Programme de baccalauréat en informatique

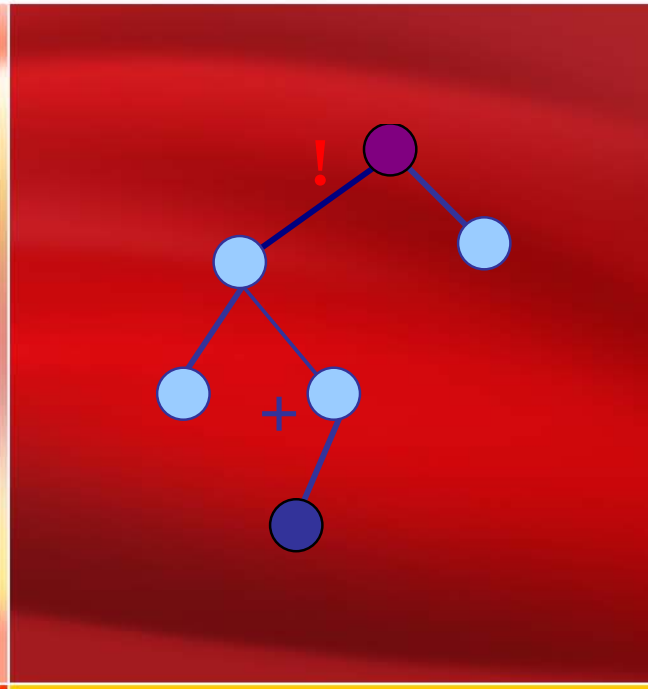
Structures de données

IFT-10541

Thierry EUDE

Module 5 : Les graphes (3)

- Connexité des graphes
- Algorithme de Dijkstra
- Algorithme de Bellman-Ford
- Les réseaux

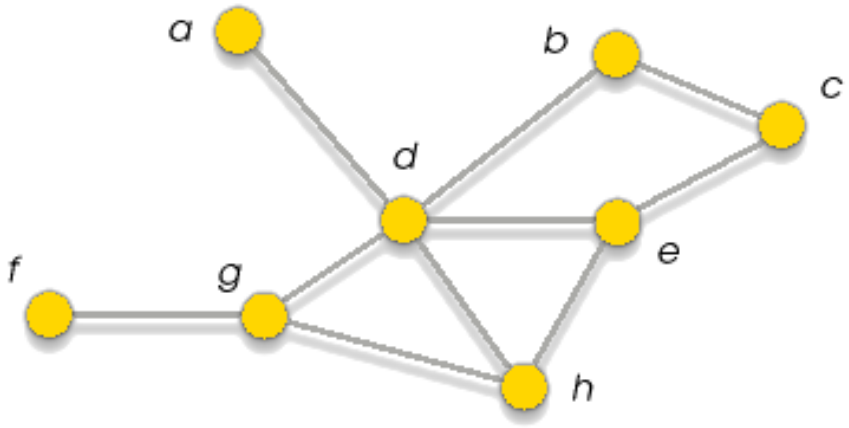


Édition été 2007

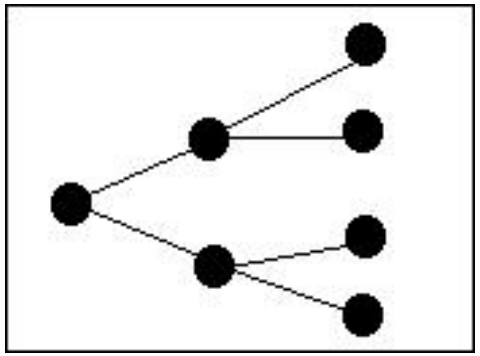
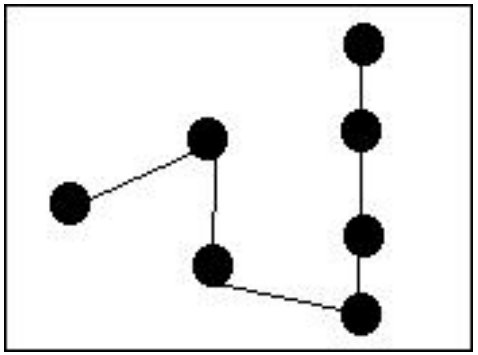
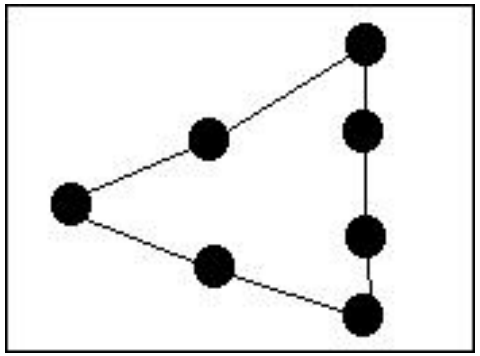
UNIVERSITÉ LAVAL
Département d'informatique
et de génie logiciel

Connexité des graphes non orientés

- Un graphe non orienté ***G*** est *connexe* ssi il existe un chemin entre n'importe quelle paire de sommets distincts du graphe



Graphes connexes (cas particuliers)

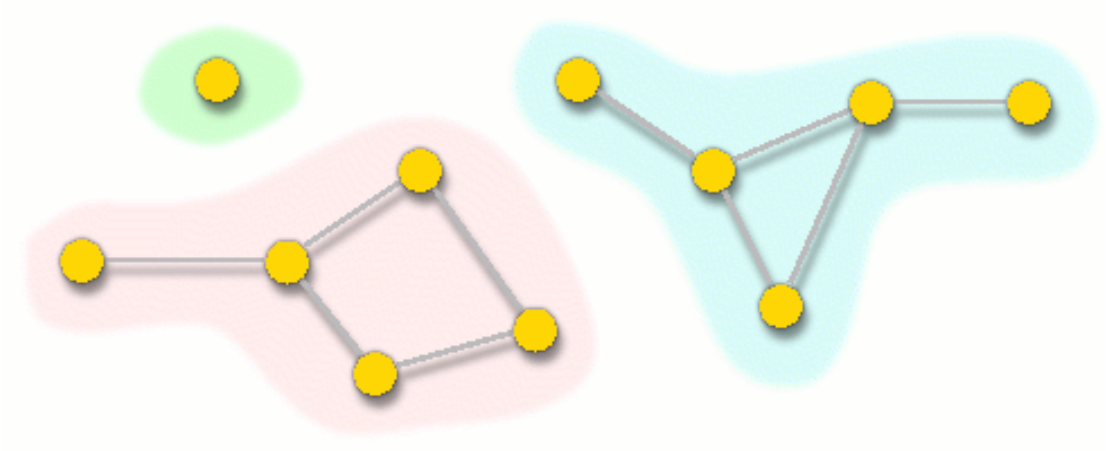
Arbre	Chemin	Circuit
		

Connexité des graphes non orientés

- Que se passe-t-il si le graphe ***G*** n'est pas connexe?
- ***G*** = ensemble de sous-graphes connexes
- On appelle chacun de ces sous-graphes une composante connexe de ***G***

Composante connexe

- ***G'=(S',A')*** une composante connexe de ***G***
- ***G'*** est un sous-graphe connexe maximal :
  $\exists$ ***S'' = S'*** $\cup$ ***{v}*** tel que ***G'' = (S'', A')*** est une composante connexe



Propriétés

- Un graphe non connexe se compose de plusieurs composantes connexes non reliées entre elles et dont le nombre est constant.
- Un graphe est connexe ssi il ne possède qu'une seule composante connexe.
- Un sommet isolé (de degré 0) constitue toujours une composante connexe à lui seul.
- Un graphe connexe d'ordre n (le nombre de sommets) comporte au moins n-1 arêtes.

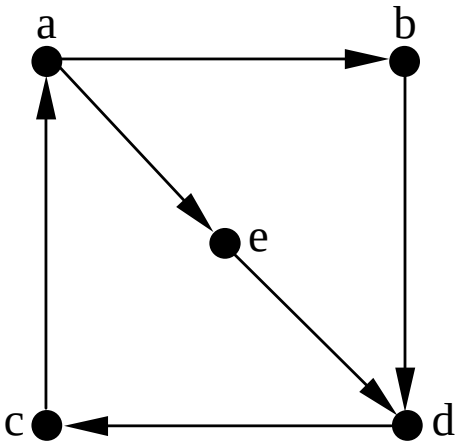
Connexité des graphes orientés

- Dans un graphe non orienté : un chemin est une suite d'arêtes
  - S'il existe un chemin entre deux sommets *a* et *b* alors celui-ci est aussi un chemin entre *b* et *a*
- Dans un graphe orienté : un chemin est une suite d'arcs
  - S'il existe un chemin entre deux sommets a et b alors on ne peut rien conclure concernant le lien de b avec a

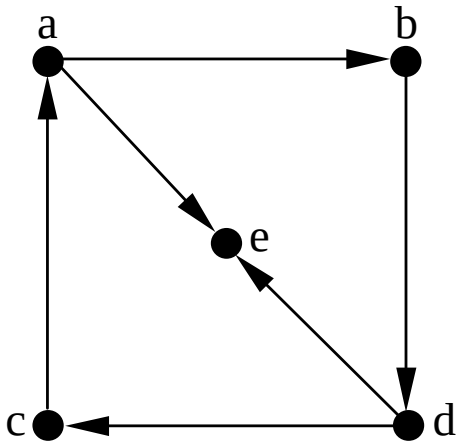
Connexité des graphes orientés

- Un graphe *G* est *fortement connexe* ssi pour n'importe quelle paire de sommets distincts *(a,b)* de *G,* il existe un chemin du sommet *a* au sommet *b* et un autre chemin du sommet *b* au sommet *a*.

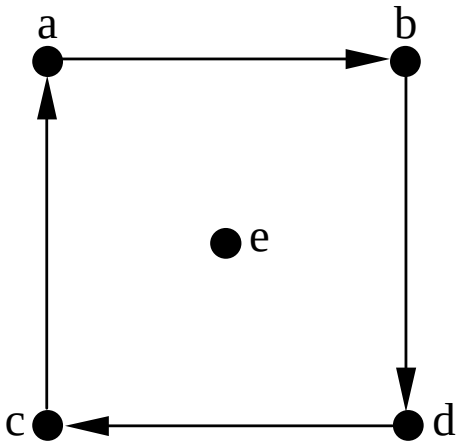
Connexité des graphes orientés (exemples)



graphe *G*



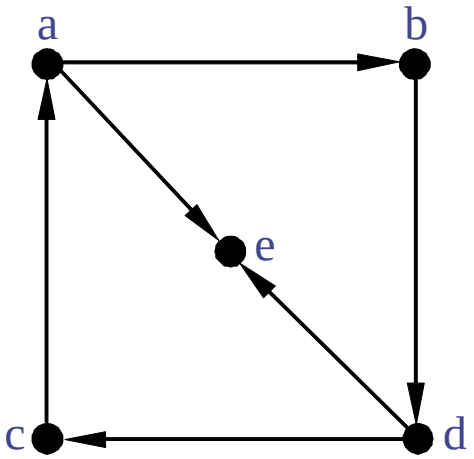
graphe *H*



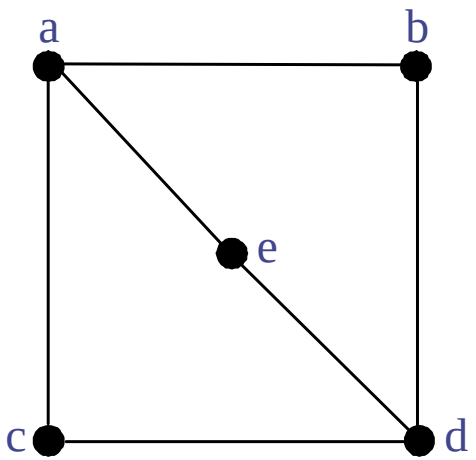
graphe *I*

Connexité des graphes orientés

- Un graphe orienté *G* est *faiblement connexe* ssi son graphe non orienté sous-jacent *G'* est connexe.
- le graphe *G'*, = le graphe *G* dans lequel les arcs ont été remplacés par des arêtes, est connexe.

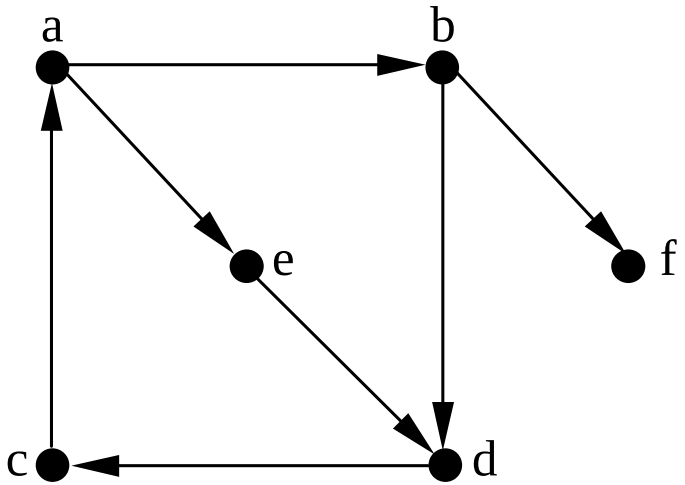


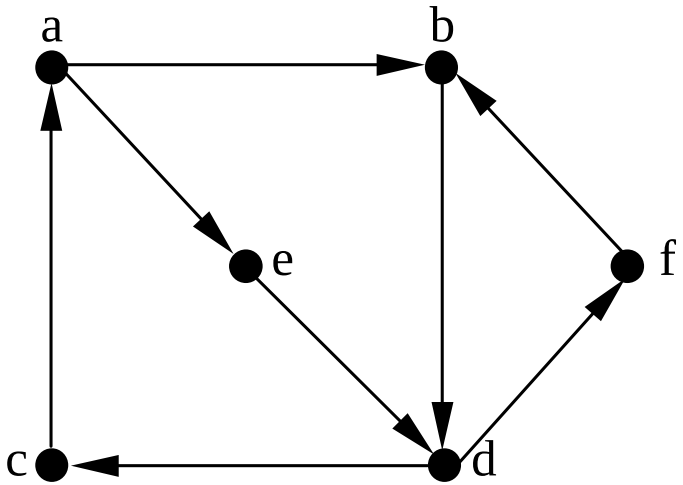
graphe *G*



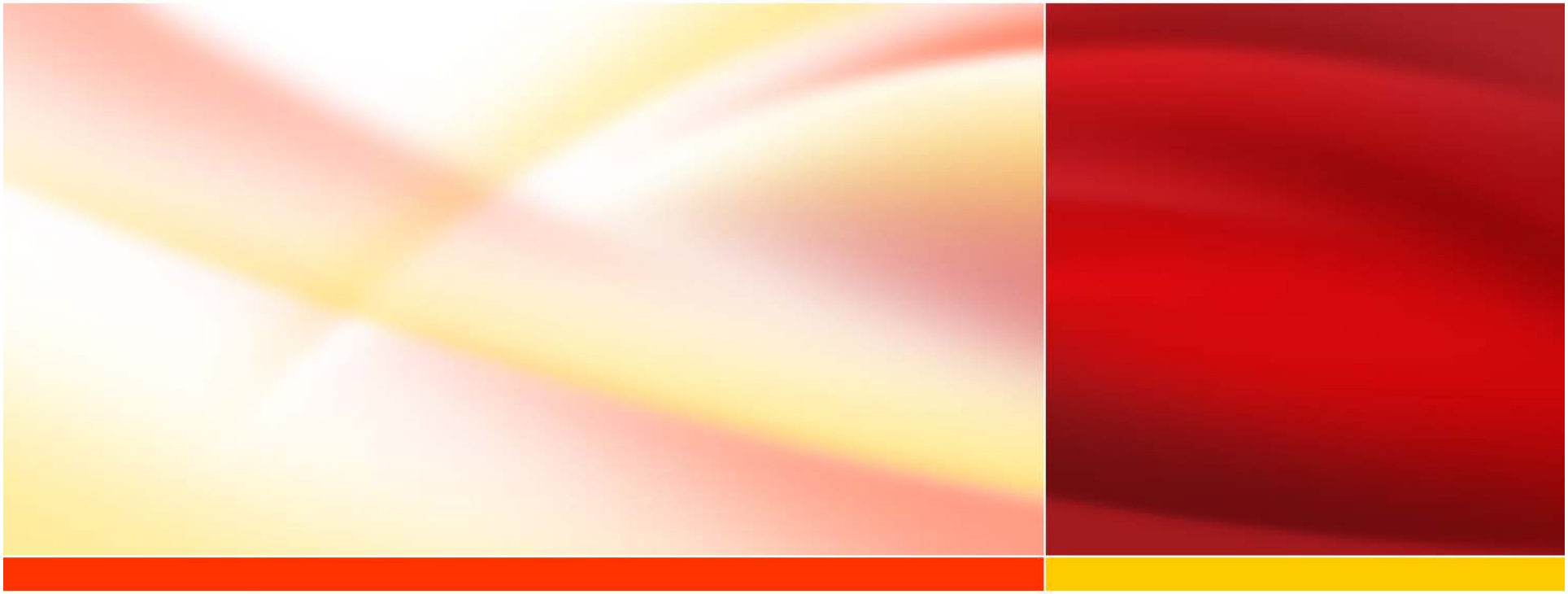
graphe *G'*

Graphes faiblement connexes (exemples)





Les plus courts chemins à origine unique



UNIVERSITÉ LAVAL

Introduction

- Soient deux sommets ***s*** et ***t*** d'un graphe ***G***
- Si ***s*** et ***t*** représentent des villes, quel est le plus court chemin entre ces deux villes sachant que :
  - les arêtes du graphe ***G*** sont pondérées
  - la pondération ***c(i,j)*** représente la distance entre les deux sommets (deux villes) ***i*** et ***j*** du graphe
  - la pondération est positive

Cas général

- Comment trouver le plus court chemin entre 2 sommets ***s*** et ***t*** d'un graphe ***G***?
- Il existe plusieurs algorithmes :
  1. Parcours exhaustif des nœuds
  2. Algorithme de Dijkstra
  3. Algorithme de Bellman-Ford (cas des pondérations négatives)

1. Parcours exhaustif des nœuds

- Déterminer tous les chemins possibles
- Choisir le plus court chemin entre ***s*** et ***t*** *(2 sommets)*
- Sur un graphe de 20 sommets avec :
  - une machine de 10^9 opérations / seconde
  - recherche d'un chemin = 1 opération

  77 ans pour avoir le résultat
- Un graphe de 26 sommets : **avant le Big Bang**
- Le nombre de chemin est de l'ordre de ***n!***

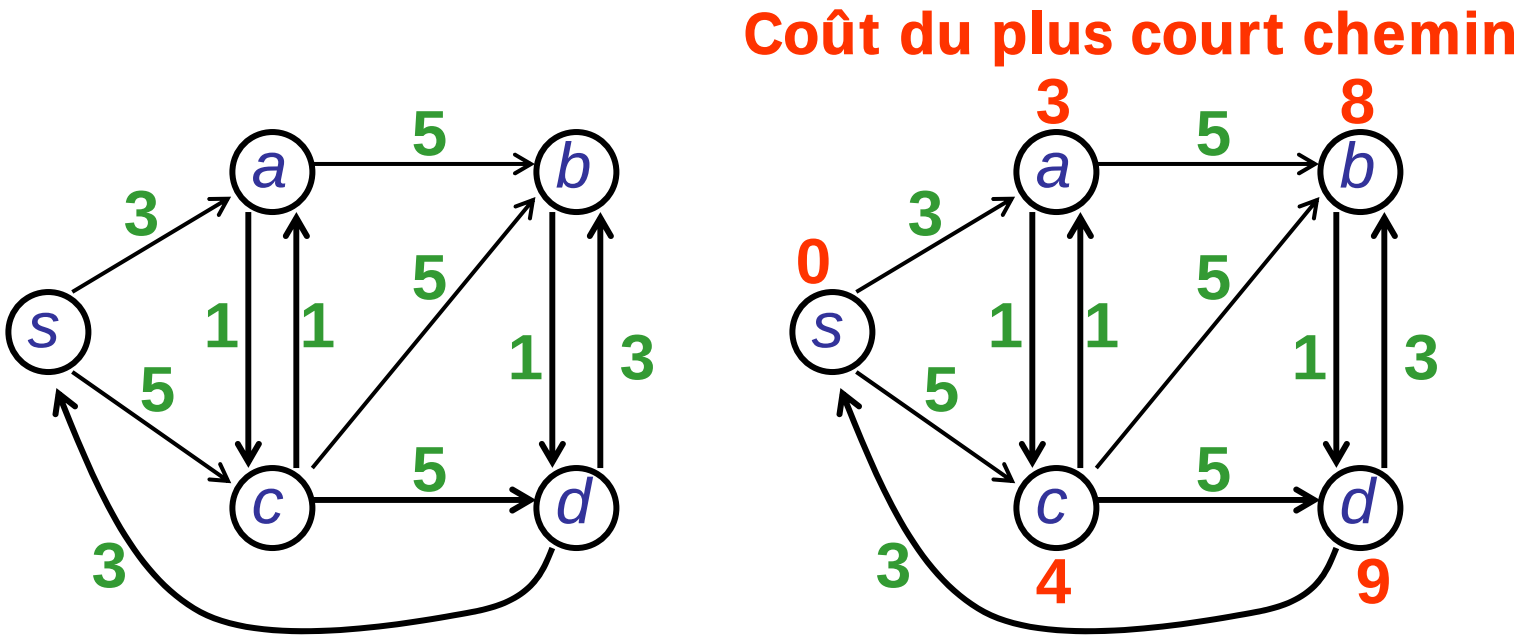
2. Algorithme de Dijkstra

- Se base sur le principe de la **sous-optimalité**
- Si ***p*** est le plus court chemin entre ***i*** et ***j*** alors:
  - $\forall$ ***k***, un sommet appartenant à ***p***
  - le sous-chemin de ***p (i, k)*** est optimal (le plus court)
  - le sous-chemin de ***p (k, j)*** est optimal (le plus court)

Algorithme de Dijkstra

Exemple du plus court chemin entre ***s*** et $\forall$ ***i*** $\in$ ***v***

(***v :*** l'ensemble des sommets)

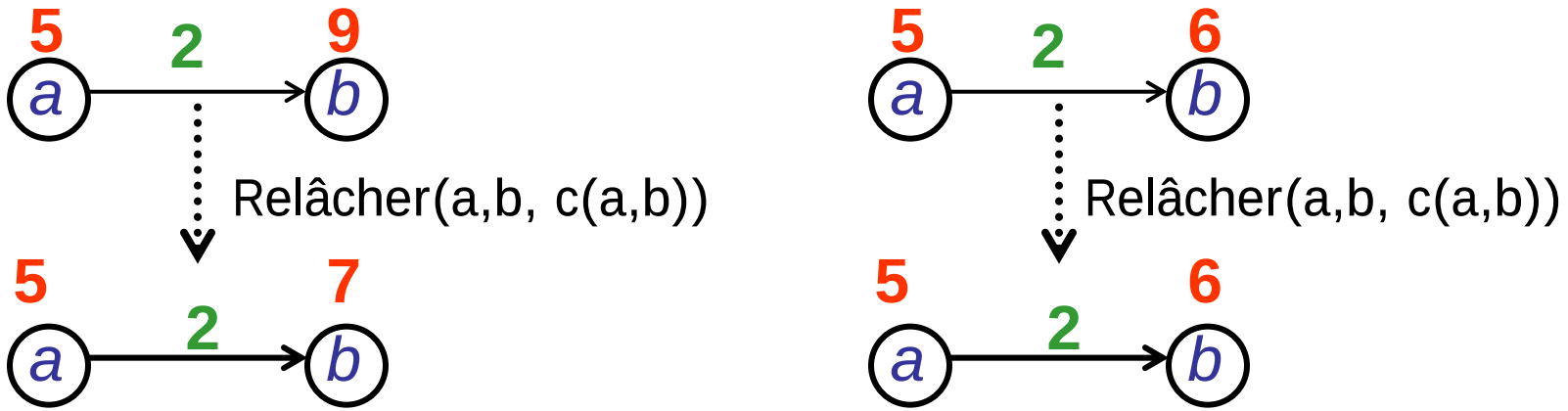


Algorithme de Dijkstra

- Basée sur la technique de ***relâchement***
- Méthode qui diminue progressivement le poids du plus cours chemin pour chaque sommet jusqu'à ce qu'il soit égal au poids du plus court chemin.
- Pour chaque sommet $i \in V$, on maintient à jour un attribut y_i: une estimation de la pondération d'un plus court chemin:
  - **État initial** :
    - $y_s = 0$ (s étant la source)
    - $y_i = +\infty$ avec $i \neq s$
  - **À chaque étape** : essayer de minimiser les y_i
  - **État final** : $y_i =$ ***lpcc(s,i)*** *(lpcc: le plus court chemin)*

Algorithme de Dijkstra

- ***RELÂCHER (a,b, c (a,b))***
  - Si $y_b > y_a + c(a,b)$ *Alors* $y_b \leftarrow y_a + c(a,b)$



- Dans l'algorithme de Dijkstra, chaque arc est relâché exactement une fois.

Algorithme de Dijkstra

Soit ***V*** est l'ensemble des sommets d'un graphes

- L'algorithme maintient à jour un ensemble ***S*** contenant les sommets dont les poids finaux de plus court chemin à partir de la source ont été calculés.
- À chaque itération, l'algorithme choisit le sommet $j \in V - S$ dont l'estimation de plus court chemin est minimale, l'insère dans ***S*** et relâche tous les arcs partant de j.
- L'algorithme gère une file T pour y emmagasiner les sommets de $V - S$ suivant leur attribut y_i .

L'Algorithme de Dijkstra...

Algorithme de Dijkstra

- Soit ***V*** l'ensemble des sommets d'un graphes
- Initialiser y_i **=** $+\infty$ pour tous les sommets i
- Initialiser y_s **=** ***0***.
- Initialiser ***S*** à l'ensemble vide, ***T*** **=** ***V***.
- Tant que ***T*** n'est pas vide

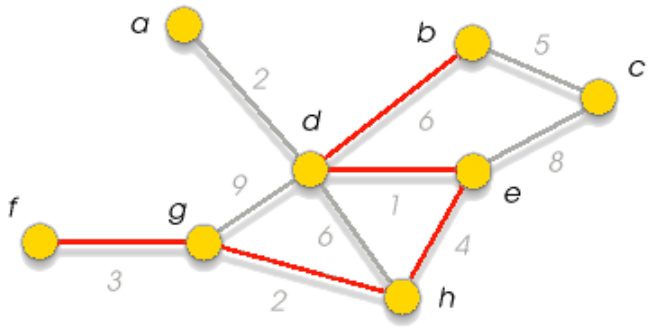
  1. Sélectionner le sommet ***j*** de ***T*** de plus petite valeur y_i

  2. Faire ***T*** **=** ***T*** **** ***j*** et ***S*** **=** ***S*** $\cup$ ***j***

  3. Pour tous les sommets ***k*** de ***T*** adjacents à ***j***, faire **RELÂCHER**(j, k, c(j,k))
- Fin Tant que.

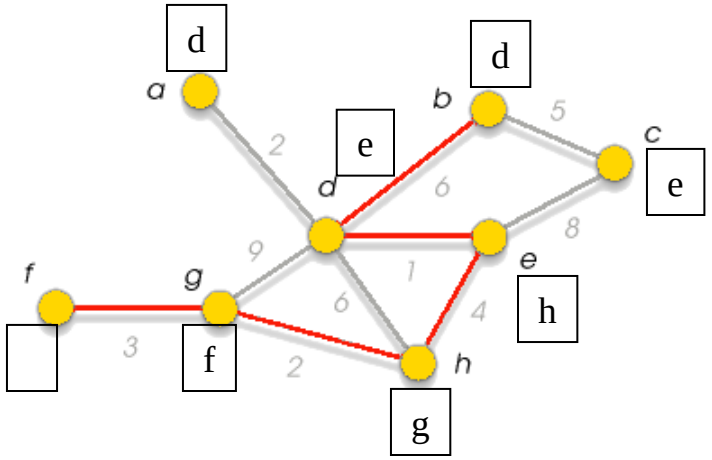
Algorithme de Dijkstra

- Pour aller d'un sommet A à un sommet B: (option A)
  - appliquer l'***algorithme de Dijkstra*** avec le sommet A comme source,
  - faire un ***parcours en profondeur*** à partir du sommet A, en choisissant toujours le prochain sommet (non visité) en ordre croissant des étiquettes calculées, et en vérifiant que la somme des poids des arêtes ne dépasse pas le minimum
  - arrêter une fois rendu au sommet B.



Algorithme de Dijkstra

- Pour aller d'un sommet A à un sommet B: (option B)
  - appliquer l'***algorithme de Dijkstra*** avec le sommet A comme source, en **conservant** pour chaque nœud le nœud origine à partir duquel sa plus petite distance de la source a été calculée.

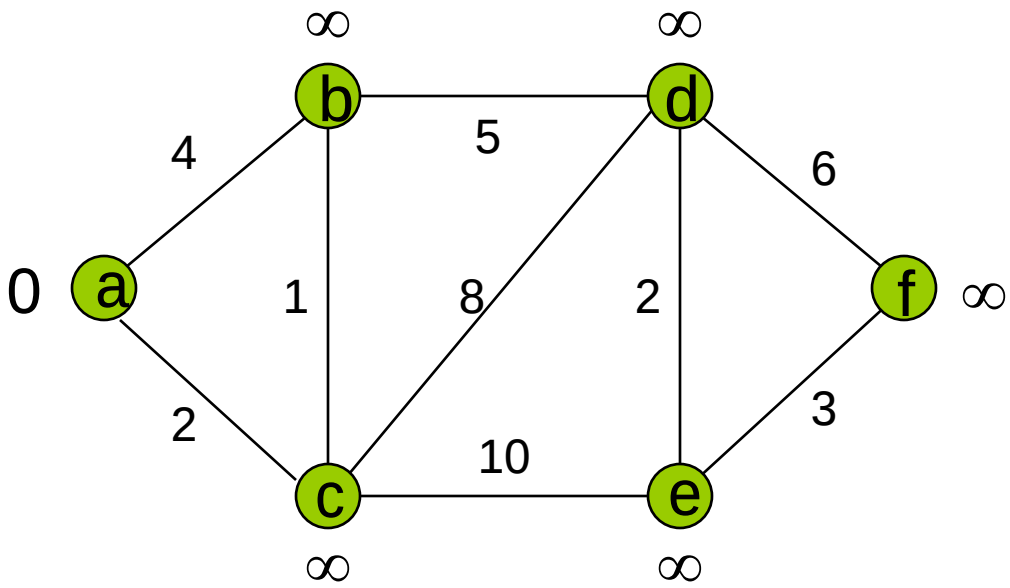


Algorithme de Dijkstra

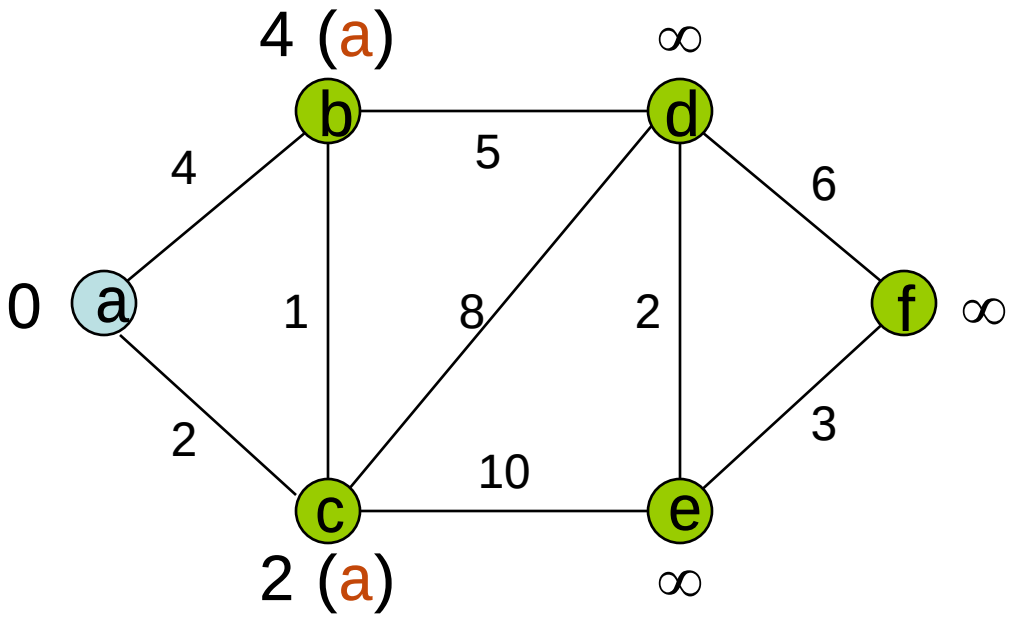
- ***RELÂCHER (a,b, c (a,b))***
  - Si $y_b > y_a + c(a,b)$
    Alors
    - $y_b \leftarrow y_a + c(a,b)$
    - $P[b] \leftarrow a$

Tous les P[i] sont initialisés à NIL au départ ($i \in v$)

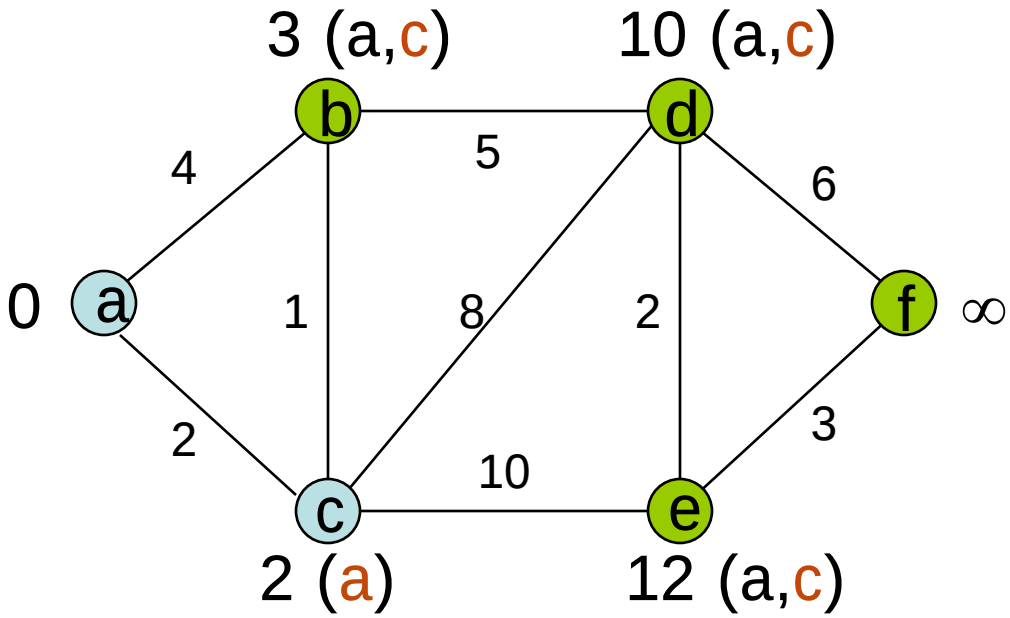
Algorithme de Dijkstra



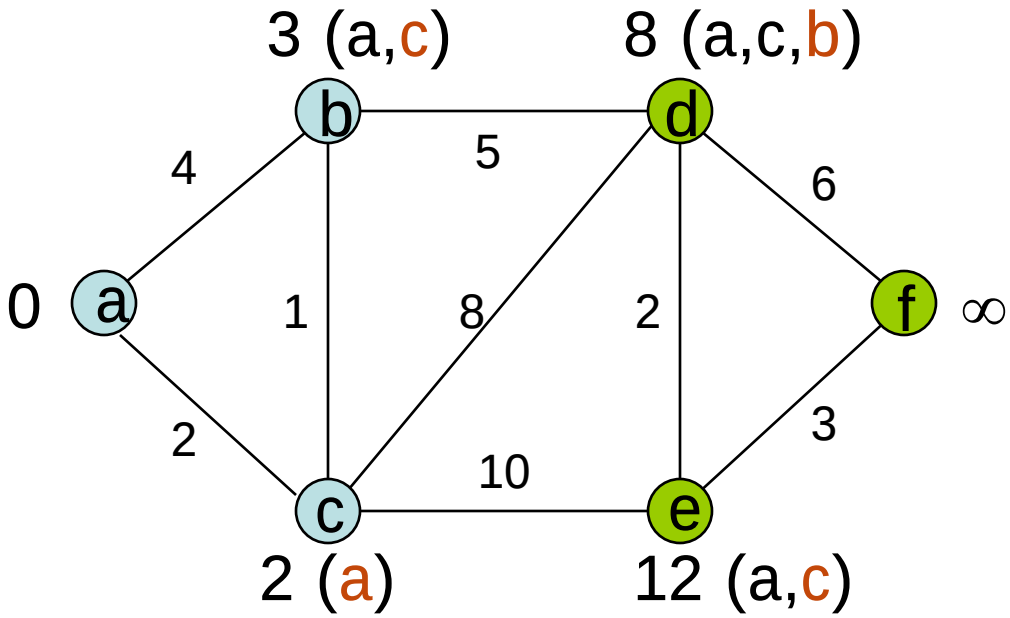
Algorithme de Dijkstra



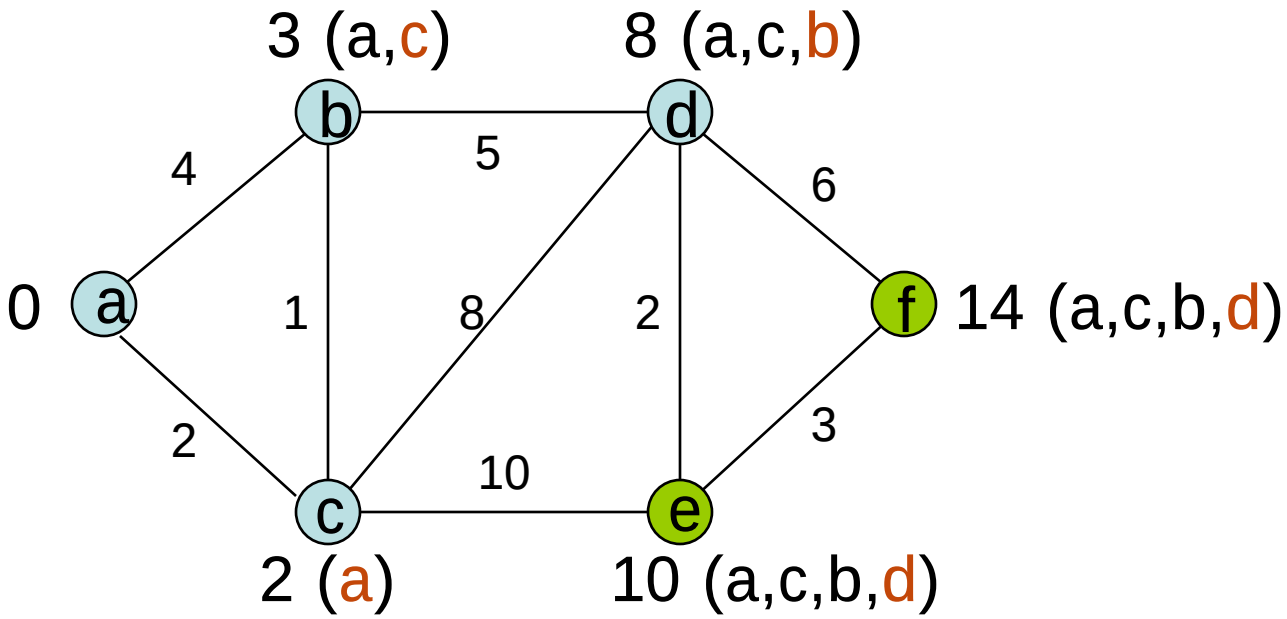
Algorithme de Dijkstra



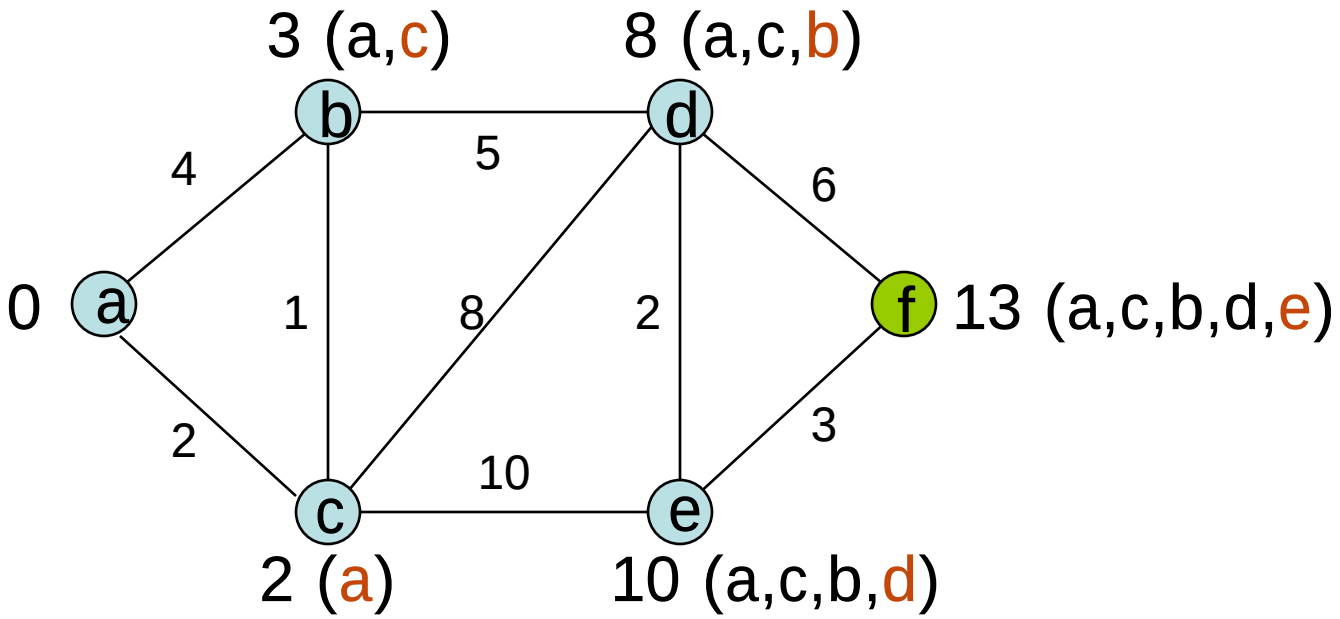
Algorithme de Dijkstra



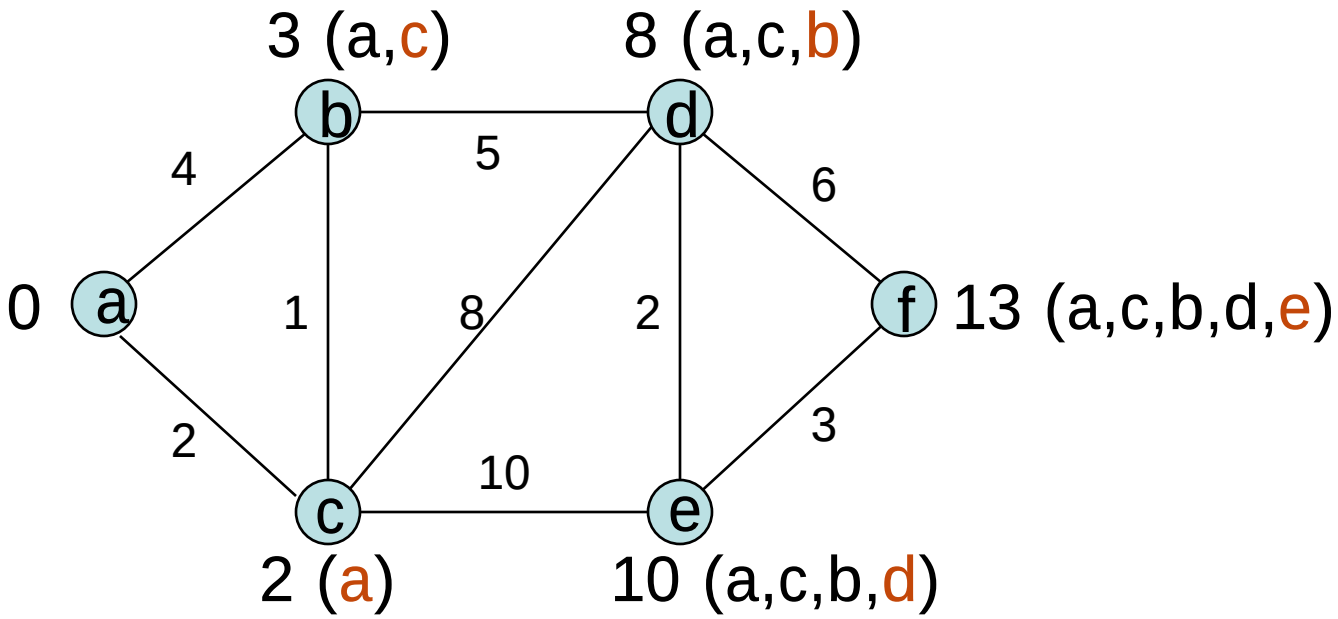
Algorithme de Dijkstra



Algorithme de Dijkstra



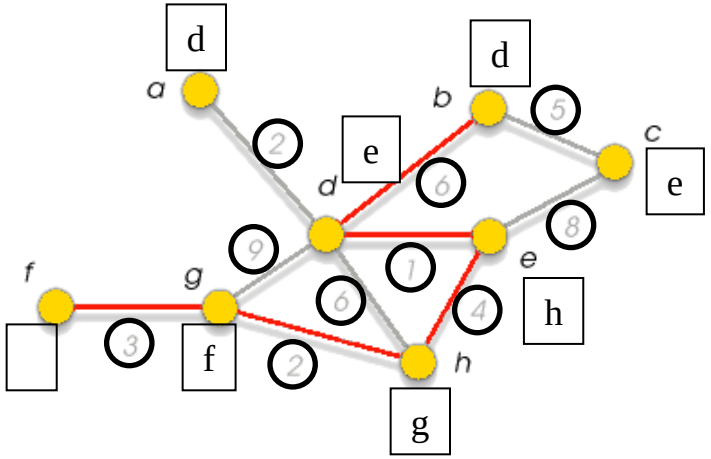
Algorithme de Dijkstra



Algorithme de Dijkstra

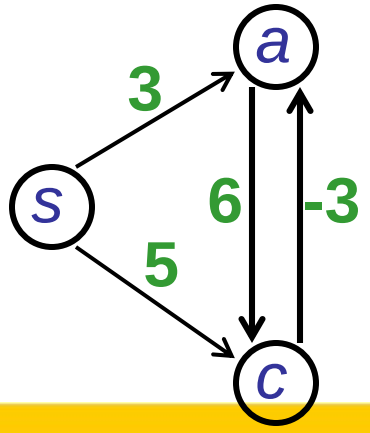
- Problème :
  - Limité à des poids positifs ➔ pas de cycle négatif

Solution : Bellman-Ford



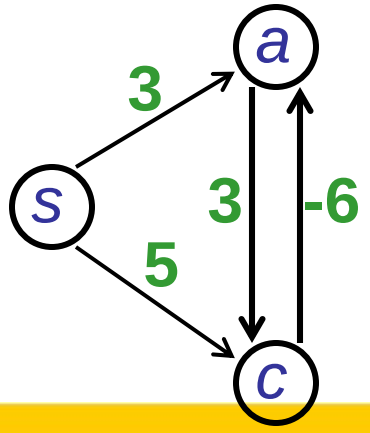
3. Algorithme de Bellman-Ford

- **Problématique des arcs de poids négatifs**
  - Si un graphe ne contient **aucun** **circuit de poids négatif** accessible à partir d'une origine s, alors, pour tout sommet i $\in$ ***v***, le poids du plus court chemin **reste bien défini**, même si sa valeur est négative.



Algorithme de Bellman-Ford

- **Problématique des arcs de poids négatifs**
  - **S'il existe un circuit de poids négatif** accessible depuis s, le poids du plus court chemin **n'est pas correctement défini**. Aucun chemin entre s et un sommet du circuit ne peut être plus court, on peut toujours trouver un encore plus court!



Algorithme de Bellman-Ford

- **Stratégie**
  - Faire les étapes nécessaires pour faire converger le poids des chemins sachant qu'un plus court chemin de s à tout autre sommet est un chemin d'ordre au plus n -1 arcs.
  - Vérifier s'ils ont tous convergé.
  - Retourner VRAI si c'est le cas.
  - Retourner FAUX sinon.
- **Utilisation de la technique du relâchement**
  - Comme dans Dijkstra, l'Algorithme utilise le relâchement pour diminuer progressivement une estimation y_v du poids d'un plus court chemin depuis l'origine.
  - À la différence de Dijkstra, chaque arc est relâché plusieurs fois.

L'algorithme de Bellman-Ford…

Algorithme de Bellman-Ford

- Soit le graphe $G(V,E)$
- Initialiser $y_i = +\infty$ pour tous les sommets i
- Initialiser $y_s = 0$.
- Répéter $|V| - 1$ FOIS

    Pour tout arc (u,v) de E

        faire **RELÂCHER**(u, v, $c(u,v)$)

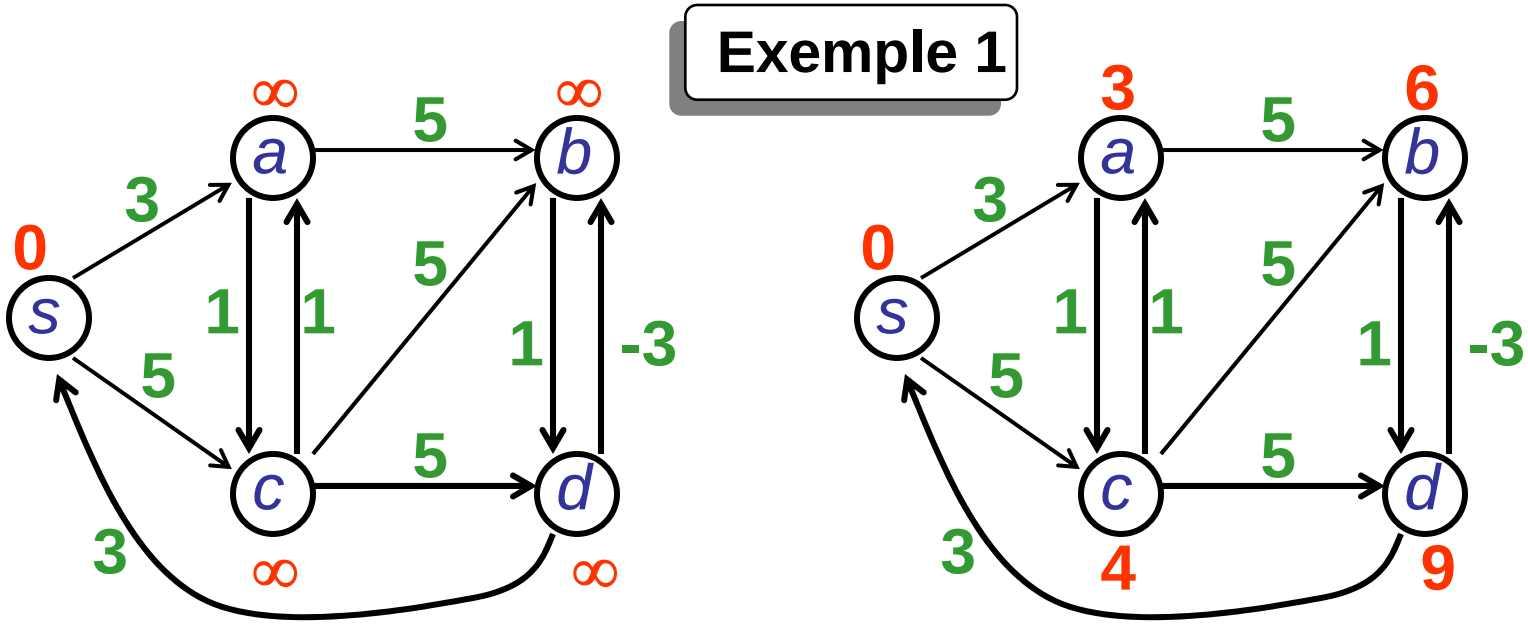
- Pour tout arc (u,v) de E faire

    Si $y_v > y_u + c(u,v)$ Alors

        Retourner FAUX

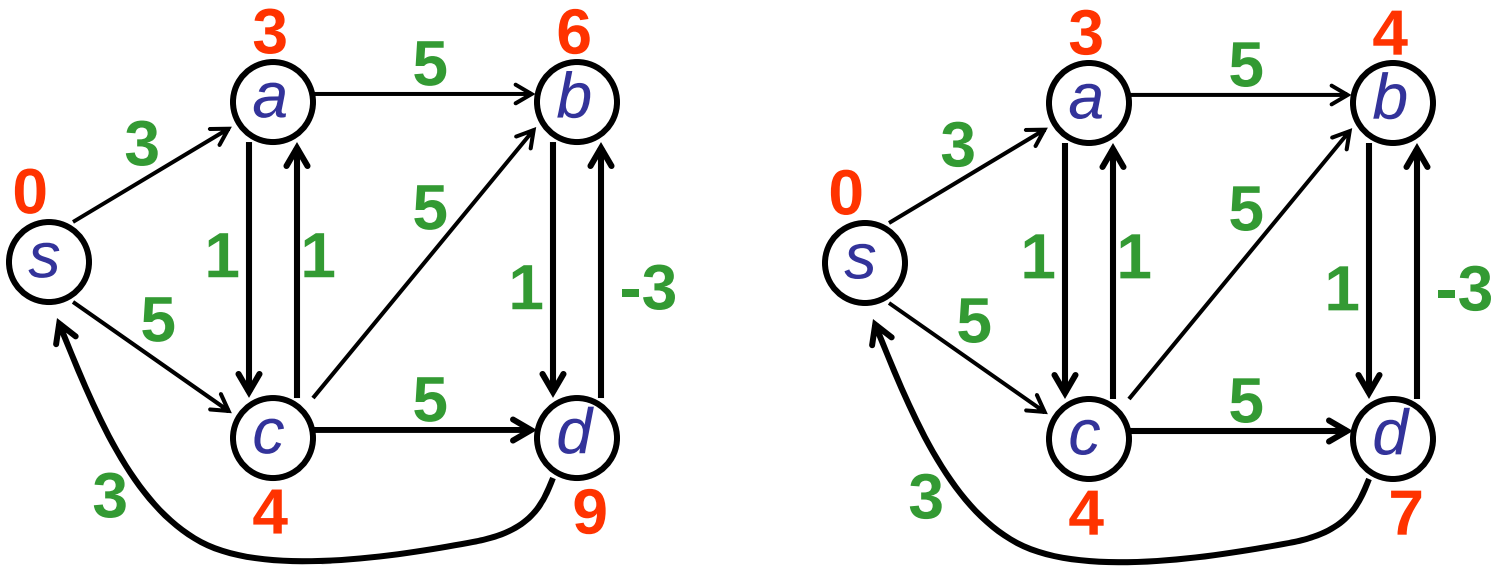
- Retourner VRAI

Algorithme de Bellman-Ford



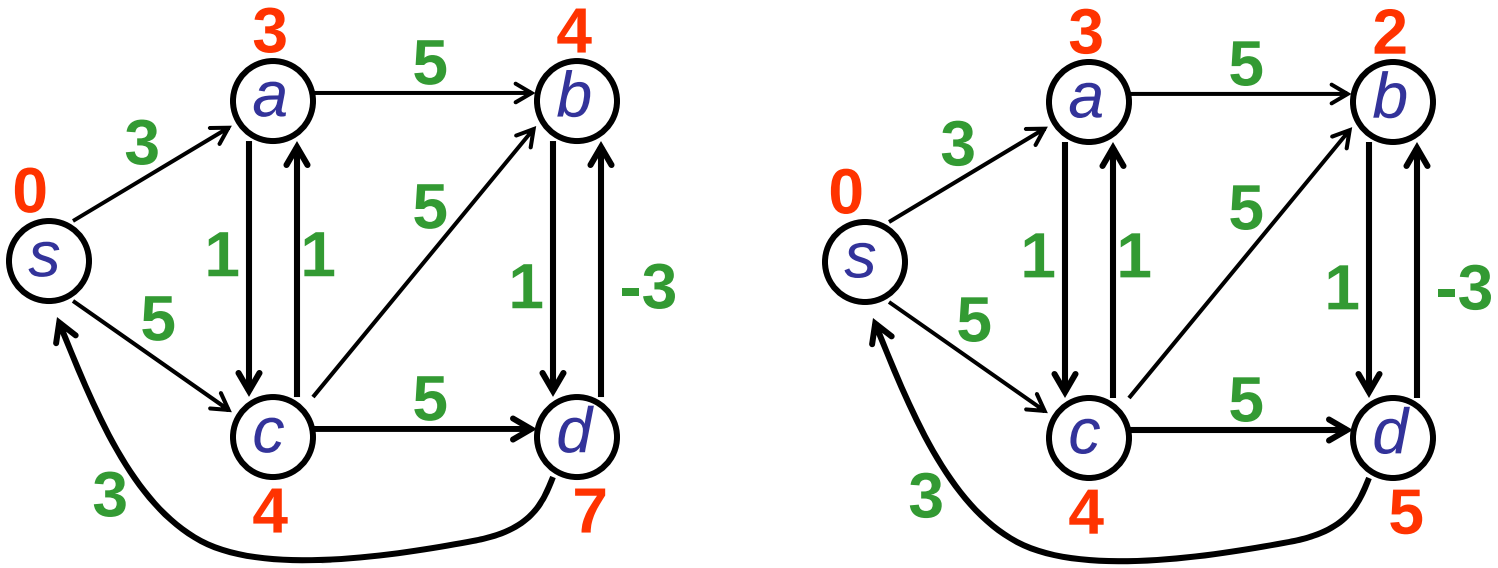
Étape 1 relaxation de tous les arcs dans l'ordre :
(*s*,*a*) (*s*,*c*) (*a*,*b*) (*a*,*c*) (*b*,*d*) (*c*,*a*) (*c*,*b*) (*c*,*d*) (*d*,*b*) (*d*,*s*)

Algorithme de Bellman-Ford



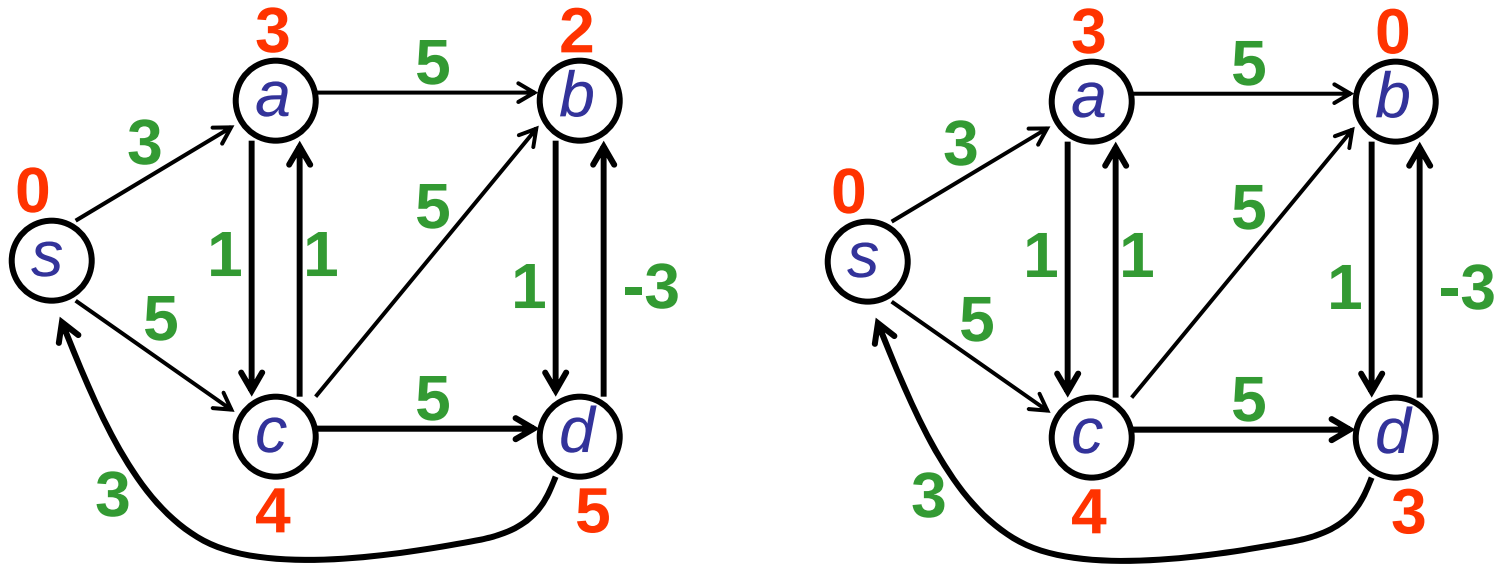
Étape 2 relaxation de tous les arcs dans l'ordre :
(*s*,*a*) (*s*,*c*) (*a*,*b*) (*a*,*c*) (*b*,*d*) (*c*,*a*) (*c*,*b*) (*c*,*d*) (*d*,*b*) (*d*,*s*)

Algorithme de Bellman-Ford



Étape 3 relaxation de tous les arcs dans l'ordre :
(*s*,*a*) (*s*,*c*) (*a*,*b*) (*a*,*c*) (*b*,*d*) (*c*,*a*) (*c*,*b*) (*c*,*d*) (*d*,*b*) (*d*,*s*)

Algorithme de Bellman-Ford



Étape 4 relaxation de tous les arcs dans l'ordre :
(*s*,*a*) (*s*,*c*) (*a*,*b*) (*a*,*c*) (*b*,*d*) (*c*,*a*) (*c*,*b*) (*c*,*d*) (*d*,*b*) (*d*,*s*)

Cycle de coût négatif: réduction encore possible !

Bellman-Ford revu….

- Soit le graphe ***G(V,E)***
- Initialiser $y_i = +\infty$ pour tous les sommets ***i***
- Initialiser $y_s = 0$.
- K ← 1
- Répéter

  Stable ← VRAI

  Pour tout arc ***(u,v)*** de ***E***

  faire **RELÂCHER**(u, v, ***c(u,v)***)

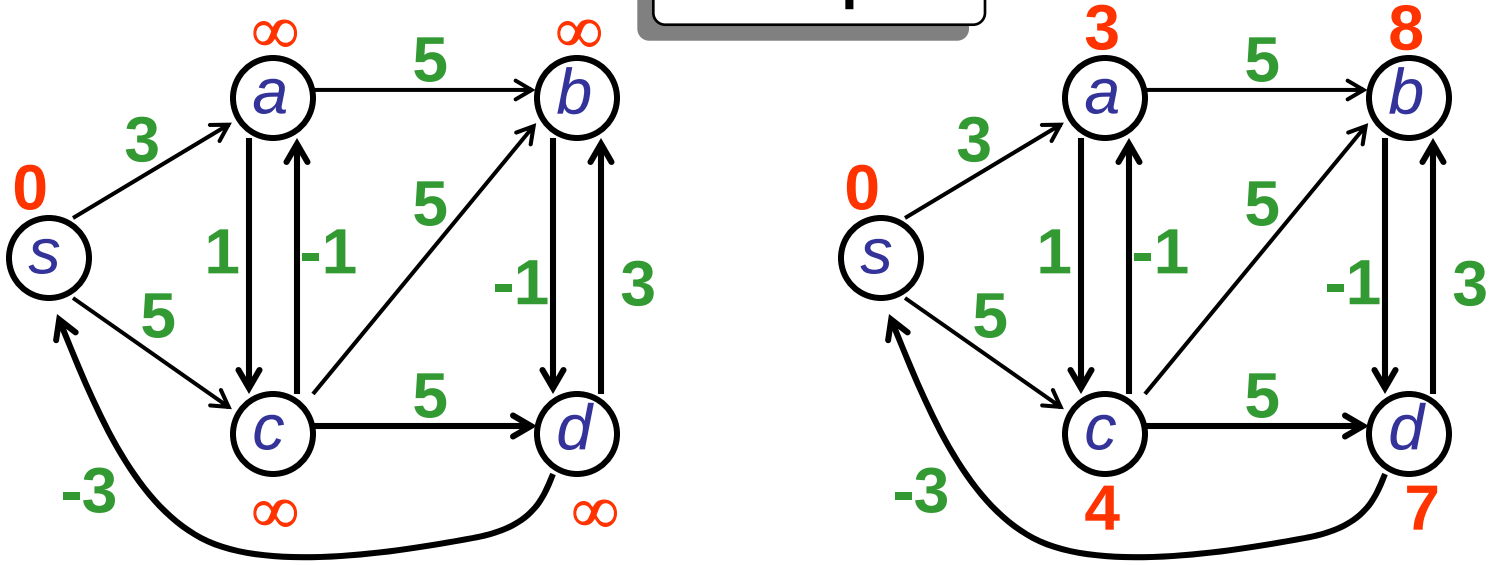
  Si ***…Alors*** Stable ← FAUX

  K ← k + 1

  Tant que Stable est FAUX ET k < n+1
- Si k=n+1 alors présence d'un circuit de poids négatif

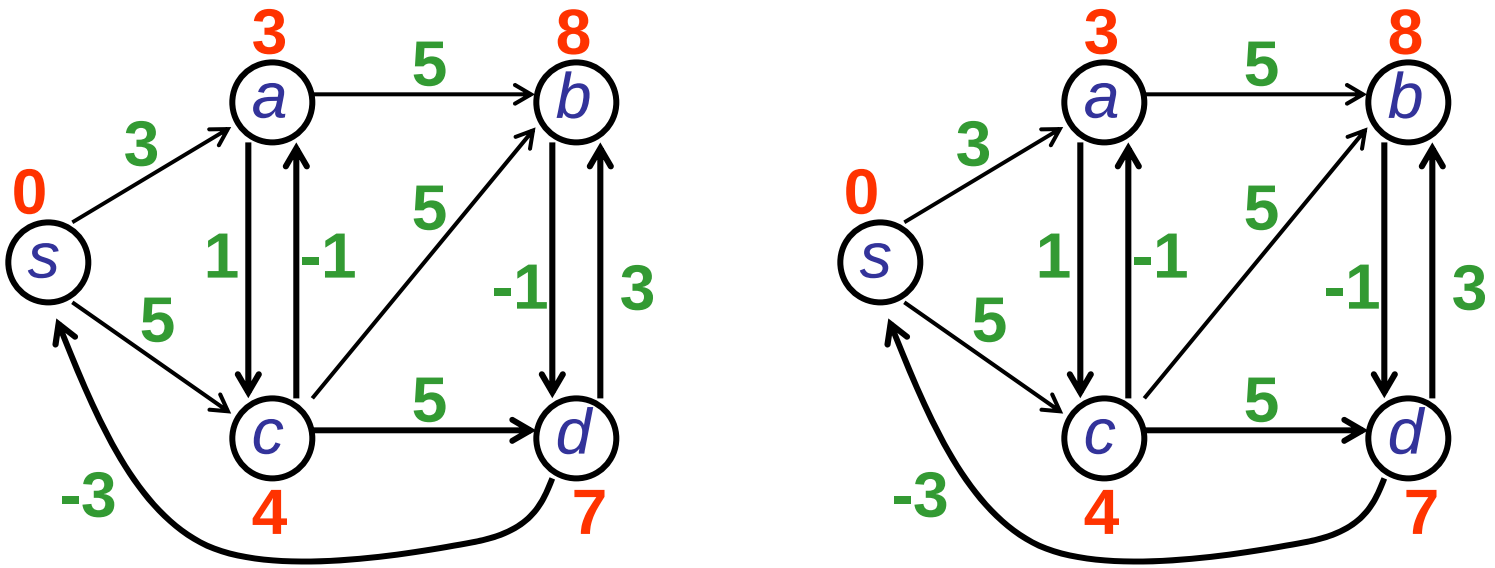
Algorithme de Bellman-Ford

Exemple 2



Étape 1 relaxation de tous les arcs dans l'ordre :
(*s*,*a*) (*s*,*c*) (*a*,*b*) (*a*,*c*) (*b*,*d*) (*c*,*a*) (*c*,*b*) (*c*,*d*) (*d*,*b*) (*d*,*s*)

Algorithme de Bellman-Ford



Étape 2 relaxation de tous les arcs dans l'ordre :
(*s*,*a*) (*s*,*c*) (*a*,*b*) (*a*,*c*) (*b*,*d*) (*c*,*a*) (*c*,*b*) (*c*,*d*) (*d*,*b*) (*d*,*s*)

Pas de réduction possible : coûts corrects!

Synthèse Plus courts chemins

- **Plus cours chemin à origine unique**
  - Algorithme de Dijkstra.
  - Algorithme de Bellman-Ford.
- **Plus cours chemin à destination unique**
  - En inversant le sens de chaque arc, on peut ramener ce problème à celui du plus court chemin à origine unique.
- **Plus cours chemin pour un couple de sommets donné**
  - Plus court chemin à origine unique avec condition d'arrêt lorsque la destination est atteinte.
- **Plus cours chemin pour tout couple de sommets**
  - On peut exécuter un algorithme à origine unique à partir de chaque sommet.
  - Mieux: algorithmes de Floyd-Warshall.

LES RÉSEAUX

Un réseau est un graphe orienté et pondéré, c'est-à-dire que chaque arc, en plus d'un sens, possède une valeur associée qui représente la *" capacité "* de liaison de l'arc.

Exemple

une distance, un coût, un débit maximum, etc.

Le problème qui consiste à déterminer quel est le flot optimum que l'on peut obtenir à travers un réseau est un problème non trivial dont l'étude détaillée est du domaine de la recherche opérationnelle.

Lecture: p. 160