Programmation 3
CM séance 3 : Graphes

Manon Scholivet, adapté des cours de Florian Bridoux

23 septembre 2020

Plan

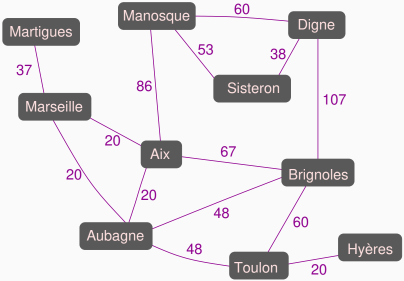
Un graphe c'est quoi ?

Définissons un graphe !

Un graphe, c'est un ensemble de sommets et d'arêtes (ou arcs), ces derniers reliant les sommets entre eux.

Un graphe permet de représenter une relation, une distance.

Exemple : des distances routières.



Ok, c'est joli et tout... mais à quoi ça sert ?

Les graphes permettent de modéliser des problèmes. De très divers et nombreux problèmes :

Par exemple :

- Réseaux de métro
- Des arbres (généalogiques, de probabilité, ...)
- Réseaux sociaux
- ...

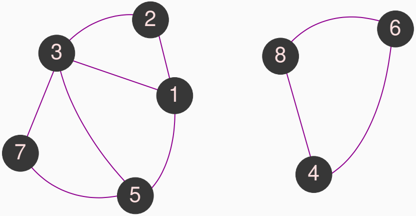
Problèmes rattachés :

- Plus court chemin
- Calcul de flots min/max
- Coloriage
- ...

Définition formelle : Graphe non-orienté

Graphe G = (S,A)

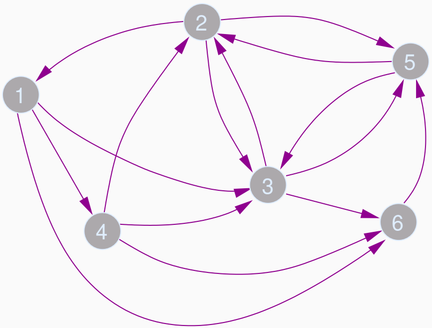
- S : ensemble des sommets
- A : ensemble des arêtes, $A = \{\{u, v\} | u, v \in S\}$



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

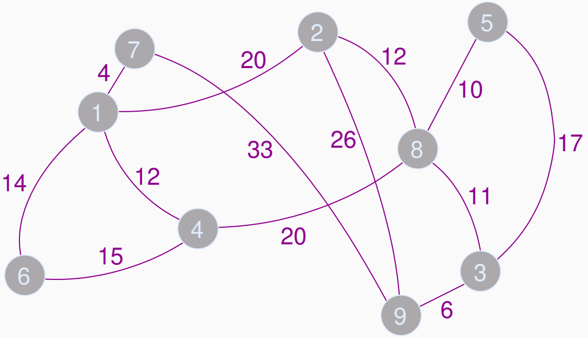
Graphes orientés

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

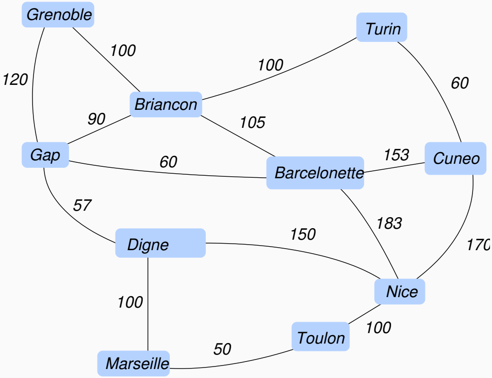


Graphes pondérés

$G = (S, A, w)$: chaque arête (u, v) a un poids, $w(u, v)$

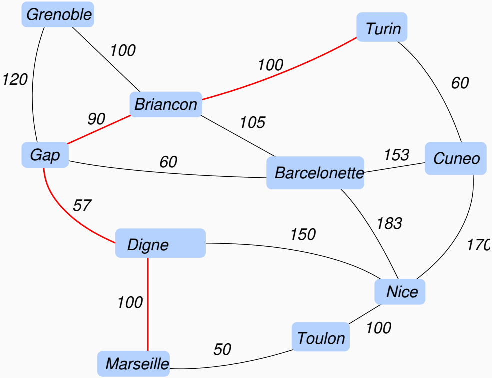


Exemple : Trajet le plus court entre deux villes.



Graphe représentant les liaisons routières et les durées de trajet.

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 :

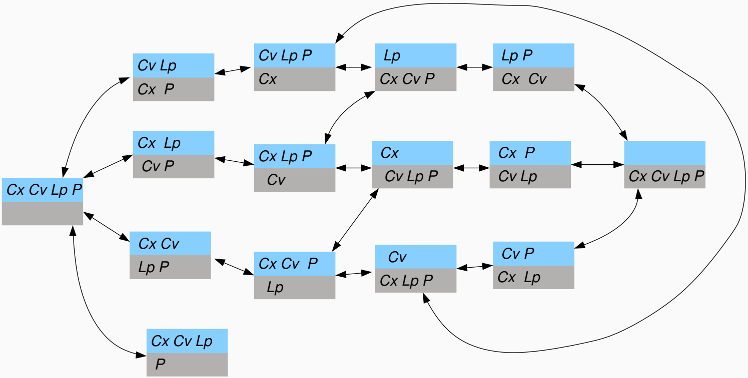
- loup et chèvre,
- chèvre et choux.

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

Exemple : le choux, la chèvre et le loup.

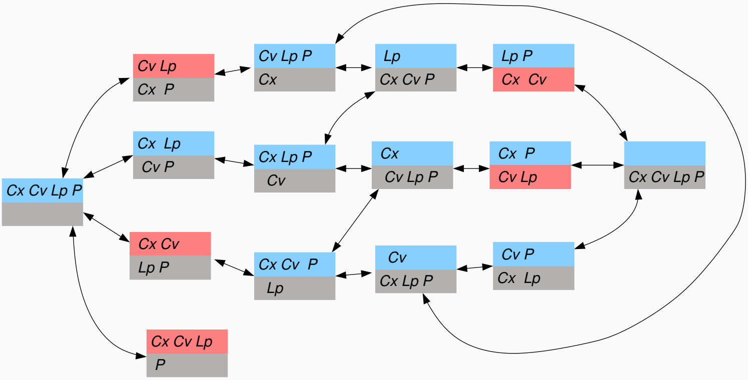
1. Quels sont les configurations possibles de cette aventure ? On pourra les décrire en observant les personnages qui peuvent se trouver sur la rive de départ et la rive d'arrivée.
2. Modéliser alors le problème sous la forme d'un graphe (non orienté) dont les sommets sont les configurations possibles et les arêtes représentent l'évolution possible de l'aventure.
3. Résoudre le problème initial. Décrire à la bergère les allers-retours qu'elle doit faire. Combien de traversées la bergère doit-elle faire au minimum ? Existe-t-il plusieurs solutions avec ce nombre minimal de traversées ?

Exemple : le choux, la chèvre et le loup.



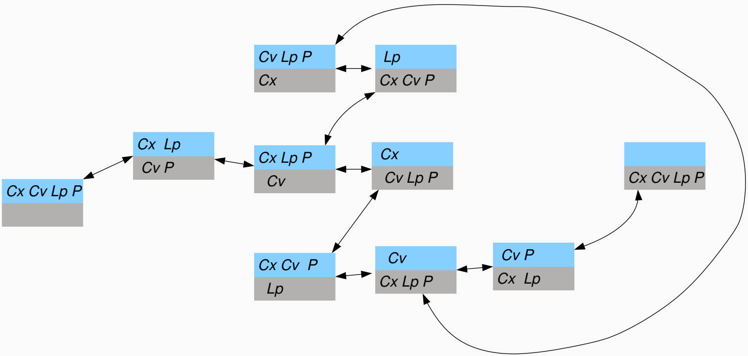
Graphe représentant les états et les transitions.

Exemple : le choux, la chèvre et le loup.

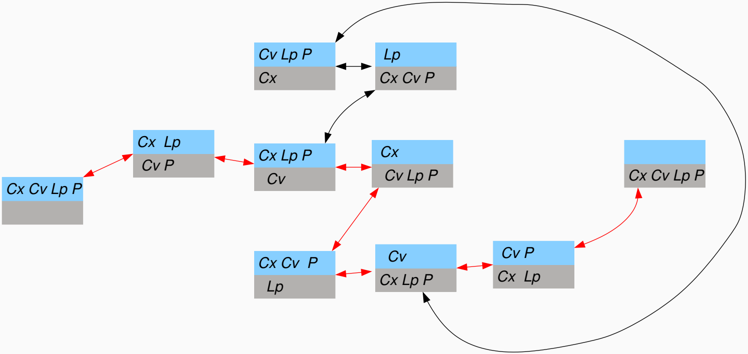


Elimination des états conflictuels.

Exemple : le choux, la chèvre et le loup.

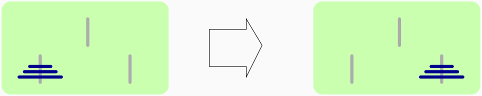


Exemple : le choux, la chèvre et le loup.



Calcul d'un chemin entre l'état initial et l'objectif.

Exemple : problème des tours de Hanoï.

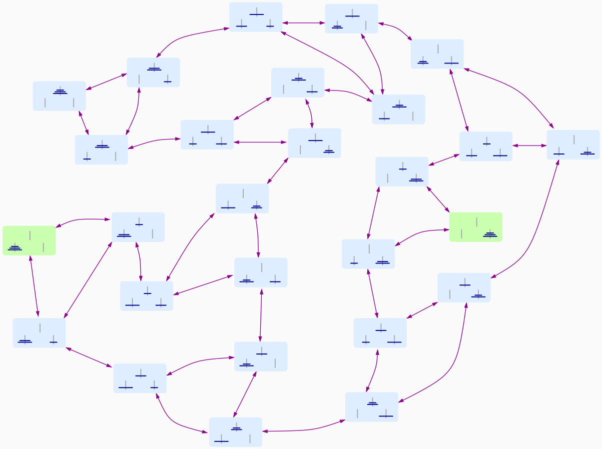


- **Mouvements :** déplacer le plus haut disque d'une pile sur une autre pile.
- **Contrainte :** respecter l'ordre des tailles,
- **Objectif :** effectuer le moins de mouvements possible.

Exemple : problème des tours de Hanoï.

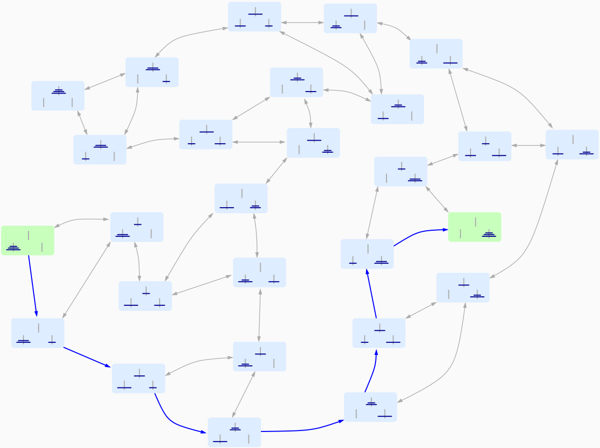
Sommets : les états valides.

Exemple : problème des tours de Hanoï.



Arêtes : les mouvements.

Exemple : problème des tours de Hanoï.



Plus court chemin entre l'état initial et l'objectif.

voisinage

cycles

arbre

chemin

graphe connexe

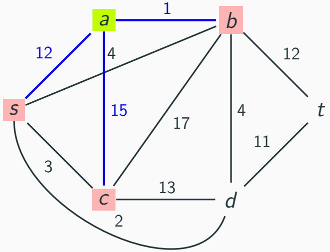
circuit

plus court chemin

composantes connexes

graphe complet

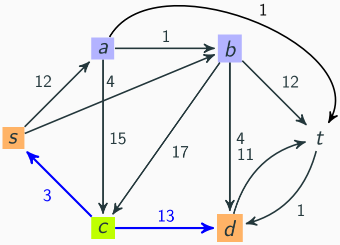
Graphes : voisinage



Graphe non orienté

voisins(u)

$= \{v \in S | \{u, v\} \in E\}$



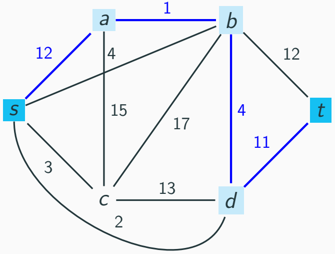
Graphe orienté

voisins_entrants(v)

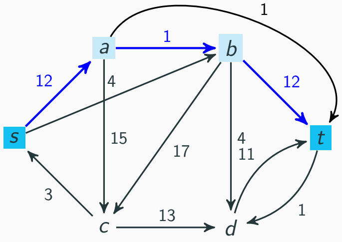
$= \{v \in S | (u, v) \in A\}$

voisins_entrants(u)

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



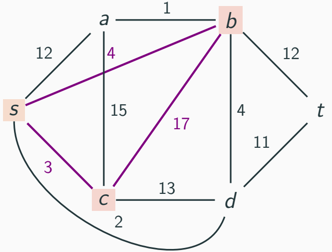
Graphe non orienté



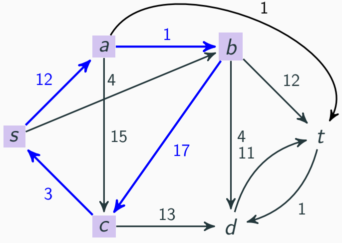
Graphe orienté

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

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



Graphe non orienté

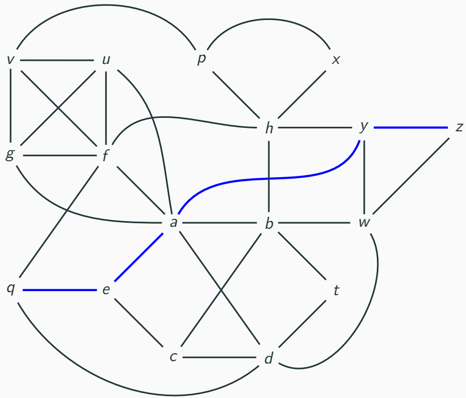


Graphe orienté (circuit)

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

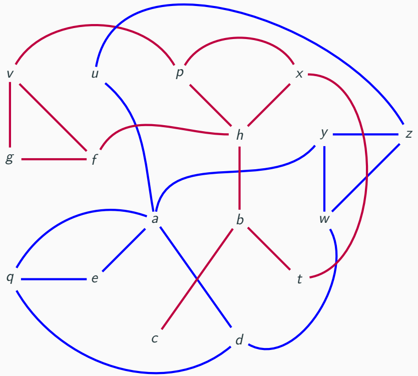
$$\text{avec } (u_1, u_2), (u_2, u_3), \ldots, (u_k, u_1) \in A.$$

Graphe connexe



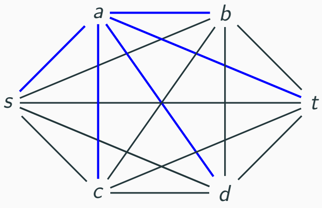
Pour tous sommets u et v il existe un chemin joignant u à v

Graphe non connexe



Composantes connexes : sous-graphes connexes maximaux

Graphe complet

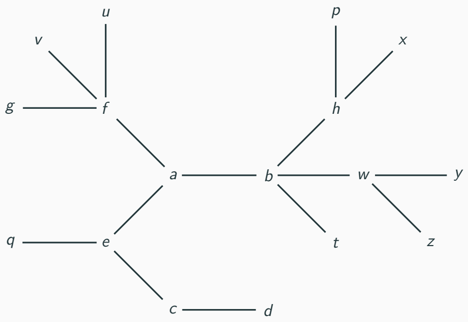


Chaque sommet est connecté avec tous les autres :

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

densité : proportion d'arêtes présentes

Arbre



Arbre : graphe connexe et sans cycle

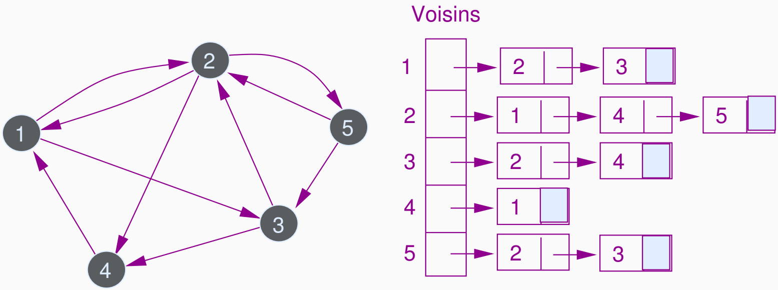
n sommets $\Rightarrow$ $n-1$ arêtes

Représentation.

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

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

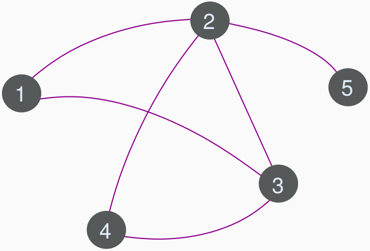
Représentation par des listes d'adjacence.



A chaque sommet est associée la liste de ses voisins.

Représentation : tableau de listes indexé sur les sommets.

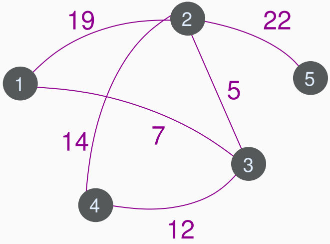
Représentation avec des matrices d’adjacence.



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

Pour chaque 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.

On peut aussi utiliser ∞ pour indiquer qu'il n'y a pas d'arête/arc.

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 :

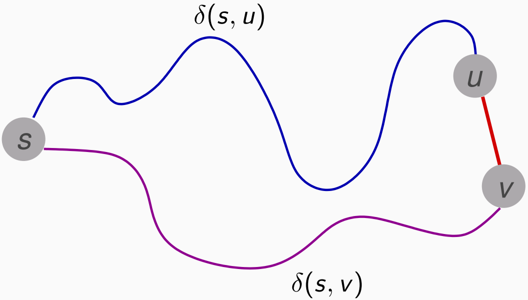
$$\delta(u, v)$$

la longueur du plus court chemin de u à v.

Plus courts chemins

Plus court chemin de s à u

Plus court chemin de s à v



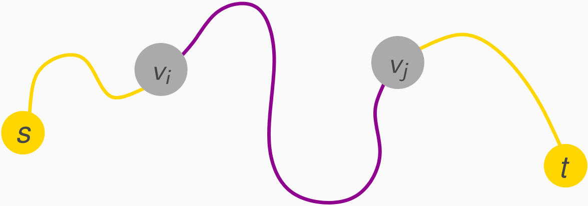
Propriété 1. $\forall s, u, v, \quad \delta(s, v) \leq \delta(s, u) + w(u, v)$

Autrement dit, tout autre chemin que $\delta(s, v)$ est plus long que ce dernier.

Plus courts chemins

Propriété 2.

Soit $(s, v_1, \dots, v_k, t)$ un plus court chemin entre s et t



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

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

$\Rightarrow$ Un sous-chemin d'un plus court chemin est un plus court chemin.

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,
- représentation du graphe par listes d'adjacence.

L'algorithme de Dijkstra découvre à chaque étape de nouveaux chemins issus du sommet source.

Notations.

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

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

$d[u]$ est la longueur du plus court chemin de s à u découvert à ce stade.

Algorithme de Dijkstra.

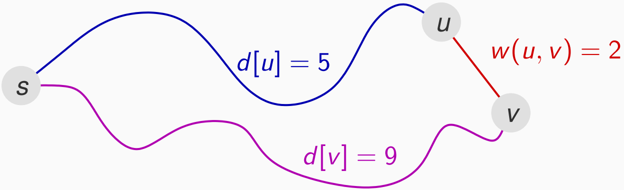
Notations.

- s : sommet source.
- $d[u]$: distance, indexée sur les sommets. A tout moment
$$\forall u \in S,\ d[u] \geq \delta(s,u)$$
  $d[u]$ est la longueur du PCC de s à u découvert à ce stade.
- Initialement
$$\forall u \in S,\ u \neq s,\ d[u] = \infty,$$
$$d[s] = 0.$$
- Finalement
$$\forall u \in S,\ d[u] = \delta(s,u).$$

Algorithme de Dijkstra.

Opération de relâchement : affinage de la borne $d[v]$.

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



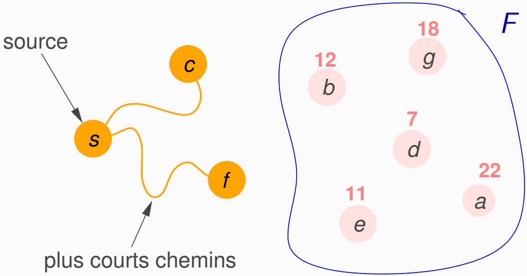
relachement de l'arc $(u, v) \Rightarrow d[v] = 7$

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.

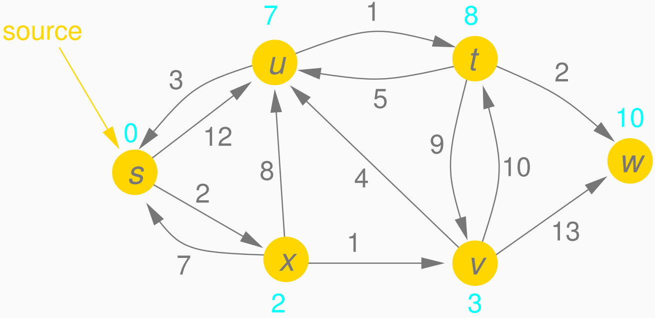
File de priorité F $(\forall u \notin F,\ d[u] = \delta(u))$

Itération : extraction d'un sommet u de F de distance $d[u]$ minimale

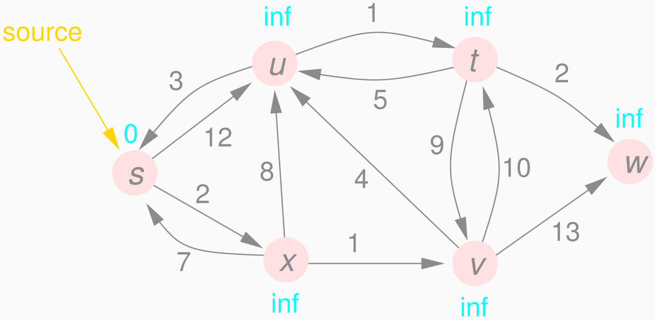


Propriété : au moment de l'extraction le sommet u vérifie $d[u] = \delta(u)$

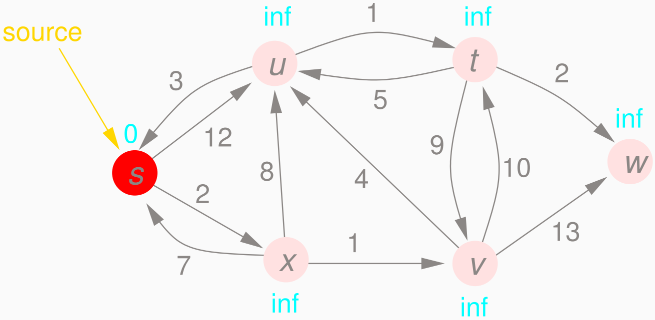
Algorithme de Dijkstra.



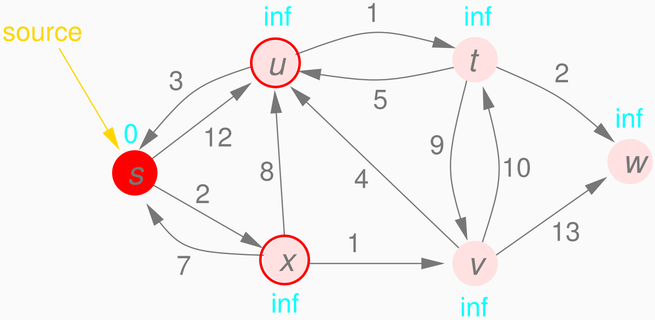
Plus courts chemins : Dijkstra.



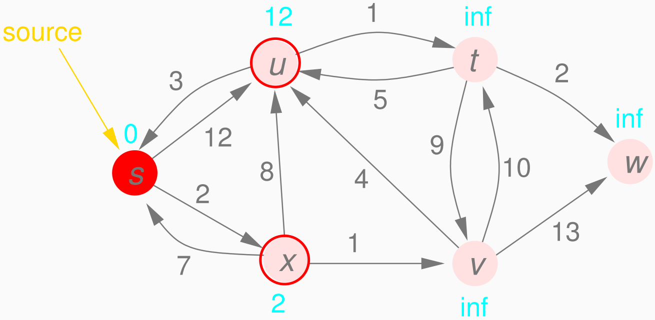
Plus courts chemins : Dijkstra.



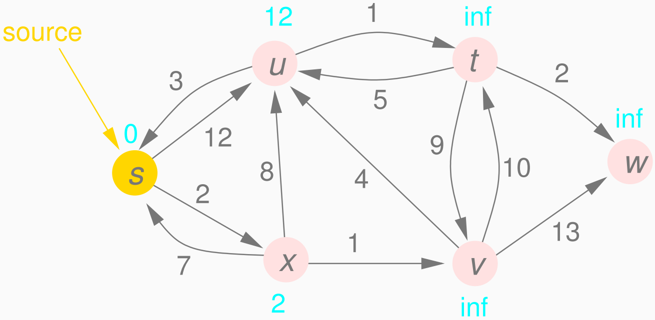
Plus courts chemins : Dijkstra.



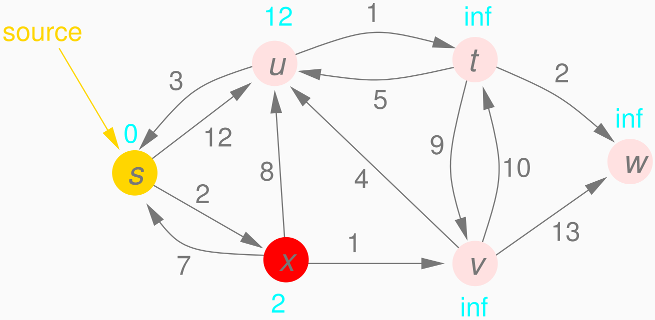
Plus courts chemins : Dijkstra.



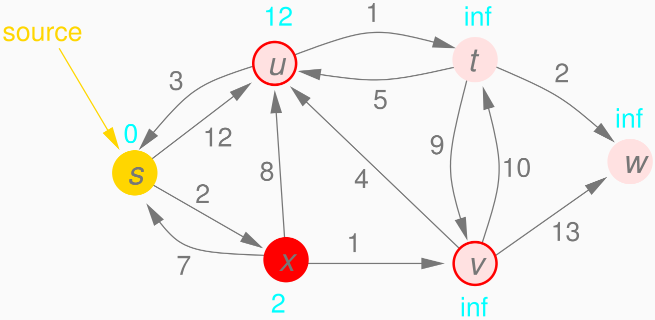
Plus courts chemins : Dijkstra.



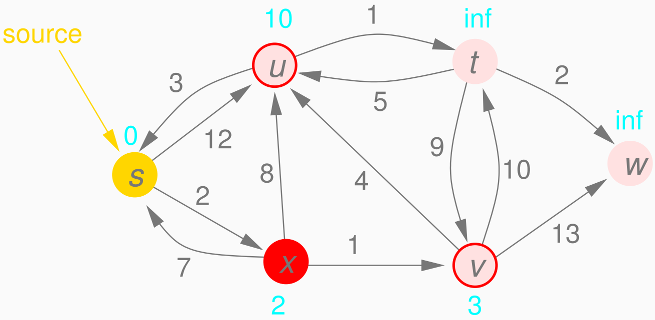
Plus courts chemins : Dijkstra.

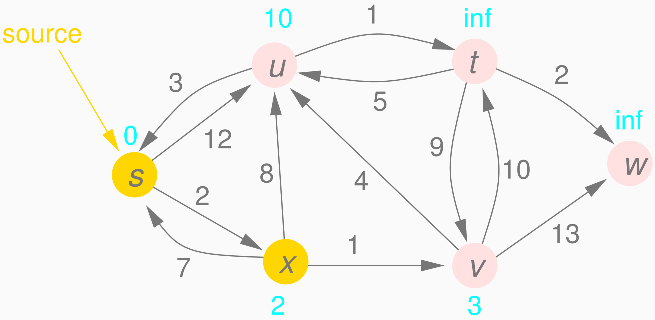


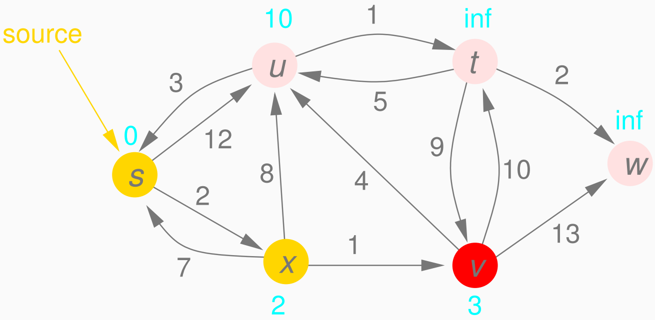
Plus courts chemins : Dijkstra.



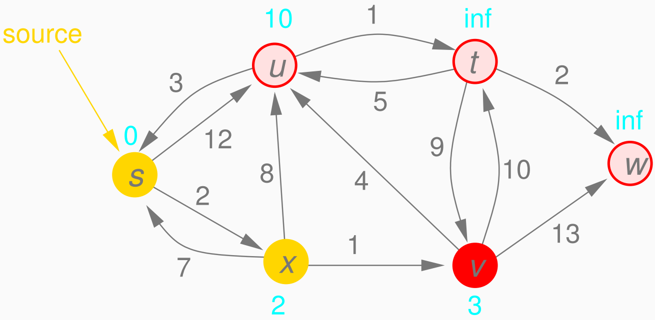
Plus courts chemins : Dijkstra.



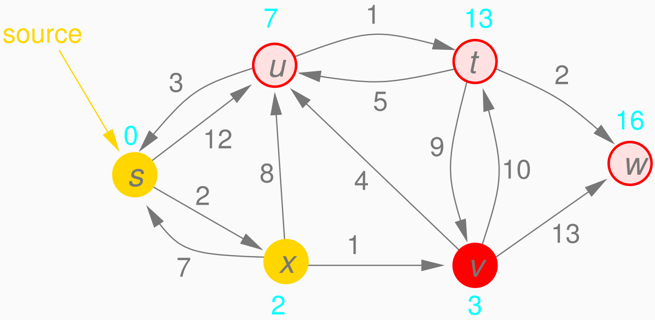

source
0
s
10
u
inf
t
inf
w
x
2
v
3
3
12
2
7
8
1
1
5
4
9
10
2
13


source
0
s
10
u
1
inf
t
2
inf
w
3
12
5
8
4
9
10
2
x
1
v
13
7
2
3

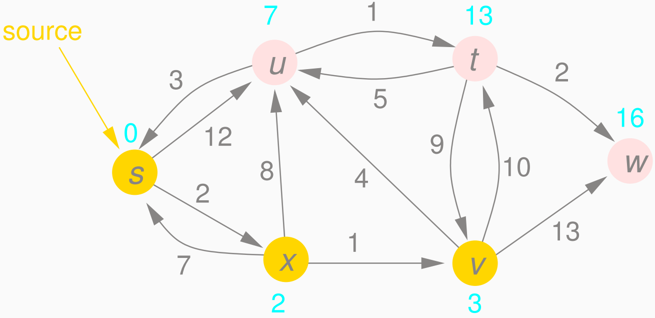
Plus courts chemins : Dijkstra.



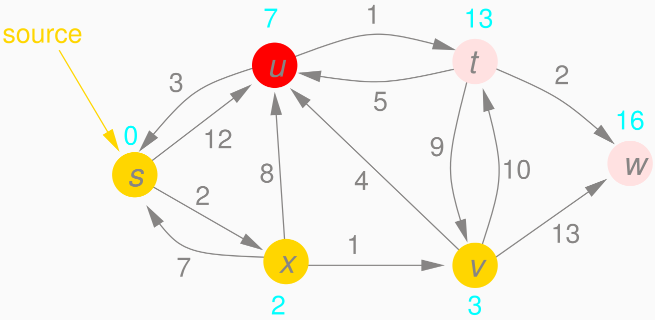
Plus courts chemins : Dijkstra.



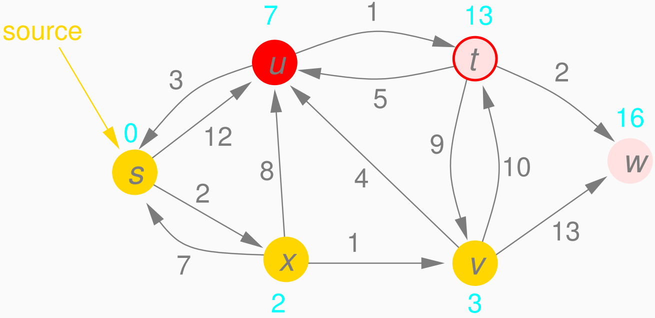
Plus courts chemins : Dijkstra.



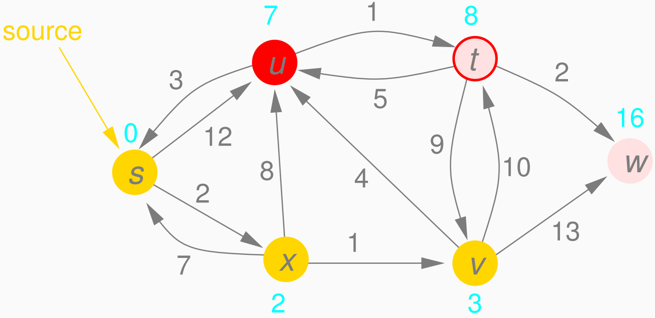
Plus courts chemins : Dijkstra.



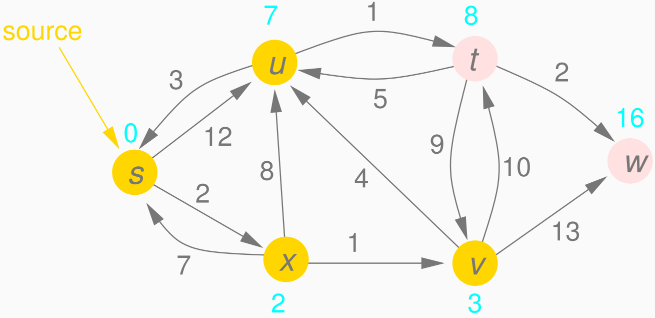
Plus courts chemins : Dijkstra.



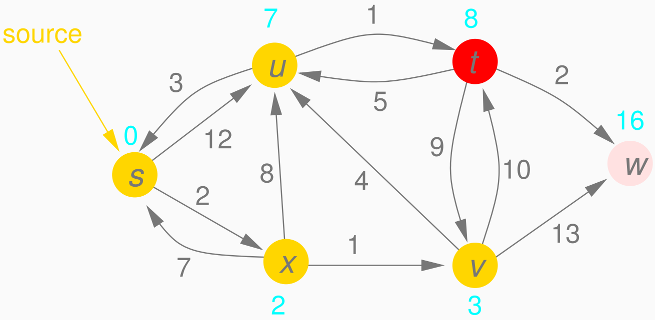
Plus courts chemins : Dijkstra.



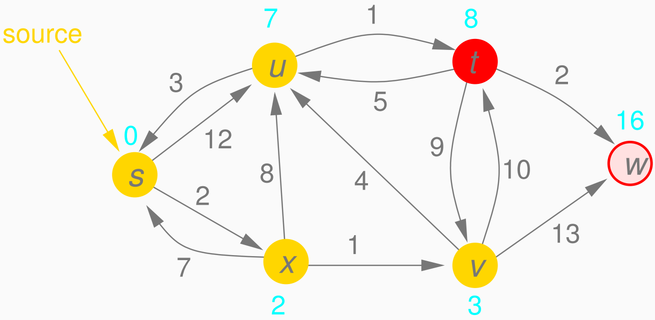
Plus courts chemins : Dijkstra.



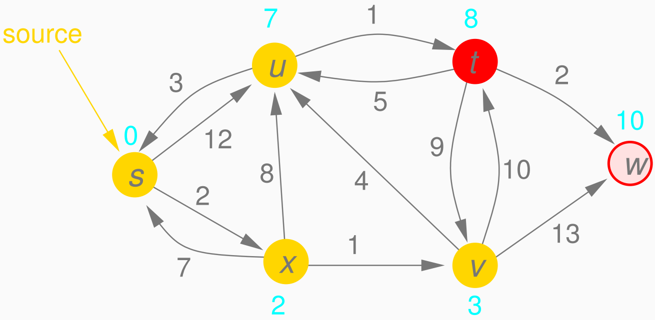
Plus courts chemins : Dijkstra.



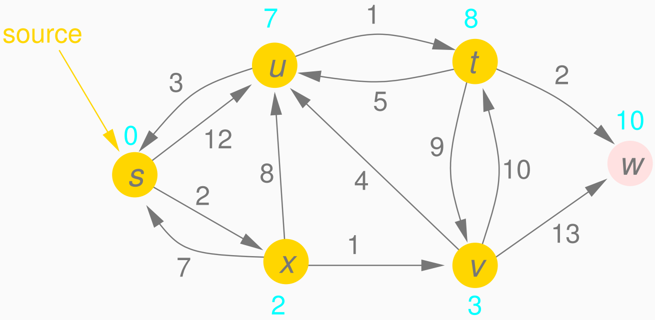
Plus courts chemins : Dijkstra.



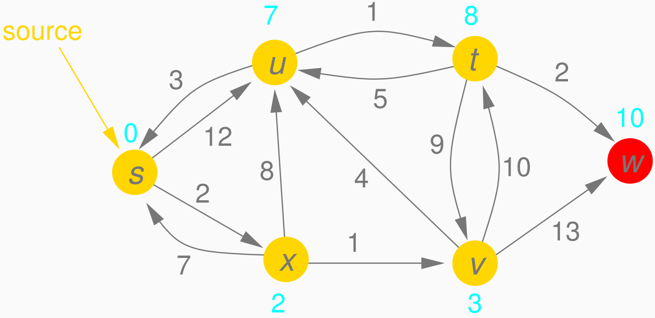
Plus courts chemins : Dijkstra.



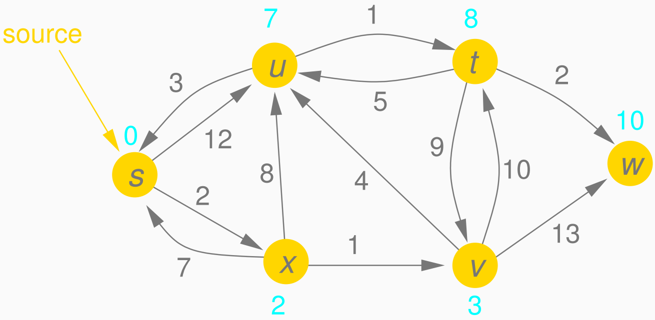
Plus courts chemins : Dijkstra.



Plus courts chemins : Dijkstra.



Plus courts chemins : Dijkstra.



Algorithme de Dijkstra.

algorithme $\textsc{Dijkstra}(G, s)$

Entrées : $G = (S, A, w)$ un graphe pondéré, s un sommet de G

Sortie : Un dictionnaire qui à chaque sommet associe sa distance à s.

$F := S$, $\{$*i.e. initialement F contient tous les sommets*$\}$

pour chaque sommet $u \in S$ **faire**

    $d[u] := \infty$,

$d[s] := 0$,

tant que $F \neq \emptyset$ **faire**

    $u := \textsc{Extraire\_le\_min}(F)$,

    pour chaque sommet $v \in Voisins[u]$ **faire**

        si $d[v] > d[u] + w(u, v)$ **alors**

            $d[v] = d[u] + w(u, v)$,

    fin pour,

fin tant que,

renvoyer d,