

MOGPL - Partie II : Graphes

Evripidis Bampis
evripidis.bampis@lip6.fr

Le programme

1 et 2. Rappels sur les graphes

Plus courts chemins

Programmation dynamique

(2. et) 3. Problèmes de flots

4. Problèmes d'affectation et de transport

5. Compléments et révisions

5 cours et 4 TDs

5ème « TD » : soutenance de projet

Bibliographie

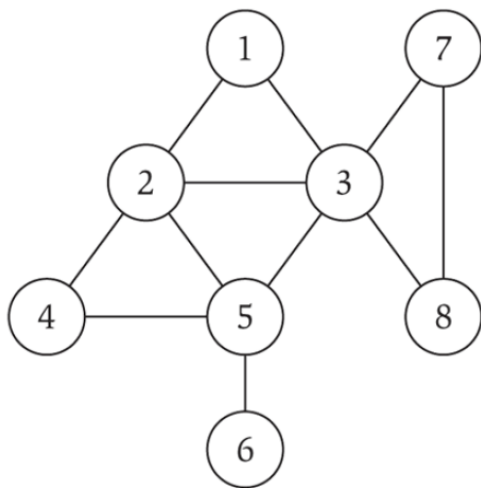
- Algorithm Design, John Kleinberg & Eva Tardos, Pearson
(une partie des transparents est basée sur ce livre)
- Algorithmique, Thomas Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein
- Graphes et Algorithmes, Michel Gondran, Michel Minoux

Rappel sur les graphes

I. Définition et représentation

Graphe non-orienté. $G = (V, E)$

- V = ensemble de sommets.
- E = ensemble d'arêtes.
- Modélise la relation par paires d'objets.
- Notations : $n = |V|$, $m = |E|$.



$V = \{ 1, 2, 3, 4, 5, 6, 7, 8 \}$

$E = \{ 1-2, 1-3, 2-3, 2-4, 2-5, 3-5, 3-7, 3-8, 4-5, 5-6, 7-8 \}$

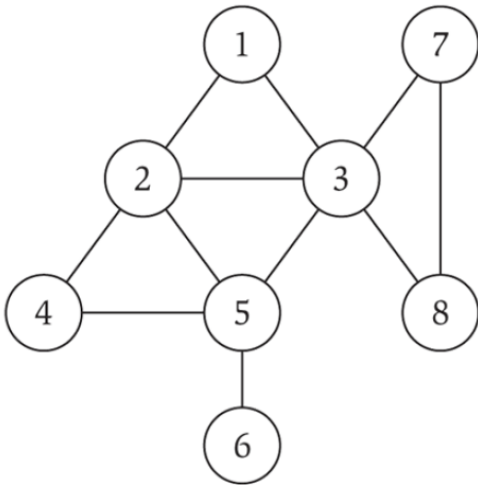
$n = 8$

$m = 11$

Rappel sur les graphes

Graphe non orienté. $G = (V, E)$

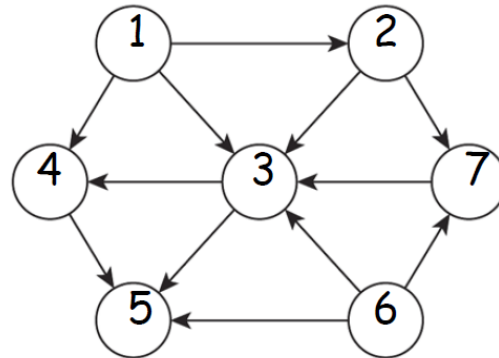
- V = ensemble de sommets.
- E = ensemble d'arêtes.
- Modélise la relation par paires d'objets.
- Notations : $n = |V|$, $m = |E|$.



$V = \{1, 2, 3, 4, 5, 6, 7, 8\}$
 $E = \{1-2, 1-3, 2-3, 2-4, 2-5, 3-4, 3-5, 3-7, 3-8, 4-5, 5-6, 7-8\}$
 $n = 8$
 $m = 11$

Graphe orienté. $G = (V, A)$

- V = ensemble de sommets.
- A = ensemble d'arcs.
- Modélise la relation par couples (non symétrique) d'objets.
- Notations : $n = |V|$, $m = |A|$.



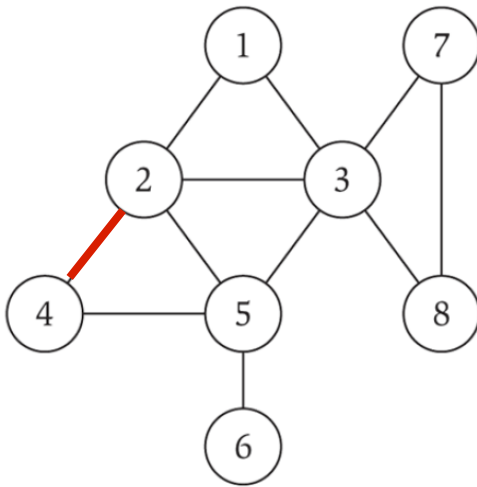
Quelques applications de graphes

Graphe	Sommets	Arêtes/arcs
transports	carrefours	routes
communication	ordinateurs	câbles de fibre optique
Web	pages web	hyperliens
réseaux sociaux	personnes	relations
réseau trophique	espèces	prédateur-proie
systèmes logiciels	fonctions	appels de fonctions
ordonnancement	tâches	contraintes de précédence

Représentation de graphes : Matrice d'adjacence

Matrice d'adjacence. matrice $n \times n$ avec $A_{uv} = 1$ si (u, v) est une arête.

- Deux apparitions pour chaque arête.
- Espace proportionnel à n^2 .
- Tester si (u, v) est une arête prend un temps en $O(1)$.
- Identifier/parcourir toutes les arêtes prend un temps en $O(n^2)$.



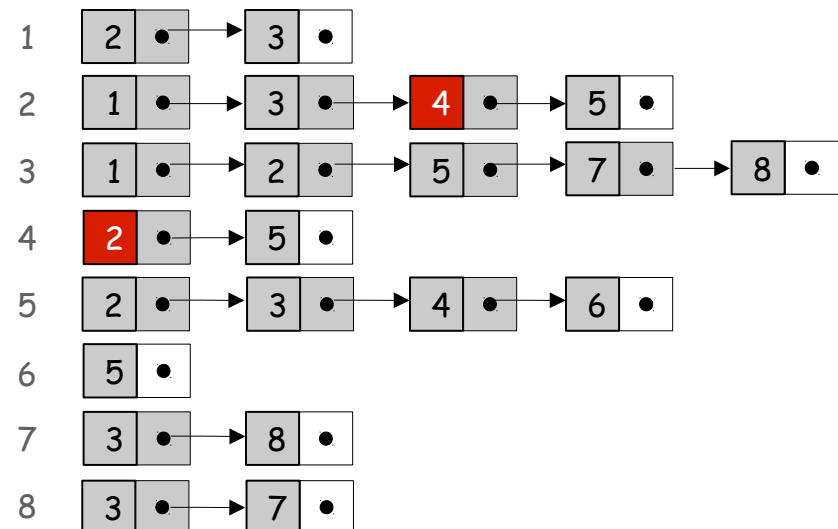
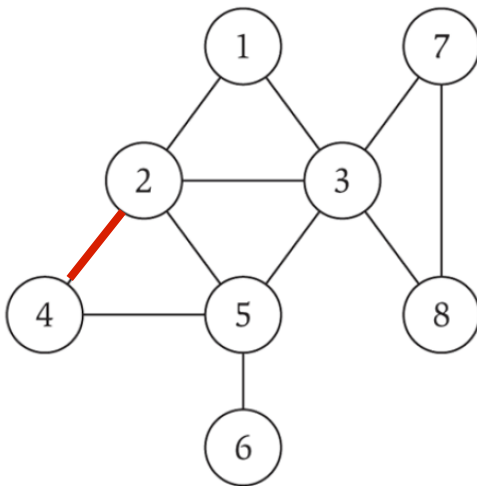
	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	1	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

Représentation de graphes: Liste d'adjacence

Liste d'adjacence. Tableau de listes indexé par les sommets.

- Deux apparitions pour chaque arête.
- Espace proportionnel à $m + n$.
- Tester si (u, v) est une arête prend un temps en $O(\text{deg}(u)+1)=O(n)$.
- Identifier/parcourir toutes les arêtes prend un temps en $O(m + n)$.

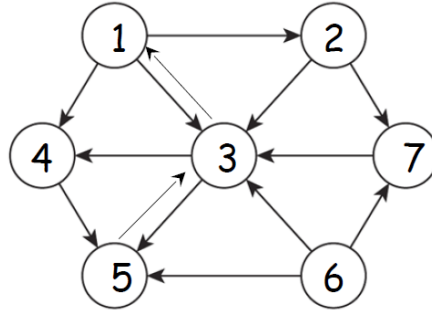
degré = nombre de voisins de u



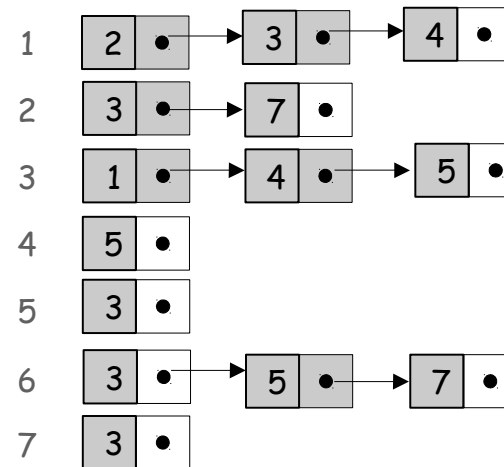
Représentation de graphes : Matrice d'adjacence

Matrice d'adjacence. matrice $n \times n$ avec $A_{uv} = 1$ si (u, v) est un arc

Liste d'adjacence. Tableau de listes indexé par les sommets.



	1	2	3	4	5	6	7
1	0	1	1	1	0	0	0
2	0	0	1	0	0	0	1
3	1	0	0	1	1	0	0
4	0	0	0	0	1	0	0
5	0	0	1	0	0	0	0
6	0	0	1	0	1	0	1
7	0	0	1	0	0	0	0

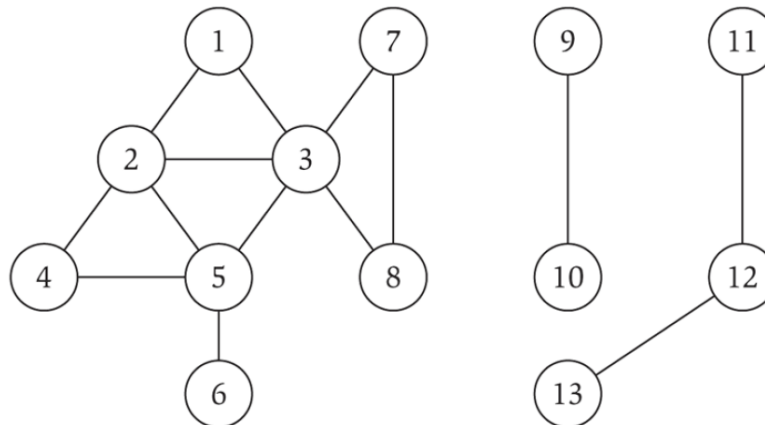


II. Chaînes/chemins, connectivité

Déf. Une **chaîne** dans un graphe non orienté $G = (V, E)$ est une séquence P de sommets $v_1, v_2, \dots, v_{k-1}, v_k$ avec la propriété que chaque paire consécutive de sommets (v_i, v_{i+1}) est reliée par une arête de E .

Def. Un graphe non-orienté est **connexe** si pour toute paire de sommets u et v , il existe une chaîne entre u et v .

Def. Une composante connexe est un sous-ensemble de sommets induisant un sous-graphe connexe et maximal pour l'inclusion.

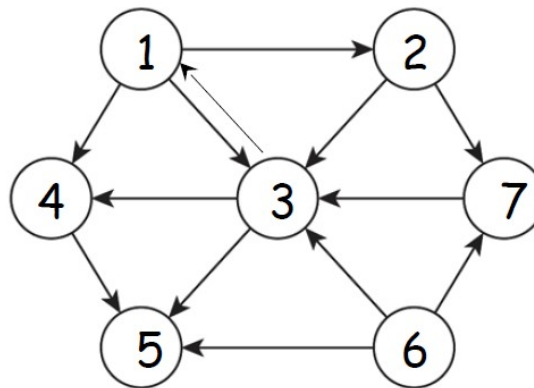


Chemins et forte connectivité

Déf. Un **chemin** dans un graphe orienté $G = (V, A)$ est une séquence P de sommets $v_1, v_2, \dots, v_{k-1}, v_k$ avec la propriété que chaque paire consécutive de sommets (v_i, v_{i+1}) est reliée par une arc (v_i, v_{i+1}) de A .

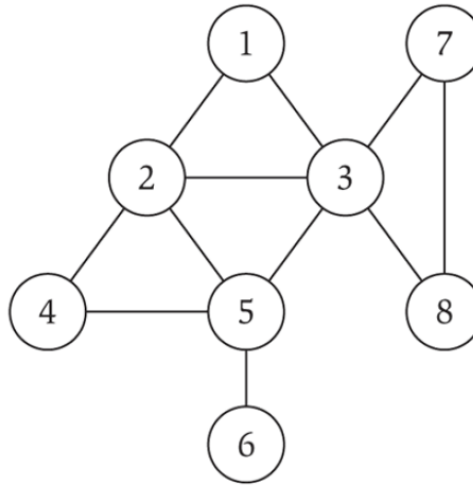
Def. Un graphe orienté est **fortement connexe** si pour toute paire de sommets u et v , il existe un chemin de u à v (et un chemin de v à u).

Def. Une **composante fortement connexe** est un ensemble de sommets induisant un sous-graphe fortement connexe et maximal pour l'inclusion.



Cycles

Déf. Un **cycle** est une chaîne $v_1, v_2, \dots, v_{k-1}, v_k$ dans laquelle $v_1 = v_k$, $k > 2$, (et deux arêtes consécutives toujours distinctes).



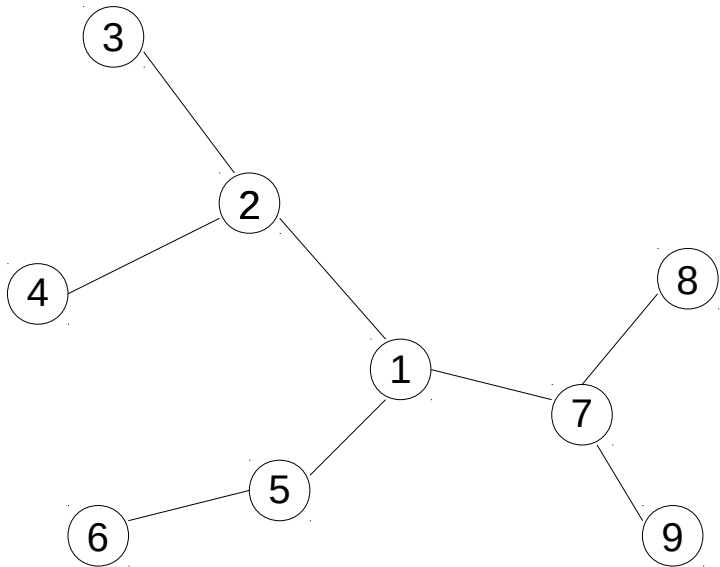
cycle $C = 1-2-4-5-3-1$

Arbres

Déf. Un graphe non orienté est un **arbre** s'il est connexe et ne contient pas de cycle.

Théorème. Soit G un graphe non orienté avec n sommets. Toute paire parmi les propriétés suivantes implique la troisième.

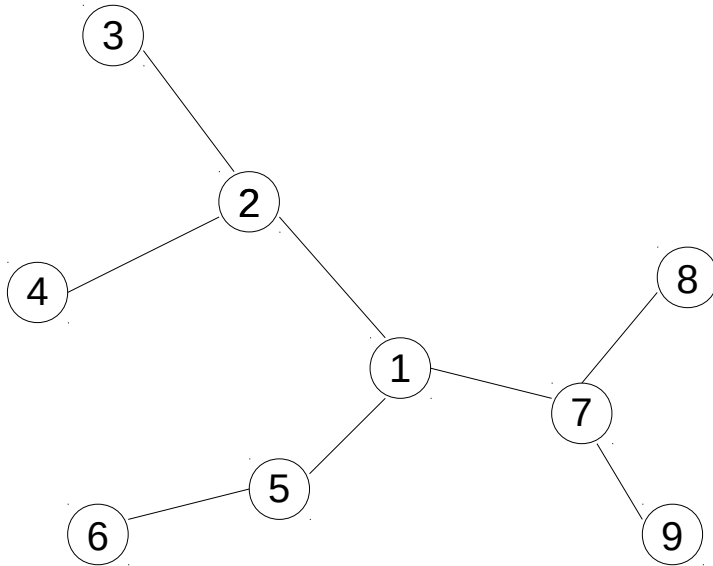
- G est connexe.
- G ne contient pas de cycle.
- G a $n-1$ arêtes.



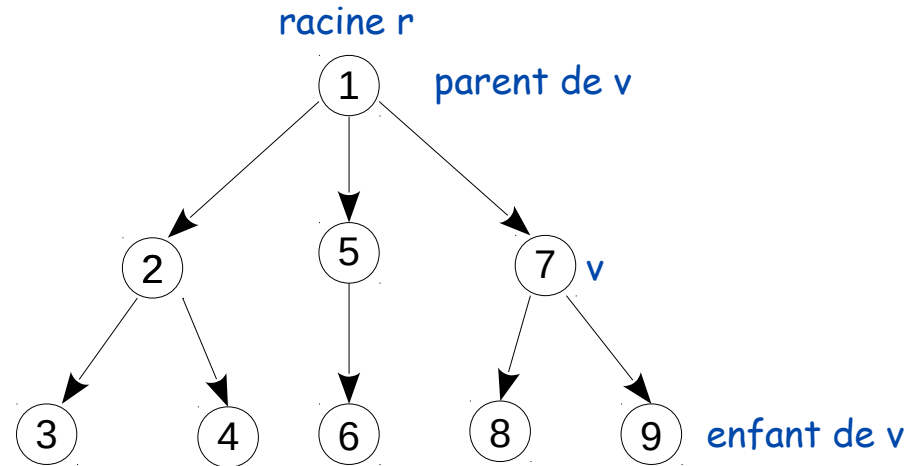
un arbre

Arbres enracinés

Arbre enraciné. Etant donné un arbre T , choisir un sommet racine r et "orienter" chaque arête "partant" de r .



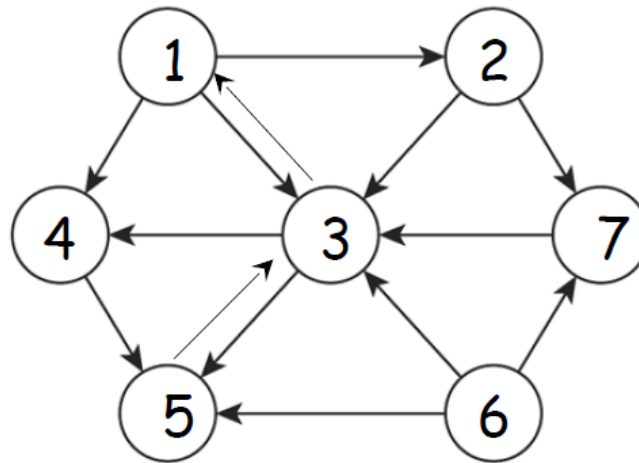
un arbre



le même arbre, enraciné à 1

Circuits

Déf. Un **circuit** est un chemin $v_1, v_2, \dots, v_{k-1}, v_k$ ($k > 1$) dans lequel $v_1 = v_k$

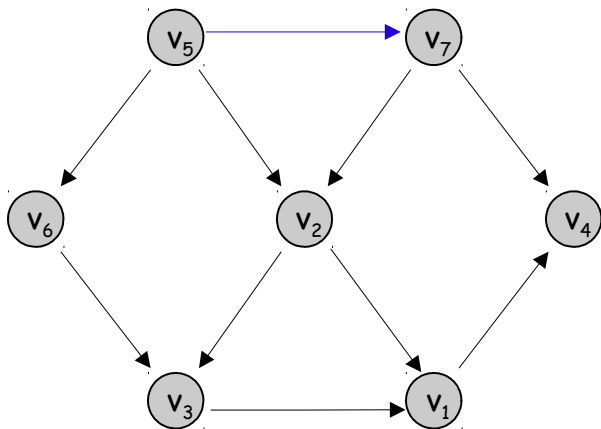


Graphes orientés acycliques (DAG)

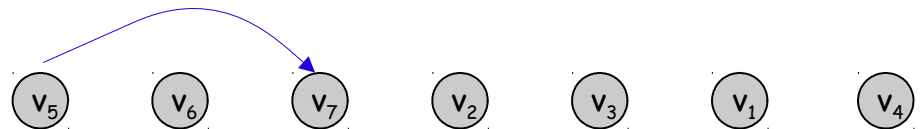
Déf. Un **DAG** est un graphe orienté ne contenant pas de circuit.

Ex. Contraintes de précédence : arc $(v_i, v_j) \Rightarrow v_i$ doit précéder v_j .

Déf. Un **ordre topologique** d'un graphe orienté $G = (V, A)$ est un ordre sur ses sommets tel que pour chaque arc (v_i, v_j) v_i est avant v_j dans l'ordre.



un DAG



un ordre topologique

Propriété. G a un ordre topologique si et seulement si c'est un DAG.

Comment déterminer un ordre topologique dans un DAG ?

Si G est un DAG, alors il contient un sommet sans arcs entrants

- 1- choisir un sommet v sans arcs entrant et le placer en premier
- 2- éliminer v de G
- 3- de manière récursive déterminer un ordre topologique dans $G - \{v\}$ et le concatener après v

Connexité

le problème de connexité s - t . Etant donnés deux sommets s et t , existe-t-il une chaîne entre s et t /un chemin de s à t ?

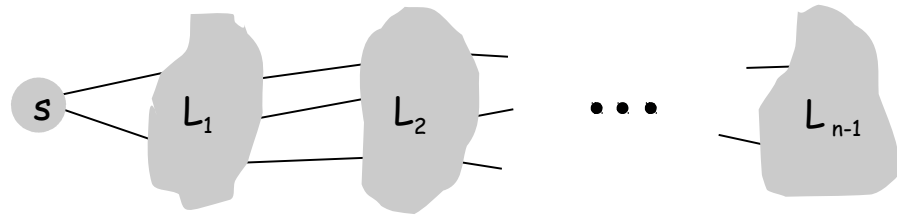
le problème de la plus courte chaîne (chemin) s - t . Etant donnés deux sommets s et t , quelle est la longueur de la plus courte chaîne (chemin) entre s et t ?

Parcours en largeur (BFS)

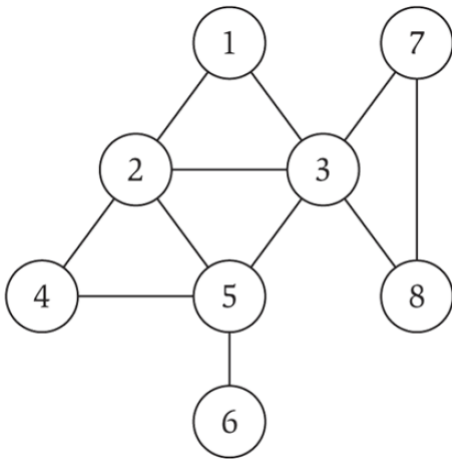
Intuition de BFS. Explorer en partant de s dans toutes les directions, en rajoutant les sommets "niveau-par niveau".

Algorithme BFS.

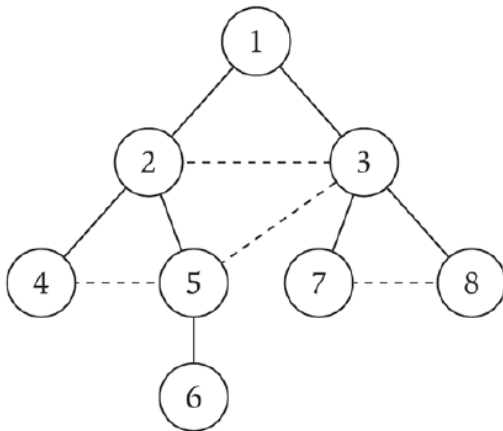
- $L_0 = \{ s \}$.
- L_1 = tous les voisins de L_0 .
- L_2 = tous les sommets n'appartenant ni à L_0 ni à L_1 , et ayant une arête vers un sommet de L_1 .
- L_{i+1} = tous les sommets n'appartenant pas à un niveau précédent, et ayant une arête vers un sommet de L_i .



Parcours en largeur (BFS)



Complexité : BFS peut être implanté en temps $O(m + n)$ si le graphe est donné sous la forme de listes d'adjacence.



L_0

L_1

L_2

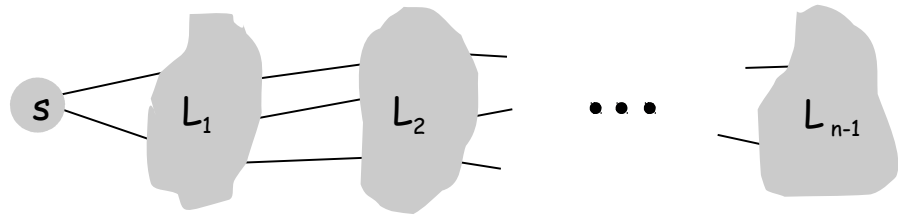
L_3

Parcours en largeur (BFS)

Intuition de BFS. Explorer en partant de s dans toutes les directions, en rajoutant les sommets "niveau-par niveau".

Algorithme BFS.

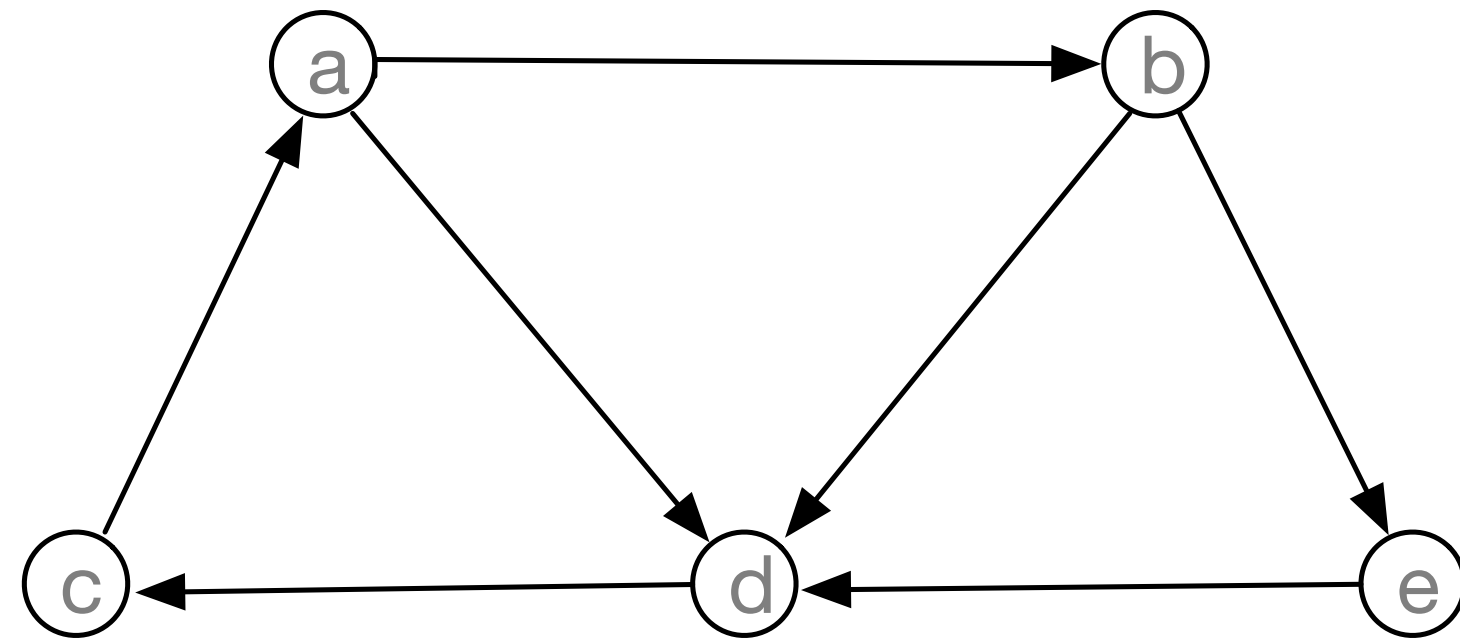
- $L_0 = \{ s \}$.
- L_1 = tous les voisins de L_0 .
- L_2 = tous les sommets n'appartenant ni à L_0 ni à L_1 , et ayant une arête vers un sommet de L_1 .
- L_{i+1} = tous les sommets n'appartenant pas à un niveau précédent, et ayant une arête vers un sommet de L_i .



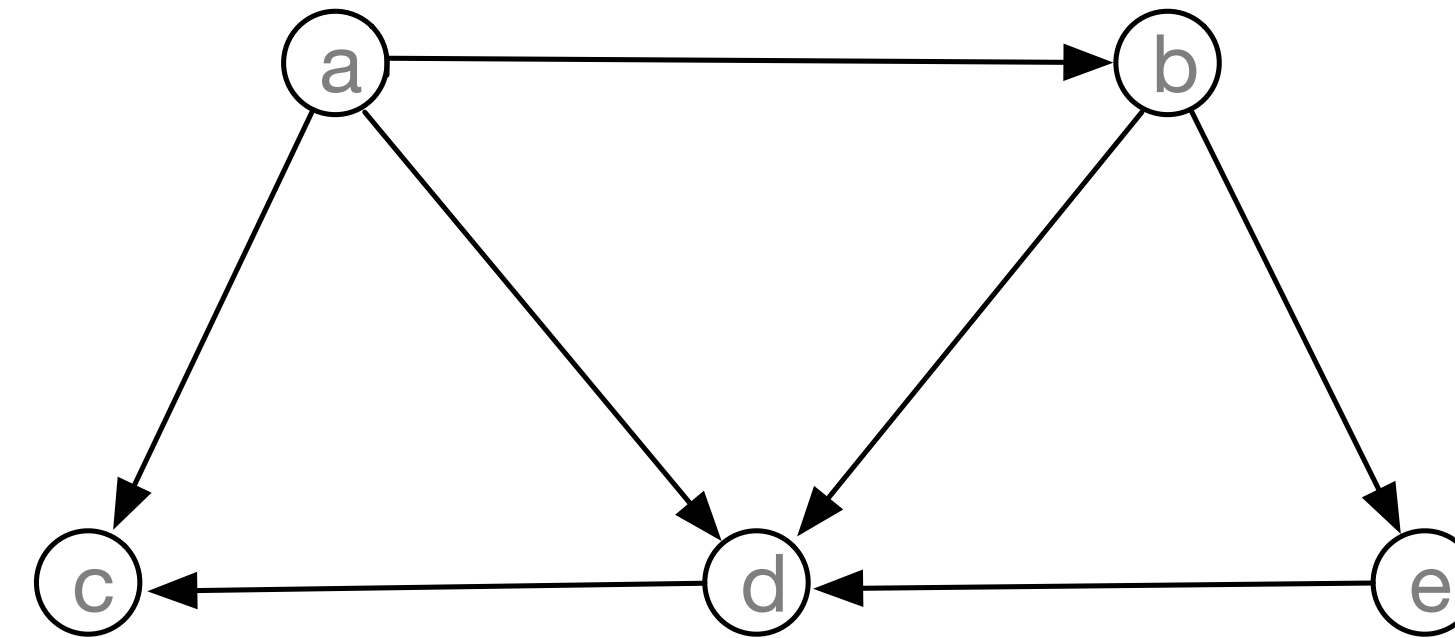
Théorème. Pour chaque valeur i , L_i contient tous les sommets à une distance de s exactement égale à i . Il existe une chaîne de s à t **ssi** t appartient à un certain niveau.

Corollaire. BFS calcule non seulement les sommets que l'on peut atteindre à partir de s , mais aussi les plus courtes chaînes vers eux.

Exemple de graphe fortement connexe



fortement connexe



non-fortement connexe

Comment savoir si un graphe G est fortement connexe ?

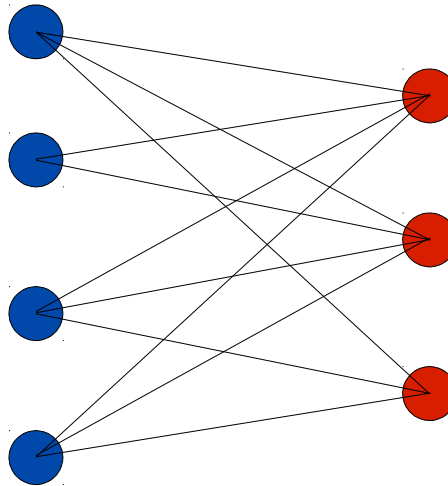
- 1- choisir un sommet quelconque s
- 2- appliquer BFS à partir de s dans G
- 3- appliquer BFS à partir de s dans le graphe G -rev où les orientations des arcs sont inversés
- 4- Retourner vrai ssi tous les sommets sont atteignables lors de 2 parcours BFS

Graphes bipartis

Déf. Un graphe non-orienté $G = (V, E)$ est **biparti** si ses sommets peuvent être coloriés bleus et rouges de sorte que chaque arête de G ait une extrémité rouge et une extrémité bleue.

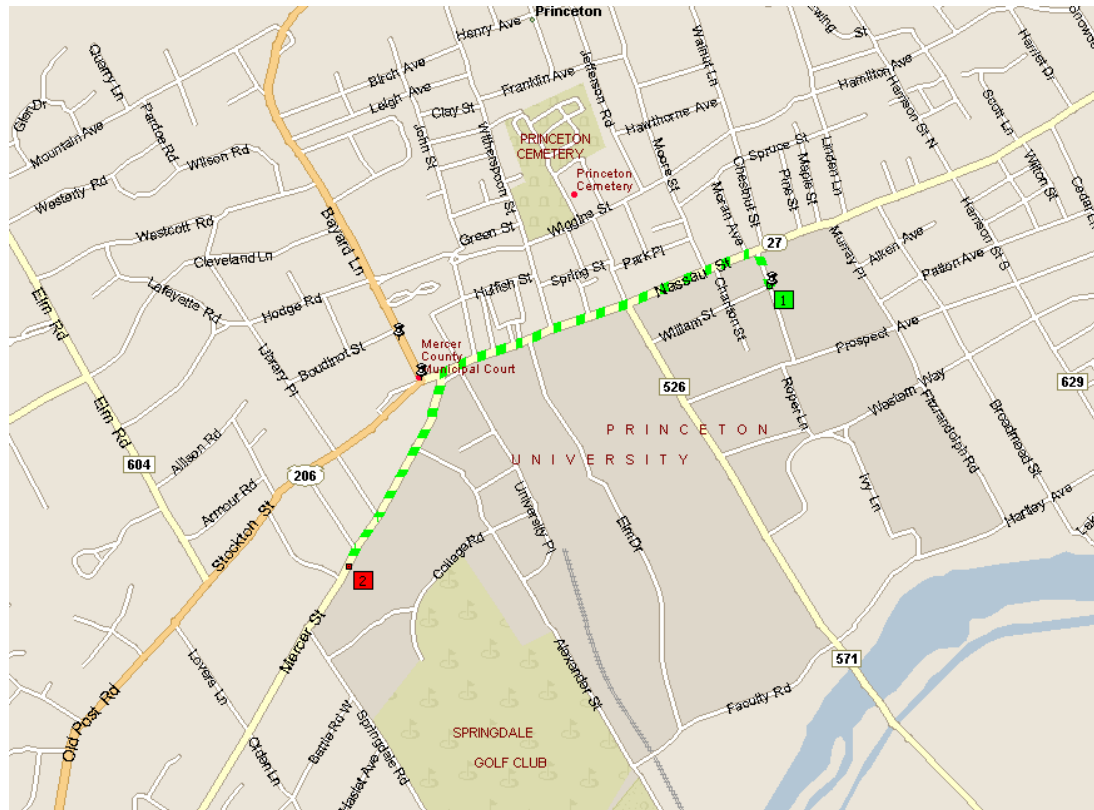
Applications.

- mariage stable : hommes = rouge, femmes = bleu.
- Ordonnancement : machines = rouge, tâches = bleu.



un graphe biparti

Plus court chemin dans un graphe



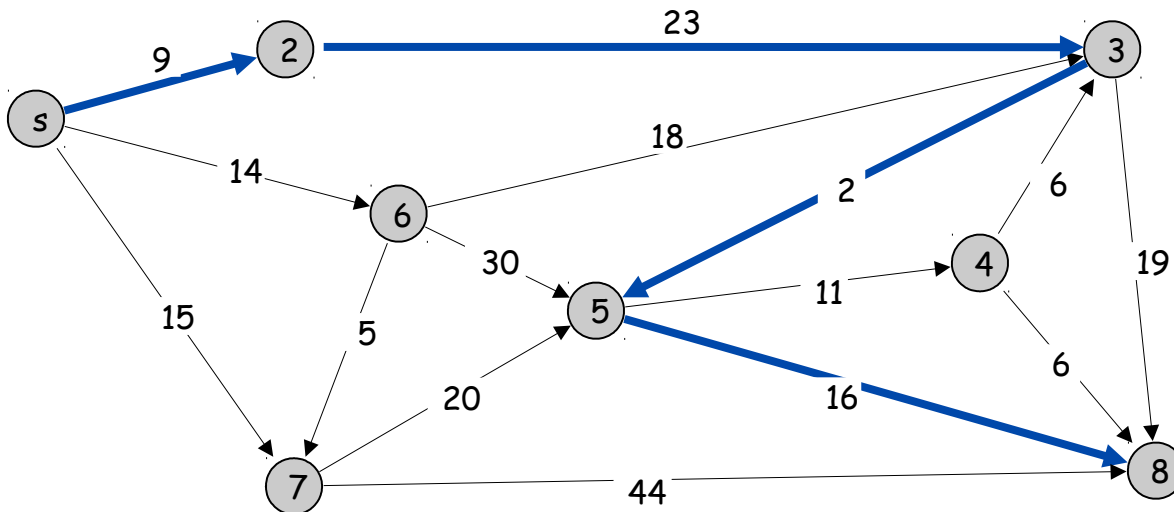
Problème du plus court chemin

Plus court chemin.

- Graphe orienté $G = (V, A)$.
- Source s , destination t
- Longueur d_e = longueur de l'arc e (supposée entière).

Problème du plus court chemin : déterminer un plus court chemin de s à t .
Parfois : déterminer un plus court chemin de s à chacun des autres sommets du graphe

coût d'un chemin = somme de coûts des arcs sur le chemin



Coût du chemin $s-2-3-5-8$
 $= 9 + 23 + 2 + 16$
 $= 50.$

Problème du plus court chemin

Résolution : dépend des données.

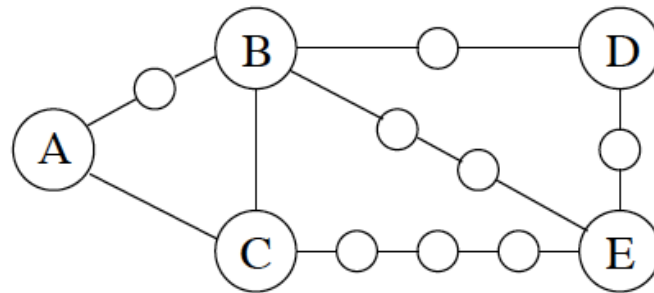
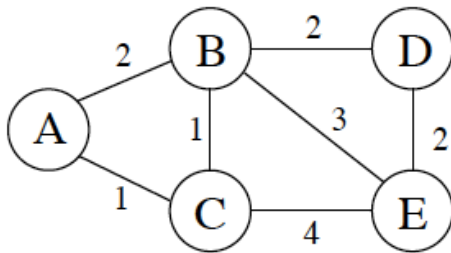
- a) La distance de chaque arc est 1 : BFS
- b) *La distance de chaque arc est positive (ou nulle)*
- c) Le graphe est sans circuit
- d) Cas général

b) PCCH avec distances non négatives

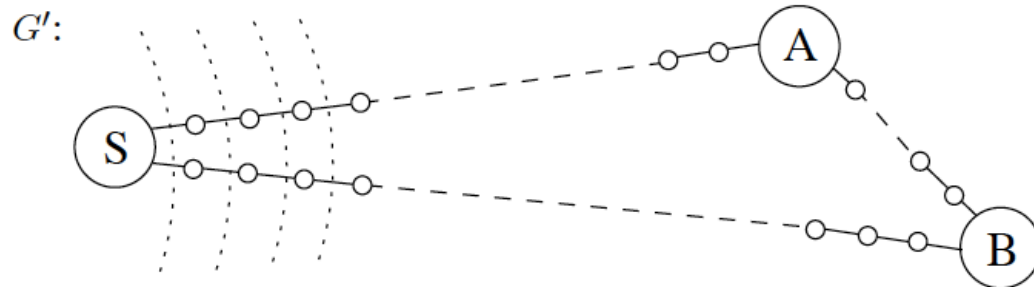
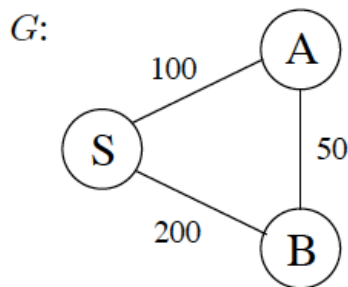
Peut-on le résoudre avec BFS ?

Il suffit pour chaque arête (ou arc dans le cas orienté) $e=(u,v)$ de la remplacer par d_e arêtes de poids 1 en rajoutant $d_e - 1$ sommets fictifs entre u et v !!!

Exemple



Problème : efficacité



Algorithme de Dijkstra (d_e non négatives)

Algorithme de Dijkstra : principe.

- Soit S l'ensemble des **sommets déjà explorés** : sommets u pour lesquels on a calculé la distance $\lambda^*(u)$ optimale de s à u .

Initialiser $S = \{s\}$, $\lambda^*(s) = 0$.

- Tant que $S \neq V$

choisir un sommet non-exploré v **minimisant**

$$\lambda(v) = \min \{ \lambda^*(u) + d(u, v) : u \in S \}$$

rajouter v à S , et poser $\lambda^*(v) = \lambda(v)$.

↙
plus court chemin à un sommet u dans la partie explorée, suivi d'un arc unique (u, v)

Dans l'algorithme :

On va maintenir pour chaque sommet v non exploré (hors de S) une marque

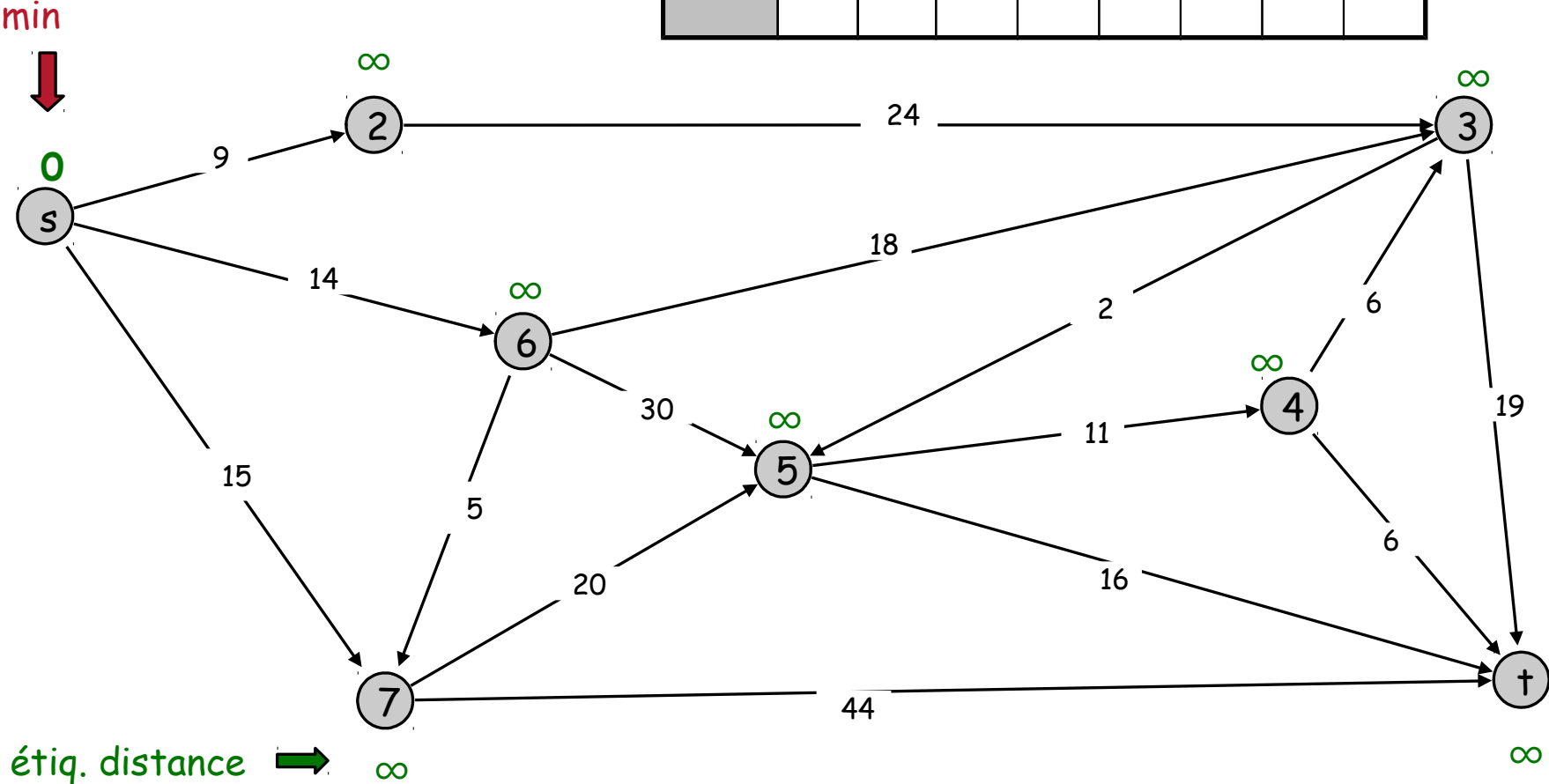
$$\lambda(v) = \min \{ \lambda^*(u) + d(u, v) : u \in S \}$$

Quand on ajoute un sommet w dans S , on met simplement à jour les marques des sommets non explorés.

Algorithme de Dijkstra

$S = \{ \}$

	s	2	3	4	5	6	7	t
Init.	0	∞	∞	∞	∞	∞	∞	∞
	-	-	-	-	-	-	-	-



Algorithme de Dijkstra

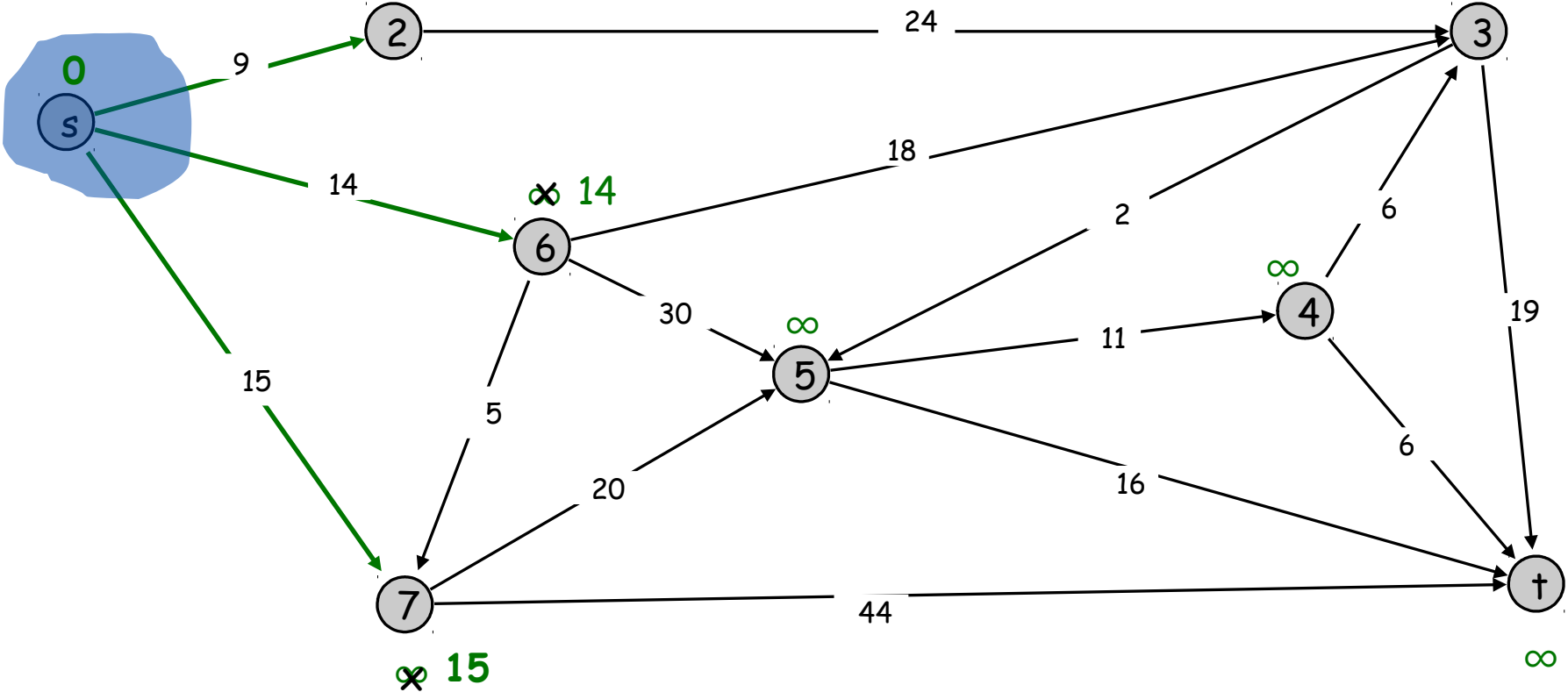
$S = \{s\}$

	s	2	3	4	5	6	7	8	t
Init.	0	∞	∞	∞	∞	∞	∞	∞	∞
k=1	-	9	∞	∞	∞	14	15	∞	∞
	-	s	-	-	-	s	s	-	-

diminuer marque

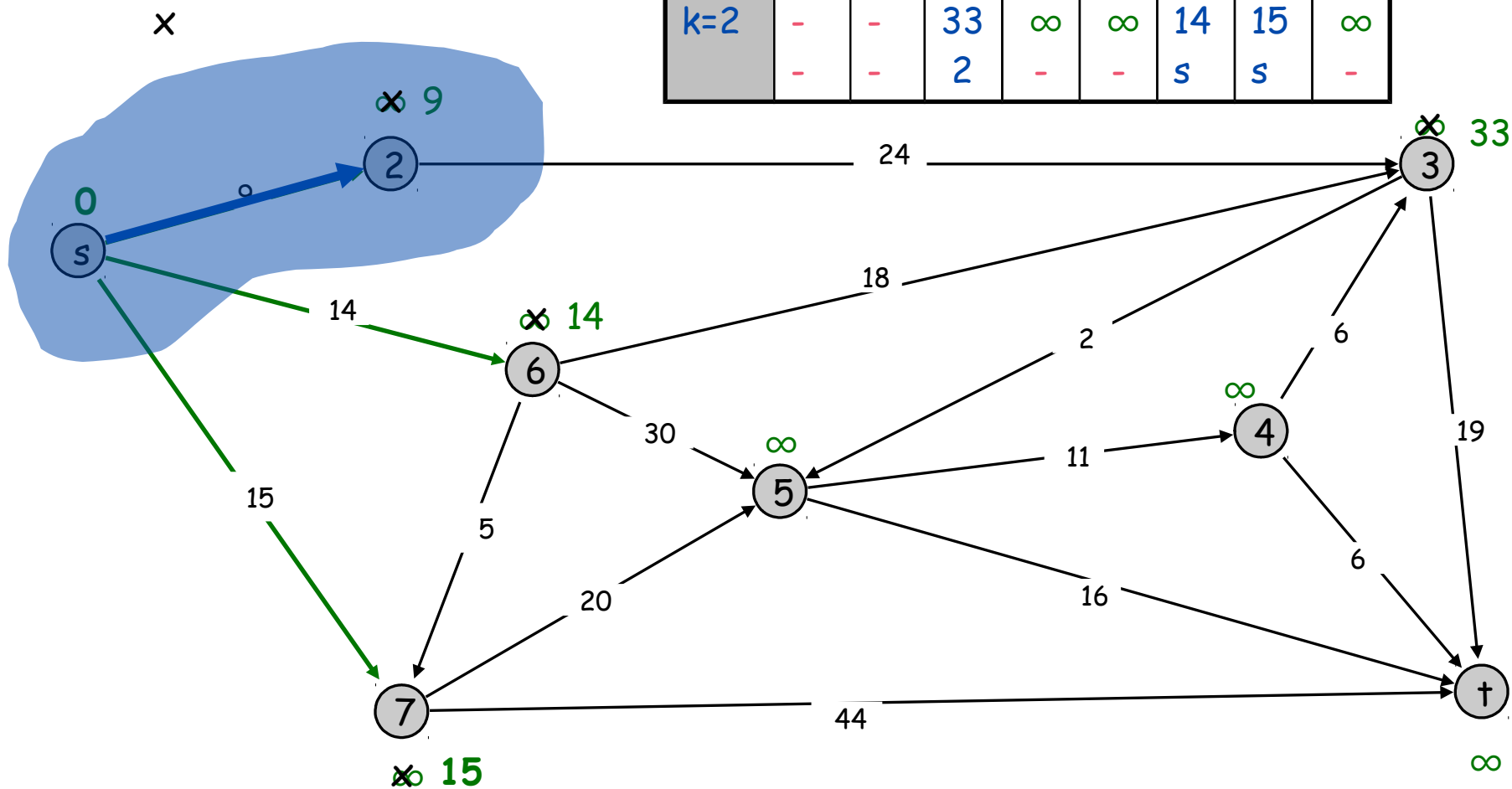


∞ 9



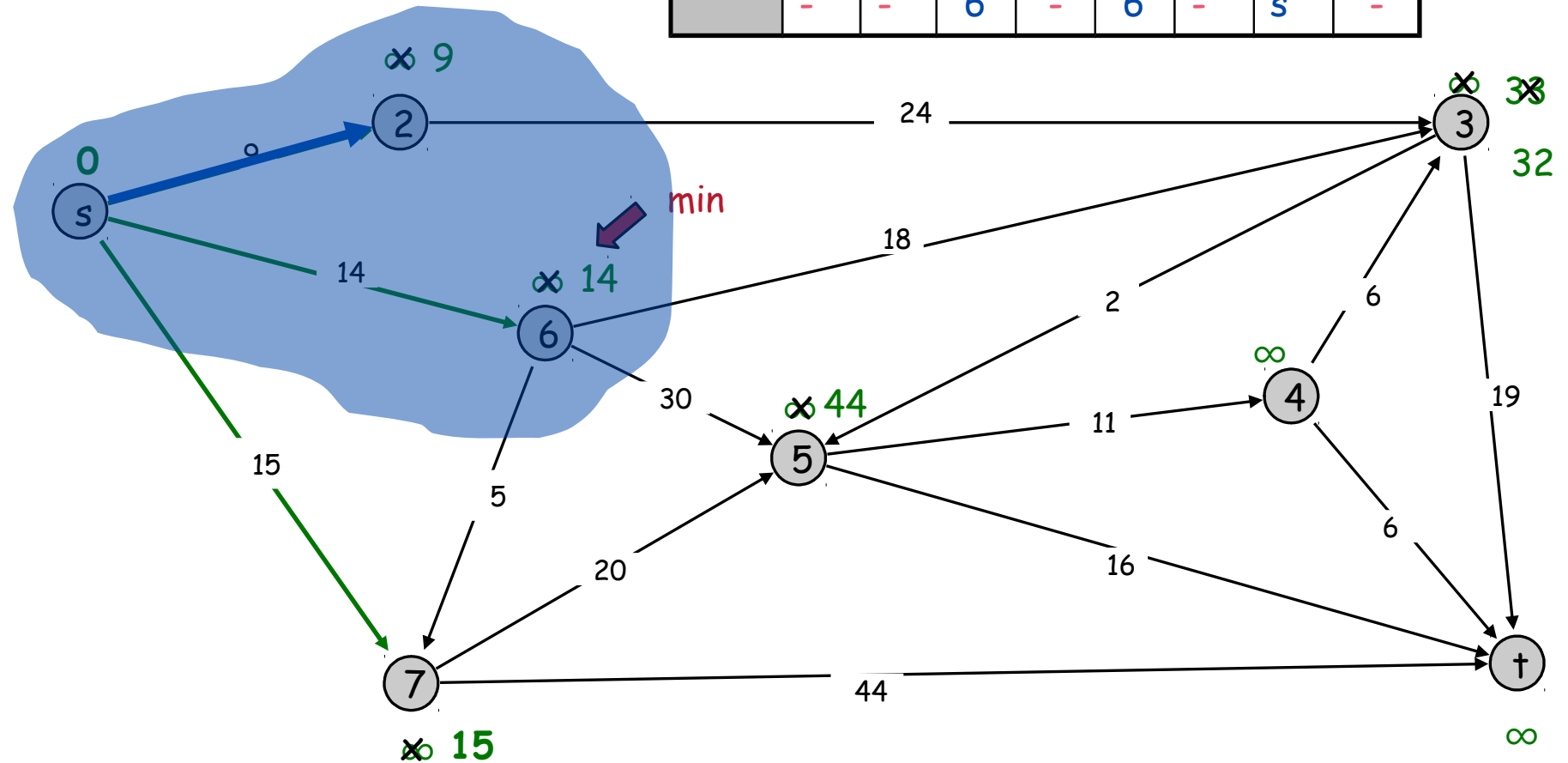
$S = \{s, 2\}$

	s	2	3	4	5	6	7	t
Init.	0	∞	∞	∞	∞	∞	∞	∞
k=1	-	9	∞	∞	∞	14	15	∞
k=2	-	-	33	∞	∞	14	15	∞



$S = \{s, 2, 6\}$

	s	2	3	4	5	6	7	t
k=1	-	9	∞	∞	∞	14	15	∞
k=2	-	-	33	∞	∞	14	15	∞
k=3	-	-	32	∞	44	-	15	∞



	s	2	3	4	5	6	7	t
Init.	0 -	∞ -	∞ -	∞ -	∞ -	∞ -	∞ -	∞ -
k=1	- -	9 s	∞ -	∞ -	∞ -	14 s	15 s	∞ -
k=2	- -	- -	33 t	∞ -	∞ -	14 s	15 s	∞ -
k=3	- -	- -	32 6	∞ -	44 6	- -	15 s	∞ -
k=4	- -	- -	32 6	∞ -	35 7	- -	- -	59 7
k=5	- -	- -	- -	∞ -	34 3	- -	- -	51 3
k=6	- -	- -	- -	45 5	- -	- -	- -	50 5
k=7	- -	- -	- -	- -	- -	- -	- -	50 5

Reconstitution des chemins : partir de t, et remonter les prédécesseurs jusqu'à s.

Algorithme de Dijkstra : idée de la preuve

Invariant. Pour chaque sommet $u \in S$, $\Lambda^*(u)$ est la longueur d'un plus court chemin de s à u .

Preuve. (par induction sur $|S|$)

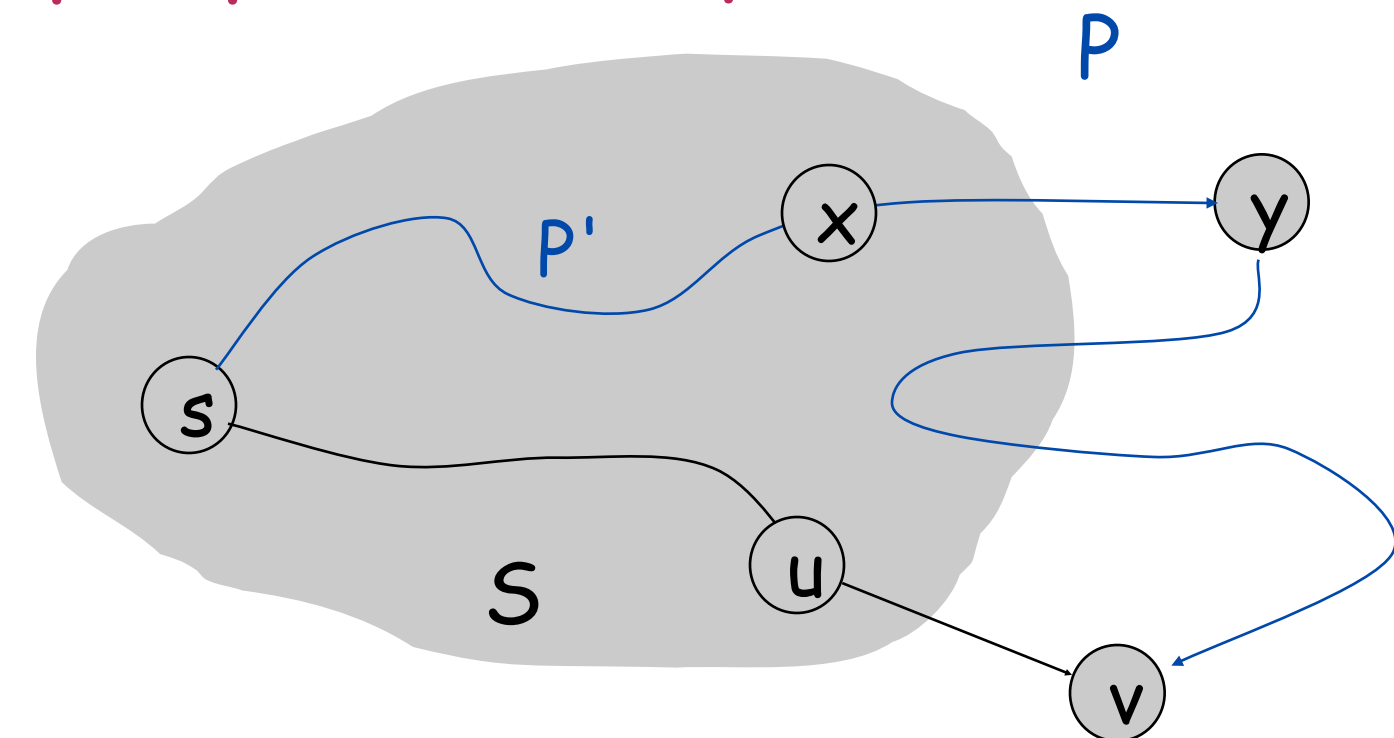
Cas de base : $|S| = 1$ est trivial.

Hypothèse de l'induction : Supposons vrai pour $|S| = k \geq 1$.

- Soit v le prochain sommet rajouté à S , et $u-v$ l'arc choisi.
Le plus court chemin $s-u$ plus (u, v) est un chemin $s-v$ de longueur $\Lambda(v)$.

Soit P un chemin $s-v$ arbitraire. Montrons qu'il n'est pas plus court que $\Lambda(v)$.

- Soit $x-y$ le premier arc dans P qui sort de S , et soit P' le sous-chemin jusqu'à x .
- P est déjà trop long lorsqu'il quitte S .



$$\begin{array}{ccccccc} \ell(P) & \geq & \ell(P') + \ell(x, y) & \geq & \Lambda^*(x) + \ell(x, y) & \geq & \Lambda(y) \geq \Lambda(v) \\ | & & | & & | & & | \\ \text{poids} & & \text{hypothèse} & & \text{defn de } \Lambda(y) & & \text{Dijkstra choisit } v \\ \text{non-négatifs} & & \text{de l'induction} & & & & \text{au lieu de } y \end{array}$$

Algorithme de Dijkstra

Implantation « naïve » (calcul du sommet de plus petite marque à chaque étape) : $O(n^2)$

En maintenant une SD permettant d'avoir ce sommet plus efficacement : $O(n+m \log n)$.

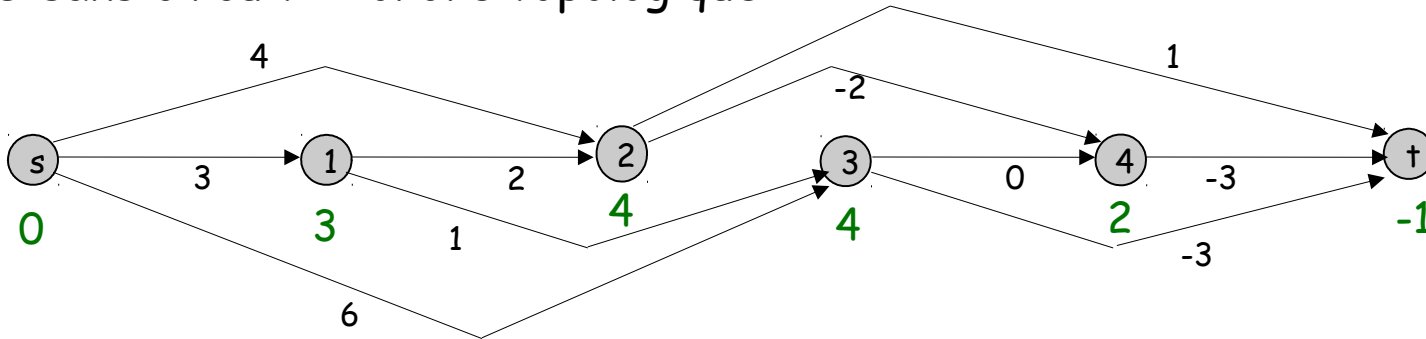
Problème du plus court chemin

Résolution : dépend des données.

- a) La distance de chaque arc est 1 : BFS
- b) La distance de chaque arc est positive (ou nulle)
- c) *Le graphe est sans circuit*
- d) Cas général

Problème du plus court chemin

G sans circuit \rightarrow ordre topologique



$$\lambda^*(s) = 0.$$

$$\lambda^*(1) = 0 + 3 = \lambda^*(s) + d(s,1)$$

$$\lambda^*(2) = \min\{3+2, 0+4\} = \min\{\lambda^*(s) + d(s,2), \lambda^*(1) + d(1,2)\}$$

...

$$\lambda^*(s) = 0.$$

$$\lambda^*(i) = \min\{\lambda^*(j) + d(i,j) : j \text{ prédécesseur de } i\}$$

(algorithme de Bellman)

Problème du plus court chemin

Résolution : dépend des données.

- a) La distance de chaque arc est 1 : BFS
- b) La distance de chaque arc est positive (ou nulle)
- c) Le graphe est sans circuit
- d) *Cas général*

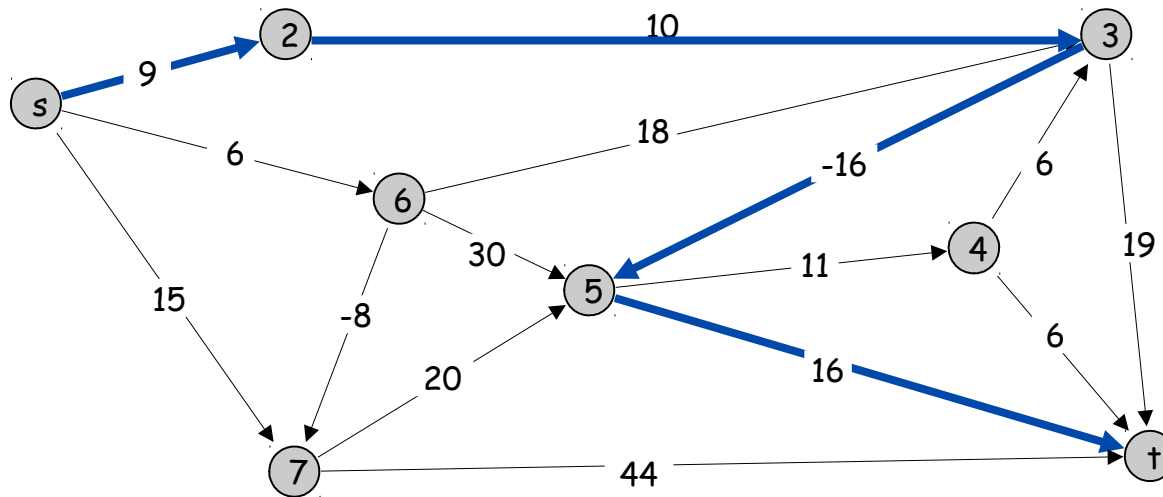
Plus court chemin avec des poids négatifs

Plus court chemin.

- Graphe orienté $G = (V, A)$.
- Source s , destination t .
- Longueur d_e = longueur de l'arc e .

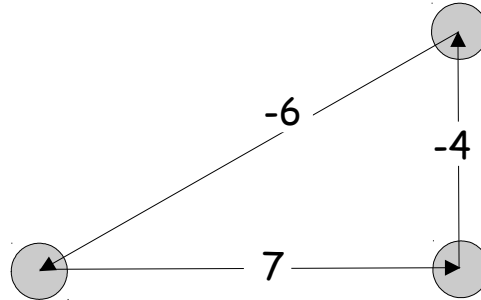
↑ autoriser des poids négatifs

Problème du plus court chemin : déterminer un plus court chemin de s à t .
(de s à tous les autres sommets)

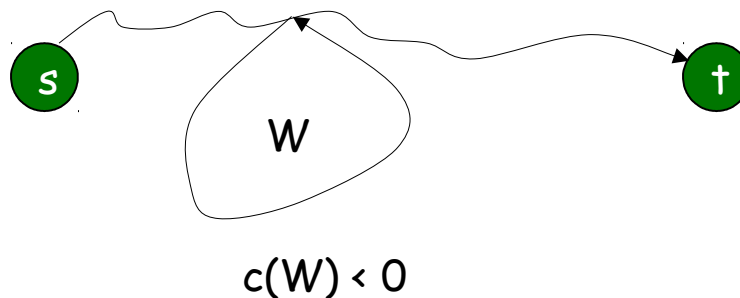


Plus court chemin : circuits absorbants

Circuit absorbant : circuit de coût strictement négatif.



Observation. Si un chemin de s à t contient un circuit absorbant, alors il n'existe pas de plus court chemin s - t ; sinon, il en existe un qui est simple (ne passe pas deux fois par le même sommet).

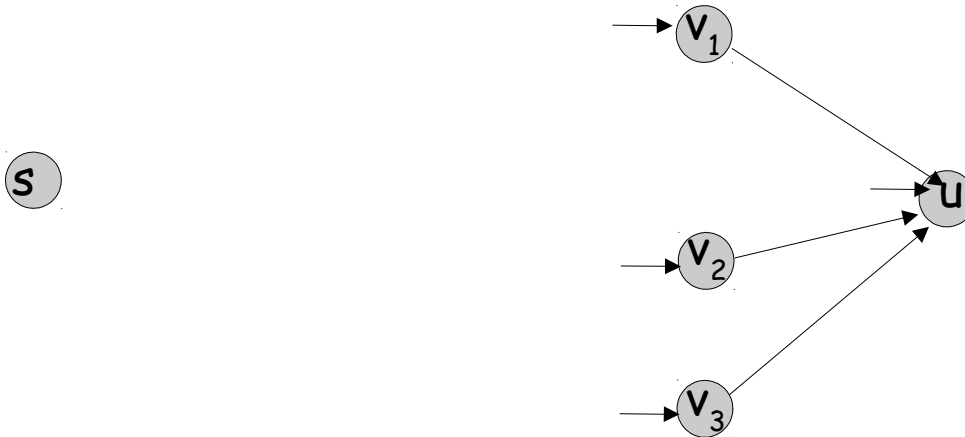


Plus court chemin : circuits absorbants

Algorithme de Bellman-Ford.

Idée : calculer $\lambda(k,u)$: valeur d'un PCCH de s à u qui emprunte au plus k arcs

- $\lambda(0,s)=0$
 $\lambda(0,u)=\infty$ pour u différent de s
- $\lambda(1,s)=0$
 $\lambda(1,u)=d(s,u)$ si arc (s,u) , $= \infty$ sinon.
- $\lambda(k+1,u)=\min\{ \lambda(k,u), \min\{\lambda(k,v)+d(v,u) : (v,u) \in A\} \}$



Algorithme de Bellman-Ford

Calcule un plus court chemin de s vers tout $v \in S$ ou déclare l'existence d'un cycle de poids négatif.

Algorithme de Bellman-Ford

pour chaque sommet u du graphe

$$\lambda(0, u) = +\infty$$

$$\lambda(0, s) = 0$$

pour k de 1 à n faire

pour tout sommet u faire

$$\lambda(k+1, u) = \min\{ \lambda(k, u), \min\{ \lambda(k, v) + d(v, u) : (v, u) \in A \} \}$$

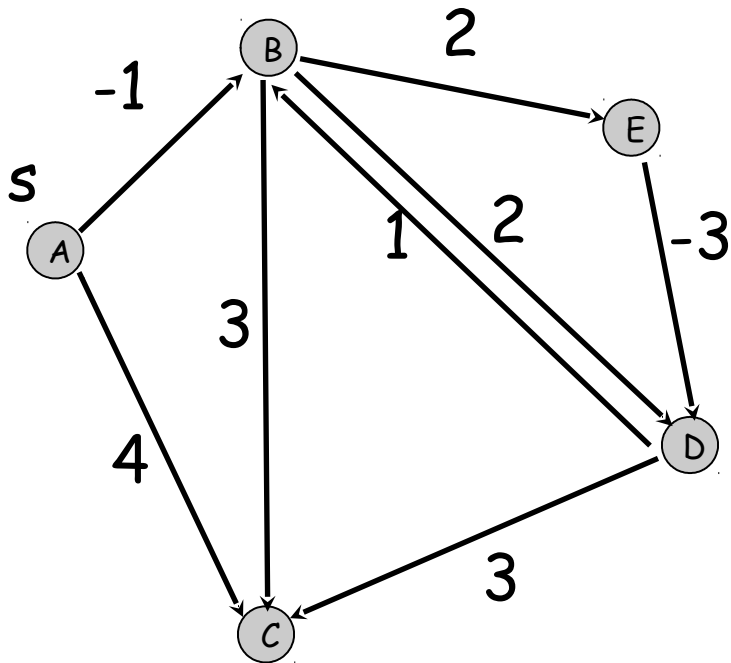
→ si $\lambda(n, v) < \lambda(n-1, v)$ pour un certain v : circuit absorbant dans le graphe

→ sinon $\lambda(n-1, u)$ = valeur d'un PCCH de s à u .

Complexité : $O(nm)$

Remarque : pour aller un peu plus vite : ne considérer à l'itération k que les successeurs des sommets dont la marque a évolué à l'itération $k-1$

Exemple



k	A	B	C	D	E
	0 -	∞ -	∞ -	∞ -	∞ -
1	0 -	-1 A	4 A	∞ -	∞ -
2	0 -	-1 A	2 B	1 B	1 B
3	0 -	-1 A	2 B	-2 E	1 B
4	0 -	-1 A	1 D	-2 E	1 B
5	0 -	-1 A	1 D	-2 E	1 B

Application : système de différences

$$Ax \leq b$$

avec pour chaque ligne de la matrice un 1, un -1 et tous les autres 0, càd

$$x_j - x_i \leq w_{ij}$$

Question : Existe-t-il une solution satisfaisant toutes les différences ?

Exemple

$$x_1 - x_2 \leq 3$$

$$x_2 - x_3 \leq -2$$

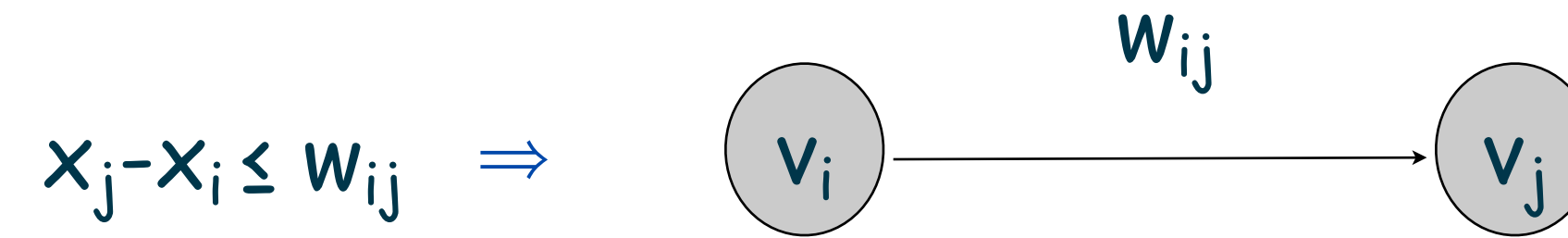
$$x_1 - x_3 \leq 2$$

Solution : $x_1=3$, $x_2=0$, $x_3=2$

Idée : Transformer le système des différences à un graphe

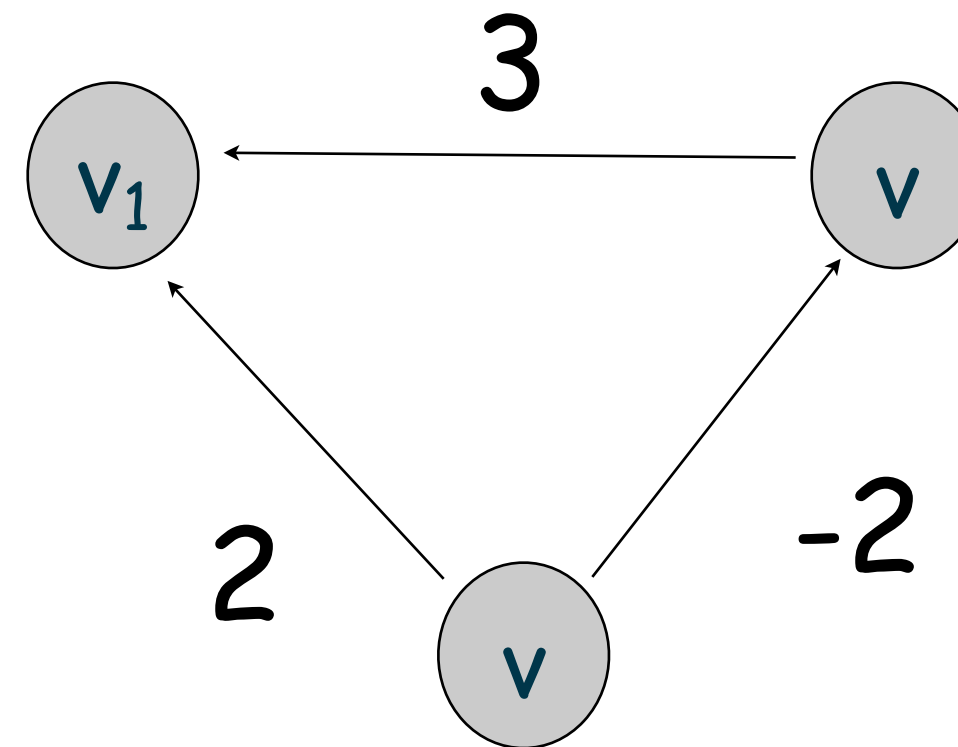
Application : système de différences

Graphe de contraintes



Exemple

$$\begin{aligned}x_1 - x_2 &\leq 3 \\x_2 - x_3 &\leq -2 \\x_1 - x_3 &\leq 2\end{aligned}$$



Observation : Il existe une relation étroite avec le plus court chemin.

$$x_j \leq x_i + w_{ij} \text{ ou } d(j) = d(i) + w(i, j)$$

Application : système de différences

Théorème. Si le graphe de contraintes a un cycle de poids négatif alors le système de différences est non-satisfiable (non réalisable).

Proof. Soit $v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow \dots \rightarrow v_k \rightarrow v_1$ un cycle de poids négatif.
Les arcs de ce cycle correspondent aux contraintes suivantes :

$$x_2 - x_1 \leq w_{12}$$

$$x_3 - x_2 \leq w_{23}$$

...

$$x_k - x_{k-1} \leq w_{k-1k}$$

$$x_1 - x_k \leq w_{k1}$$

En additionnant $0 \leq w(\text{cycle}) < 0$, contradiction

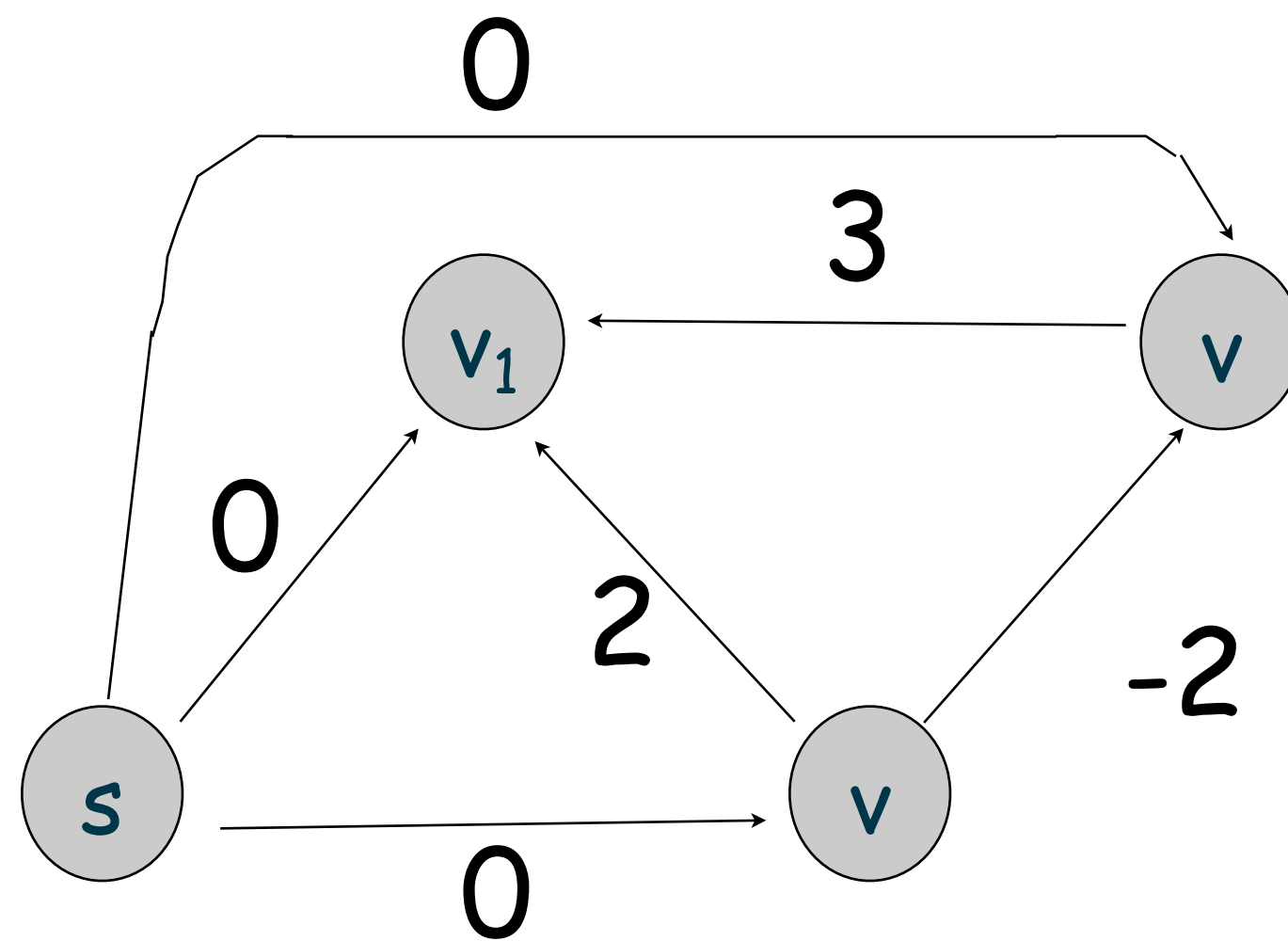
i.e. les contraintes ne sont pas satisfiables.

Application : système de différences

Théorème. Si il n'existe pas de cycle de poids négatif dans le graphe de contraintes G , alors le système de différence est satisfiable.

Preuve.

Rajouter à G , un sommet s et des arcs de poids 0 pour tout $v \in V$



Obs. Il n'existe pas de cycle de poids négatif et il existe de chemins de s vers tout $v \in V$.

Assertion. Il suffit de poser $x_i = d(v_i)$

Application : système de différences

Assertion. Il suffit de poser $x_i = d(v_i)$

Preuve. On veut $x_j - x_i \leq w_{ij}$ pour tout $(v_i, v_j) \in E$.

Ou de manière équivalente :

$$d(v_j) - d(v_i) \leq w(v_i, v_j)$$

$d(v_j) \leq d(v_i) + w(v_i, v_j)$ ce qui est vrai par l'inégalité triangulaire.

Corollaire. Bellman-Ford résout un système à m contraintes de différence sur n variables en temps $O(mn)$.