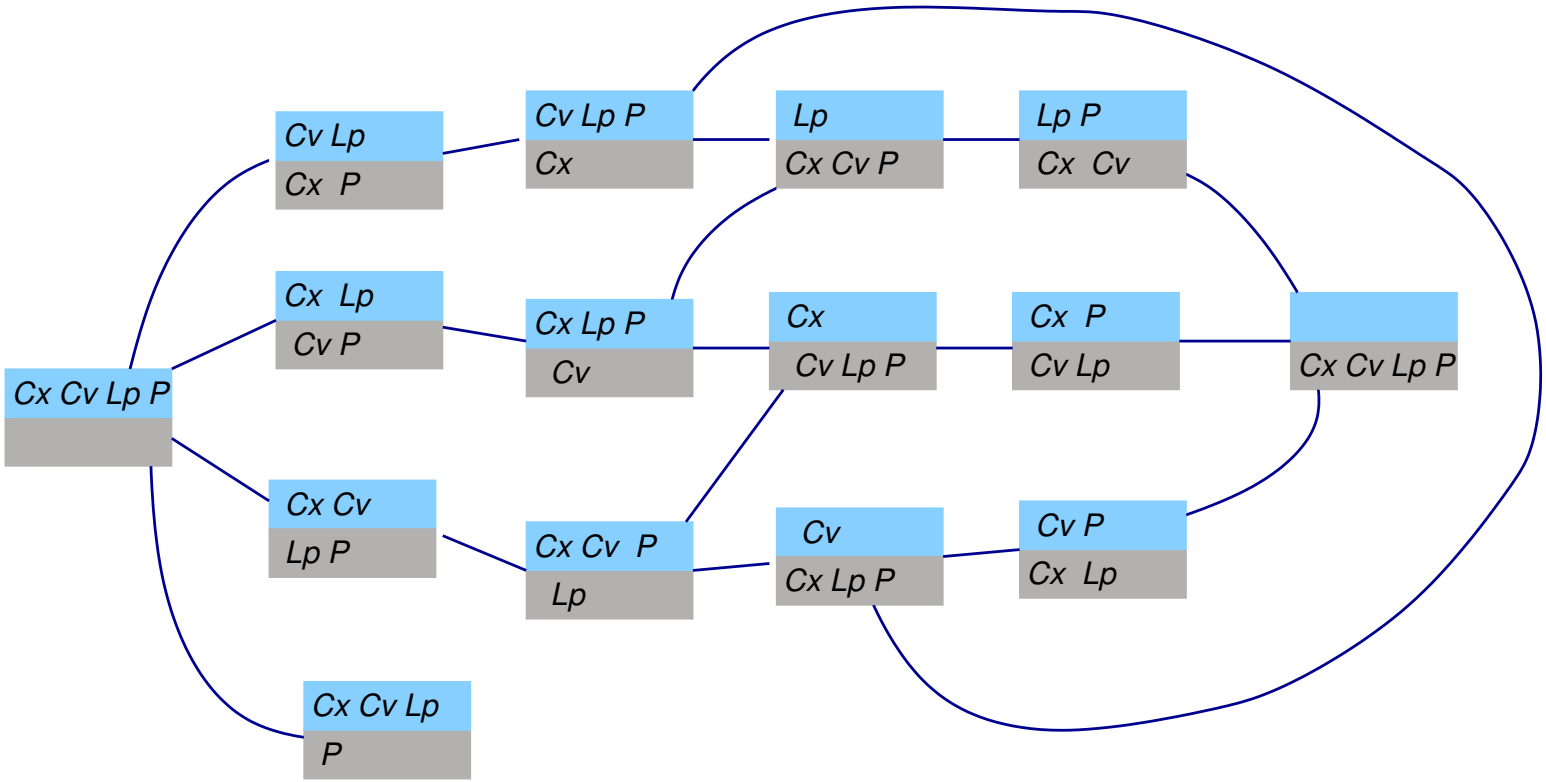
Conflits :

- loup et chèvre,
- chèvre et choux.

Problème : est-il possible de faire traverser la rivière au choux, à la chèvre et au loup, sans perte ?

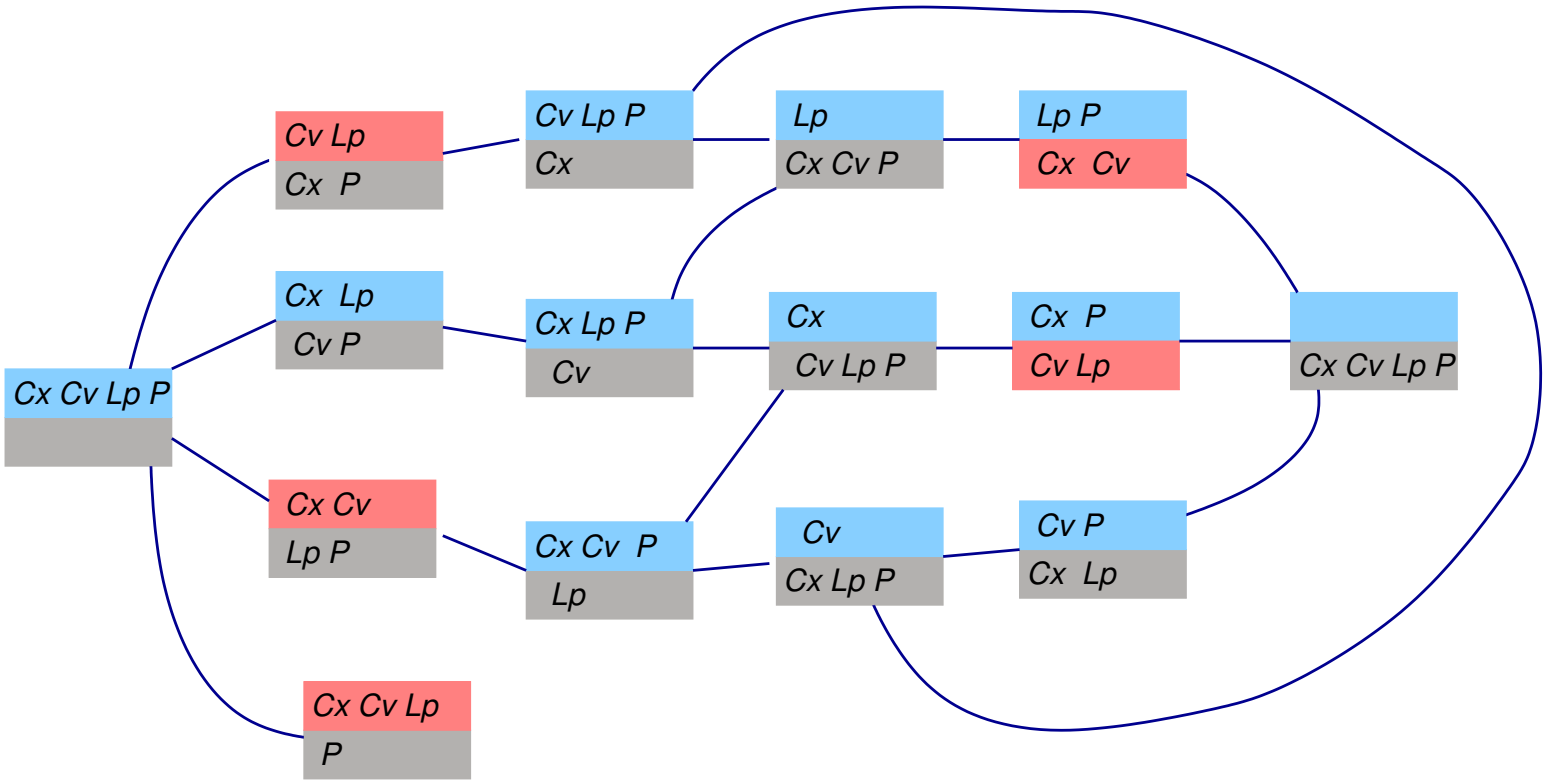
On modélise le problème par un calcul de plus court chemin dans un graphe.

Exemple : le choux, la chèvre et le loup



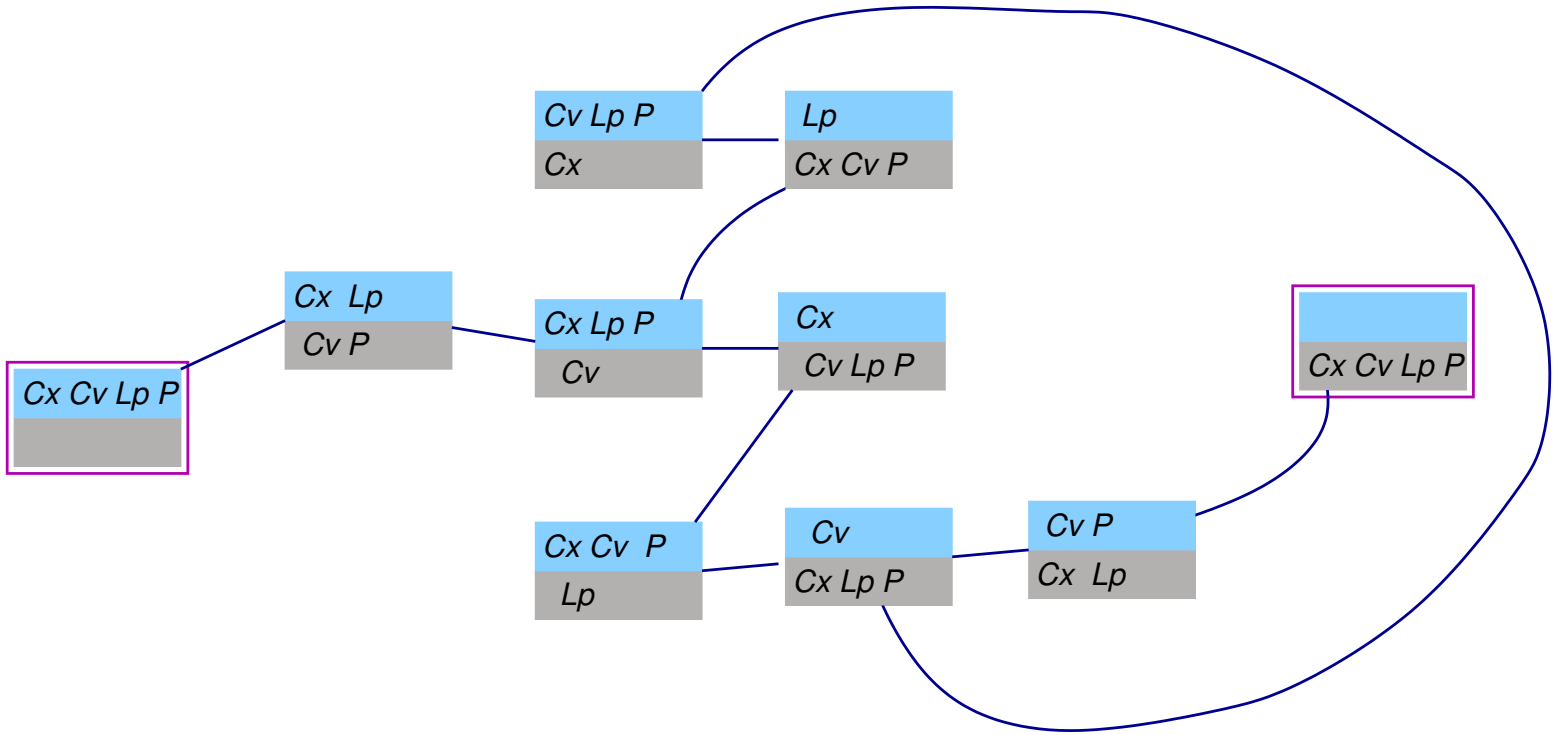
Graphe représentant les états et les transitions

Exemple : le choux, la chèvre et le loup



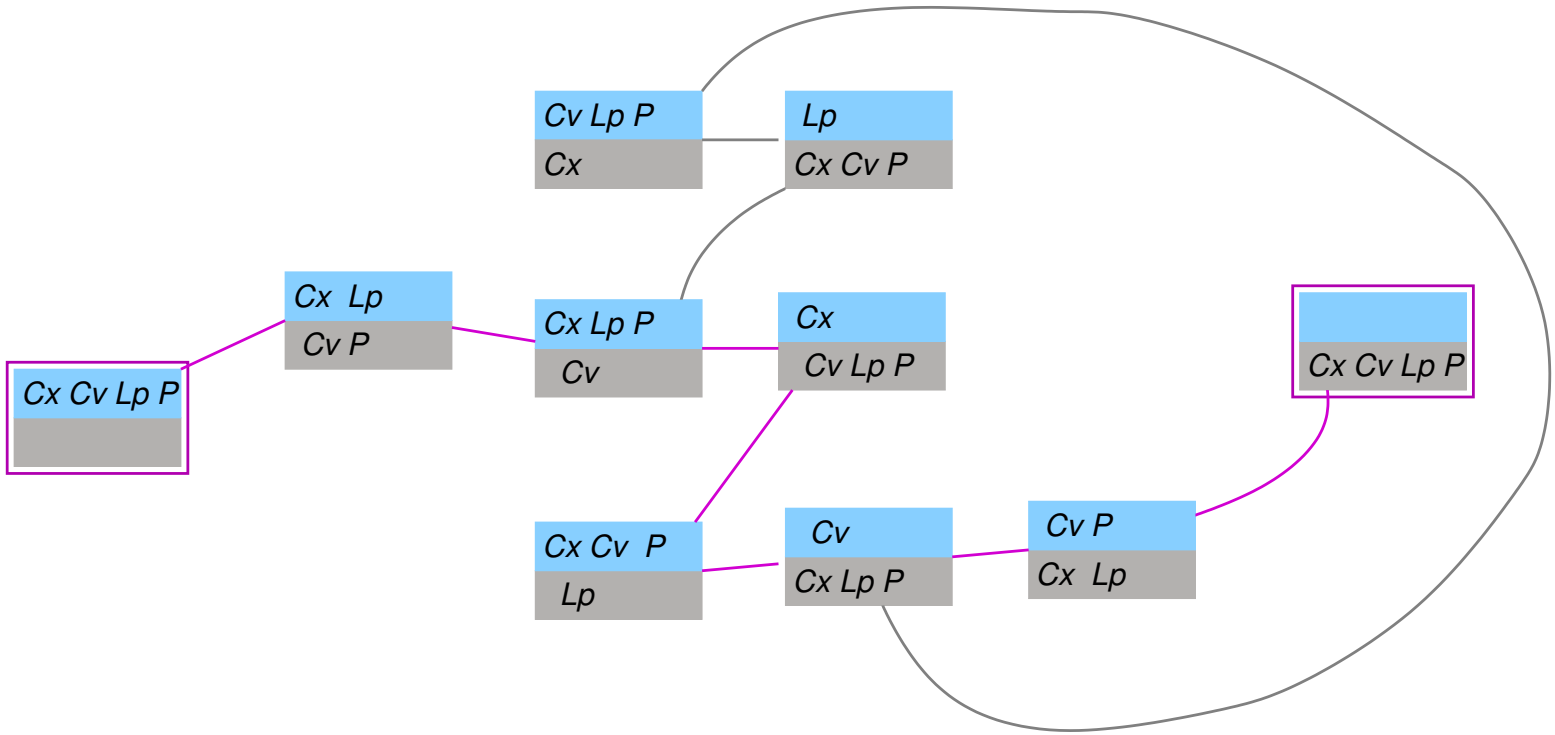
Elimination des états conflictuels

Exemple : le choux, la chèvre et le loup



Problème : trouver un chemin dans le graphe entre l'état initial et l'état final, avec le moins d'arêtes possible.

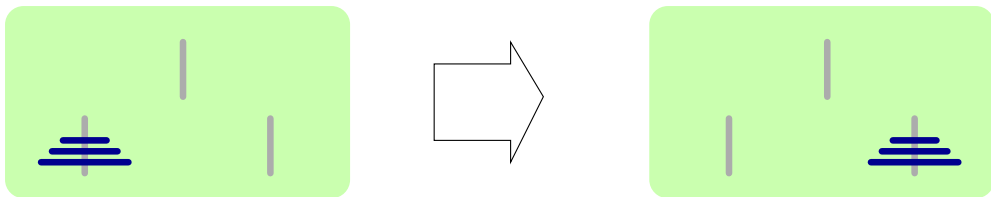
Exemple : le choux, la chèvre et le loup



Un algorithme de calcul de plus court chemin (Dijkstra par exemple) produit une solution avec le moins de traversées possible.

Exemple : problème des tours de Hanoï

Trois tours, n disques de tailles toutes différentes

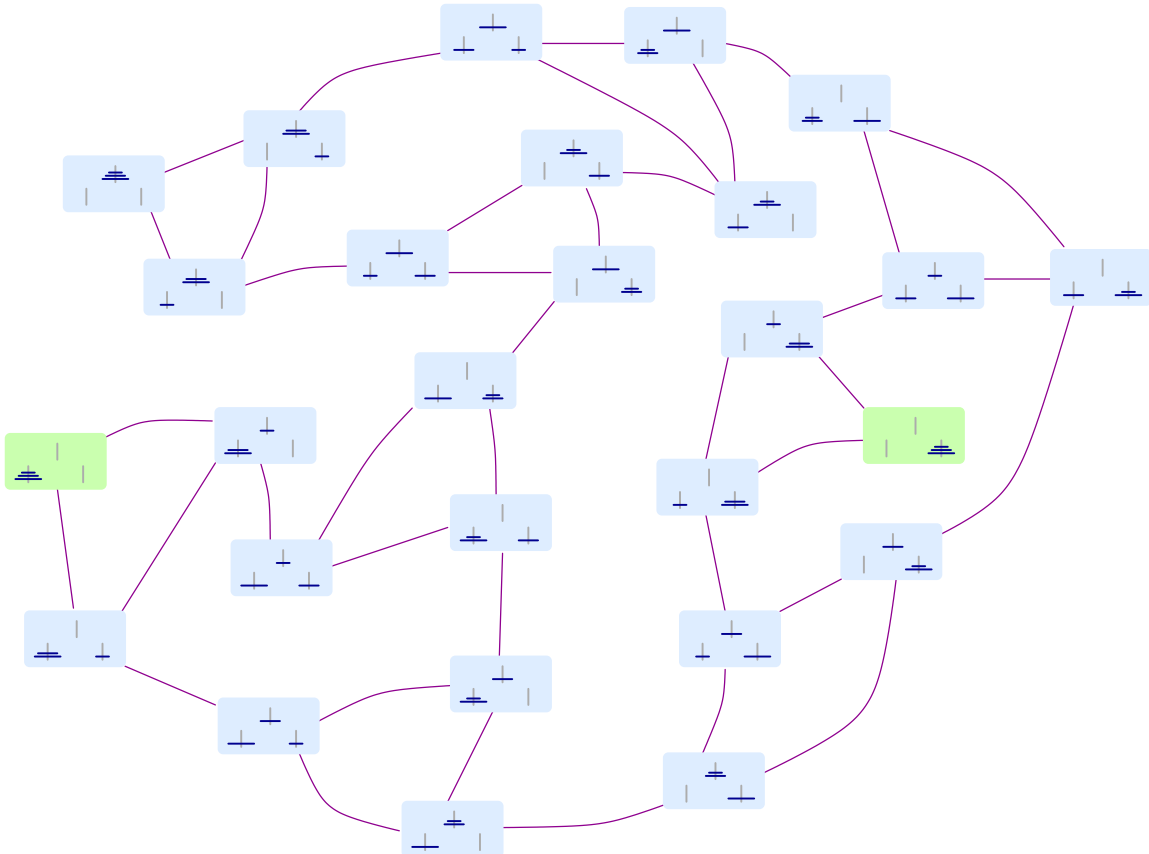


- **mouvements :** déplacer un disque sur une autre pile,
- **contrainte :** respecter l'ordre des tailles,
- **objectif :** déplacer tous les disques d'une tour à une autre en effectuant le moins de mouvements possible.

Exemple : problème des tours de Hanoï

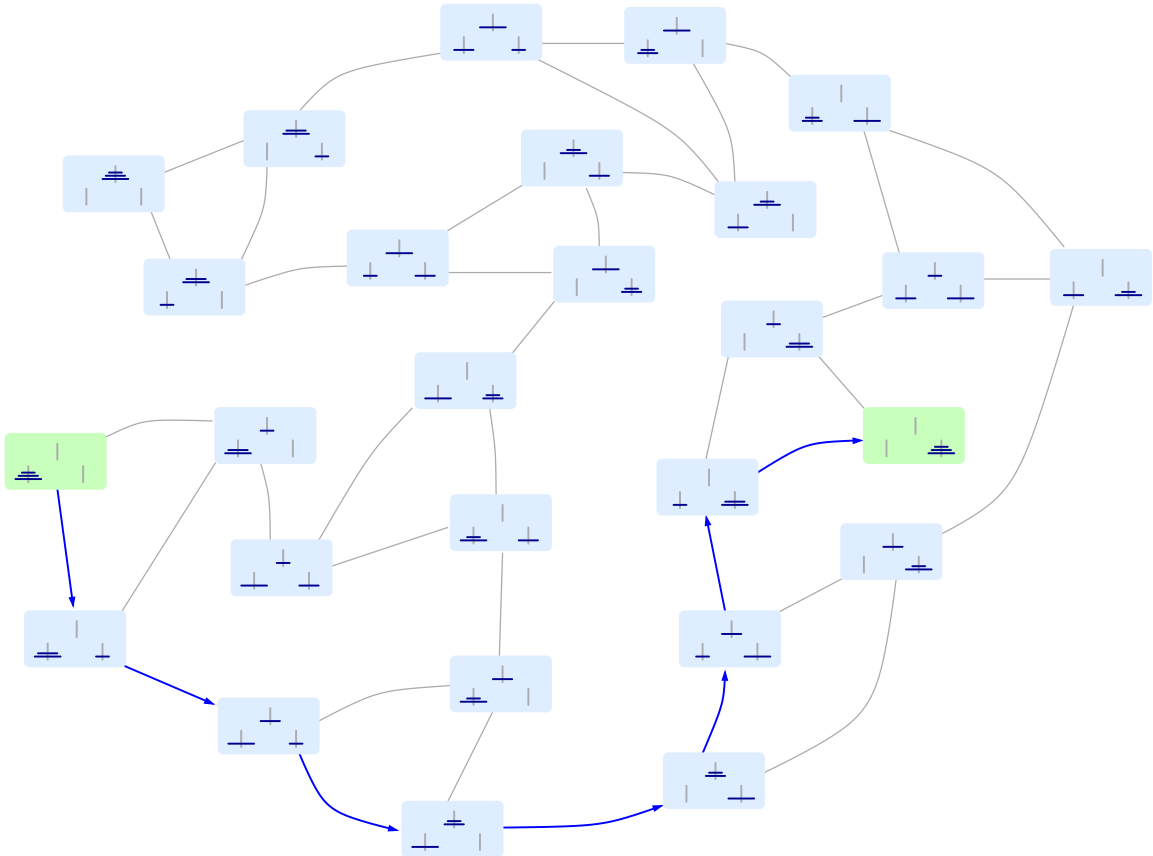
Sommets : tous les états valides (3^n états)

Exemple : problème des tours de Hanoï.



Arêtes : les mouvements

Exemple : problème des tours de Hanoï



Un **plus court chemin** entre l'état initial et l'objectif représente une plus courte séquence de mouvements pour déplacer les disques (il faut $2^n - 1$ mouvements).

Graphes : vocabulaire

voisinage

cycles

arbre

chemin

graphe connexe

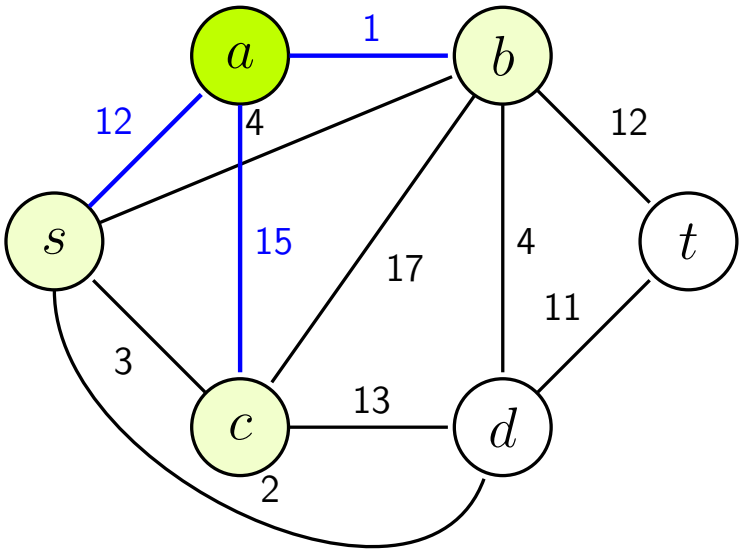
circuit

plus court chemin

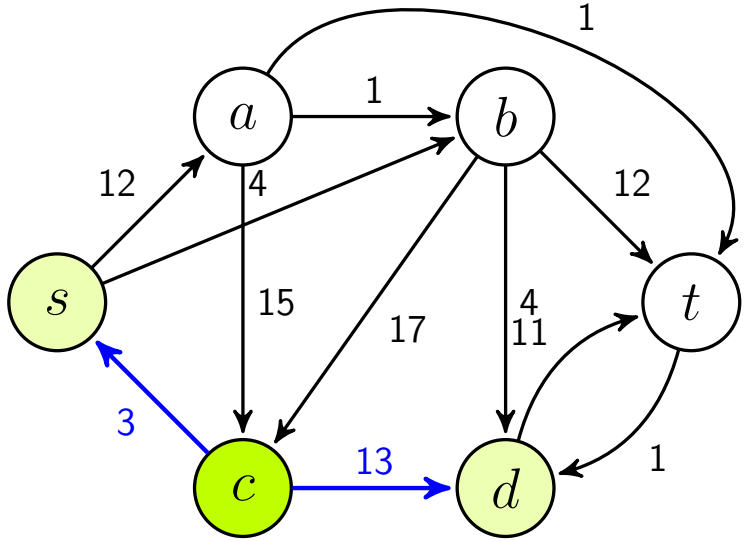
composantes connexes

graphe complet

Graphes : voisinage



Graphe non orienté

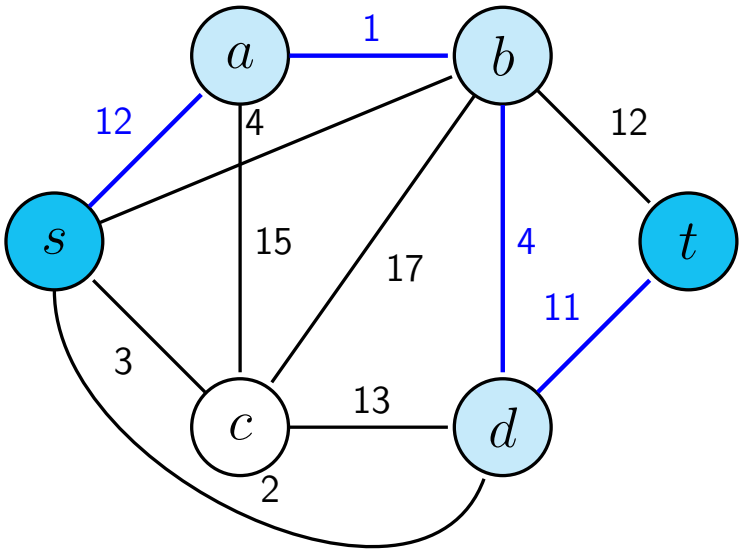


Graphe orienté

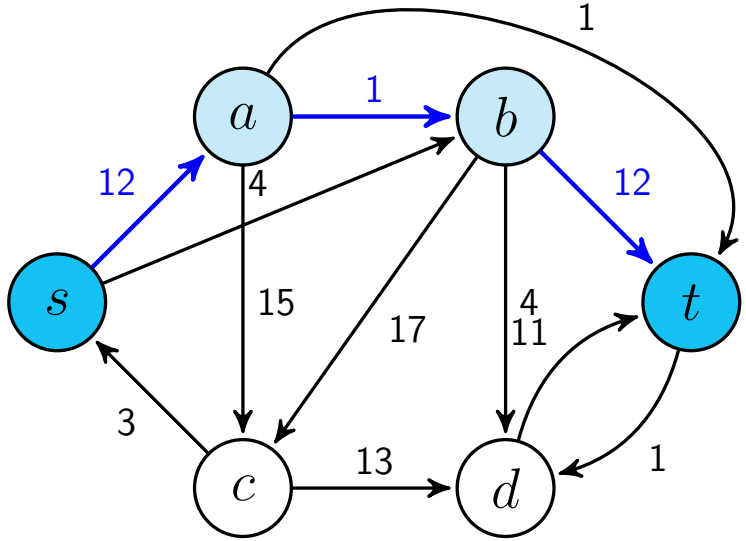
$$Voisins(u) = \{v \in S | (u, v) \in A\}$$

Les voisins de u sont les sommets v tels que le graphe contient l'arête (u, v) (resp. l'arc (u, v)).

Graphes : chemins



Graphe non orienté

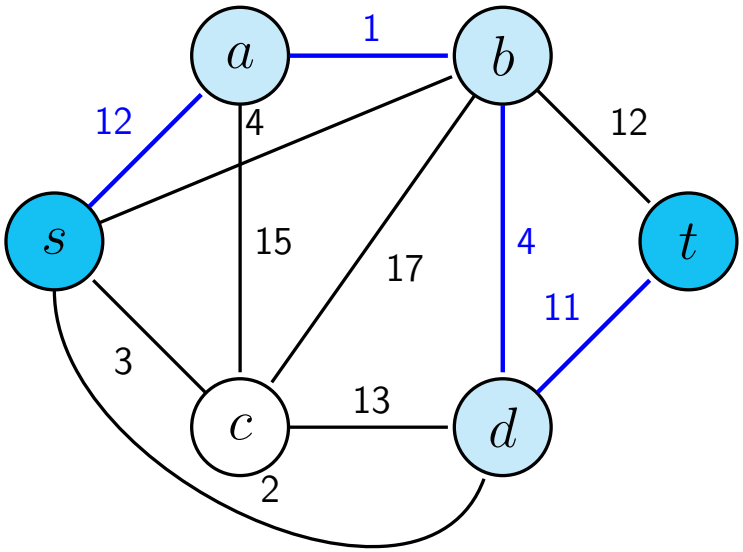


Graphe orienté

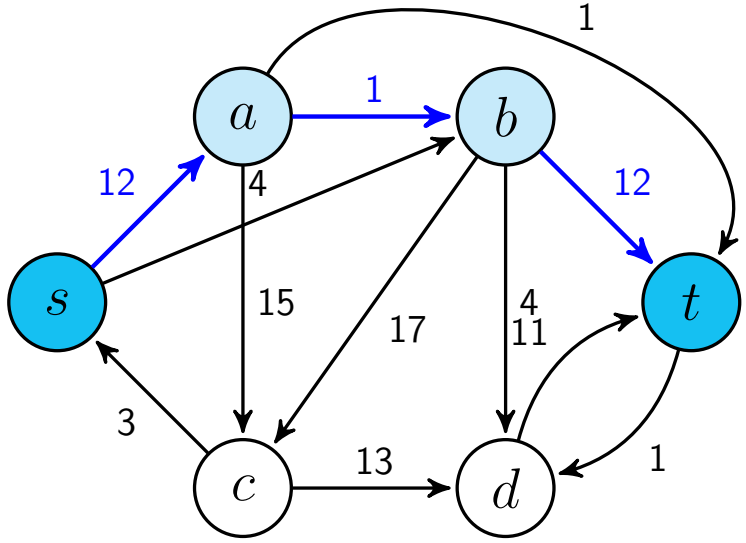
Chemin : $(u_1, u_2, \ldots, u_k)$ avec $(u_1, u_2), \ldots, (u_{k-1}, u_k) \in A$.

Suite de sommets telle que pour tout sommet u_i, $i < k$, le graphe contient l'arête (resp. l'arc) (u_i, u_{i+1}).

Graphes : chemins



Graphe non orienté

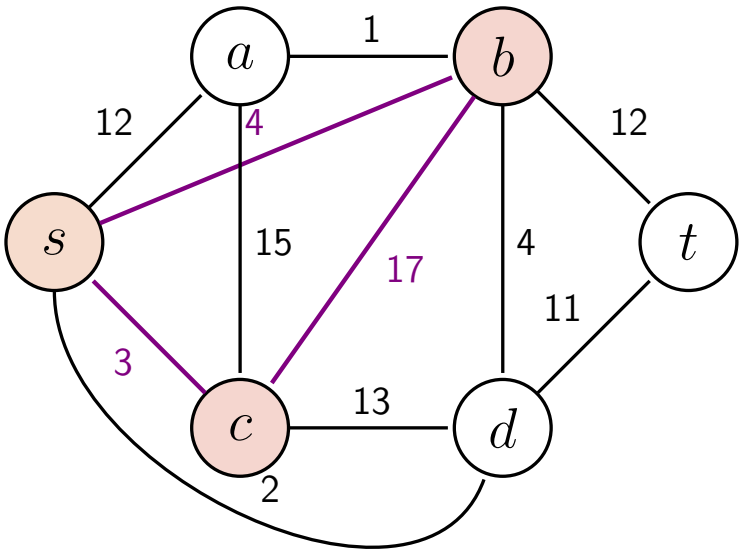


Graphe orienté

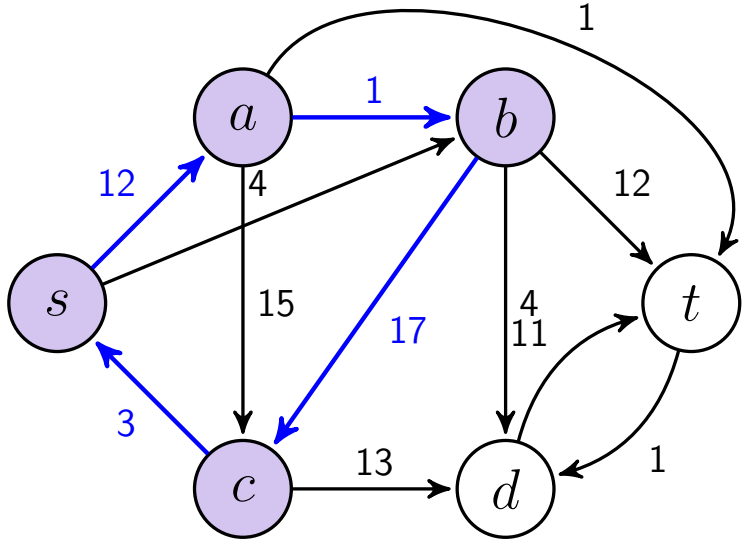
Chemin : $(u_1, u_2, \ldots, u_k)$ avec $(u_1, u_2), \ldots, (u_{k-1}, u_k) \in A$.

Longueur : $\sum_{i=1}^{k-1} w(u_i, u_{i+1})$

Graphes : cycles, circuits



Graphe non orienté

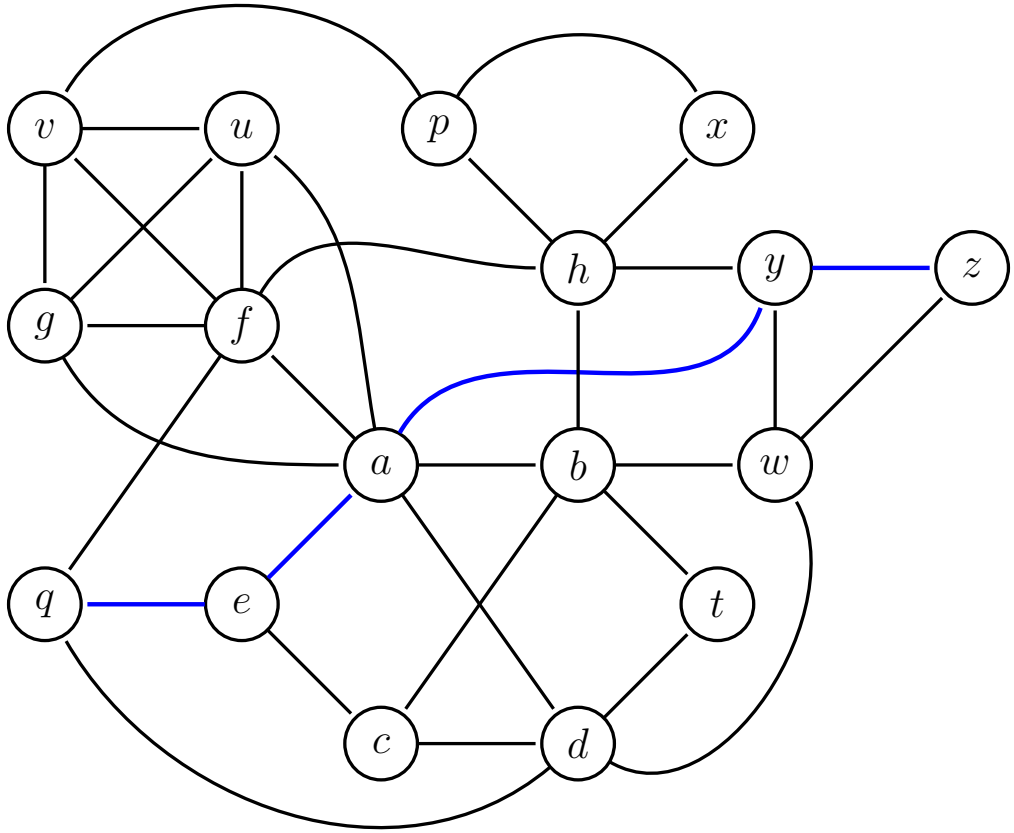


Graphe orienté (circuit)

Cycle : $(u_1, u_2, \ldots, u_k)$

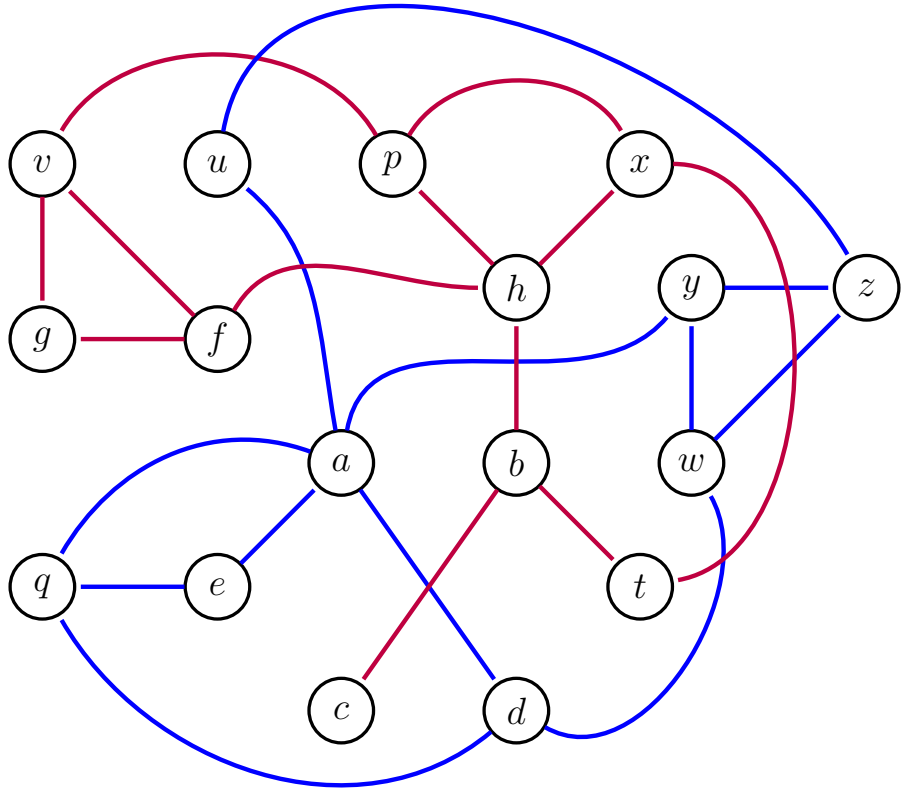
avec $(u_1, u_2), (u_2, u_3), \ldots, (u_k, u_1) \in A$.

Graphe connexe



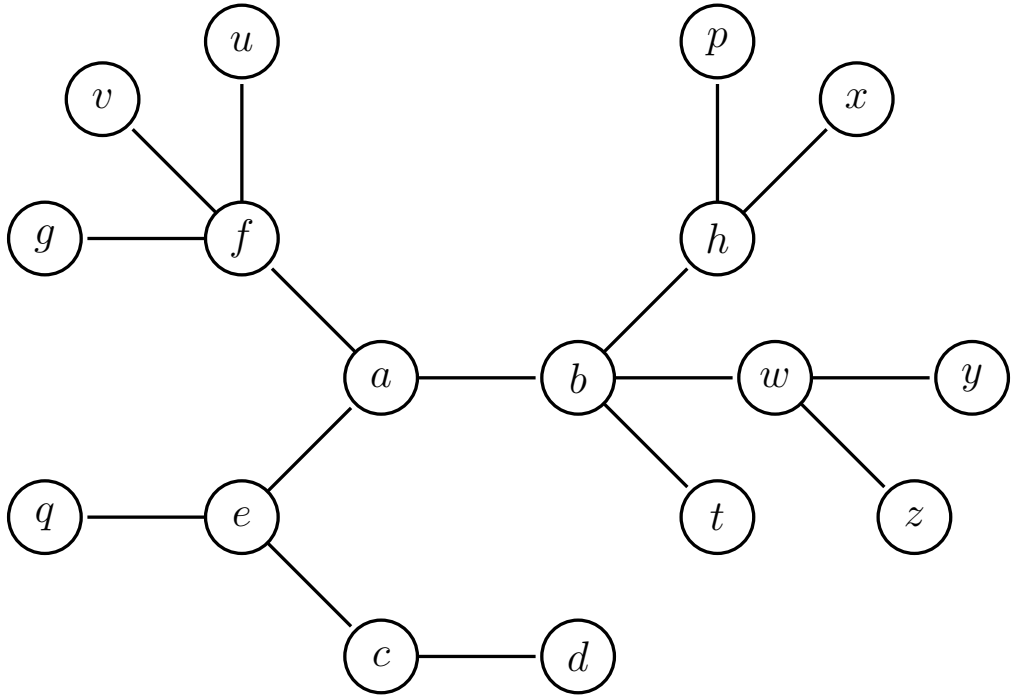
Le graphe est connexe si pour tous sommets u et v il existe un chemin joignant u à v

Graphe non connexe



Composantes connexes : sous-graphes connexes maximaux

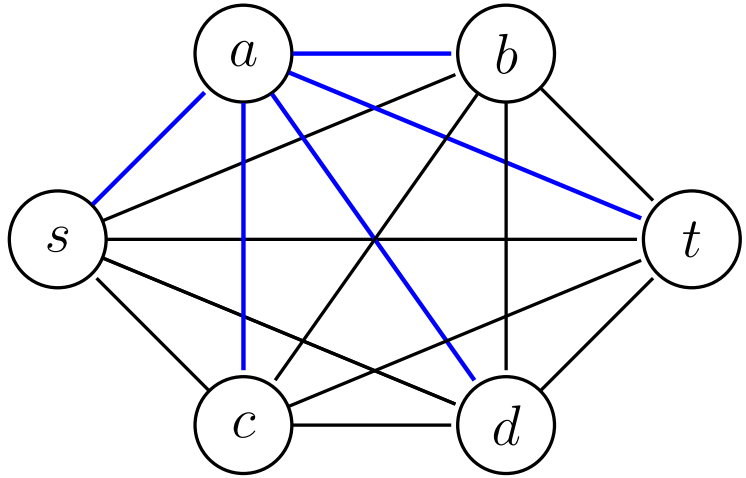
Arbre



Arbre : graphe connexe et sans cycle

n sommets $\Rightarrow n-1$ arêtes

Graphe complet

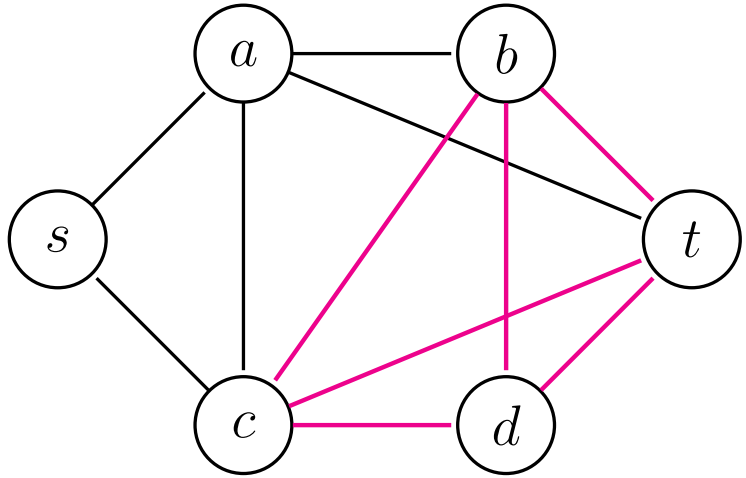


Chaque sommet est connecté avec tous les autres :

$$(n-1) + (n-2) + \ldots + 1 = n \times (n-1)/2 \text{ arêtes}$$

densité : proportion d'arêtes présentes (100% pour un graphe complet)

Clique



Une **clique** est un sous-graphe complet (graphe induit par un sous-ensemble de sommets)

Clique max : plus grande clique

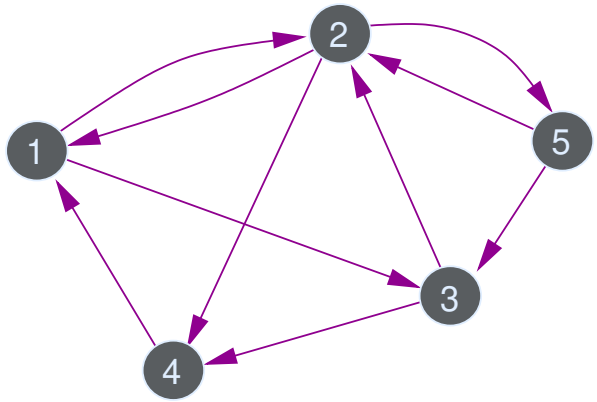
Représentation

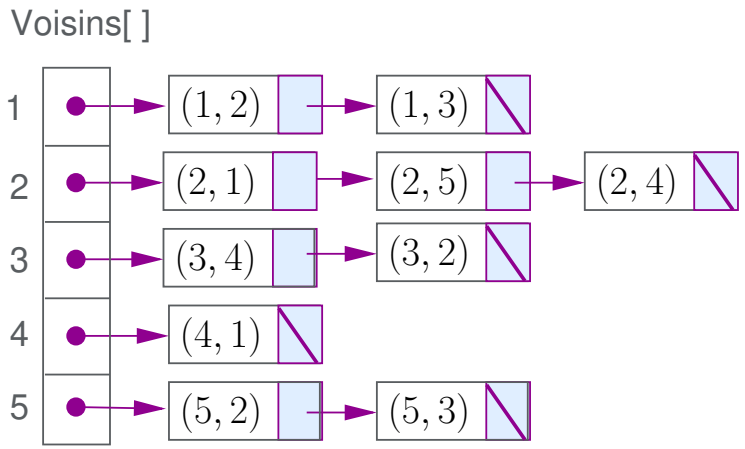
- listes d'adjacences,
- matrice d'adjacence,
- liste des arêtes (ou arcs).

La représentation doit être adaptée aux algorithmes que l'on va utiliser.

Dans ce qui suit les sommets sont numérotés de 1 à n. Lors de la mise en oeuvre (dans un programme C, Java,...) on utilisera la numérotation $0, \ldots, n-1$ correspondant à l'indexation standard dans un tableau.

Représentation par des listes d'adjacence



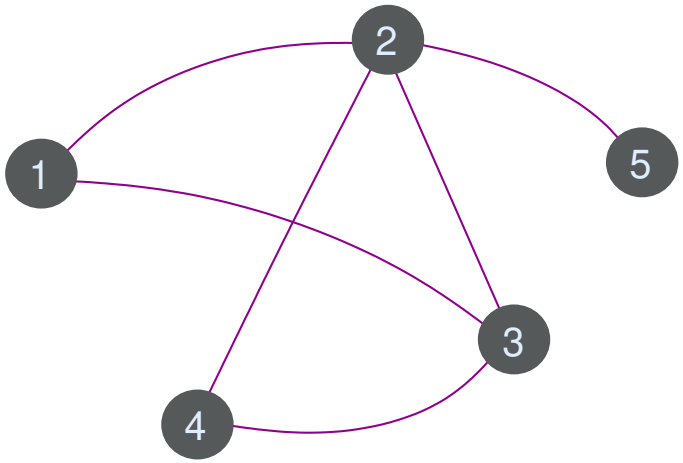


A chaque sommet est associée la **liste** de ses voisins.

Représentation : tableau de listes, indexé sur les sommets

(généralement des listes chainées contenant les arcs)

Représentation avec des matrices d'adjacence



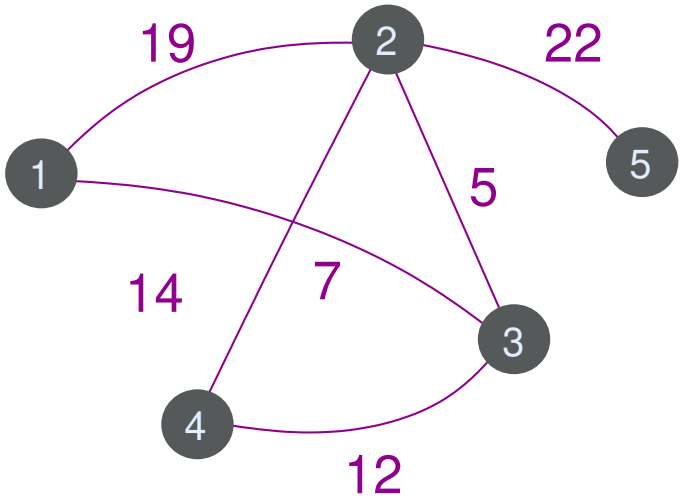
M

	1	2	3	4	5
1	F	V	V	F	F
2	V	F	V	V	V
3	V	V	F	V	F
4	F	V	V	F	F
5	F	V	F	F	F

$$M_{ij} = \begin{cases} V & si\ le\ graphe\ contient\ l'arête\ (i,j) \\ F & sinon \end{cases}$$

Pour chaque paire/couple de sommets un **booléen** indique s'ils sont voisins.

Représentation avec des matrices d'adjacence



	1	2	3	4	5
1	0	19	7	∞	∞
2	19	0	5	14	22
3	7	5	0	12	∞
4	∞	14	12	0	∞
5	∞	22	∞	∞	0

Graphe pondéré : une deuxième matrice est nécessaire.

Quand la pondération représente une distance on utilise une seule matrice avec la valeur $+\infty$ pour indiquer qu'il n'y a pas d'arête.

Parcours en profondeur

- exploration récursive (semblable au parcours en profondeur pour les arbres),
- marquage des sommets afin de ne pas parcourir indéfiniment les mêmes sommets s'il y a des cycles,
- produit plusieurs arborescences (une forêt).

Parcours en profondeur

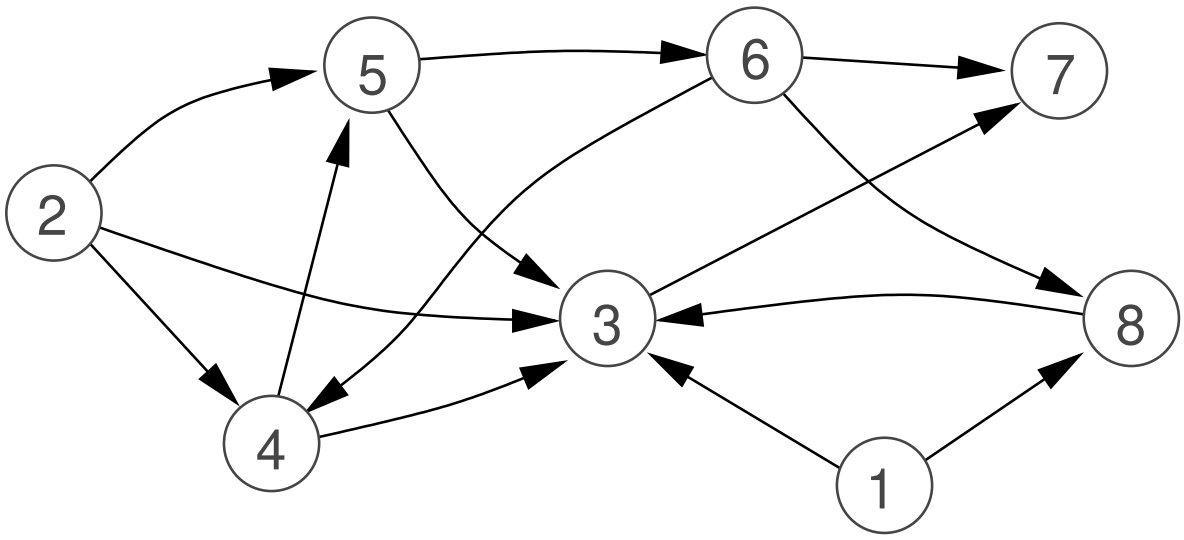
Coloration des sommets :

- BLANC : sommet non découvert
- GRIS : sommet découvert, arborescence en cours d'exploration
- NOIR : sommet visité, arborescence sous-jacente parcourue

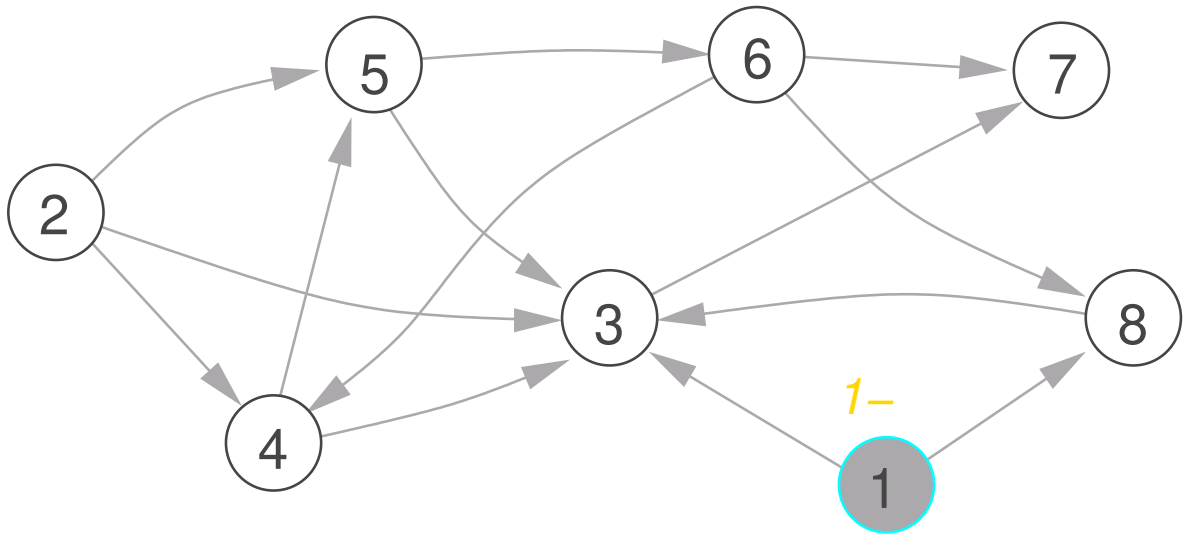
Dates :

- $debut[u]$: date de découverte du sommet u,
- $fin[u]$: date de fin de traitement du sommet u (fin de l'exploration à partir du sommet u).

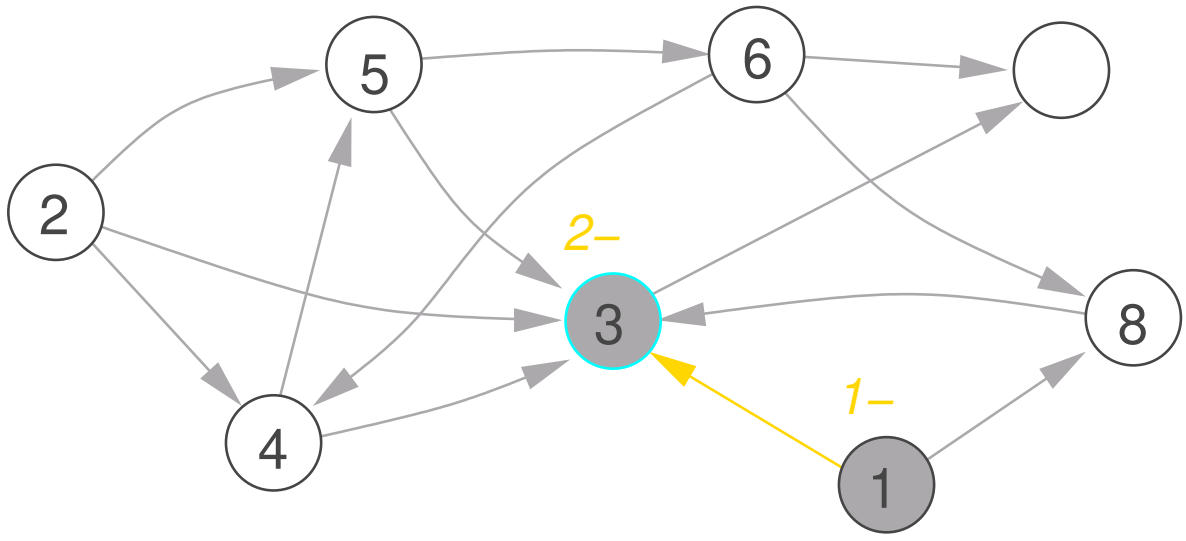
Parcours en profondeur



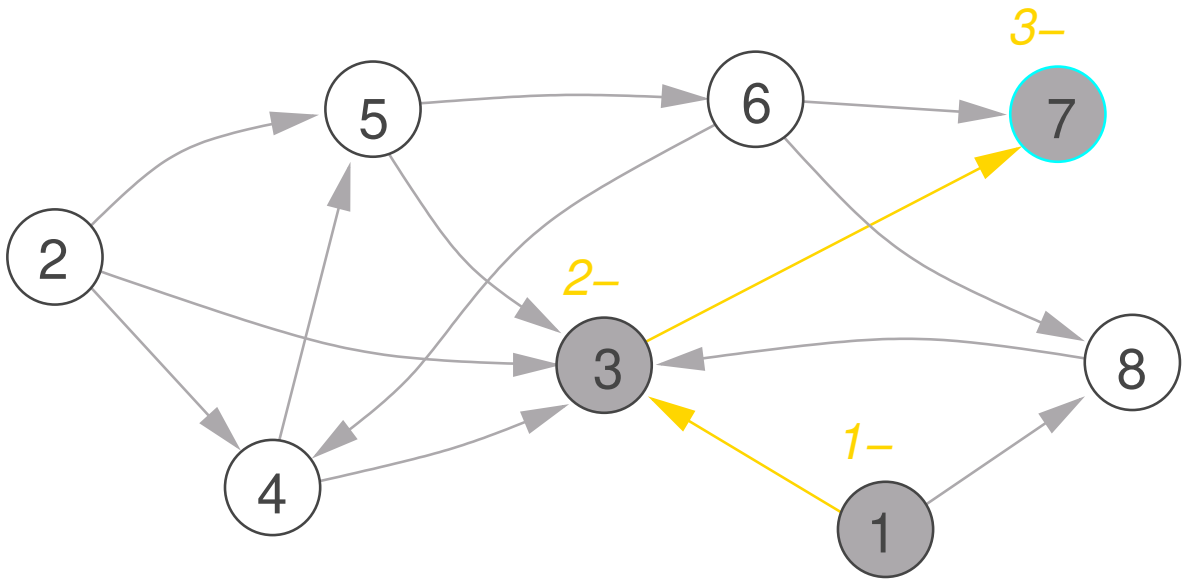
Parcours en profondeur



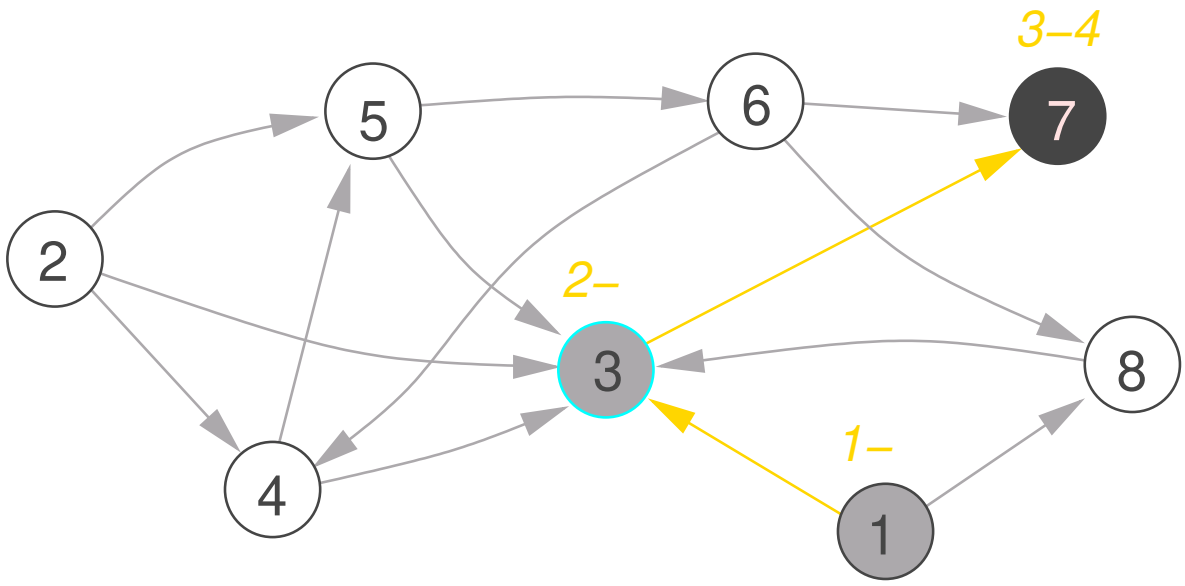
Parcours en profondeur



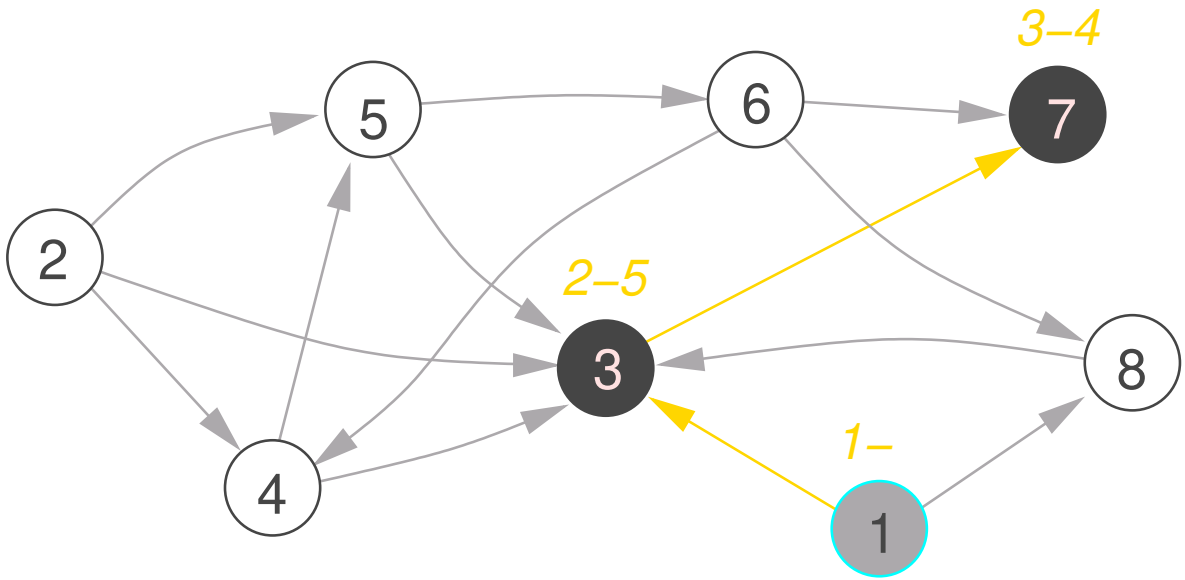
Parcours en profondeur



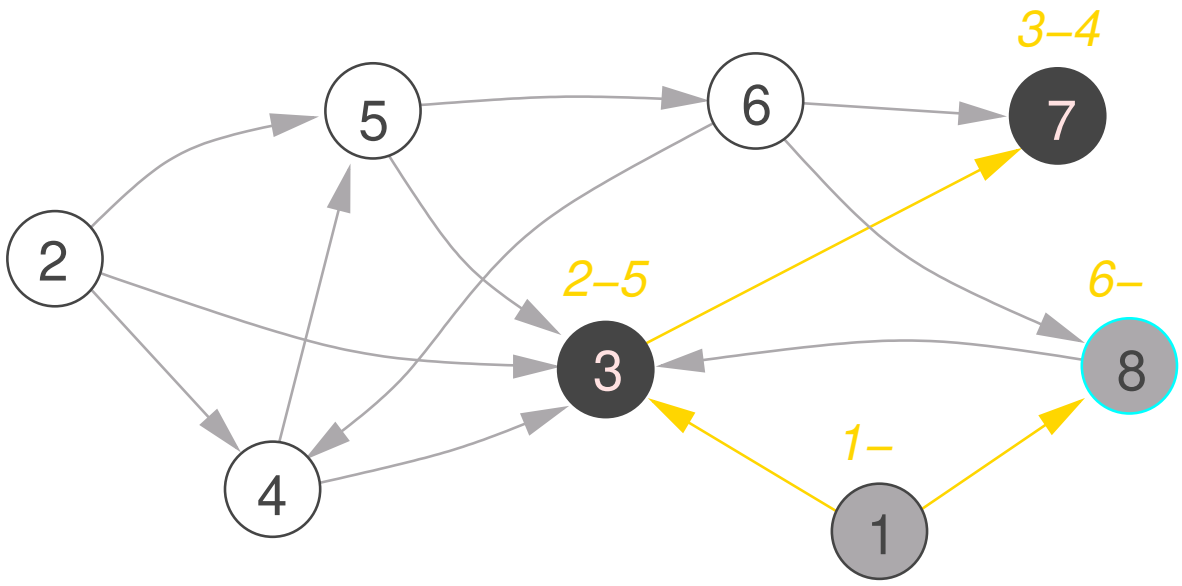
Parcours en profondeur



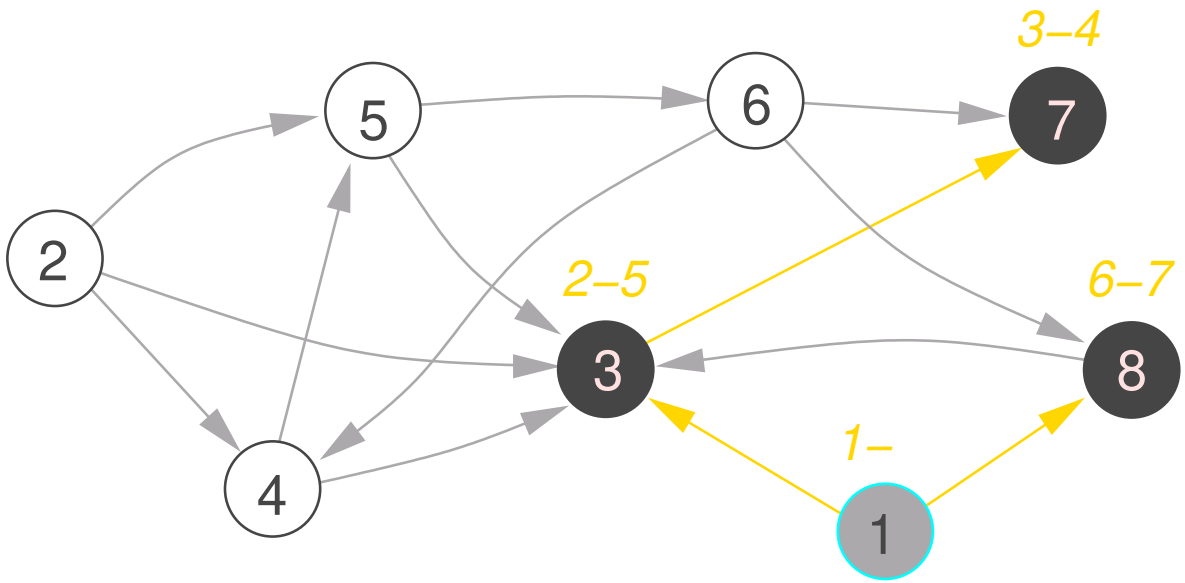
Parcours en profondeur



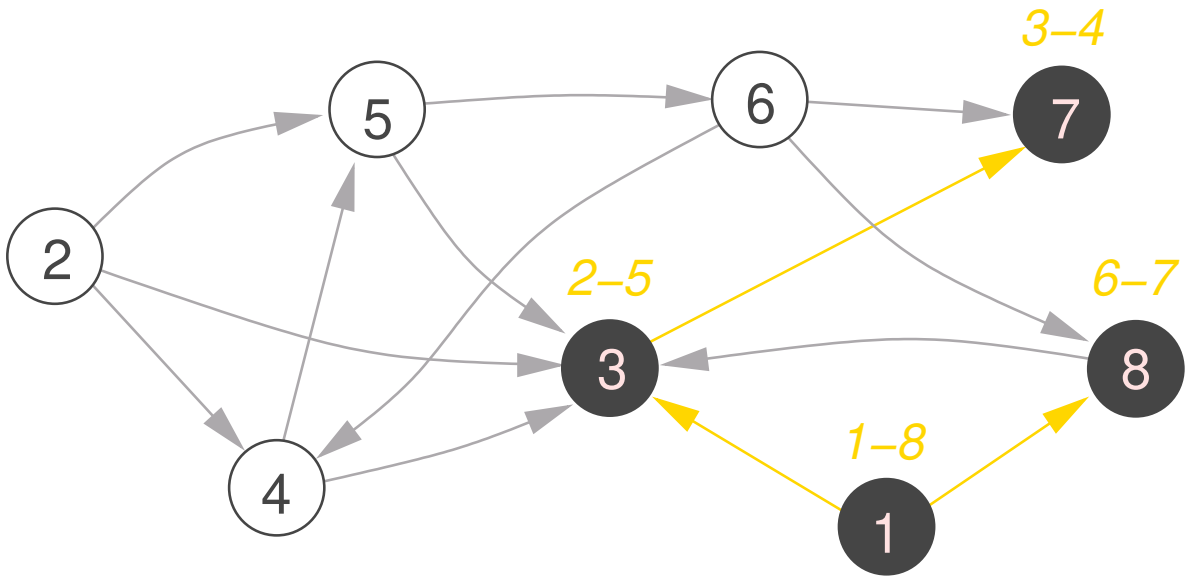
Parcours en profondeur



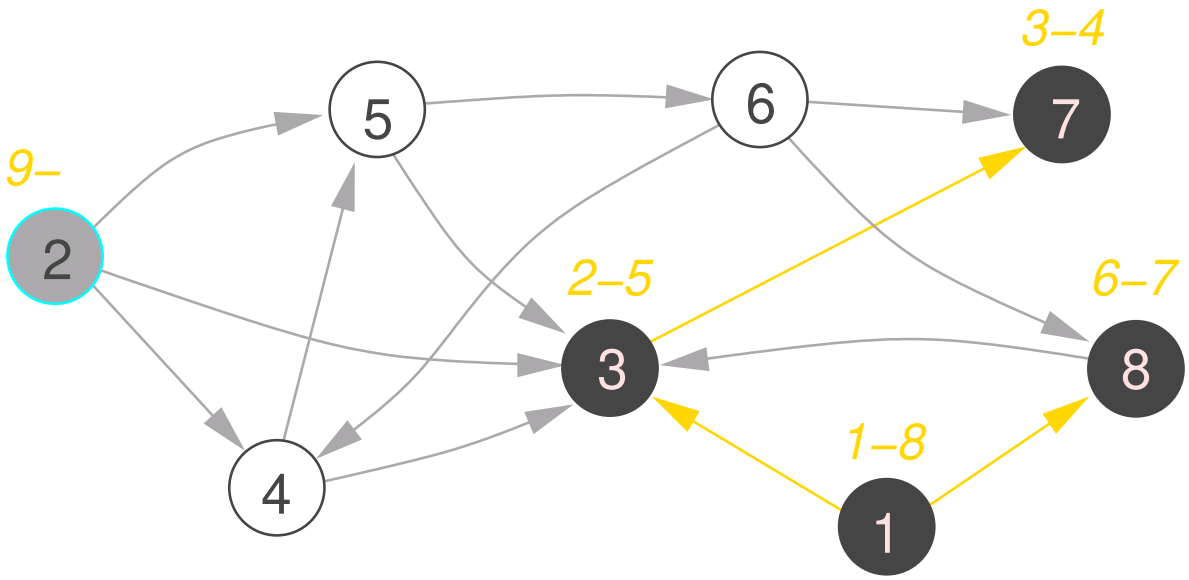
Parcours en profondeur



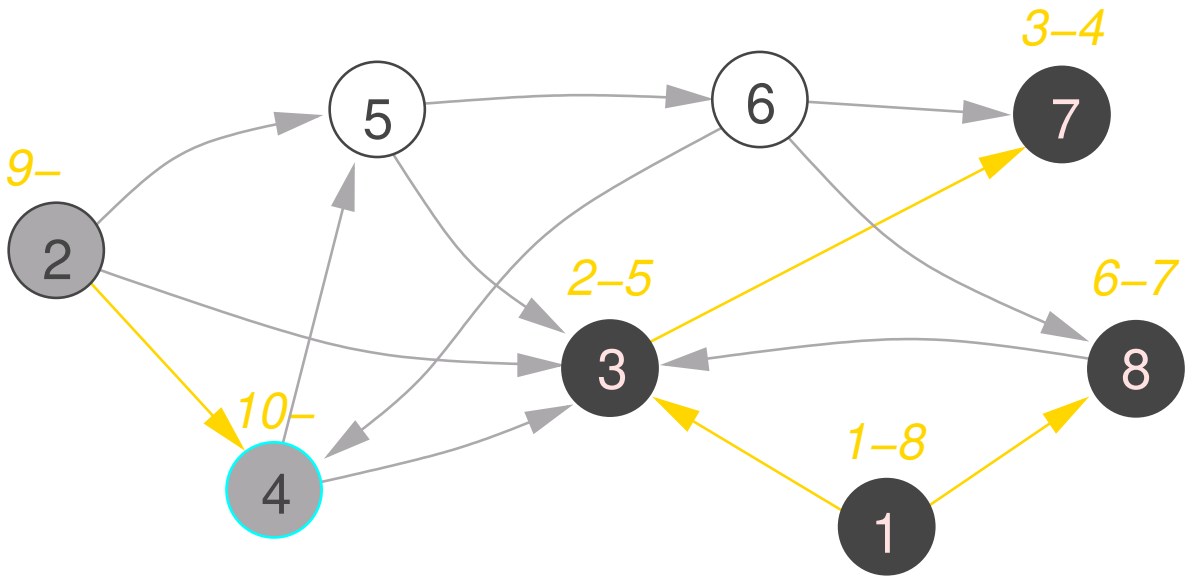
Parcours en profondeur



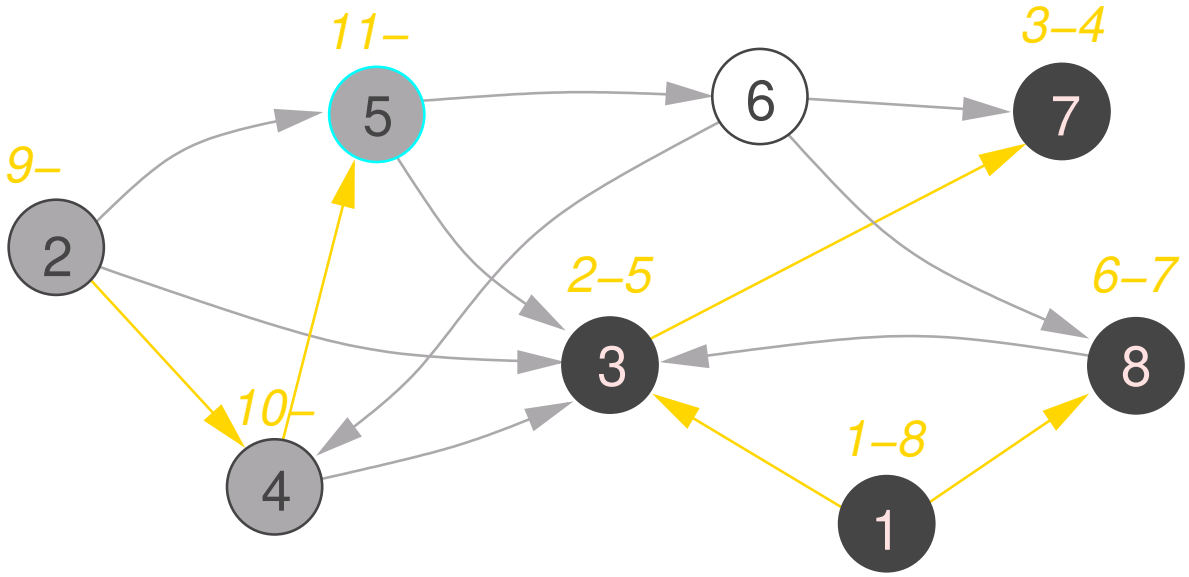
Parcours en profondeur



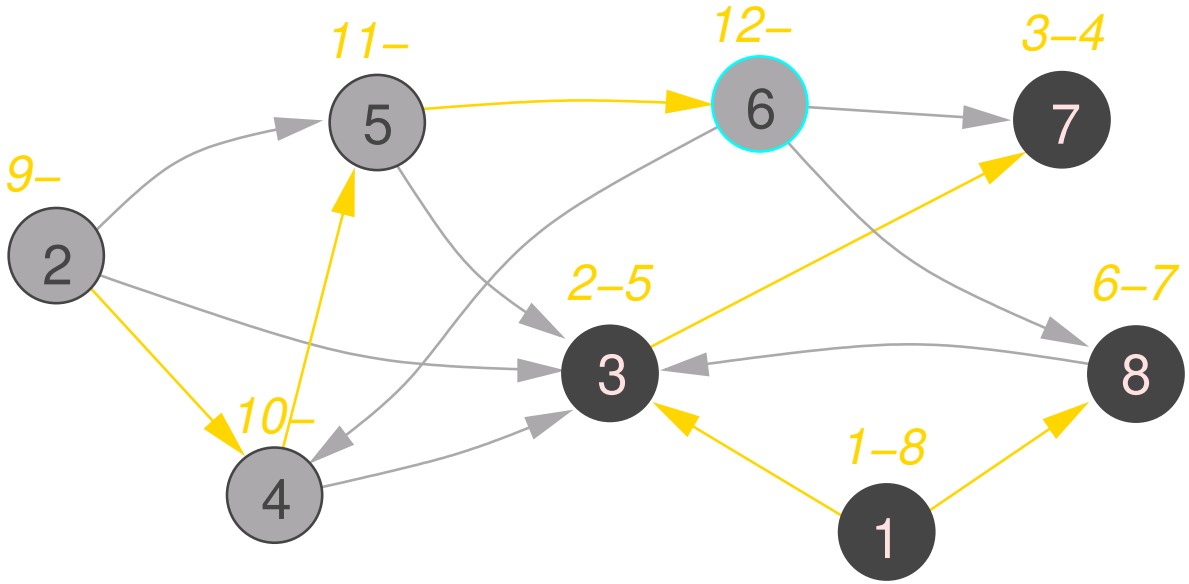
Parcours en profondeur



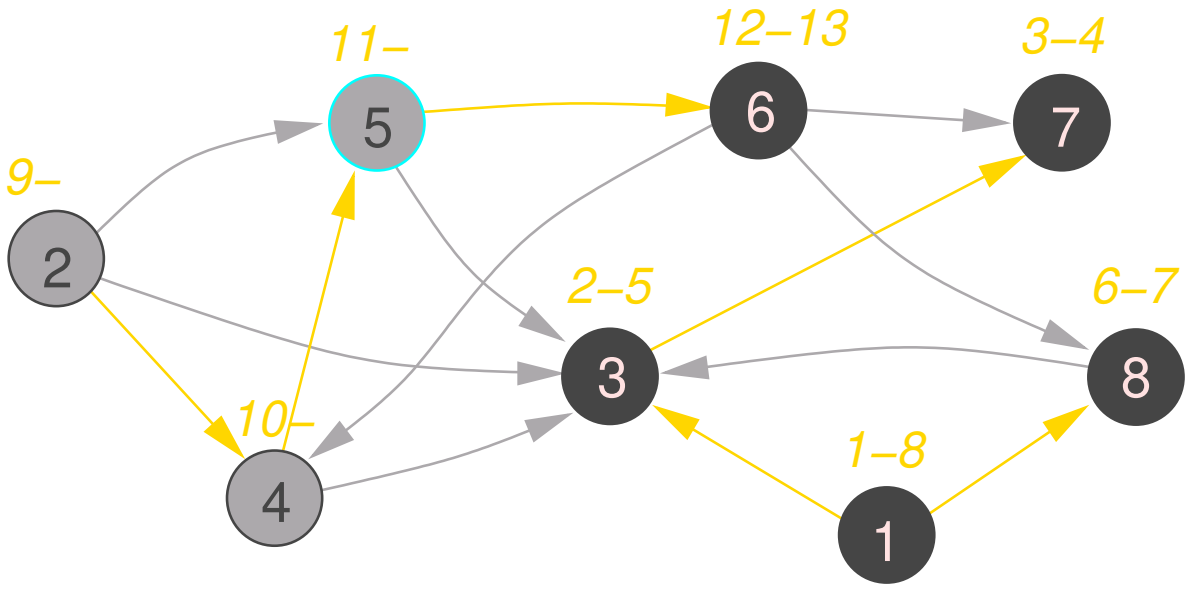
Parcours en profondeur



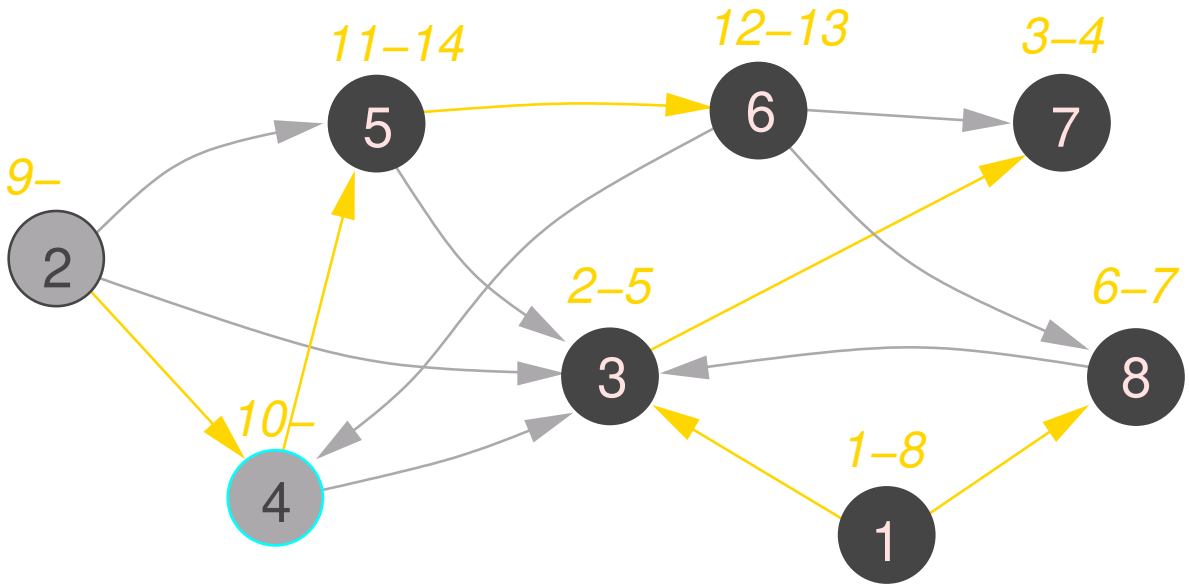
Parcours en profondeur



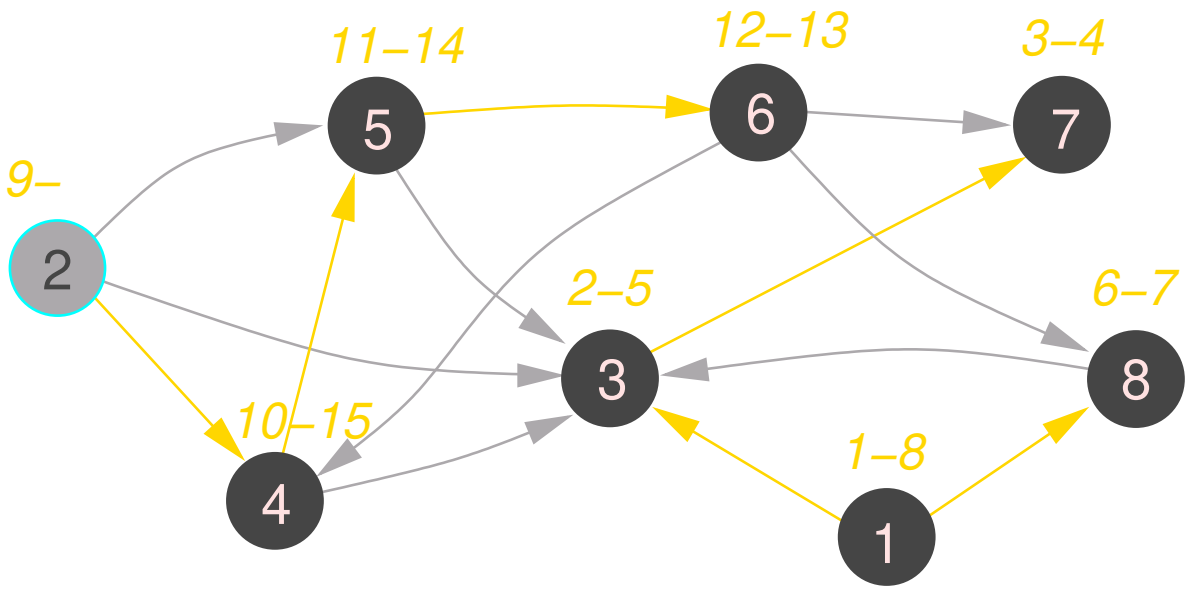
Parcours en profondeur



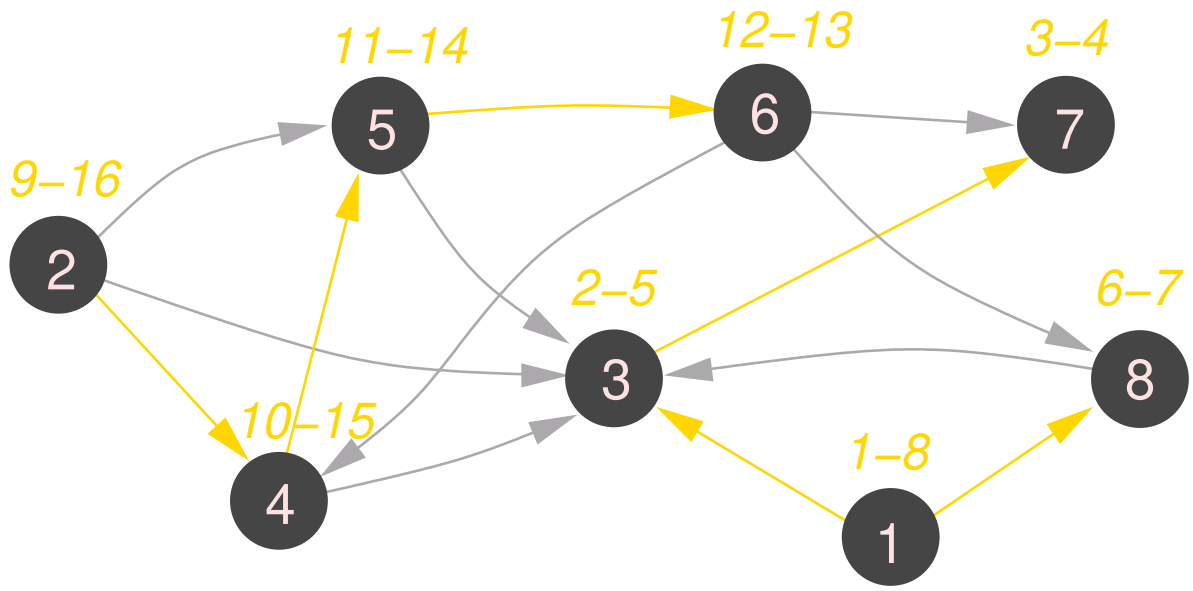
Parcours en profondeur



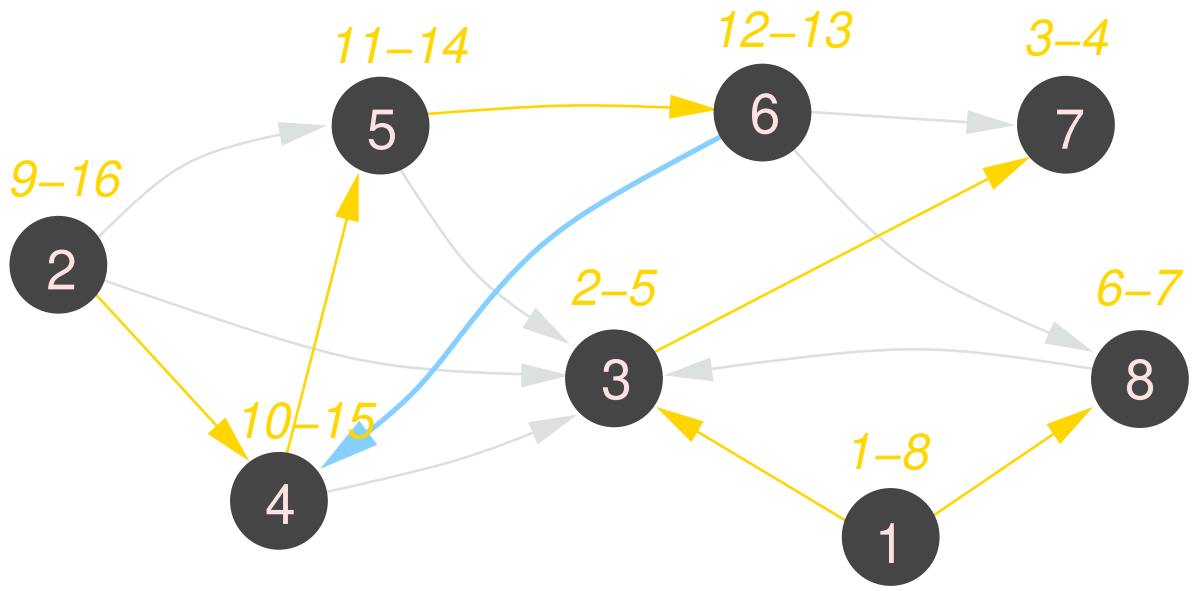
Parcours en profondeur



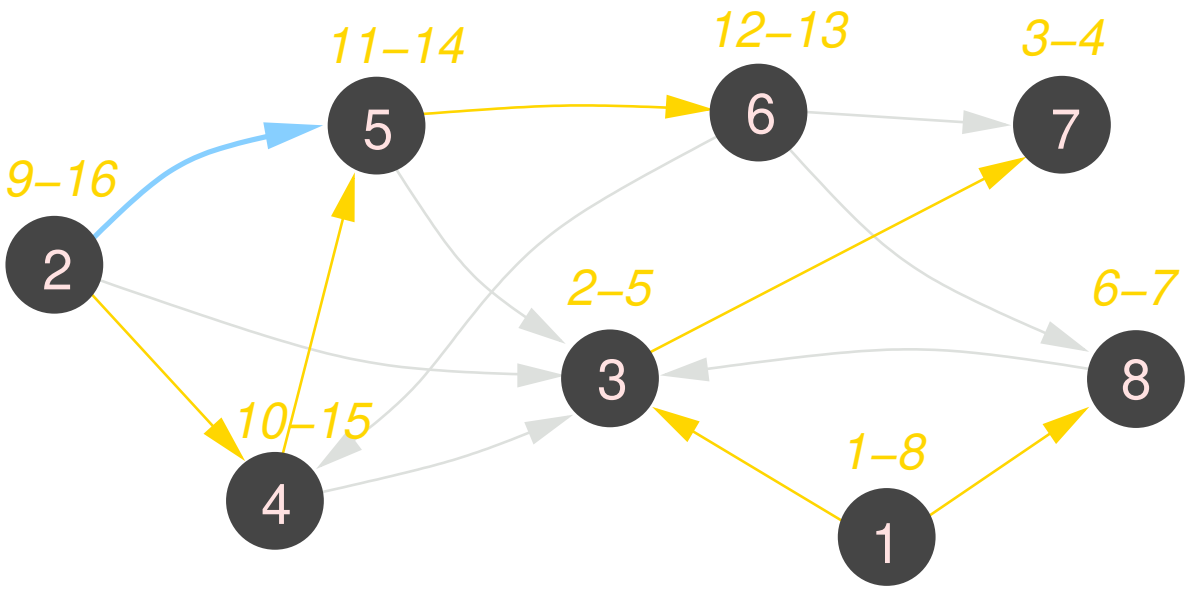
Parcours en profondeur : arborescence



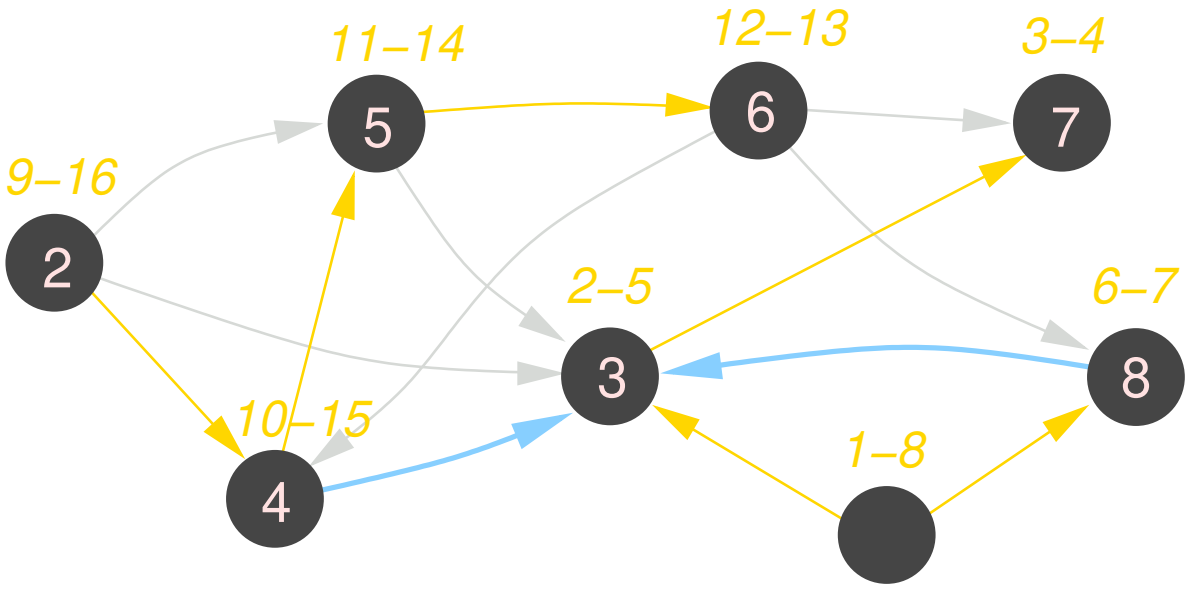
Parcours en profondeur : arcs arrières



Parcours en profondeur : arcs avants



Parcours en profondeur : arcs transverses



Parcours en profondeur : mise en place

fonction PARCOURS_EN_PROFONDEUR(G)
(**In :** $G = (S, A)$ un graphe)

pour chaque sommet $u \in S$ **faire**
$pred[u] :=$ AUCUN,
$couleur[u] :=$ BLANC,
$date := 0$,
pour chaque sommet $u \in S$ **faire**
si $couleur[u] =$BLANC **alors**
VISITER(u),
fin fonction

Parcours en profondeur : exploration

fonction VISITER(u, G) // u un sommet du graphe G

$couleur[u] :=$ GRIS,
$date := date + 1$,
$debut[u] := date$,
pour chaque sommet $v \in Voisins(u)$ **faire**
  si $couleur[v] =$ BLANC **alors**
    $pred[v] := u$,
    VISITER(v),
  fin si,
fin pour,
$couleur[u] :=$ NOIR,
$date := date + 1$,
$fin[u] := date$,
fin fonction

Parcours en profondeur

Complexité $O(|S| + |A|)$

Initialisation $O(|S|)$
Parcours des voisins de chaque sommet $O(|A|)$

Parcours en profondeur : dates (théorème des parenthèses)

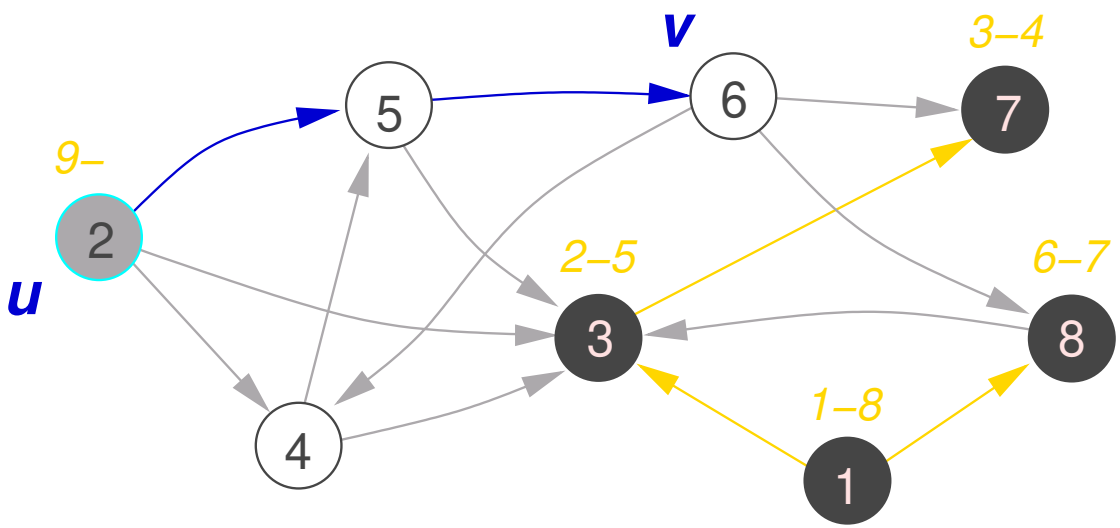
soient u et v deux sommets et d_u, f_u, d_v, f_v les dates associées :

- soit $[d_u, f_u]$ et $[d_v, f_v]$ sont disjoints (arborescences disjointes)
- soit $d_v < d_u < f_u < f_v$ (u est un descendant de v)
- soit $d_u < d_v < f_v < f_u$ (v est un descendant de u)

En effet, si $d_u < d_v < f_u$, v a été découvert alors que u était GRIS, le traitement de v s'est terminé alors que u était toujours GRIS, donc $f_v < f_u$ et v est un descendant de u.

Parcours en profondeur : théorème du chemin blanc

v est un descendant de u dans la forêt du parcours en profondeur si et ssi à la date d_u où u est découvert il existe un chemin blanc allant de u à v



chemin blanc : chemin constitué uniquement de sommets blancs

Parcours en profondeur : théorème du chemin blanc

si v est un descendant de u **alors** à la date d_u où u est découvert il existe un chemin blanc allant de u à v

En effet, tout sommet p figurant dans la branche allant de u à v dans l'arborescence est un descendant de u et donc vérifie $d_p > d_u$. Il est donc BLANC au moment où u est découvert.

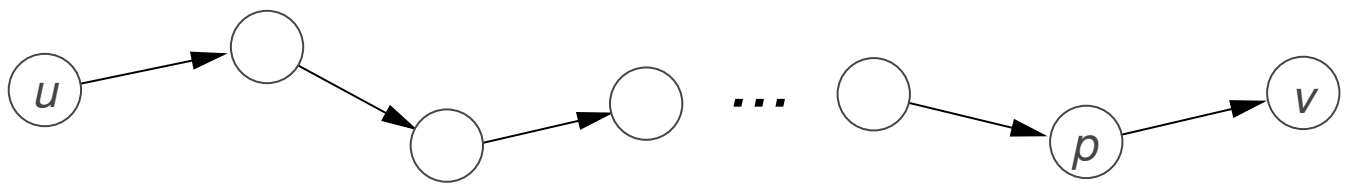
Parcours en profondeur : théorème du chemin blanc

si à la date d_u il existe un chemin blanc allant de u à v
alors v sera un descendant de u

Supposons que v ne soit pas un descendant de u et que c'est le premier dans ce cas, on note p son prédecesseur dans le chemin blanc.

- $d_u < d_v$ du fait que v est blanc à la date d_u
- $d_v < f_p$ puisque p a pour voisin v
- $f_p < f_u$ puisque p est un descendant de u

donc $d_u < d_v < f_p < f_u$ et v est donc un descendant de u (th. des parenthèses).



Plus courts chemins

$G = (S, A, w)$ un graphe pondéré.

La longueur du chemin $(u_1, \ldots, u_k)$ est

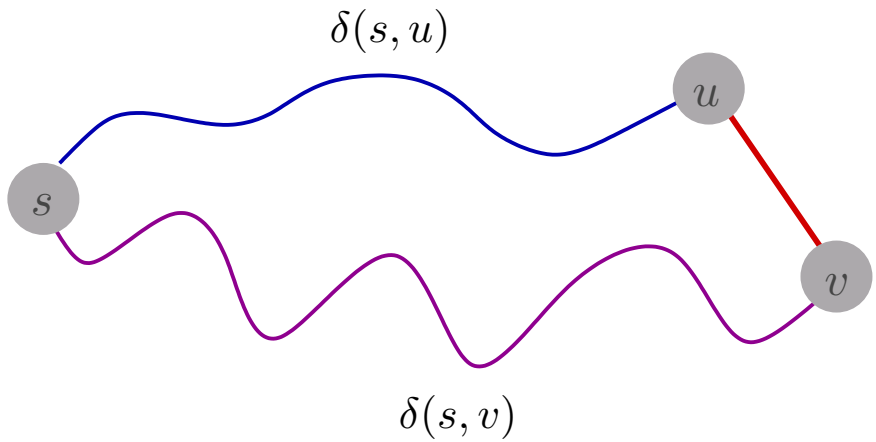
$$\sum_{i=1}^{k-1} w(u_i, u_{i+1})$$

Notation :

$$\delta(u, v)$$

la longueur du plus court chemin de u à v.

Plus courts chemins



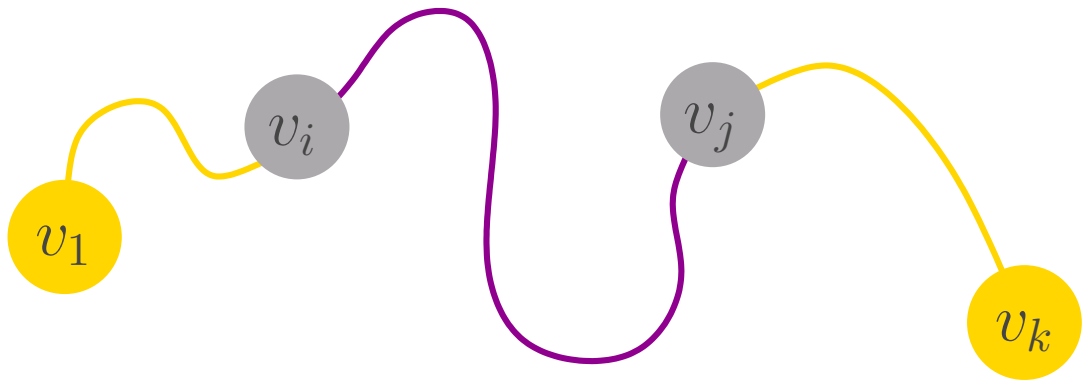
Propriété 1. Soit $s, u, v \in S$ tels que $(u, v) \in A$,

$$\delta(s, v) \leq \delta(s, u) + w(u, v)$$

Si ce n'était pas le cas on aurait un chemin de longueur inférieure à $\delta(s, v)$ finissant par l'arête (u, v), ce qui contredit la définition de $\delta(s, v)$.

Plus courts chemins

Propriété 2. Soit $(v_1, \ldots, v_k)$ un plus court chemin,



alors, pour tous sommets v_i, v_j de ce chemin,

$(v_i, v_{i+1}, \ldots, v_j)$ est un plus court chemin entre v_i et v_j.

Si ce n'était pas le cas on aurait un chemin plus court entre v_1 et v_k empruntant le plus court chemin entre v_i et v_j.

Algorithme de Dijkstra.

Calcul des plus courts chemins

- à partir d'un sommet donné, le sommet **source,**
- graphes orientés ou non orientés,
- arcs de poids **positifs ou nuls,**
- la bonne représentation du graphe : des **listes d'adjacence.**

L'algorithme de Dijkstra *découvre* les plus courts chemins vers les autres sommets du graphe en commençant par les plus courts.

Algorithme de Dijkstra.

Notations.

- s : source.
- $d[u]$: distance, indexée sur les sommets. A tout moment

$$\forall u \in S,\ d[u] \geq \delta(s,u)$$

Pendant le déroulement de l'algorithme, $d[u]$ représente la longueur du chemin le plus court entre s et u découvert **à ce stade.**

Algorithme de Dijkstra.

- Initialement

$$\forall u \in S,\ u \neq s,\ d[u] = \infty,$$
$$d[s] = 0.$$ (longueur du PCC de s à s)

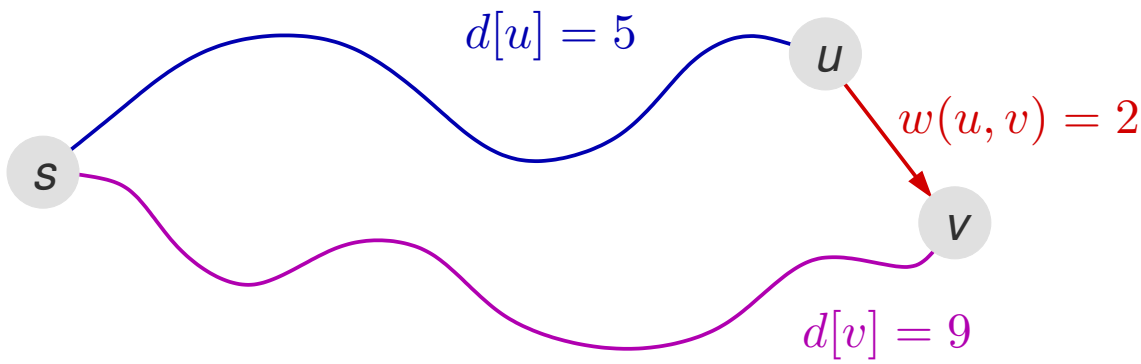
- Lorsque l'algorithme termine

$$\forall u \in S,\ d[u] = \delta(s, u).$$

Algorithme de Dijkstra : opération de relachement.

Relachement de l'arc $(u, v) \longrightarrow$ affinage de la borne $d[v]$.

Si $d[v] > d[u] + w(u, v)$ alors on remplace $d[v]$ par $d[u] + w(u, v)$

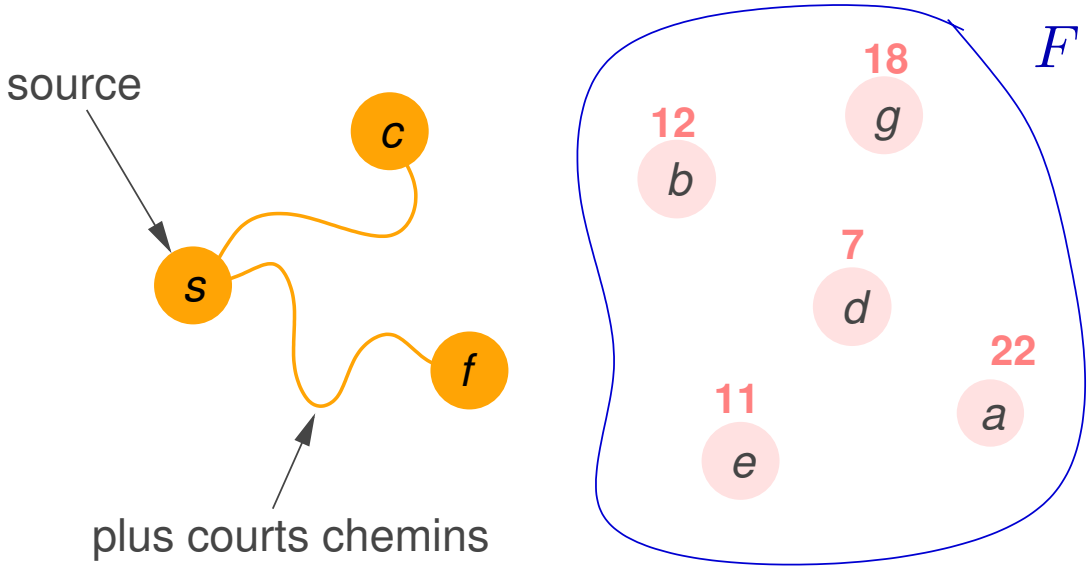


relachement de l'arc $(u, v) \Rightarrow d[v] = 7$

On peut étendre le chemin de longueur 5 de s à u, en un chemin de longueur 7 de s à v.

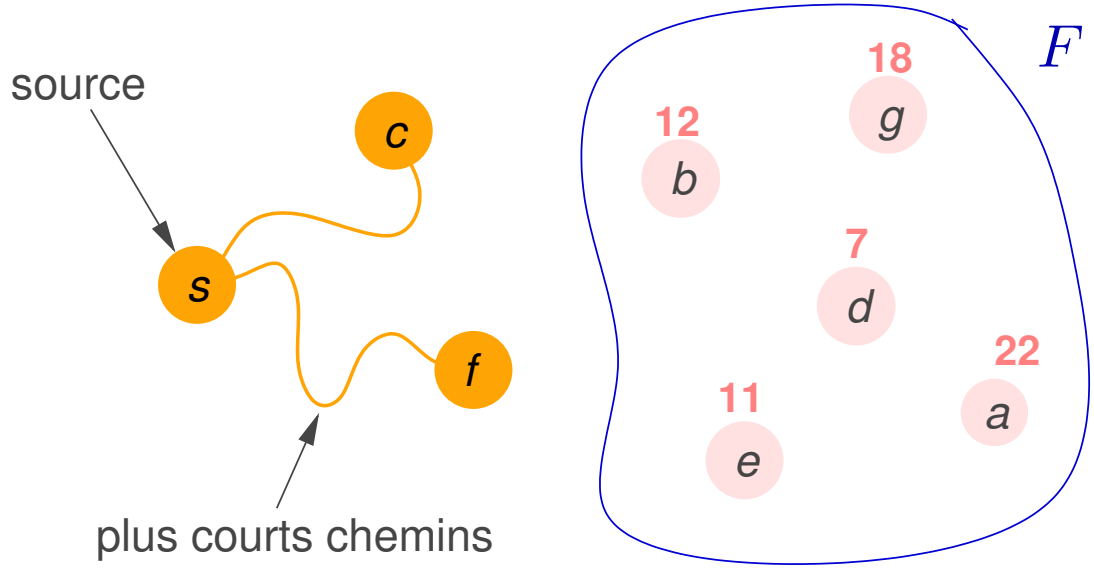
Algorithme de Dijkstra.

File de priorité F



Pour tout sommet u non dans F, on a $d[u] = \delta(u)$.

Algorithme de Dijkstra.

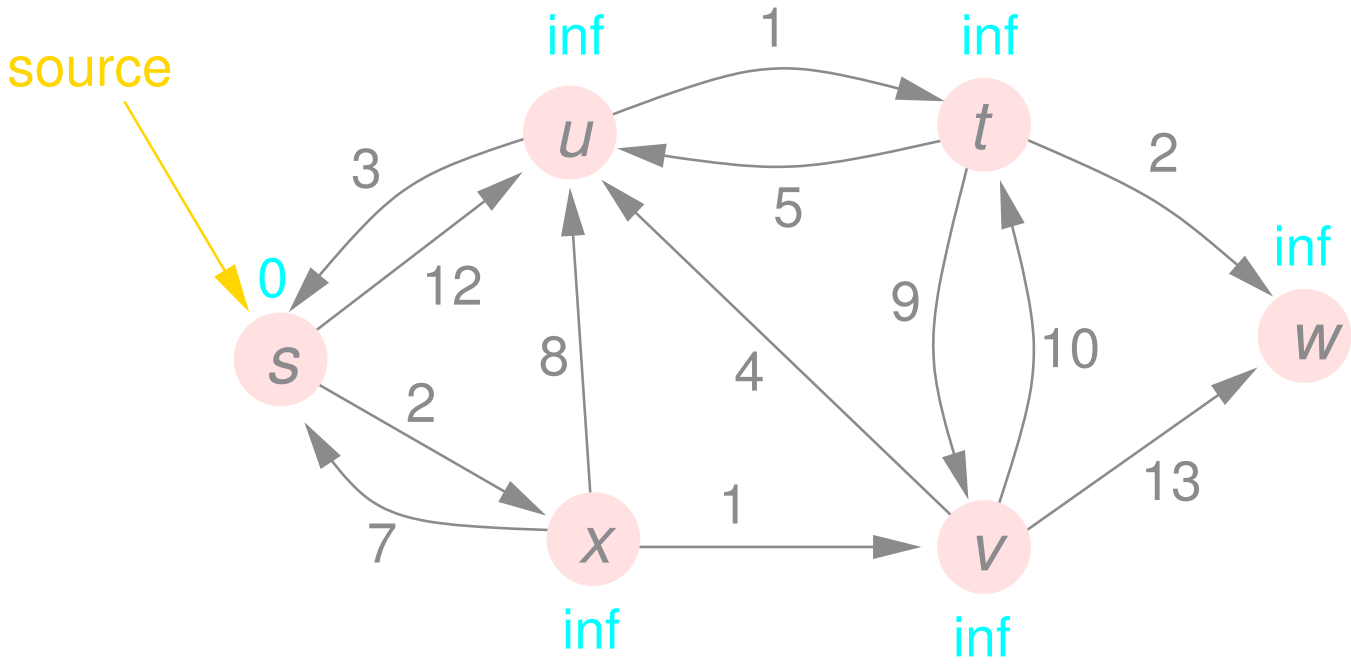


Itération :

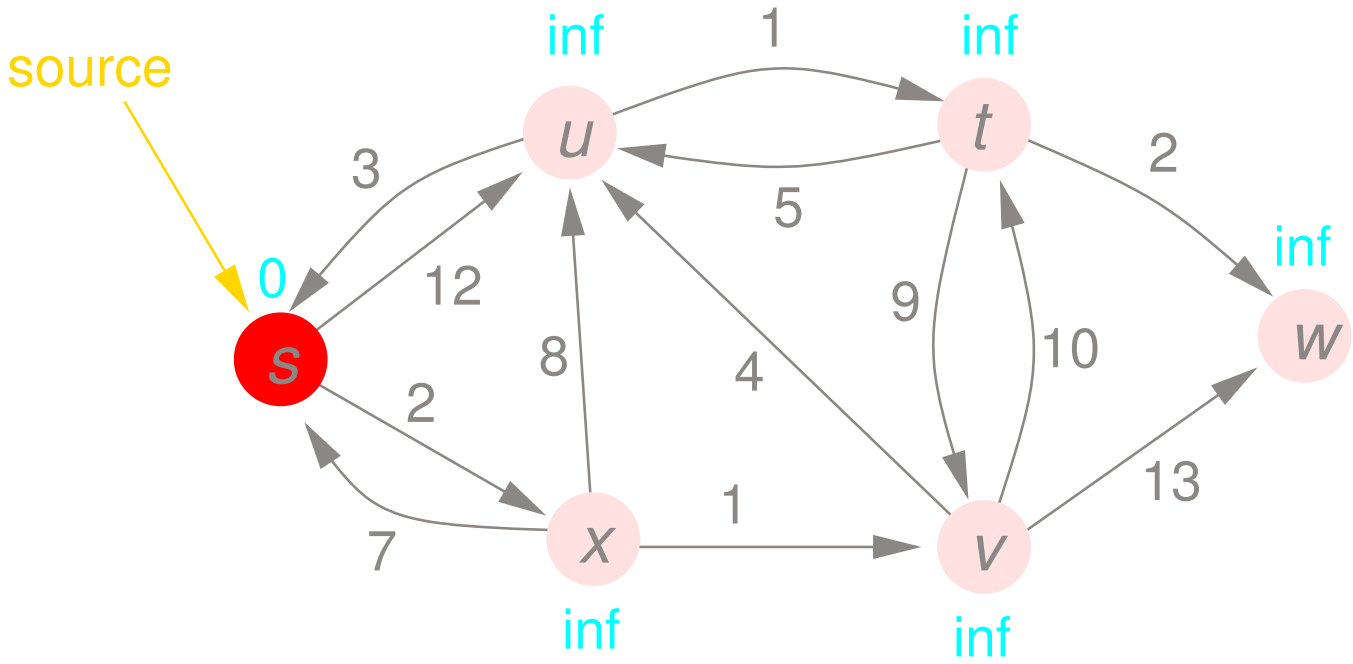
- extraction d'un sommet u de F de distance $d[u]$ minimale,
- relachement des arcs sortants de u.

Propriété : au moment de l'extraction le sommet u vérifie $d[u] = \delta(u)$

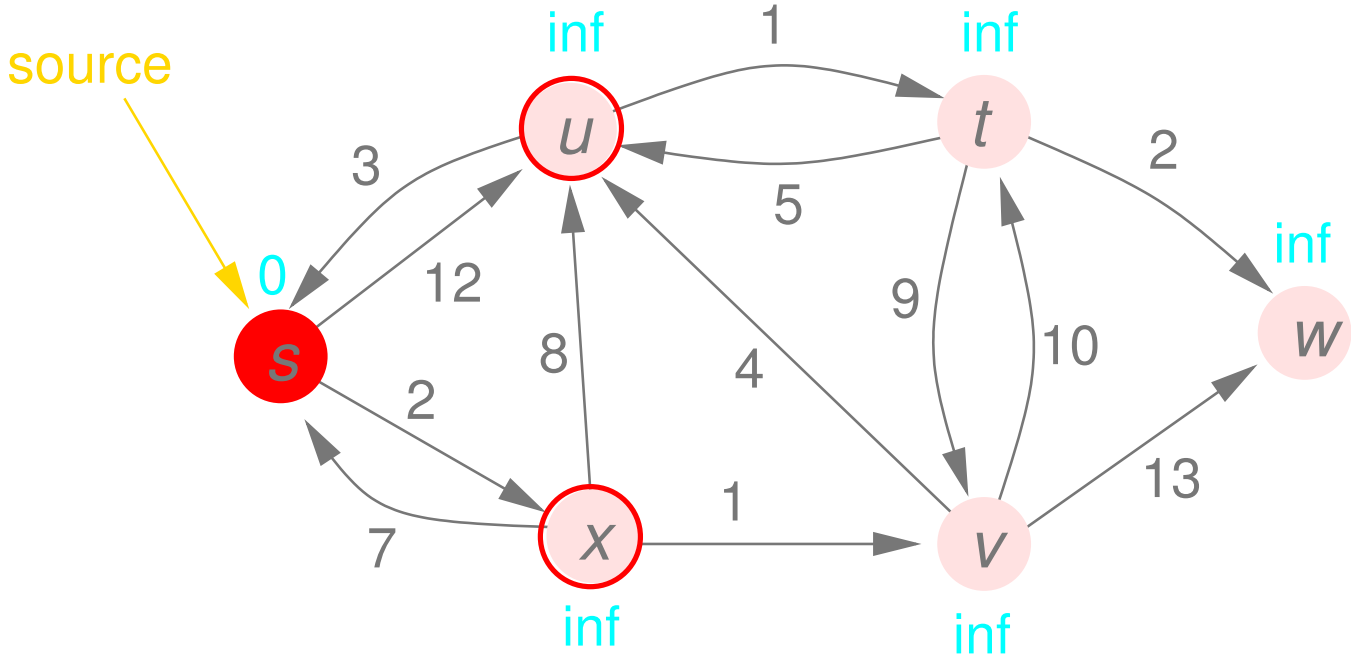
Plus courts chemins : Dijkstra.



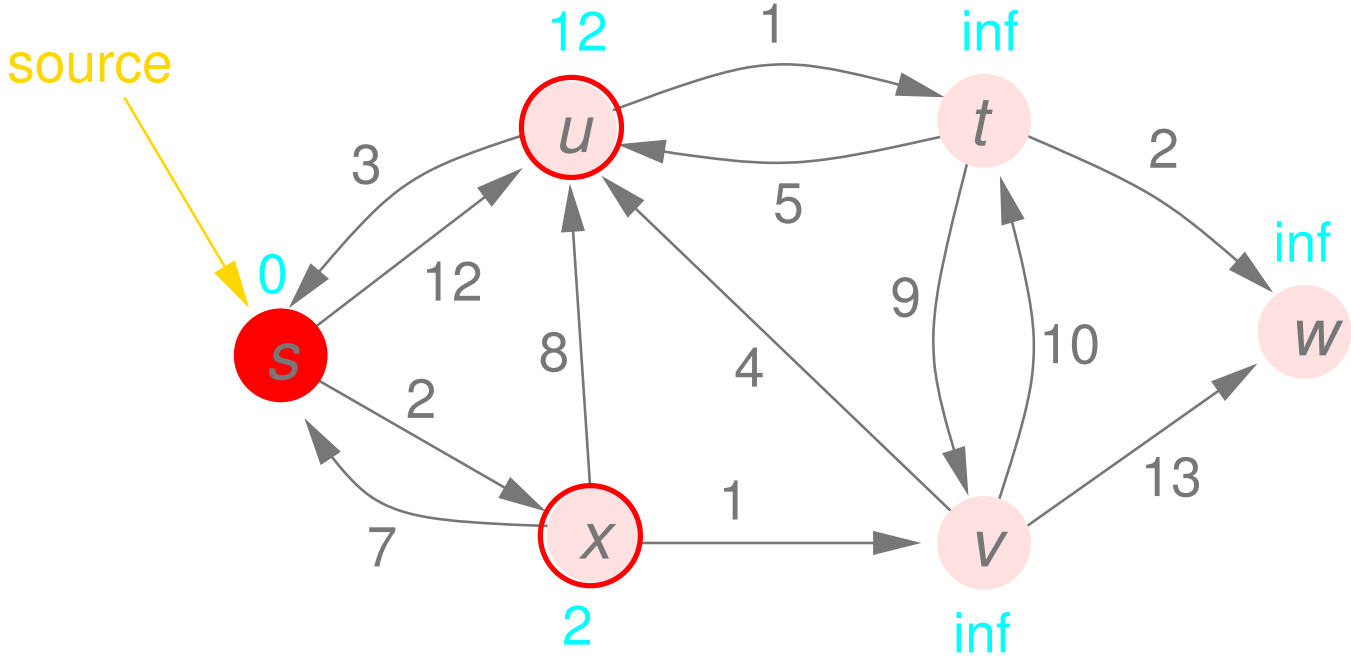
Plus courts chemins : Dijkstra.



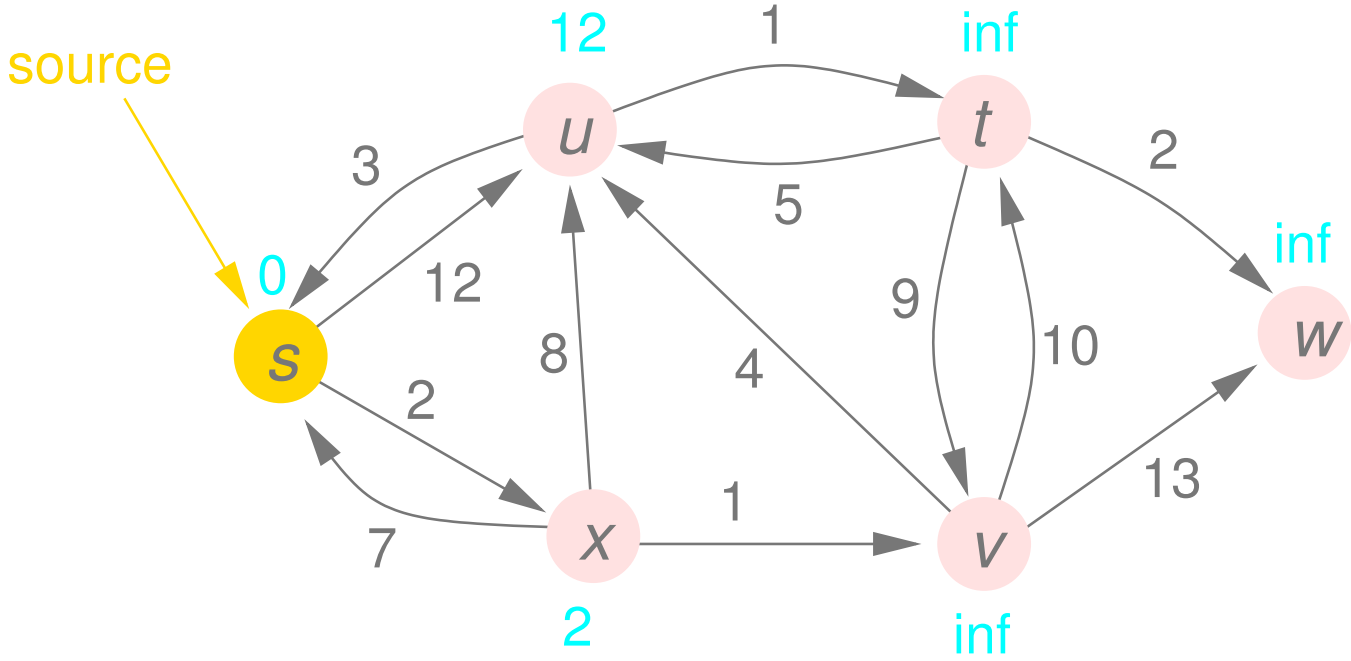
Plus courts chemins : Dijkstra.



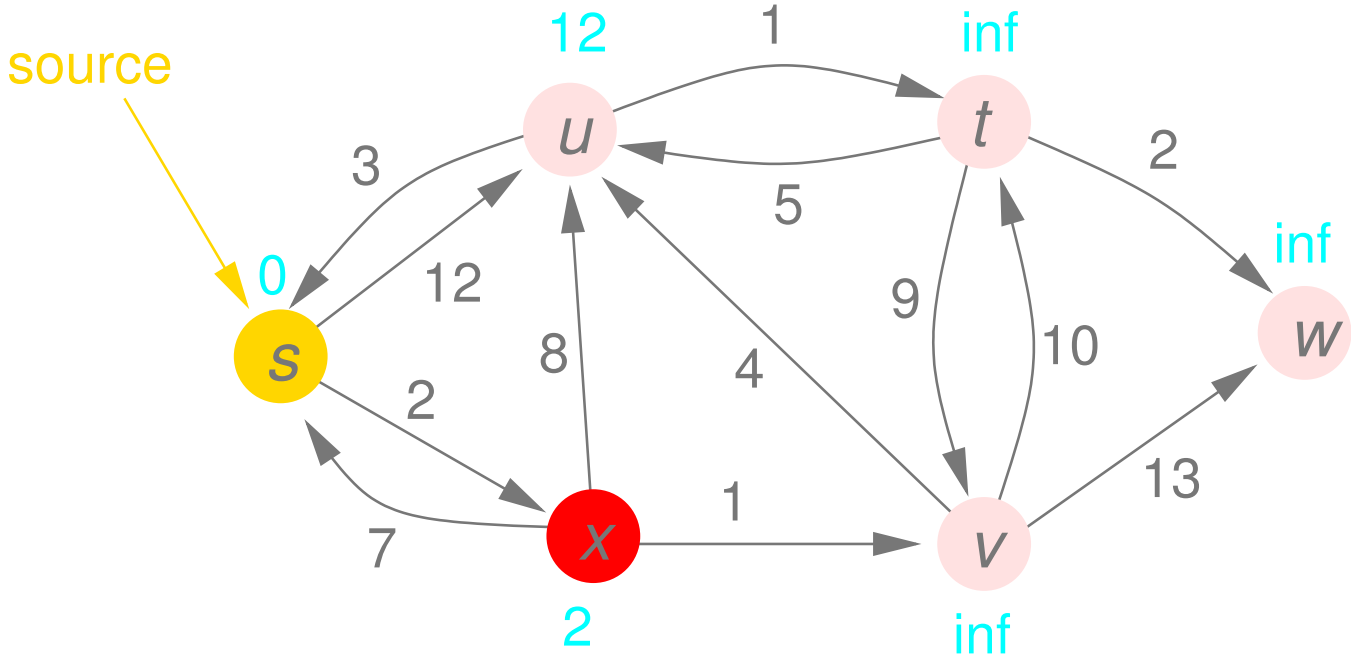
Plus courts chemins : Dijkstra.



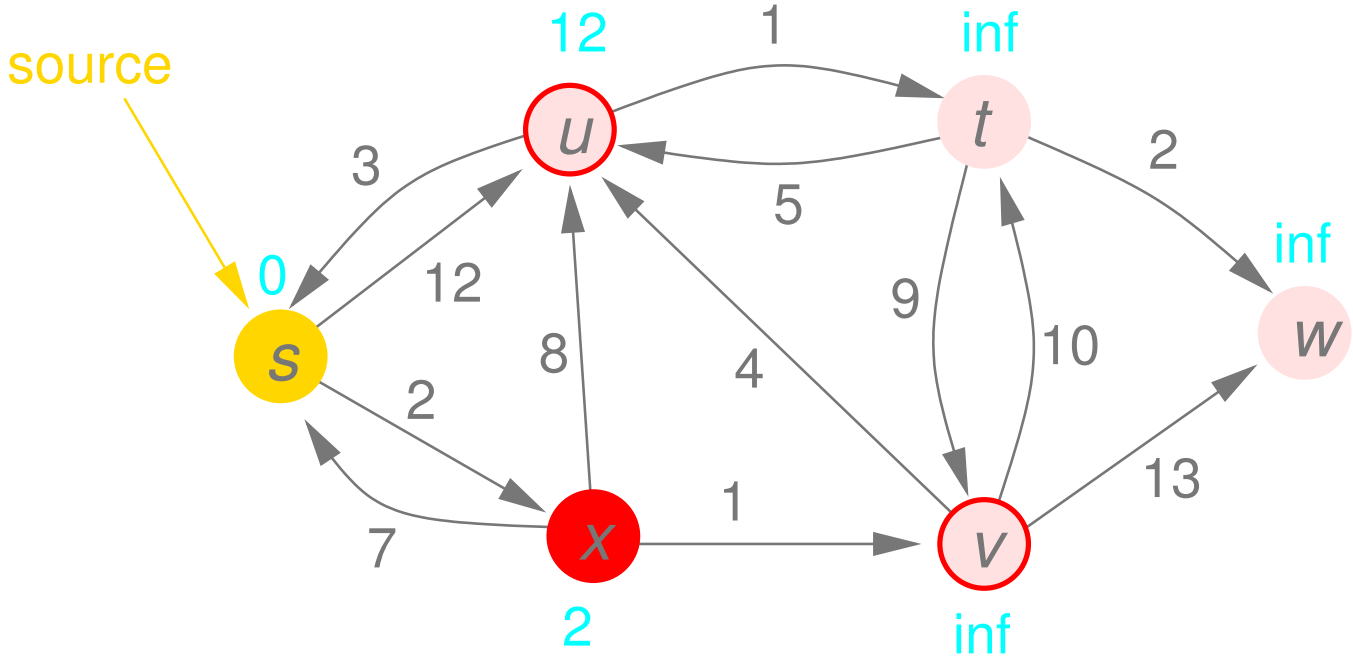
Plus courts chemins : Dijkstra.



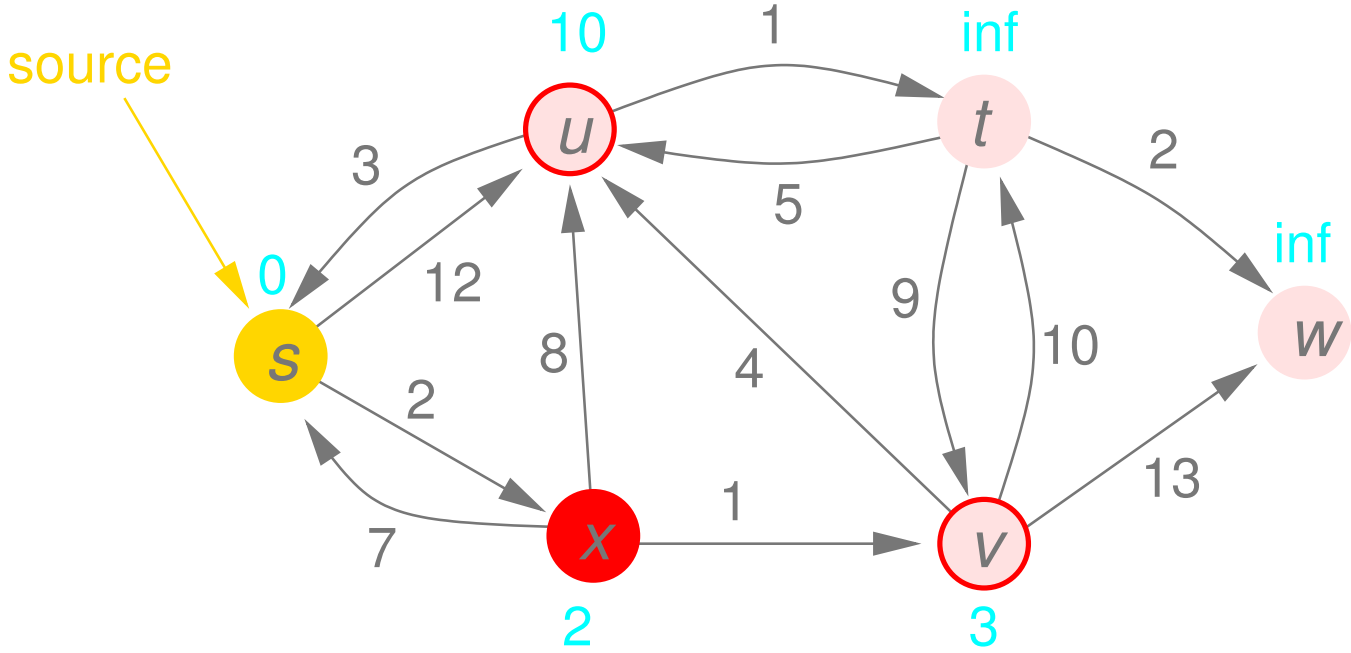
Plus courts chemins : Dijkstra.



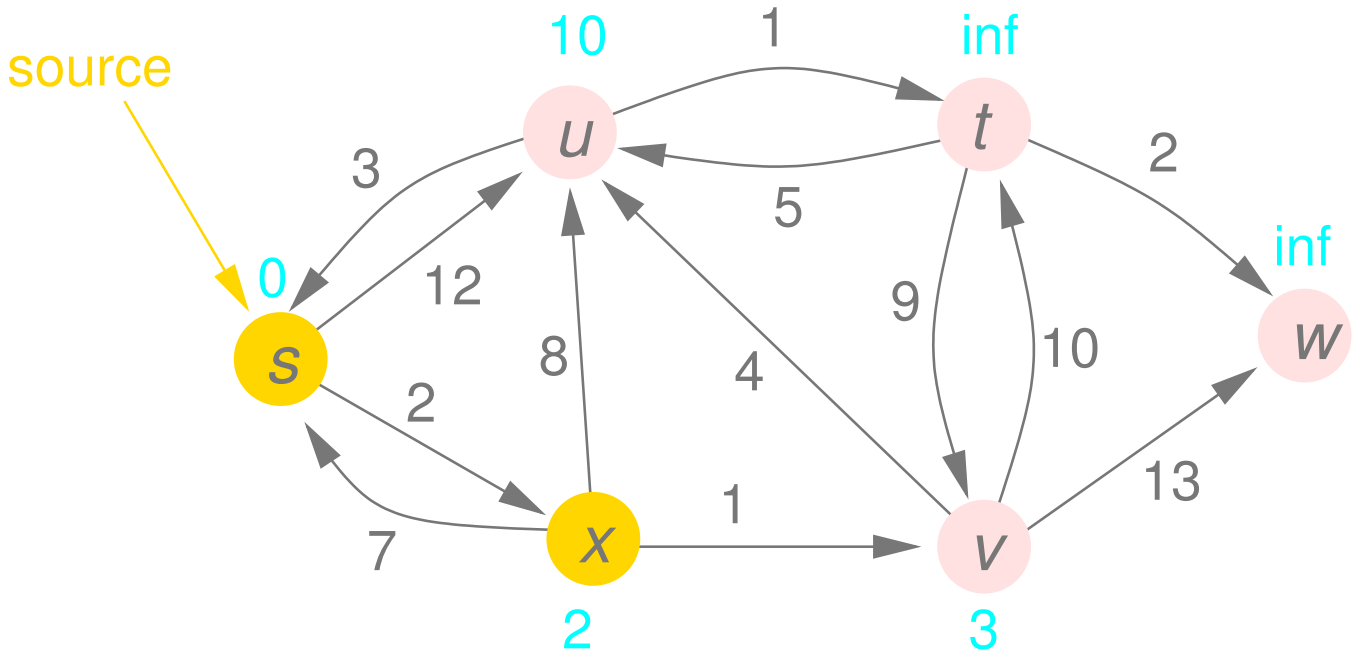
Plus courts chemins : Dijkstra.



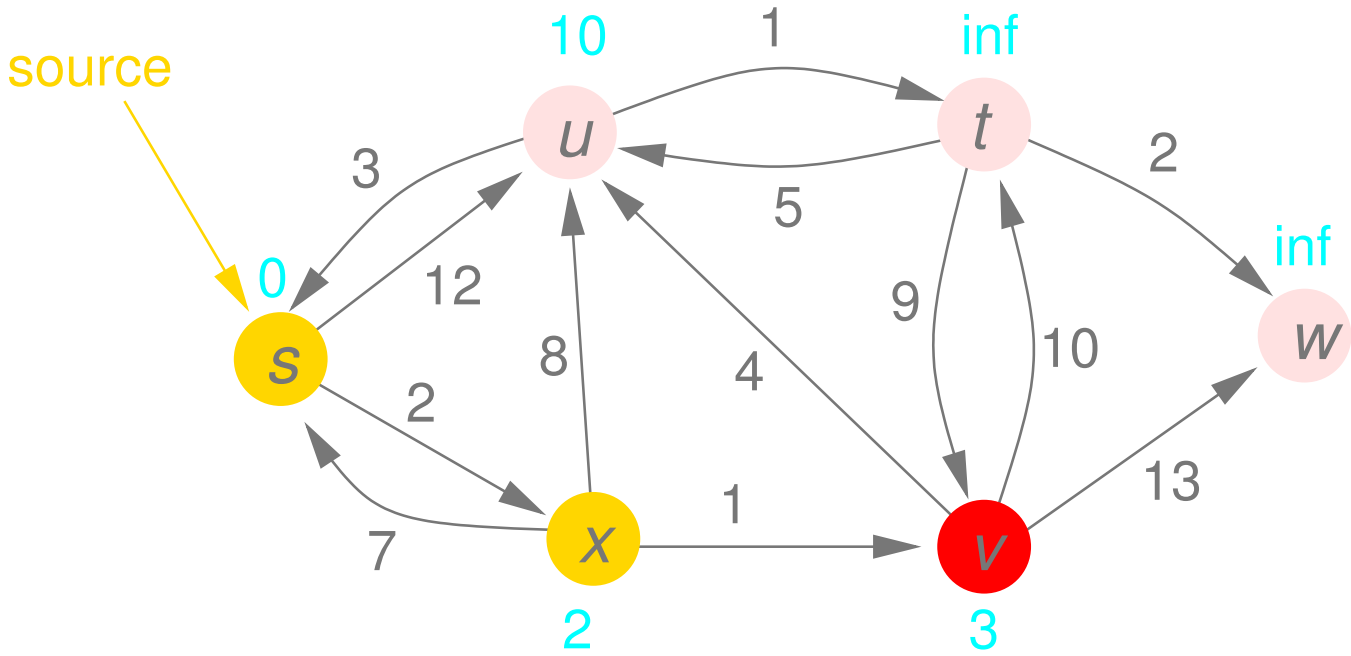
Plus courts chemins : Dijkstra.



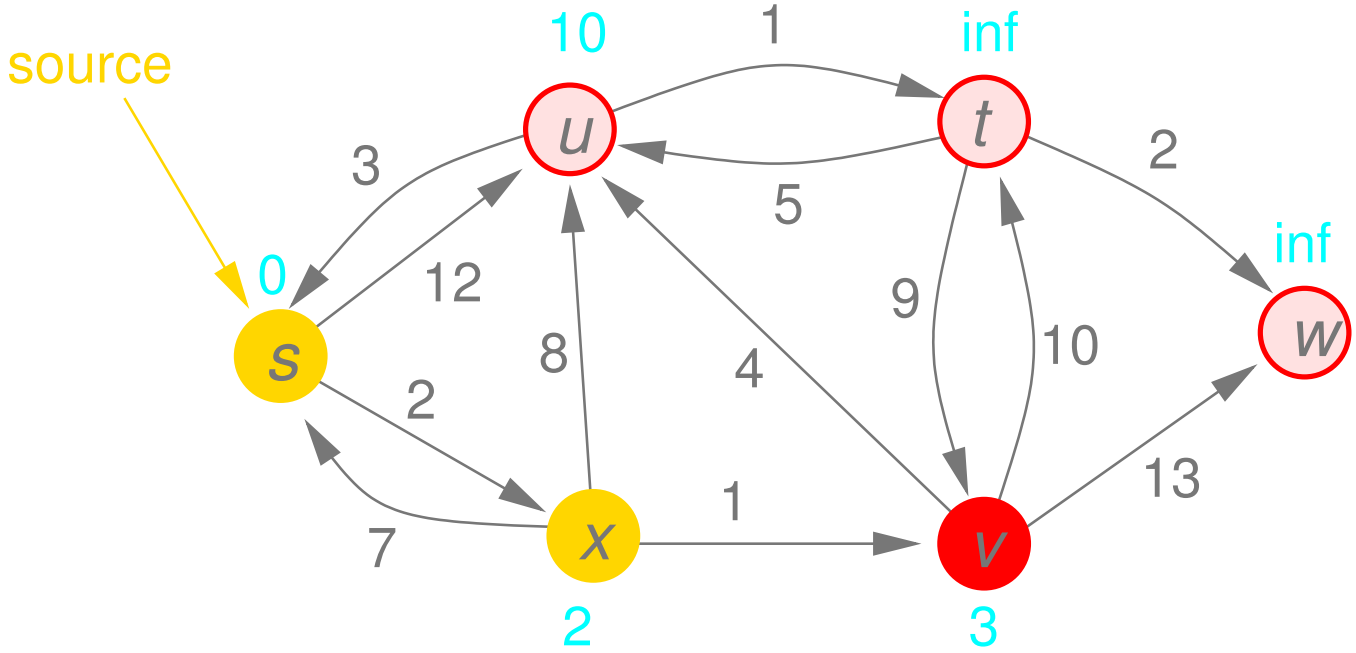
Plus courts chemins : Dijkstra.



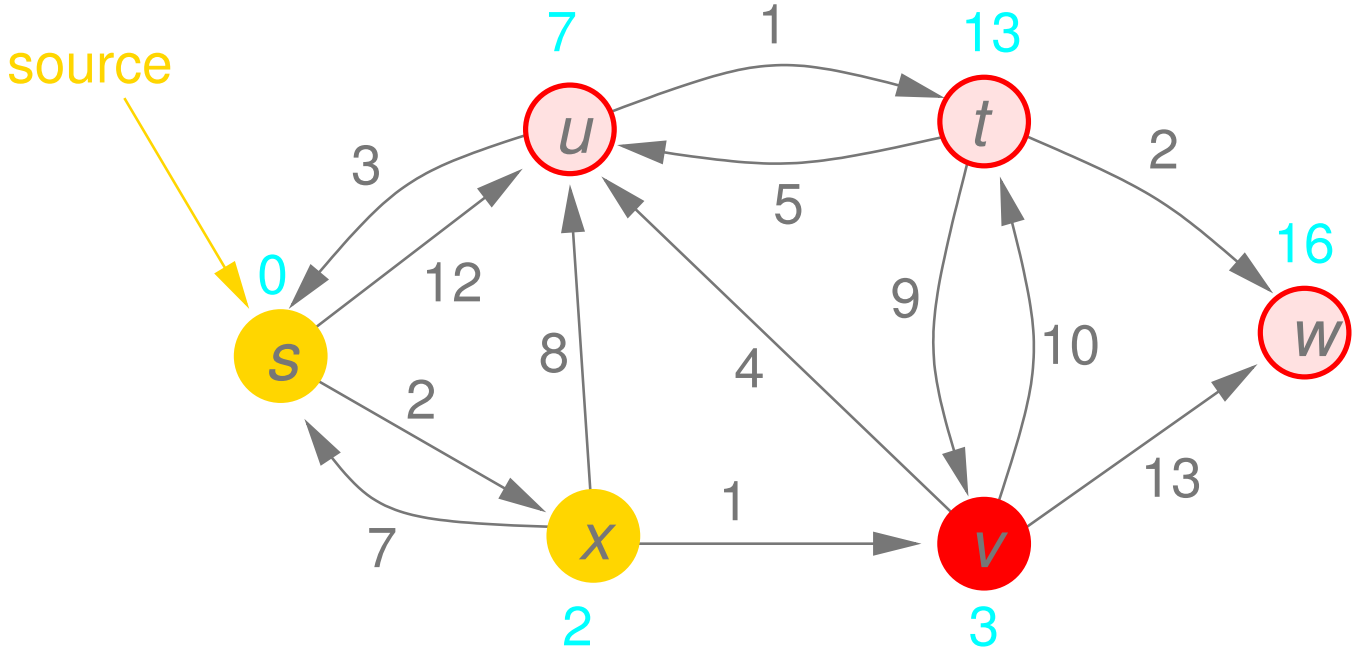
Plus courts chemins : Dijkstra.



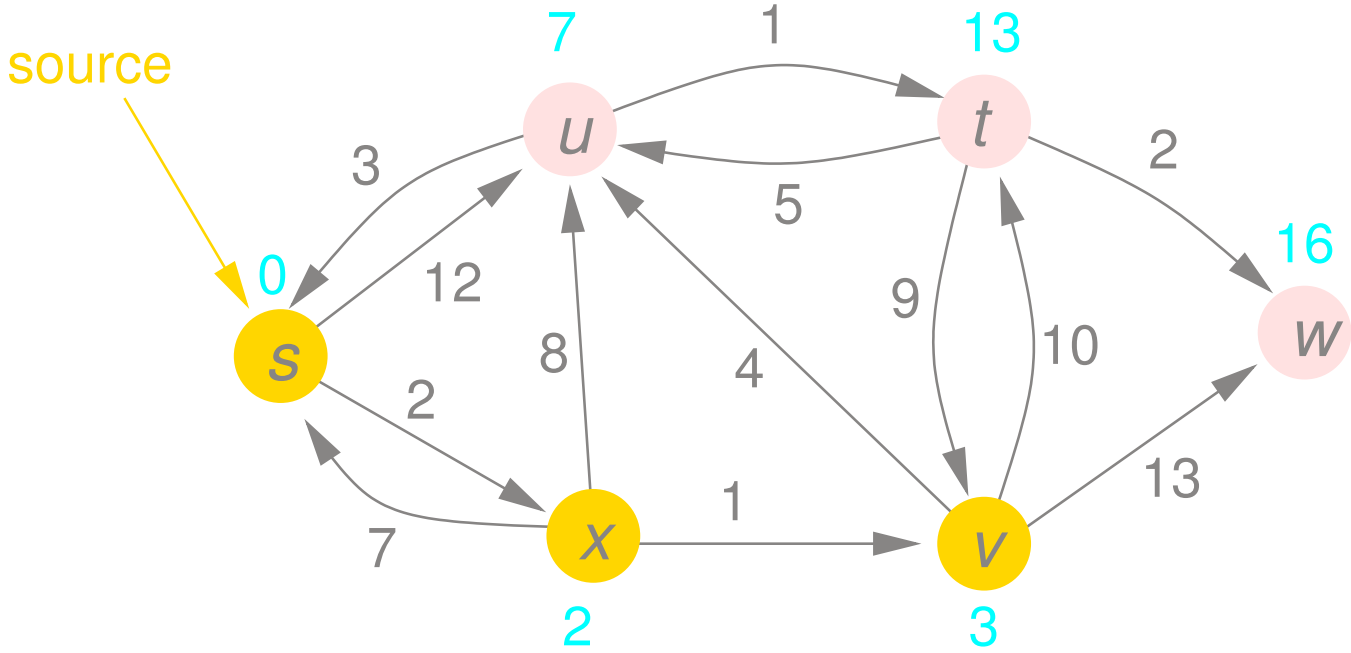
Plus courts chemins : Dijkstra.



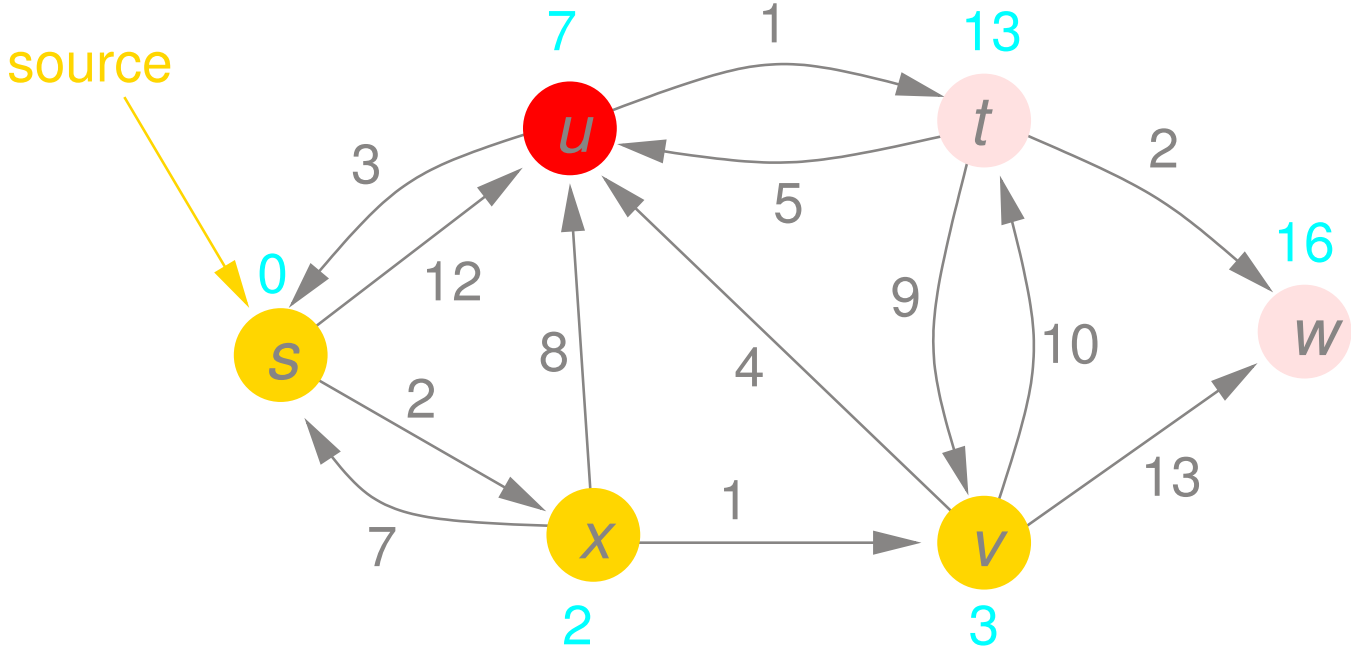
Plus courts chemins : Dijkstra.



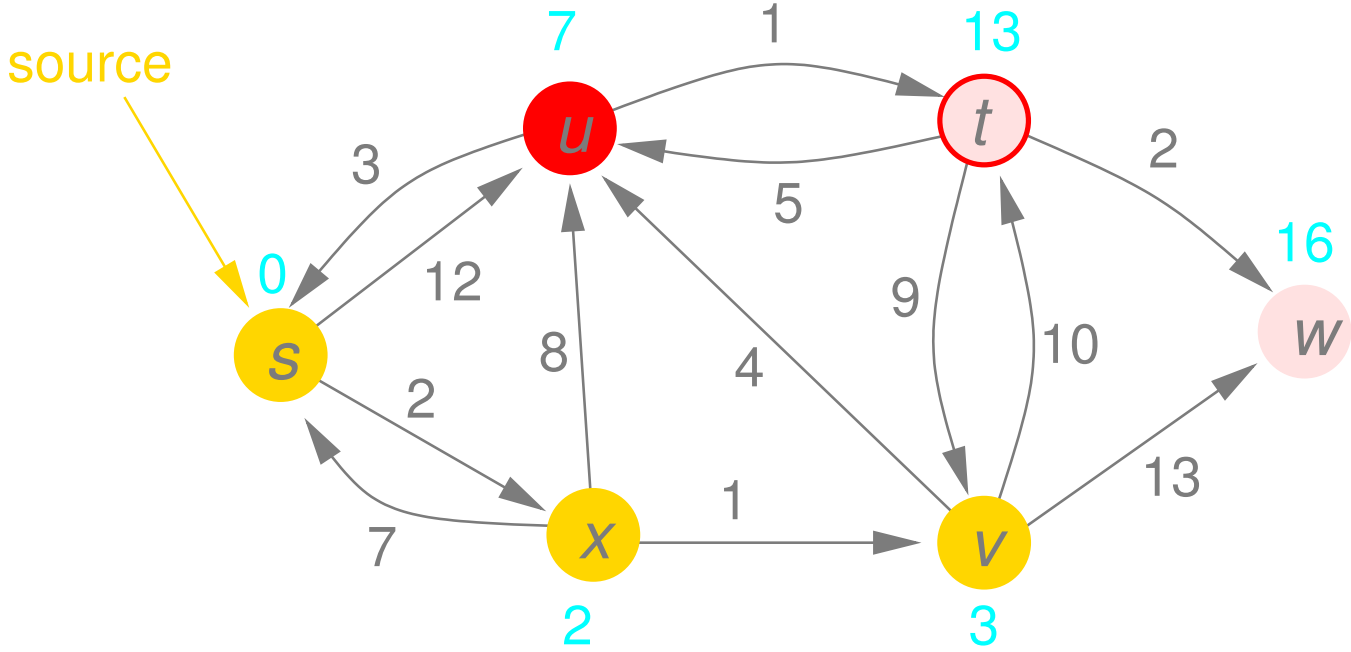
Plus courts chemins : Dijkstra.



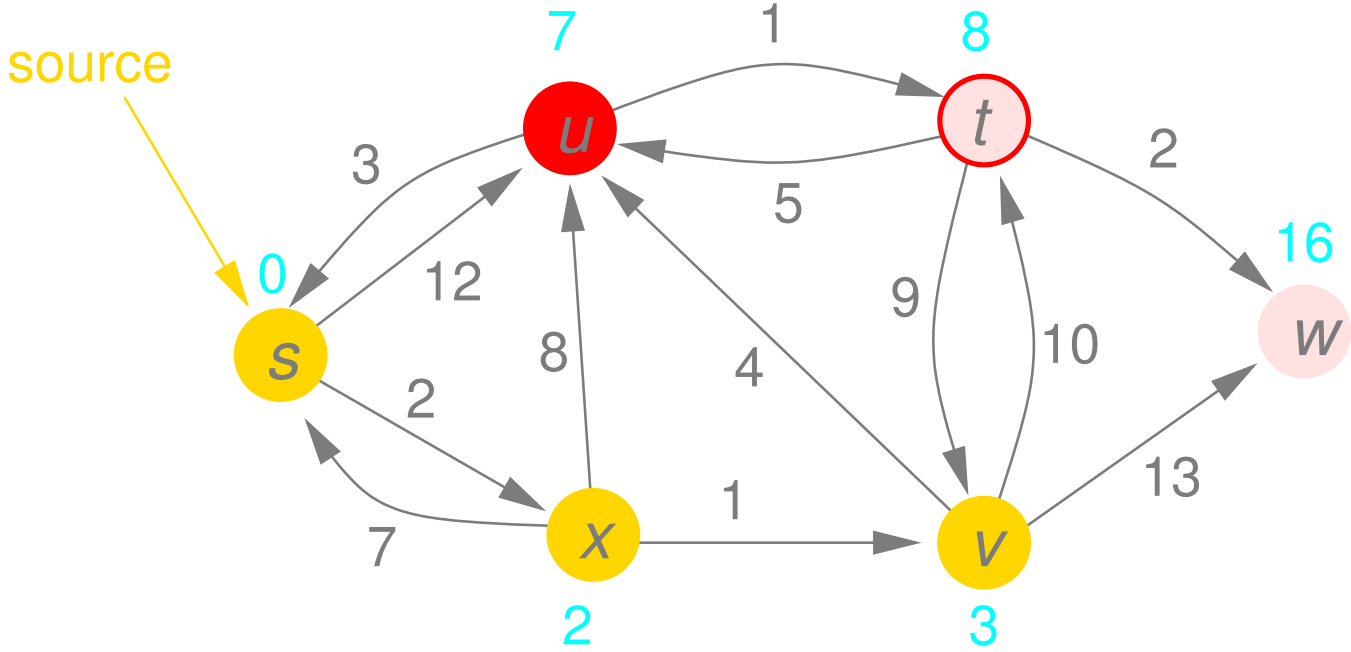
Plus courts chemins : Dijkstra.



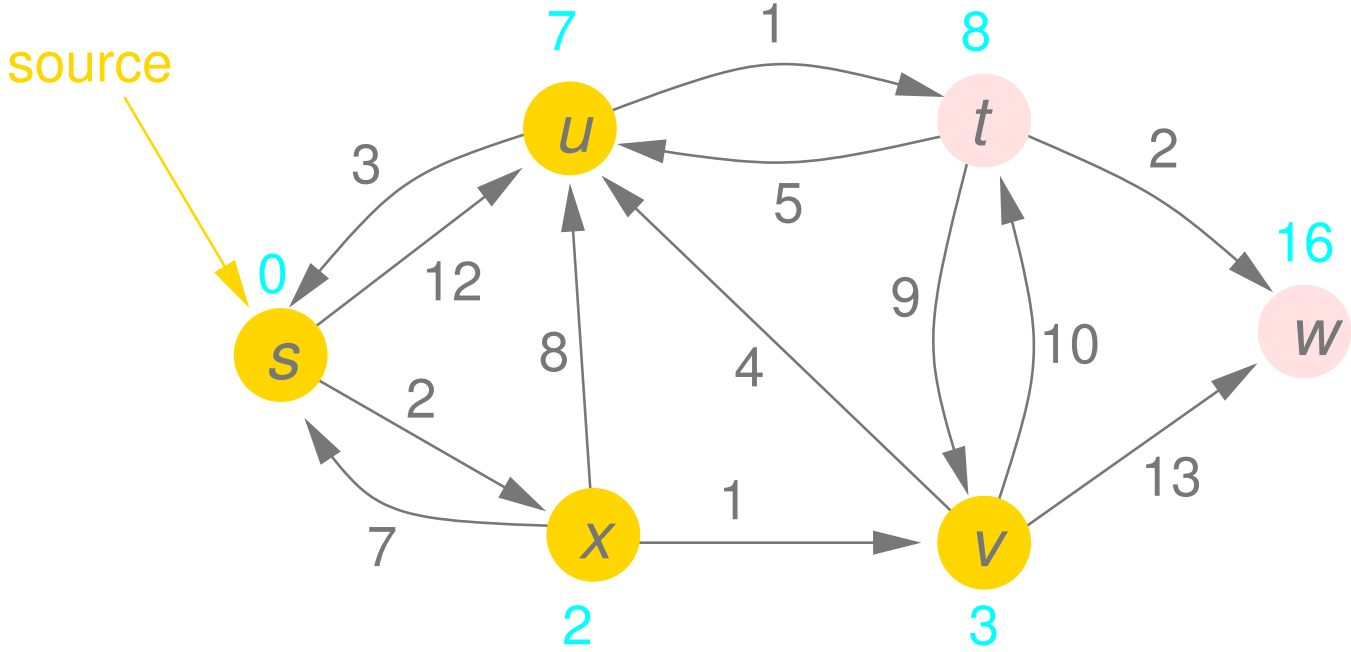
Plus courts chemins : Dijkstra.



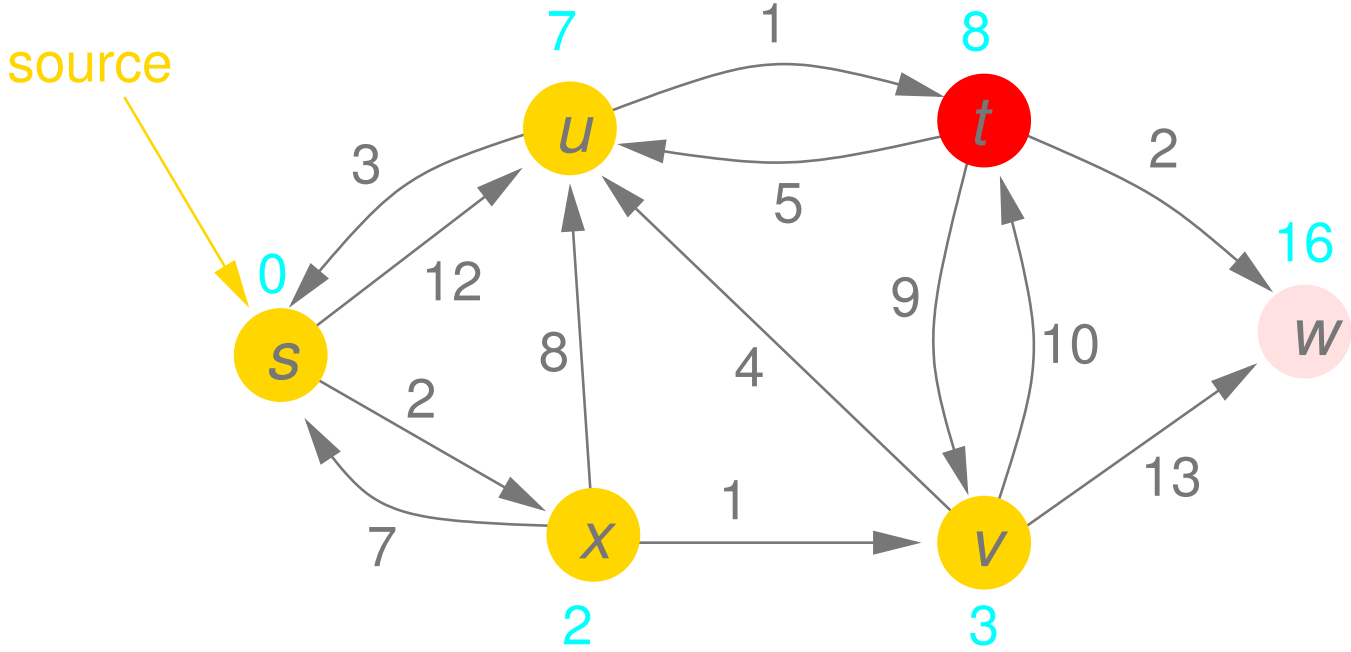
Plus courts chemins : Dijkstra.



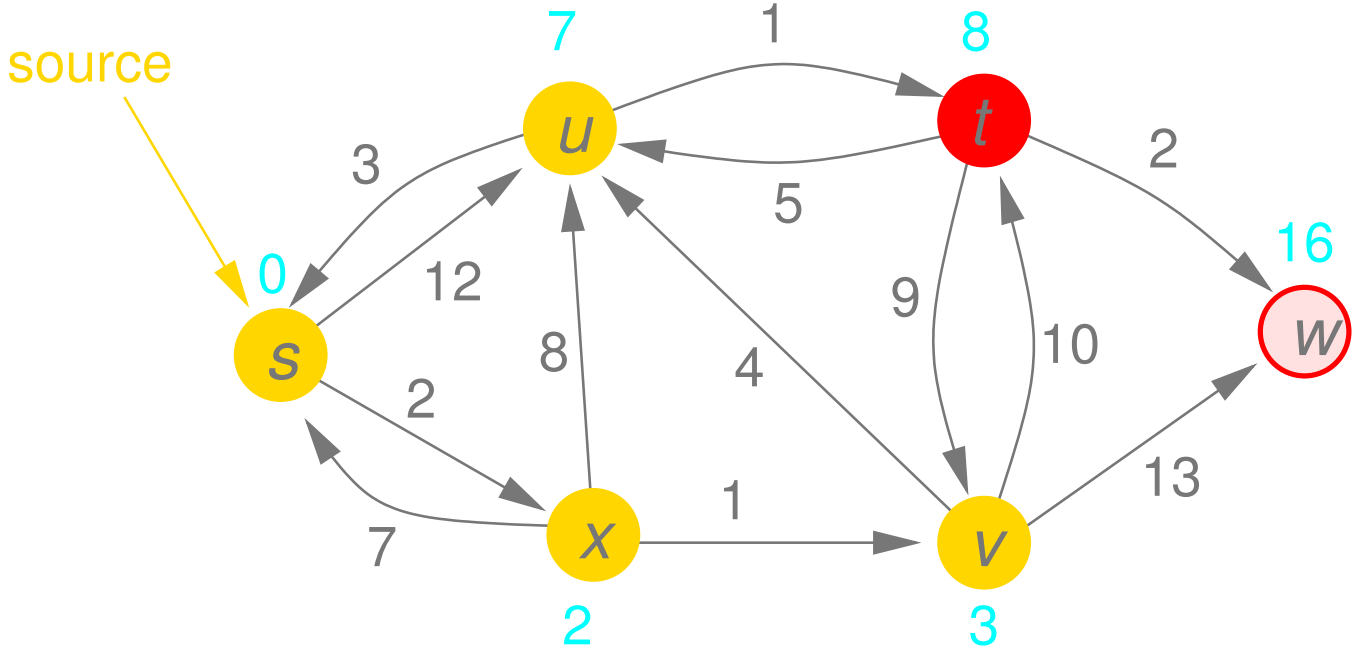
Plus courts chemins : Dijkstra.



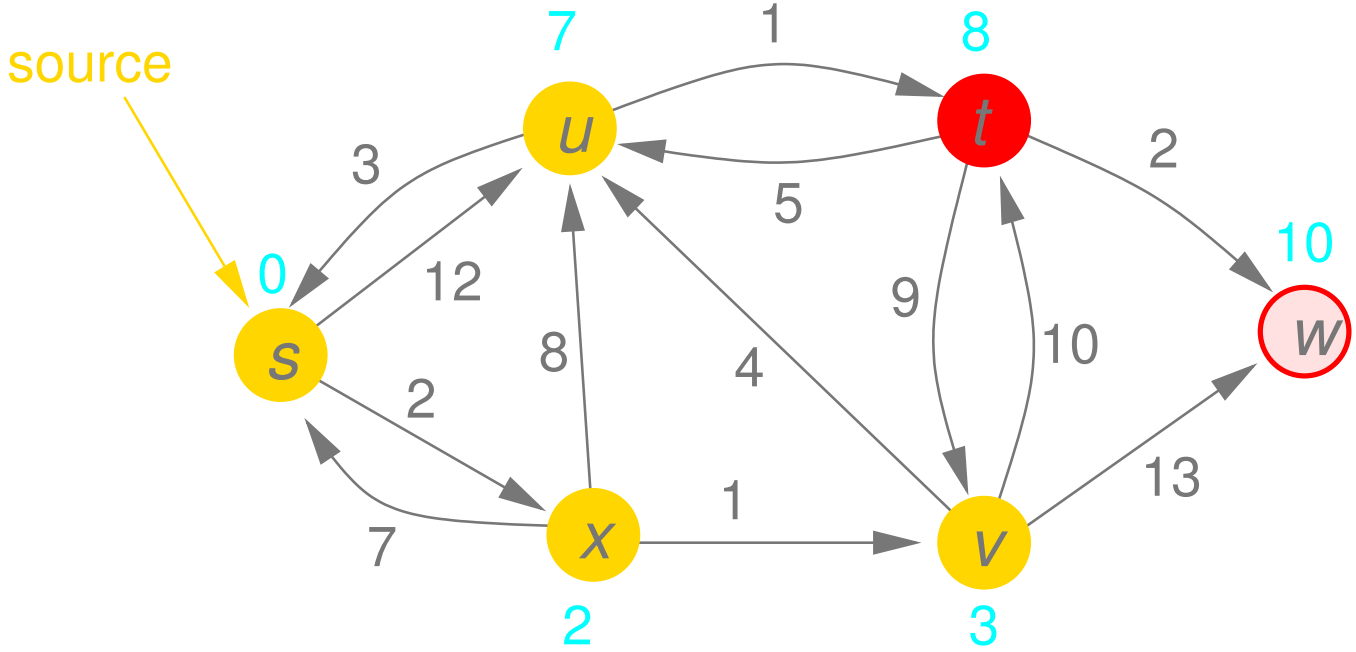
Plus courts chemins : Dijkstra.



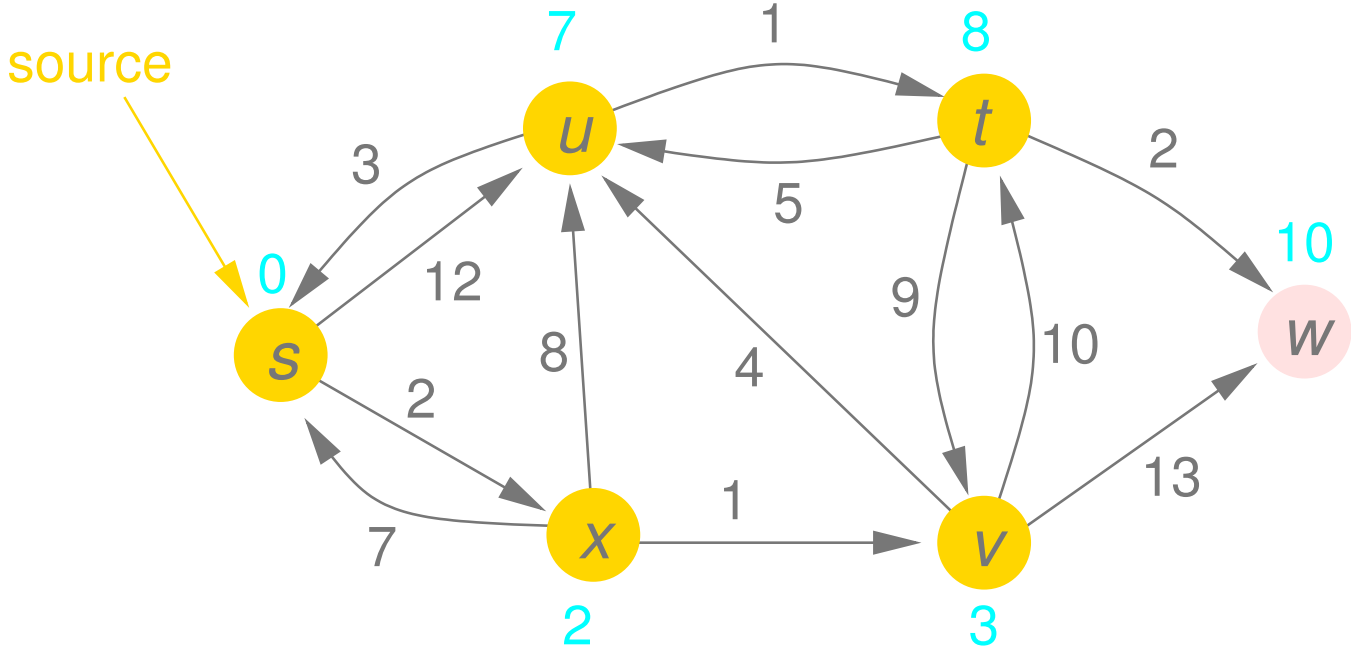
Plus courts chemins : Dijkstra.



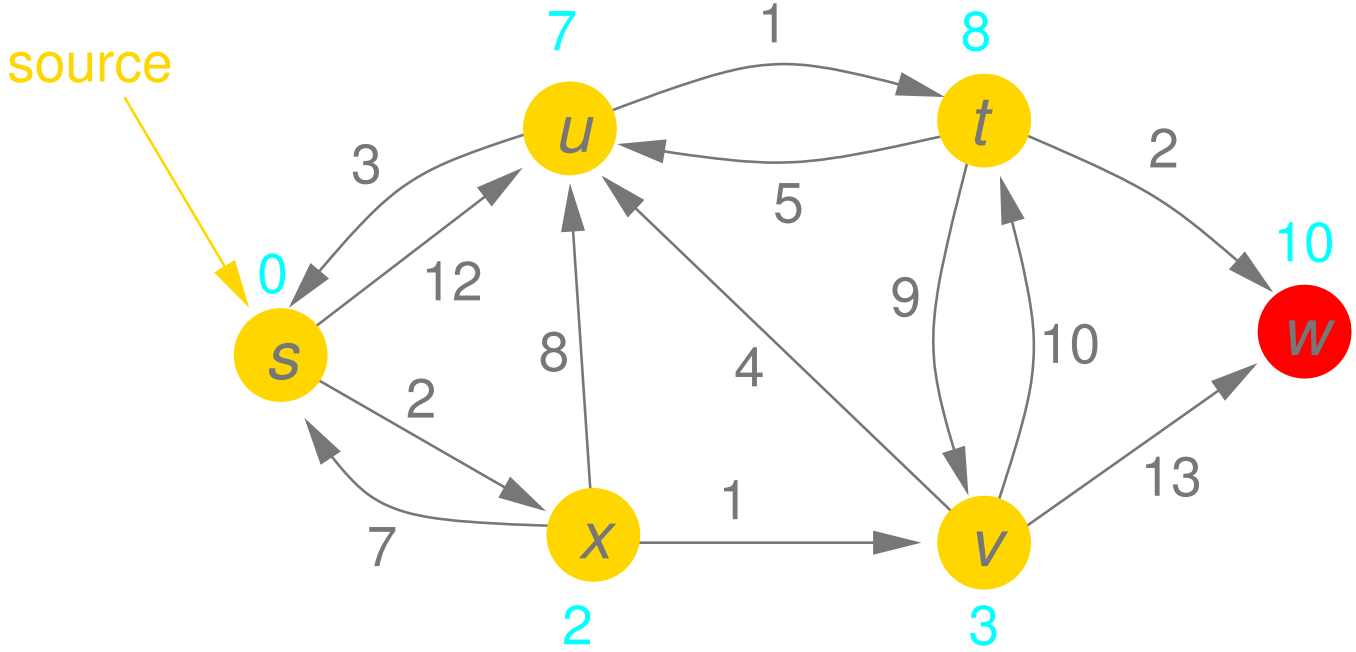
Plus courts chemins : Dijkstra.



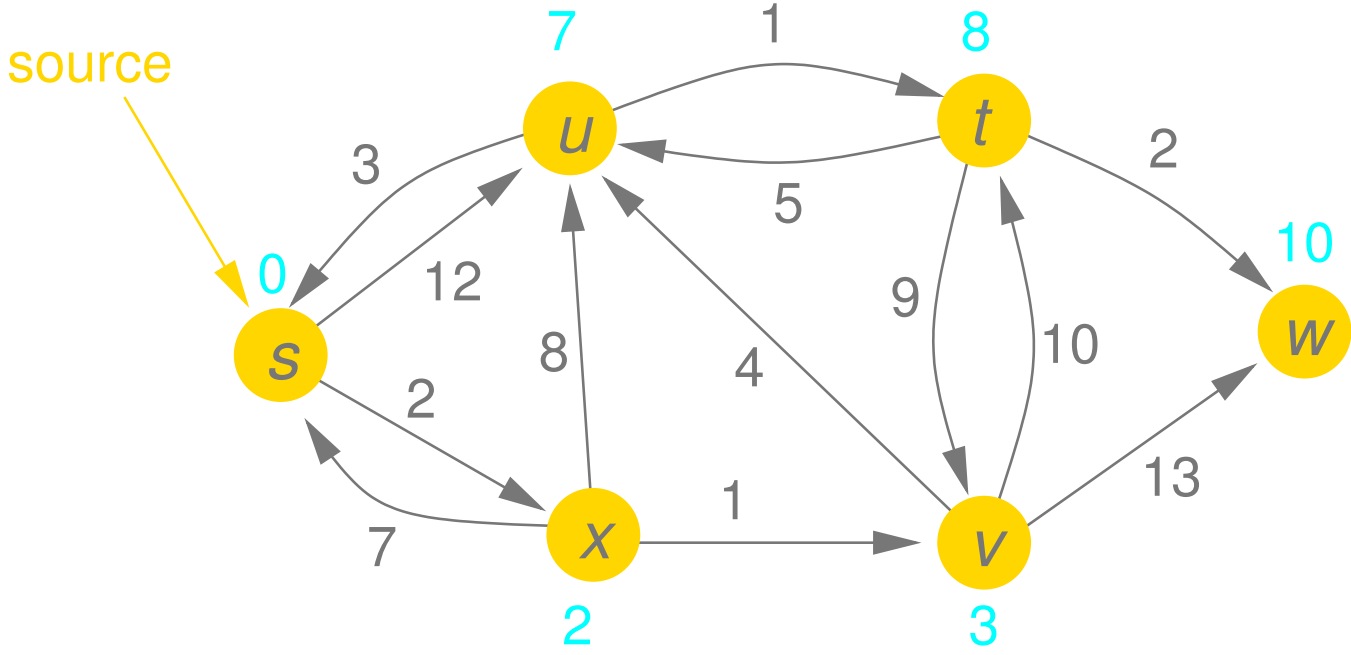
Plus courts chemins : Dijkstra.



Plus courts chemins : Dijkstra.



Plus courts chemins : Dijkstra.



Algorithme de Dijkstra.

algorithme PCC-DIJKSTRA(G, s)
in : $G = (S, A, w)$ un graphe pondéré, s un sommet de G
(F une file de priorité)

1 $F := S$, {*i.e. initialement F contient tous les sommets*}
2 **pour chaque sommet** $u \in S$ **faire**
3 $d[u] := \infty$,
4 $d[s] := 0$, ($pred[s] :=$NONE)
5 **tant que** $F \neq \emptyset$ **faire**
6 $u :=$EXTRAIRE_LE_MIN(F, d),
7 **pour chaque arc** $(u, v) \in A$ **faire**
8 **si** $d[v] > d[u] + w(u, v)$ **alors**
9 $d[v] = d[u] + w(u, v)$, ($pred[v] := u$)
10 **fin pour**,
11 **fin tant que**,
12 **renvoyer** d, (et $pred[]$)

Algorithme de Dijkstra.

algorithme PCC-DIJKSTRA(G, s)
in : $G = (S, A, w)$ un graphe pondéré, s un sommet de G
(n le nombre de sommets, m le nombre d'arcs)

```
1   F := S,                                              O(n)
2   pour chaque sommet u ∈ S faire
3       d[u] := ∞,                                       O(n)
4   d[s] := 0,
5   tant que F ≠ ∅ faire
6       u :=EXTRAIRE_LE_MIN(F, d),                       n × O(n)
7       pour chaque arc (u, v) ∈ A faire
8           si d[v] > d[u] + w(u, v) alors                m fois
9               d[v] = d[u] + w(u, v),                   (m : nombre d'arêtes)
10      fin pour,
11  fin tant que,
12  renvoyer d,
```

F : tableau contenant des sommets
EXTRAIRE_LE_MIN(F) : parcours du tableau
Coût total : $O(n^2)$

Algorithme de Dijkstra.

EXTRAIRE_LE_MIN(F) : parcours du tableau

	0	1	2	3	4	5	6	7	8	9	10
F	6	8	10	3	7	2	4	1	9	5	0

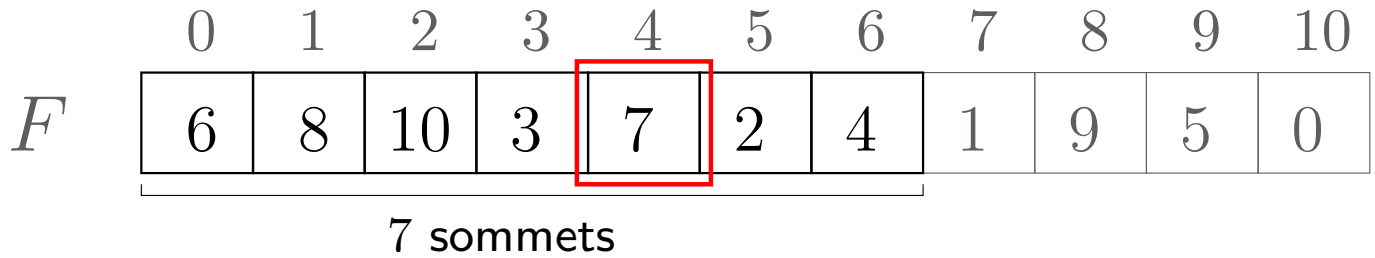
7 sommets

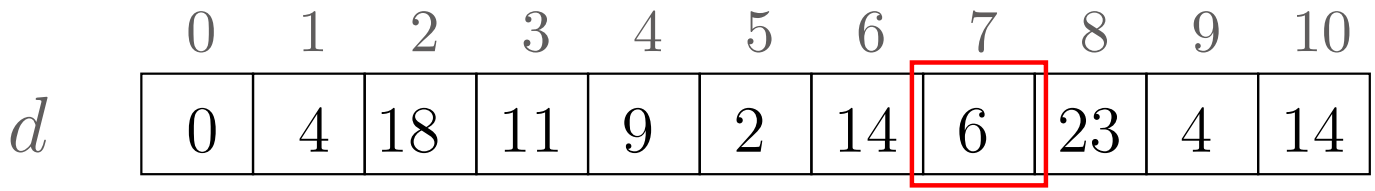
	0	1	2	3	4	5	6	7	8	9	10
d	0	4	18	11	9	2	14	6	23	4	14

F représente l'ensemble $\{6, 8, 10, 3, 7, 2, 4\}$

Algorithme de Dijkstra.

EXTRAIRE_LE_MIN(F) : parcours du tableau

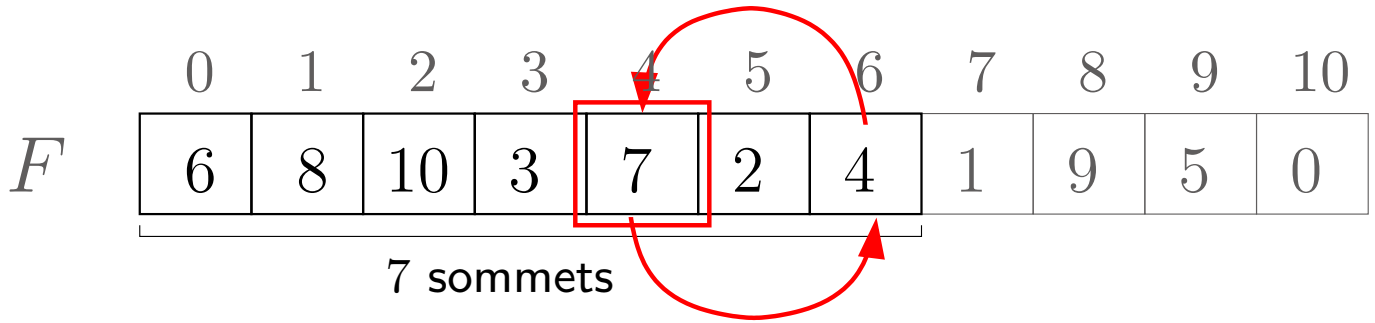


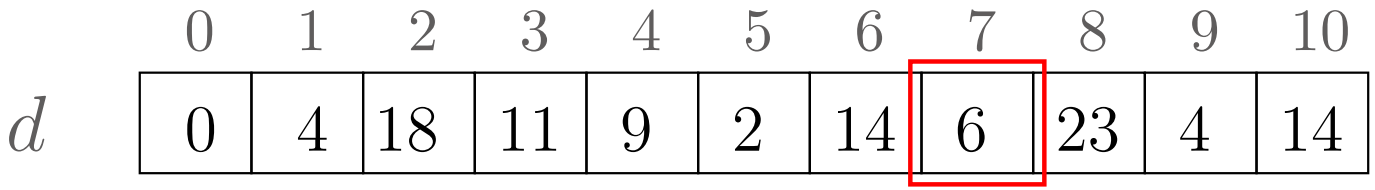


Le sommet 7 est celui qui a la plus petite distance $d[7] = 6$.

Algorithme de Dijkstra.

EXTRAIRE_LE_MIN(F) : parcours du tableau





On extrait de F le sommet 7.

Algorithme de Dijkstra.

EXTRAIRE_LE_MIN(F) : parcours du tableau

	0	1	2	3	4	5	6	7	8	9	10
F	6	8	10	3	4	2	7	1	9	5	0

6 sommets

	0	1	2	3	4	5	6	7	8	9	10
d	0	4	18	11	9	2	14	6	23	4	14

F représente l'ensemble $\{6, 8, 10, 3, 4, 2\}$

Algorithme de Dijkstra.

algorithme PCC-DIJKSTRA(G, s)
in : $G = (S, A, w)$ un graphe pondéré, s un sommet de G
(n le nombre de sommets, m le nombre d'arcs)

1	$F := S$,	$O(n)$
2	**pour chaque sommet** $u \in S$ **faire**	
3	$\quad d[u] := \infty$,	$O(n)$
4	$d[s] := 0$,	
5	**tant que** $F \neq \emptyset$ **faire**	
6	$\quad u :=$EXTRAIRE_LE_MIN(F, d),	$n \times O(\log\, n)$
7	$\quad$ **pour chaque arc** $(u, v) \in A$ **faire**	
8	$\quad\quad$ **si** $d[v] > d[u] + w(u, v)$ **alors**	
9	$\quad\quad\quad d[v] = d[u] + w(u, v)$,	$m \times O(\log\, n)$
9	$\quad\quad\quad$DIMINUER_LA_CLE$(v, d[v], F)$,	$m \times O(\log\, n)$
10	$\quad$ **fin pour**,	
11	**fin tant que**,	
12	**renvoyer** d,	

F : tas binaire.
Extraire le min, diminuer la clé : $O(\log\, n)$.
Coût total : $O(m \times \log\, n)$

Algorithme de Dijkstra : complexité

En faisant une recherche linéaire dans F : $\mathcal{O}(|S|^2 + |A|)$

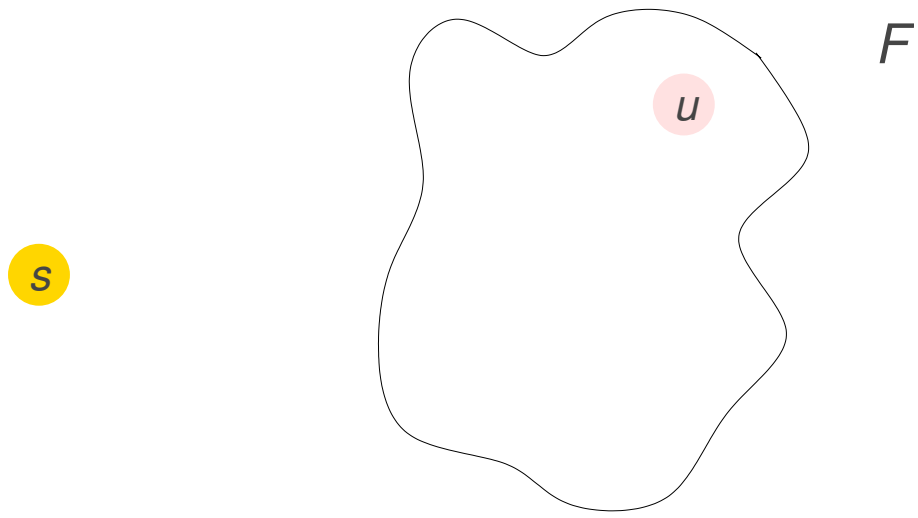
En implémentant F avec un tas binaire :

- *initialisations :* $\mathcal{O}(|S|)$
- *Construction du tas :* $\mathcal{O}(|S|)$
- *maintien du tas :*
  $\mathcal{O}(|S| \times \log |S| + |A| \times \log |S|) = \mathcal{O}(|A| \times \log |S|)$

Le coût total est $\mathcal{O}(|A| \times \log |S|)$ si le graphe est connexe

($\mathcal{O}(|S| \times \log |S| + |A|)$ avec un tas de Fibonacci)

Algorithme de Dijkstra : preuve.

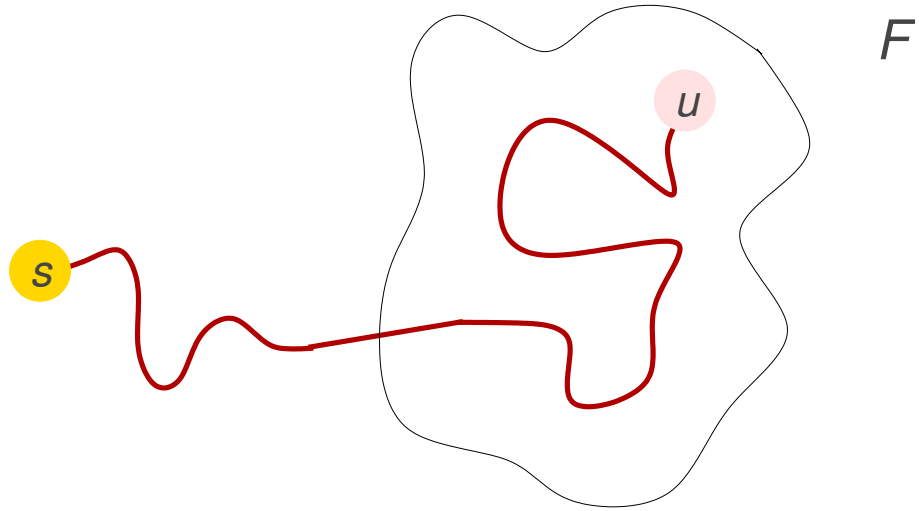


Supposition. quand u est extrait de F, $d[u] > \delta(s, u)$

[on suppose que u est le premier dans ce cas]

▶

Algorithme de Dijkstra : preuve.

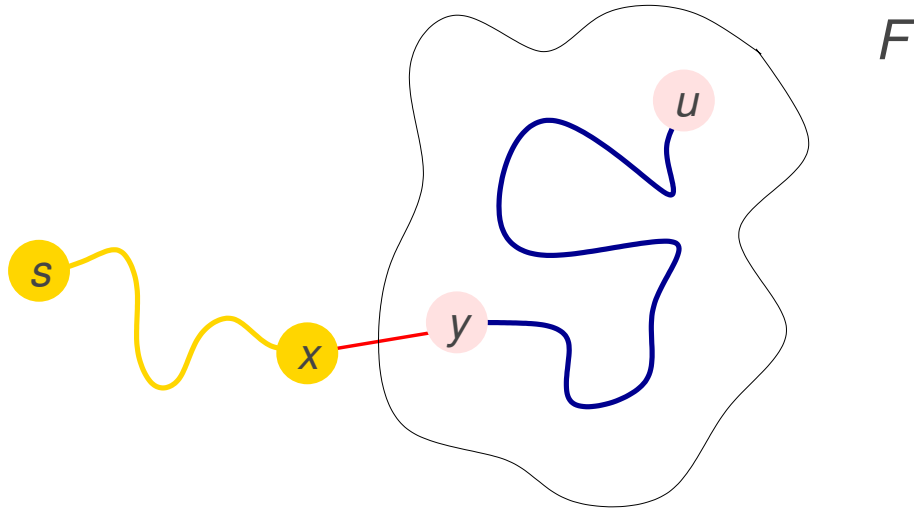


Supposition. quand u est extrait de F, $d[u] > \delta(s, u)$

[on suppose que u est le premier dans ce cas]

- considérons un plus court chemin entre s et u,

Algorithme de Dijkstra : preuve.

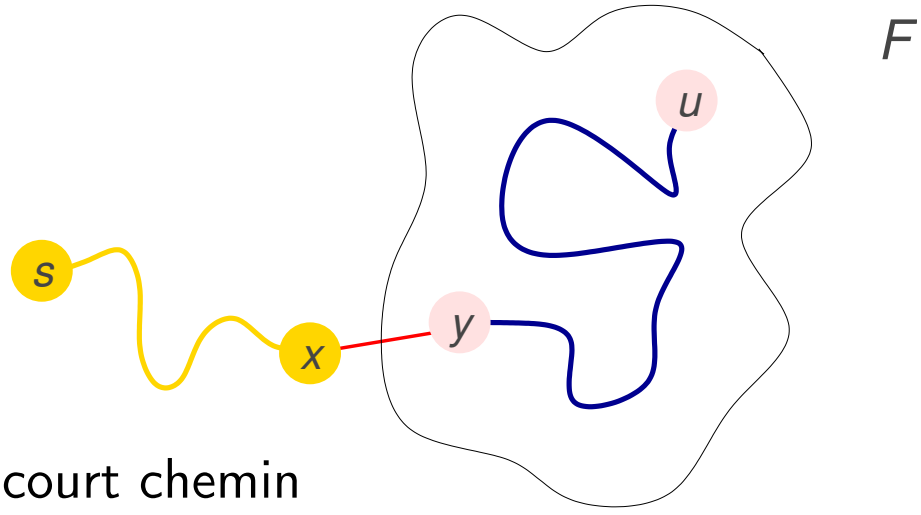


Supposition. quand u est extrait de F, $d[u] > \delta(s, u)$

[on suppose que u est le premier dans ce cas]

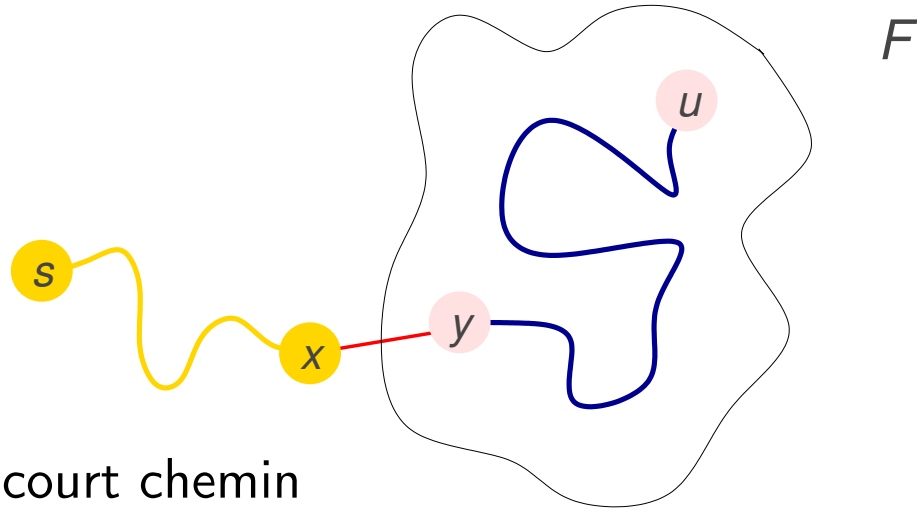
- considérons un plus court chemin entre s et u,
- soit y le premier sommet de ce chemin qui est dans F, et x le sommet précédent y dans le chemin.

Algorithme de Dijkstra : preuve.



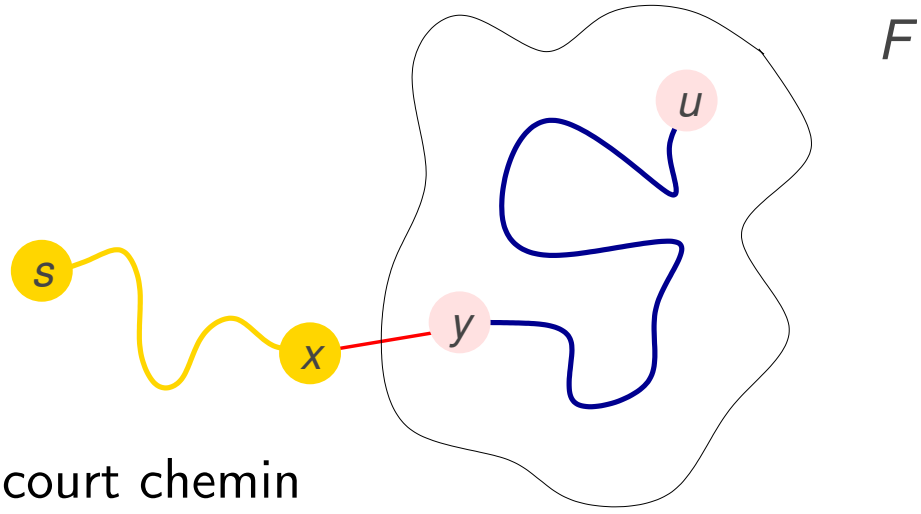
- le chemin jaune est un plus court chemin
  (car sous chemin d'un plus court chemin entre s et u)

Algorithme de Dijkstra : preuve.



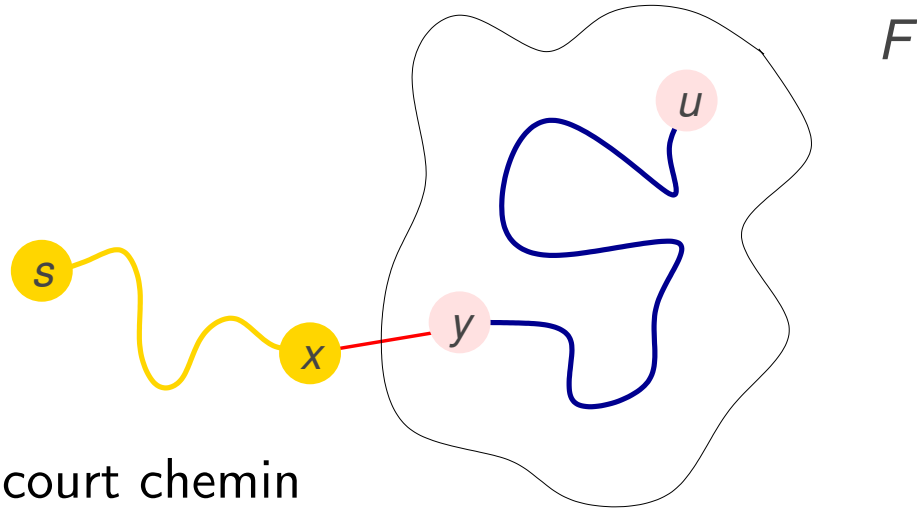
- le chemin jaune est un plus court chemin
  (car sous chemin d'un plus court chemin entre s et u)
- le chemin jaune et l'arête rouge idem (même raison)

Algorithme de Dijkstra : preuve.



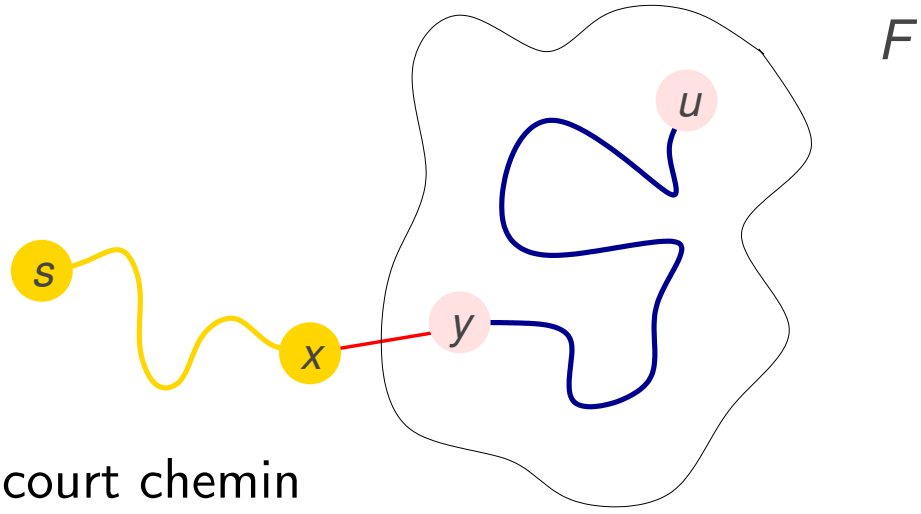
- le chemin jaune est un plus court chemin
  (car sous chemin d'un plus court chemin entre s et u)
- le chemin jaune et l'arête rouge idem (même raison)
- $\delta(s, y) \leq \delta(s, u)$ et $d[y] = \delta(s, y)$ car (x, y) a été relaché

Algorithme de Dijkstra : preuve.



- le chemin jaune est un plus court chemin
  (car sous chemin d'un plus court chemin entre s et u)
- le chemin jaune et l'arête rouge idem (même raison)
- $\delta(s, y) \leq \delta(s, u)$ et $d[y] = \delta(s, y)$ car (x, y) a été relaché
- $d[u] \leq d[y]$ puisqu'on a extrait u et non pas y

Algorithme de Dijkstra : preuve.



- le chemin jaune est un plus court chemin
  (car sous chemin d'un plus court chemin entre s et u)
- le chemin jaune et l'arête rouge idem (même raison)
- $\delta(s, y) \leq \delta(s, u)$ et $d[y] = \delta(s, y)$ car (x, y) a été relaché
- $d[u] \leq d[y]$ puisqu'on a extrait u et non pas y

donc $d[u] \leq \delta(s, u)$ ce qui contredit l'hypothèse $d[u] > \delta(s, u)$