Graphes et langages (II)
Algorithmes sur un graphe valué

bruno.colombel@univ-amu.fr

DUT Informatique
IUT d'Aix-Marseille
Site d'Arles

2018 — 2019

Graphes pondérés

Recherche d'une plus courte chaîne

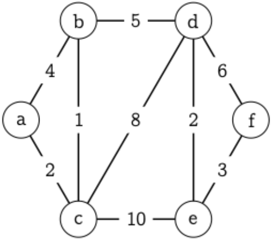
Arbre couvrant optimal

Flots et réseaux de transport

Sommaire

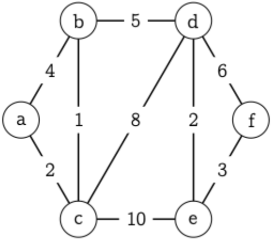
Graphes pondérés

C'est un graphe dont les arêtes sont affectées d'un « poids ».



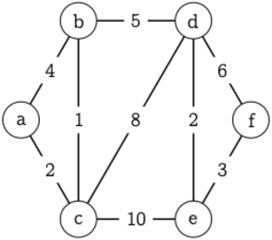
Graphes pondérés

C'est un graphe dont les arêtes sont affectées d'un « poids ».



Graphes pondérés

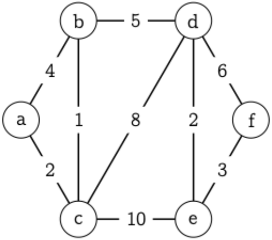
C'est un graphe dont les arêtes sont affectées d'un « poids ».



- on peut penser à un réseau routier,

Graphes pondérés

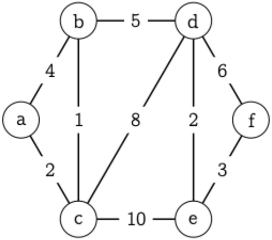
C'est un graphe dont les arêtes sont affectées d'un « poids ».



- on peut penser à un réseau routier,
  - les sommets étant des villes,
  - les arêtes des routes entre ces villes pondérées par la distance séparant les deux villes sommets de l'arête.

Graphes pondérés

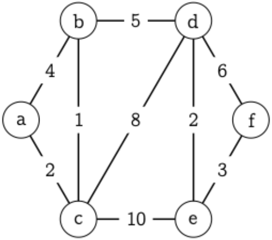
C'est un graphe dont les arêtes sont affectées d'un « poids ».



- on peut penser à un réseau routier,
  - les sommets étant des villes,
  - les arêtes des routes entre ces villes pondérées par la distance séparant les deux villes sommets de l'arête.
- On peut alors chercher :

Graphes pondérés

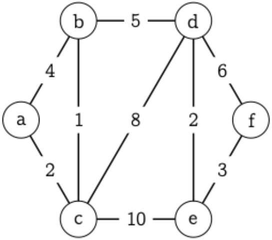
C'est un graphe dont les arêtes sont affectées d'un « poids ».



- on peut penser à un réseau routier,
  - les sommets étant des villes,
  - les arêtes des routes entre ces villes pondérées par la distance séparant les deux villes sommets de l'arête.
- On peut alors chercher :
  - le plus court chemin d'une ville à l'autre

Graphes pondérés

C'est un graphe dont les arêtes sont affectées d'un « poids ».



- on peut penser à un réseau routier,
  - les sommets étant des villes,
  - les arêtes des routes entre ces villes pondérées par la distance séparant les deux villes sommets de l'arête.
- On peut alors chercher :
  - le plus court chemin d'une ville à l'autre
  - le plus court chemin passant par tous les sommets (problème du voyageur de commerce)

Graphes pondérés

Définition

Un graphe pondéré est un quadruplet $\mathscr{G} = (\mathcal{S}; \mathcal{A}; \delta; p)$ où :

- $(\mathcal{S}; \mathcal{A}; \delta)$ est un graphe ;

Graphes pondérés

Définition

Un graphe pondéré est un quadruplet $\mathscr{G} = (\mathcal{S}; \mathcal{A}; \delta; p)$ où :

- $(\mathcal{S}; \mathcal{A}; \delta)$ est un graphe ;
- p est une fonction dite de *poids*

$$p : \mathcal{A} \to \mathbb{R}$$

Graphes pondérés

Définition

Dans un graphe $\mathscr{G}$, le poids d'une chaine est la somme du poids des arêtes qui la compose.

Graphes pondérés

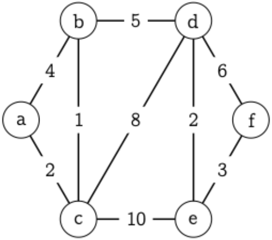
Définition

Dans un graphe $\mathscr{G}$, le poids d'une chaine est la somme du poids des arêtes qui la compose.

Exemple

Le poids de la chaine est $a - c - d$ est :

$$8 + 2 = 10$$



Graphes pondérés

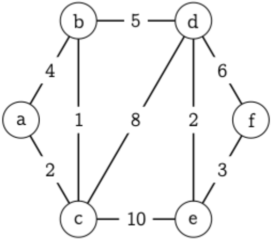
Définition

On appelle *matrice de pondération* d'un graphe $\mathscr{G}$ la matrice dont les coefficients correspondant aux sommets s et t valent :

$$A = \begin{cases} 0 & \text{si } t = s \\ \infty & \text{si } \{s, t\} \text{ n'est pas une arête} \\ p & \text{si } \{s, t\} \text{ est une arête de poids } p \end{cases}$$

Exemple

$$\begin{pmatrix} 0 & 4 & 2 & \infty & \infty & \infty \\ 4 & 0 & 1 & 5 & \infty & \infty \\ 2 & 1 & 0 & 8 & 10 & \infty \\ \infty & 5 & 8 & 0 & 2 & 6 \\ \infty & \infty & 10 & 2 & 0 & 3 \\ \infty & \infty & \infty & 6 & 3 & 0 \end{pmatrix}$$



Sommaire

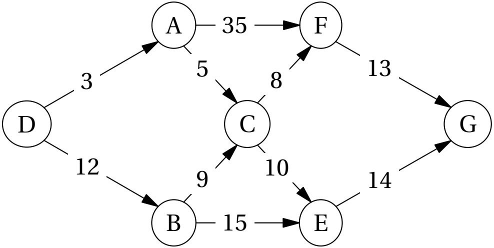
Algorithme de Dijkstra

L'algorithme de Dijkstra répond au problème du plus court chemin dans la cas où les poids sont positifs.

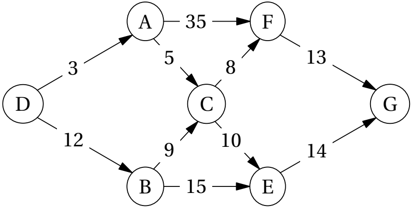
Regardons un exemple.

Algorithme de Dijkstra

On cherche à déterminer la plus courte chaîne entre les sommets D et G.



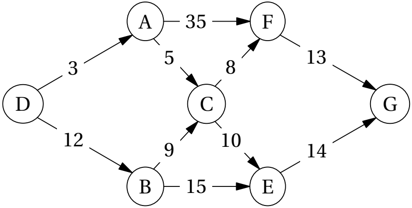
Algorithme de Dijkstra



Initialisation : Dans un tableau :

- Sur la première ligne, on écrit les sommets du graphes , en commençant par le sommet de départ D.

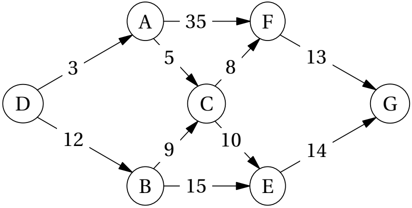
Algorithme de Dijkstra



Initialisation : Dans un tableau :

- Sur la première ligne, on écrit les sommets du graphes , en commençant par le sommet de départ D.
- Sur la deuxième ligne, on écrit la marque des sommets
  - On affecte en rouge le coefficient 0 à D ; le départ est ainsi fixé

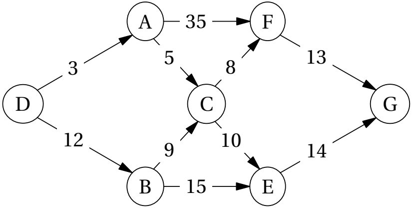
Algorithme de Dijkstra



Initialisation : Dans un tableau :

- Sur la première ligne, on écrit les sommets du graphes , en commençant par le sommet de départ D.
- Sur la deuxième ligne, on écrit la marque des sommets
  - On affecte en rouge le coefficient 0 à D ; le départ est ainsi fixé
  - Pour chaque sommet adjacent à D, on marque le poids de l'arête qui le relie à D et on indique le sommet de provenance D

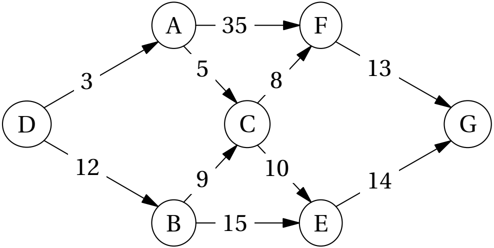
Algorithme de Dijkstra



Initialisation : Dans un tableau :

- Sur la première ligne, on écrit les sommets du graphes , en commençant par le sommet de départ D.
- Sur la deuxième ligne, on écrit la marque des sommets
  - On affecte en rouge le coefficient 0 à D ; le départ est ainsi fixé
  - Pour chaque sommet adjacent à D, on marque le poids de l'arête qui le relie à D et on indique le sommet de provenance D
  - On marque le poids ∞ à tous les autres sommets.

Algorithme de Dijkstra



D	A	B	C	E	F	G	sommet sélectionné
0	3(D)	12(D)	∞	∞	∞	∞	D

Itération

Itération

- On sélectionne le sommet X ayant le plus petit poids non marqué en rouge (ici A)

Itération

- On sélectionne le sommet X ayant le plus petit poids non marqué en rouge (ici A)
  - Si le sommet Y n'est pas adjacent au sommet X, on recopie la marque précédente du sommet Y.

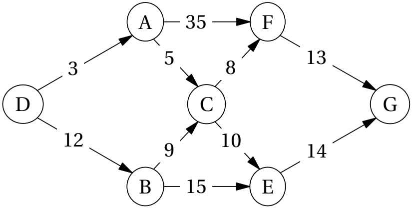
Itération

- On sélectionne le sommet X ayant le plus petit poids non marqué en rouge (ici A)
  - Si le sommet Y n'est pas adjacent au sommet X, on recopie la marque précédente du sommet Y.
  - Si le sommet Y est adjacent à X, on ajoute au poids de X. Lorsque cette somme est plus petite que la marque précédente de Y, on la remplace par la marque améliorée suivie du sommet de provenance.

Itération

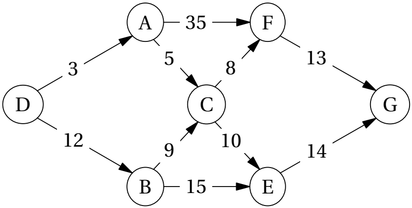
- On sélectionne le sommet X ayant le plus petit poids non marqué en rouge (ici A)
  - Si le sommet Y n'est pas adjacent au sommet X, on recopie la marque précédente du sommet Y.
  - Si le sommet Y est adjacent à X, on ajoute au poids de X. Lorsque cette somme est plus petite que la marque précédente de Y, on la remplace par la marque améliorée suivie du sommet de provenance.
  - Lorsqu'un sommet est fixé, on ne note plus rien sur sa colonne. On peut l'indiquer en tirant une double barre dans le reste de la colonne.

Algorithme de Dijkstra



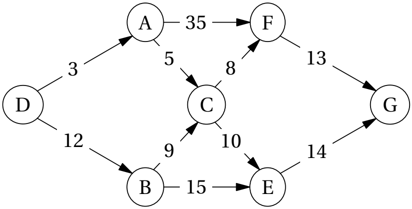
D	A	B	C	E	F	G
0	3, D	12, D	∞	∞	∞	∞

Algorithme de Dijkstra



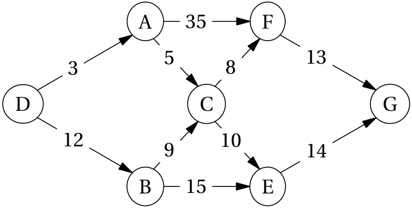
D	A	B	C	E	F	G
0	3, D	12, D	∞	∞	∞	∞
	3,D	12, D	8, A	∞	38, A	∞

Algorithme de Dijkstra



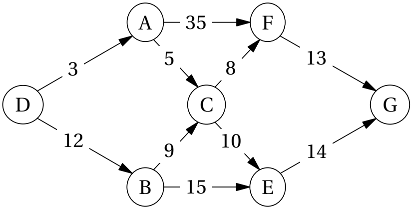
D	A	B	C	E	F	G
0	3, D	12, D	∞	∞	∞	∞
	3,D	12, D	8, A	∞	38, A	∞
		12, D	8, A	18, C	16, C	∞

Algorithme de Dijkstra



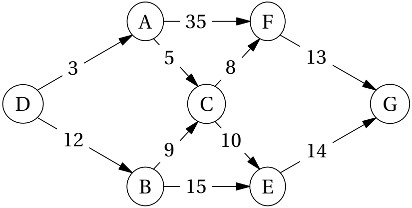
D	A	B	C	E	F	G
0	3, D	12, D	∞	∞	∞	∞
	3,D	12, D	8, A	∞	38, A	∞
		12, D	8, A	18, C	16, C	∞
		12, D		18, C	16, C	∞

Algorithme de Dijkstra



D	A	B	C	E	F	G
0	3, D	12, D	∞	∞	∞	∞
	3,D	12, D	8, A	∞	38, A	∞
		12, D	8, A	18, C	16, C	∞
		12, D		18, C	16, C	∞
				18, C	16, C	29, F

Algorithme de Dijkstra



D	A	B	C	E	F	G
0	3, D	12, D	∞	∞	∞	∞
	3,D	12, D	8, A	∞	38, A	∞
		12, D	8, A	18, C	16, C	∞
		12, D		18, C	16, C	∞
				18, C	16, C	29, F
				18,C		29, F

Algorithme de Dijkstra

D	A	B	C	E	F	G
0	3, D	12, D	∞	∞	∞	∞
	3,D	12, D	8, A	∞	38, A	∞
		12, D	8, A	18, C	16, C	∞
		12, D		18, C	16, C	∞
				18, C	16, C	29, F
				18,C		29, F

Algorithme de Dijkstra

D	A	B	C	E	F	G
0	3, D	12, D	∞	∞	∞	∞
	3,D	12, D	8, A	∞	38, A	∞
		12, D	8, A	18, C	16, C	∞
		12, D		18, C	16, C	∞
				18, C	16, C	29, F
				18,C		29, F

Pour lire la chaîne la plus courte, on part de la fin de parcours et on « remonte » la chaîne suivant les sommets de provenance

Algorithme de Dijkstra

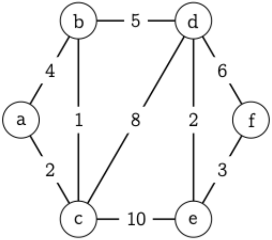
D	A	B	C	E	F	G
0	3, D	12, D	∞	∞	∞	∞
	3,D	12, D	8, A	∞	38, A	∞
		12, D	8, A	18, C	16, C	∞
		12, D		18, C	16, C	∞
				18, C	16, C	29, F
				18,C		29, F

Pour lire la chaîne la plus courte, on part de la fin de parcours et on « remonte » la chaîne suivant les sommets de provenance

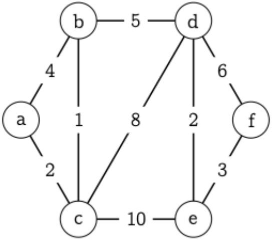
$$D - A - C - F - G \text{ de poids } 29$$

Algorithme de Dijkstra

Déterminer le plus court chemin entre a et f

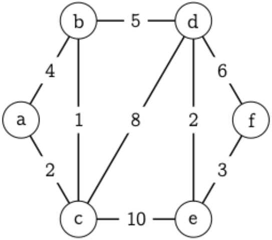


Algorithme de Dijkstra



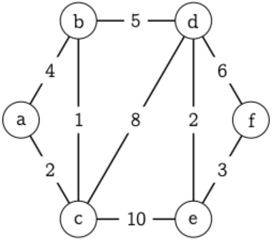
sommet	b	c	d	e	f
a	$4,a$	$2,a$	∞	∞	∞

Algorithme de Dijkstra



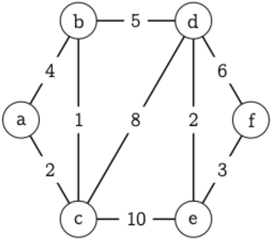
sommet	b	c	d	e	f
a	$4,a$	$2,a$	∞	∞	∞
c	3,c		$10,c$	$12,\ c$	∞

Algorithme de Dijkstra



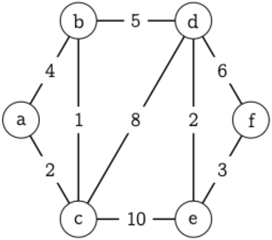
sommet	b	c	d	e	f
a	$4,a$	$2,a$	∞	∞	∞
c	$3,c$		$10,c$	$12,\ c$	∞
b			$8,\ b$	$12,\ c$	∞

Algorithme de Dijkstra



sommet	b	c	d	e	f
a	$4,a$	$2,a$	∞	∞	∞
c	3,c		$10,c$	$12,\ c$	∞
b			$8,\ b$	$12,\ c$	∞
d				$10,\ d$	$14,\ d$

Algorithme de Dijkstra



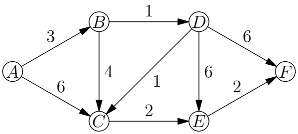
sommet	b	c	d	e	f
a	$4,a$	$2,a$	∞	∞	∞
c	3,c		$10,c$	$12,\ c$	∞
b			$8,\ b$	$12,\ c$	∞
d				$10,\ d$	$14,\ d$
e					$13,\ e$

Exemple : Open Short Path First

Dans le protocole de routage

Open Short Path First (OSPF)

chaque routeur mémorise ses voisins directs en envoyant aux autres routeurs des requêtes régulières

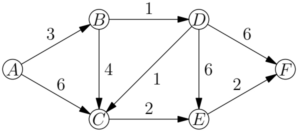


Exemple : Open Short Path First

Dans le protocole de routage

Open Short Path First (OSPF)

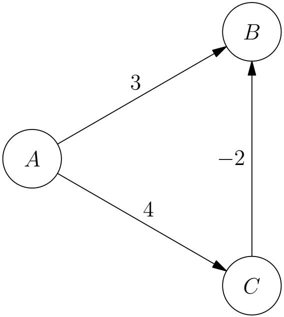
chaque routeur mémorise ses voisins directs en envoyant aux autres routeurs des requêtes régulières



Le chemin le plus court entre deux routeurs est calculé avec l'algorithme de Dijkstra

Algorithme de Dijkstra : limites

Et si les poids sont négatifs ?

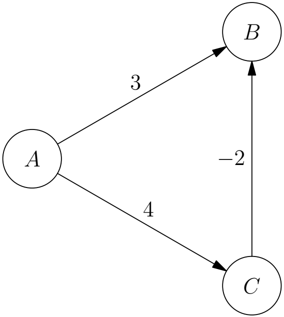


Algorithme de Dijkstra : limites

Et si les poids sont négatifs ?

La plus courte chaîne du sommet A au sommet B est manifestement :

$$A \to C \to B \text{ de poids } 2$$

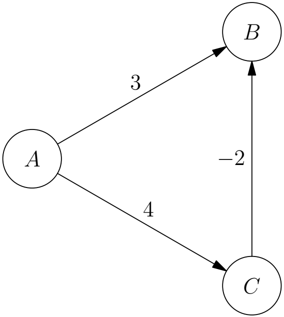


Algorithme de Dijkstra : limites

Et si les poids sont négatifs ?

La plus courte chaîne du sommet A au sommet B est manifestement :

$$A \to C \to B \text{ de poids } 2$$



L'algorithme de Dijkstra donne :

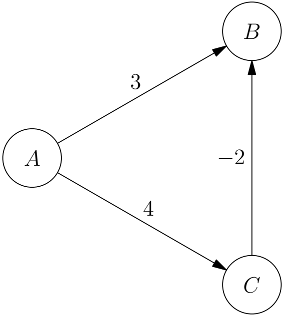
Sommet	B	C
A	3, A	4, A
B	–	4, A

Algorithme de Dijkstra : limites

Et si les poids sont négatifs ?

La plus courte chaîne du sommet A au sommet B est manifestement :

$$A \to C \to B \text{ de poids } 2$$



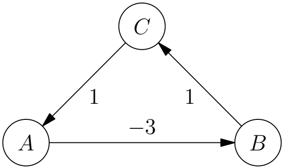
L'algorithme de Dijkstra donne :

Sommet	B	C
A	3, A	4, A
B	—	4, A

L'algorithme de Dijkstra ne donne pas la bonne réponse !

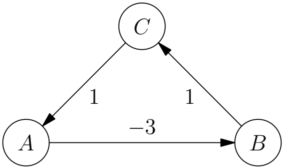
Algorithme de Dijkstra : limites

Et si les poids sont négatifs ?



Algorithme de Dijkstra : limites

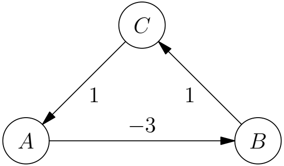
Et si les poids sont négatifs ?



- Le circuit $A \to B \to C \to A$ a un poids négatif : c'est un *circuit négatif*

Algorithme de Dijkstra : limites

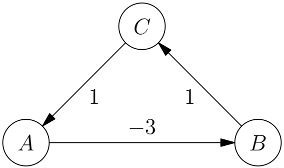
Et si les poids sont négatifs ?



- Le circuit $A \to B \to C \to A$ a un poids négatif : c'est un *circuit négatif*
  Le problème de la plus courte chaîne n'a pas de solution ici !

Algorithme de Dijkstra : limites

Et si les poids sont négatifs ?



- Le circuit $A \to B \to C \to A$ a un poids négatif : c'est un *circuit négatif*
  Le problème de la plus courte chaîne n'a pas de solution ici !
- Il est possible de diminuer indéfiniment le poids d'une chaîne en utilisant ce circuit

Algorithme de Bellman-Ford

On cherche un algorithme qui :

Algorithme de Bellman-Ford

On cherche un algorithme qui :

- recherche les plus courtes chaîne entre deux sommets . . .

Algorithme de Bellman-Ford

On cherche un algorithme qui :

- recherche les plus courtes chaîne entre deux sommets ...
- ... mais qui permet l'utilisation de poids négatifs

Algorithme de Bellman-Ford

On cherche un algorithme qui :

- recherche les plus courtes chaîne entre deux sommets ...
- ... mais qui permet l'utilisation de poids négatifs
- détecte les circuits négatifs

Algorithme de Bellman-Ford

On cherche un algorithme qui :

- recherche les plus courtes chaîne entre deux sommets ...
- ... mais qui permet l'utilisation de poids négatifs
- détecte les circuits négatifs

Cet algorithme est l'algorithme de **Bellman-Ford**

Algorithme de Bellman-Ford

Principe :

- Le principe est le même que pour l'algorithme de Dijkstra

Algorithme de Bellman-Ford

Principe :

- Le principe est le même que pour l'algorithme de Dijkstra
- mais les sommets ne sont plus marqués

Algorithme de Bellman-Ford

Principe :

- Le principe est le même que pour l'algorithme de Dijkstra
- mais les sommets ne sont plus marqués
  - il est possible de revenir sur certains sommets jusqu'à la fin de l'algorithme

Principe :

- Le principe est le même que pour l'algorithme de Dijkstra
- mais les sommets ne sont plus marqués
  - il est possible de revenir sur certains sommets jusqu'à la fin de l'algorithme
  - avec des poids positifs, le poids total ne peut qu'augmenter. Ce n'est pas le cas ici

Algorithme de Bellman-Ford

Arrêt :

Arrêt :

1. Nombre d'itérations maximal : ordre du graphe

Algorithme de Bellman-Ford

Arrêt :

1. Nombre d'itérations maximal : ordre du graphe
2. aucune valeur n'est modifiée entre deux itérations

Arrêt :

1. Nombre d'itérations maximal : ordre du graphe
2. aucune valeur n'est modifiée entre deux itérations

Propriété

Si $\mathcal{G}$ est d'ordre n, et si les valeurs sont encore modifiées après n étapes, alors :

Algorithme de Bellman-Ford

Arrêt :

1. Nombre d'itérations maximal : ordre du graphe
2. aucune valeur n'est modifiée entre deux itérations

Propriété

Si $\mathcal{G}$ est d'ordre n, et si les valeurs sont encore modifiées après n étapes, alors :
il existe un cycle négatif et il est inutile de continuer

Arrêt :

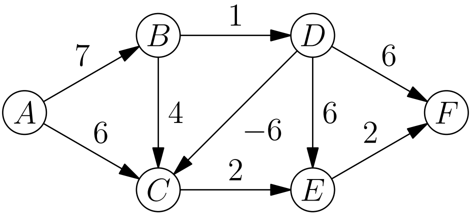
1. Nombre d'itérations maximal : ordre du graphe
2. aucune valeur n'est modifiée entre deux itérations

Propriété

Si $\mathcal{G}$ est d'ordre n, et si les valeurs sont encore modifiées après n étapes, alors :
il existe un cycle négatif et il est inutile de continuer

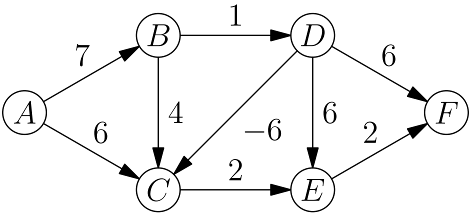
Regardons un exemple

Algorithme de Bellman-Ford



Au plus 6 étapes

Algorithme de Bellman-Ford

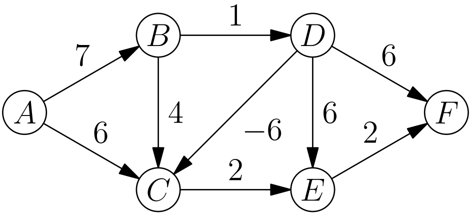


Au plus 6 étapes

La procédure d'initialisation est la même que dans l'algorithme de Dijkstra :

A	B	C	D	E	F
$(0, A)$	$(7, A)$	$(6, A)$	∞	∞	∞

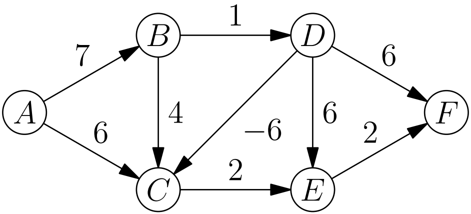
Algorithme de Bellman-Ford



Aucun sommet n'est marqué : on regarde où on peut aller à partir de A et de B :

A	B	C	D	E	F
$(0, A)$	$(7, A)$	$(6, A)$	∞	∞	∞

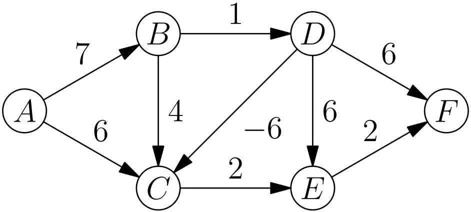
Algorithme de Bellman-Ford



Aucun sommet n'est marqué : on regarde où on peut aller à partir de A et de B :

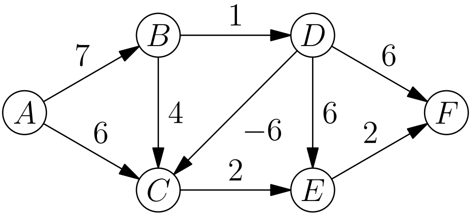
A	B	C	D	E	F
$(0, A)$	$(7, A)$	$(6, A)$	∞	∞	∞
$(0, A)$	$(7, A)$	$(6, A)$	$(8, B)$	$(8, C)$	∞

Algorithme de Bellman-Ford



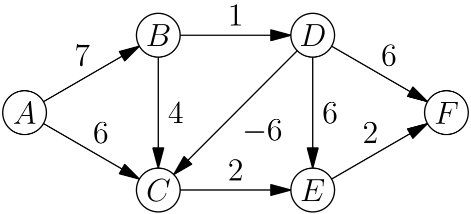
A	B	C	D	E	F
$(0, A)$	$(7, A)$	$(6, A)$	∞	∞	∞
$(0, A)$	$(7, A)$	$(6, A)$	$(8, B)$	$(8, C)$	∞

Algorithme de Bellman-Ford



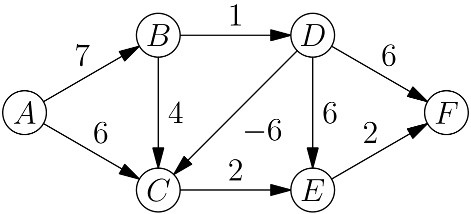
A	B	C	D	E	F
$(0, A)$	$(7, A)$	$(6, A)$	∞	∞	∞
$(0, A)$	$(7, A)$	$(6, A)$	$(8, B)$	$(8, C)$	∞
$(0, A)$	$(7, A)$	$(2, D)$	$(8, B)$	$(8, C)$	$(10, E)$

Algorithme de Bellman-Ford



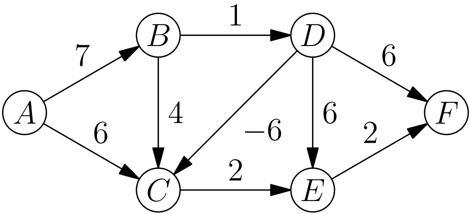
A	B	C	D	E	F
$(0, A)$	$(7, A)$	$(6, A)$	∞	∞	∞
$(0, A)$	$(7, A)$	$(6, A)$	$(8, B)$	$(8, C)$	∞
$(0, A)$	$(7, A)$	$(2, D)$	$(8, B)$	$(8, C)$	$(10, E)$
$(0, A)$	$(7, A)$	$(2, D)$	$(8, B)$	$(4, C)$	$(10, E)$

Algorithme de Bellman-Ford



A	B	C	D	E	F
$(0, A)$	$(7, A)$	$(6, A)$	∞	∞	∞
$(0, A)$	$(7, A)$	$(6, A)$	$(8, B)$	$(8, C)$	∞
$(0, A)$	$(7, A)$	$(2, D)$	$(8, B)$	$(8, C)$	$(10, E)$
$(0, A)$	$(7, A)$	$(2, D)$	$(8, B)$	$(4, C)$	$(10, E)$
$(0, A)$	$(7, A)$	$(2, D)$	$(8, B)$	$(4, C)$	$(6, E)$

Algorithme de Bellman-Ford



A	B	C	D	E	F
$(0, A)$	$(7, A)$	$(6, A)$	∞	∞	∞
$(0, A)$	$(7, A)$	$(6, A)$	$(8, B)$	$(8, C)$	∞
$(0, A)$	$(7, A)$	$(2, D)$	$(8, B)$	$(8, C)$	$(10, E)$
$(0, A)$	$(7, A)$	$(2, D)$	$(8, B)$	$(4, C)$	$(10, E)$
$(0, A)$	$(7, A)$	$(2, D)$	$(8, B)$	$(4, C)$	$(6, E)$
$(0, A)$	$(7, A)$	$(2, D)$	$(8, B)$	$(4, C)$	$(6, E)$

Algorithme de Bellman-Ford

A	B	C	D	E	F
$(0, A)$	$(7, A)$	$(6, A)$	∞	∞	∞
$(0, A)$	$(7, A)$	$(6, A)$	$(8, B)$	$(8, C)$	∞
$(0, A)$	$(7, A)$	$(2, D)$	$(8, B)$	$(8, C)$	$(10, E)$
$(0, A)$	$(7, A)$	$(2, D)$	$(8, B)$	$(4, C)$	$(10, E)$
$(0, A)$	$(7, A)$	$(2, D)$	$(8, B)$	$(4, C)$	$(6, E)$
$(0, A)$	$(7, A)$	$(2, D)$	$(8, B)$	$(4, C)$	$(6, E)$

Algorithme de Bellman-Ford

A	B	C	D	E	F
$(0, A)$	$(7, A)$	$(6, A)$	∞	∞	∞
$(0, A)$	$(7, A)$	$(6, A)$	$(8, B)$	$(8, C)$	∞
$(0, A)$	$(7, A)$	$(2, D)$	$(8, B)$	$(8, C)$	$(10, E)$
$(0, A)$	$(7, A)$	$(2, D)$	$(8, B)$	$(4, C)$	$(10, E)$
$(0, A)$	$(7, A)$	$(2, D)$	$(8, B)$	$(4, C)$	$(6, E)$
$(0, A)$	$(7, A)$	$(2, D)$	$(8, B)$	$(4, C)$	$(6, E)$

- L'algorithme s'est terminé en 5 étapes

Algorithme de Bellman-Ford

A	B	C	D	E	F
$(0, A)$	$(7, A)$	$(6, A)$	∞	∞	∞
$(0, A)$	$(7, A)$	$(6, A)$	$(8, B)$	$(8, C)$	∞
$(0, A)$	$(7, A)$	$(2, D)$	$(8, B)$	$(8, C)$	$(10, E)$
$(0, A)$	$(7, A)$	$(2, D)$	$(8, B)$	$(4, C)$	$(10, E)$
$(0, A)$	$(7, A)$	$(2, D)$	$(8, B)$	$(4, C)$	$(6, E)$
$(0, A)$	$(7, A)$	$(2, D)$	$(8, B)$	$(4, C)$	$(6, E)$

- L'algorithme s'est terminé en 5 étapes
- il n'y a donc pas de circuit négatif

Algorithme de Bellman-Ford

Remarque :

- Avec Dijkstra, le sommet C aurait été marqué à la 1^{re} étape

Algorithme de Bellman-Ford

Remarque :

- Avec Dijkstra, le sommet C aurait été marqué à la 1re étape
- On aurait obtenu le résultat (faux) :

A	B	C	D	E	F
$(0, A)$	$(7, A)$	$(6, A)$	$(8, B)$	$(8, C)$	$(10, E)$

Algorithme de Bellman-Ford

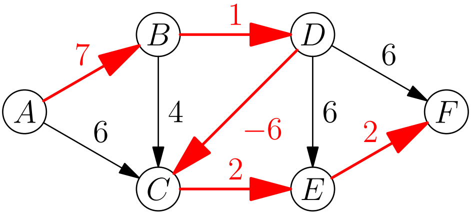
Le chemin le plus court est déduit comme dans l'algorithme de Dijkstra :

A	B	C	D	E	F
$(0, A)$	$(7, A)$	$(2, D)$	$(8, B)$	$(4, C)$	$(6, E)$

Algorithme de Bellman-Ford

Le chemin le plus court est déduit comme dans l'algorithme de Dijkstra :

A	B	C	D	E	F
$(0, A)$	$(7, A)$	$(2, D)$	$(8, B)$	$(4, C)$	$(6, E)$



Plus court chemin

Les algorithmes de Dijkstra et Bellman-Ford :

Plus court chemin

Les algorithmes de Dijkstra et Bellman-Ford :

- servent à déterminer le plus court chemin d'un sommet à tous les autres

Plus court chemin

Les algorithmes de Dijkstra et Bellman-Ford :

- servent à déterminer le plus court chemin d'un sommet à tous les autres
- pour des graphes simples pondérés
  - Pondérations positives pour Dijkstra

Plus court chemin

Les algorithmes de Dijkstra et Bellman-Ford :

- servent à déterminer le plus court chemin d'un sommet à tous les autres
- pour des graphes simples pondérés
  - Pondérations positives pour Dijkstra
  - Pondérations quelconques pour Bellman-Ford

Plus court chemin

Les algorithmes de Dijkstra et Bellman-Ford :

- servent à déterminer le plus court chemin d'un sommet à tous les autres
- pour des graphes simples pondérés
  - Pondérations positives pour Dijkstra
  - Pondérations quelconques pour Bellman-Ford

Variantes

Plus court chemin

Les algorithmes de Dijkstra et Bellman-Ford :

- servent à déterminer le plus court chemin d'un sommet à tous les autres
- pour des graphes simples pondérés
  - Pondérations positives pour Dijkstra
  - Pondérations quelconques pour Bellman-Ford

Variantes

- chemin le plus long :

Plus court chemin

Les algorithmes de Dijkstra et Bellman-Ford :

- servent à déterminer le plus court chemin d'un sommet à tous les autres
- pour des graphes simples pondérés
  - Pondérations positives pour Dijkstra
  - Pondérations quelconques pour Bellman-Ford

Variantes

- chemin le plus long : Inverser les signes

Plus court chemin

Les algorithmes de Dijkstra et Bellman-Ford :

- servent à déterminer le plus court chemin d'un sommet à tous les autres
- pour des graphes simples pondérés
  - Pondérations positives pour Dijkstra
  - Pondérations quelconques pour Bellman-Ford

Variantes

- chemin le plus long : Inverser les signes
- chemin le plus court de tous les sommets à tous les sommets :

Plus court chemin

Les algorithmes de Dijkstra et Bellman-Ford :

- servent à déterminer le plus court chemin d'un sommet à tous les autres
- pour des graphes simples pondérés
  - Pondérations positives pour Dijkstra
  - Pondérations quelconques pour Bellman-Ford

Variantes

- chemin le plus long : Inverser les signes
- chemin le plus court de tous les sommets à tous les sommets : Appliquer les algorithmes à tous les sommets

Sommaire

Arbre couvrant optimal

Soit $\mathscr{G} = (\mathcal{S}, \mathcal{A})$ un graphe.

Définition (Graphe partiel)

Un graphe partiel de $\mathcal{G}$ est un graphe $\mathscr{G}' = (\mathcal{S}; \mathcal{A}')$ avec

$$\mathcal{S}' = \mathcal{S} \quad \text{et} \quad \mathcal{A}' \subset \mathcal{A}$$

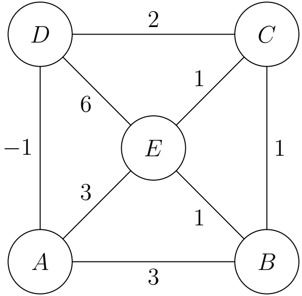
Arbre couvrant optimal

Soit $\mathscr{G} = (\mathcal{S}, \mathcal{A})$ un graphe.

Définition (Graphe partiel)

Un graphe partiel de $\mathcal{G}$ est un graphe $\mathscr{G}' = (\mathcal{S}; \mathcal{A}')$ avec

$$\mathcal{S}' = \mathcal{S} \quad \text{et} \quad \mathcal{A}' \subset \mathcal{A}$$



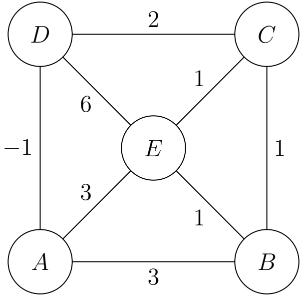
Arbre couvrant optimal

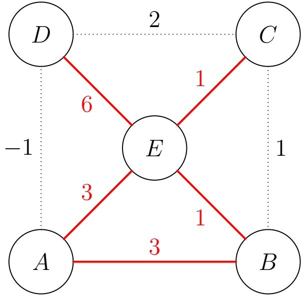
Soit $\mathscr{G} = (\mathcal{S}, \mathcal{A})$ un graphe.

Définition (Graphe partiel)

Un graphe partiel de $\mathcal{G}$ est un graphe $\mathscr{G}' = (\mathcal{S}; \mathcal{A}')$ avec

$$\mathcal{S}' = \mathcal{S} \quad \text{et} \quad \mathcal{A}' \subset \mathcal{A}$$





Arbre couvrant optimal

Définition (Arbre)

Un *arbre* est un graphe connexe non orienté sans circuit

Arbre couvrant optimal

Définition (Arbre)

Un *arbre* est un graphe connexe non orienté sans circuit

Définition (Arbre de recouvrement)

Un arbre de recouvrement de $\mathscr{G}$ est un graphe partiel de $\mathscr{G}$ qui est un arbre

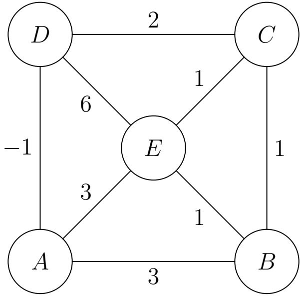
Arbre couvrant optimal

Définition (Arbre)

Un *arbre* est un graphe connexe non orienté sans circuit

Définition (Arbre de recouvrement)

Un arbre de recouvrement de $\mathscr{G}$ est un graphe partiel de $\mathscr{G}$ qui est un arbre

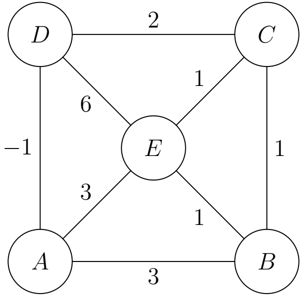


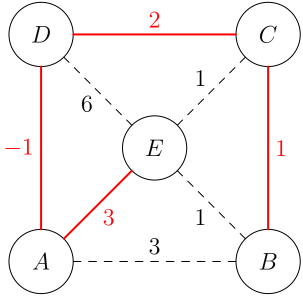
Définition (Arbre)

Un *arbre* est un graphe connexe non orienté sans circuit

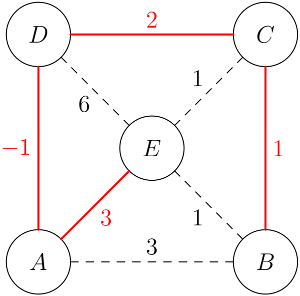
Définition (Arbre de recouvrement)

Un arbre de recouvrement de $\mathscr{G}$ est un graphe partiel de $\mathscr{G}$ qui est un arbre



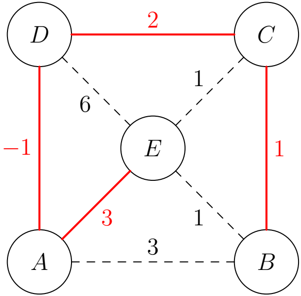


Arbre couvrant optimal



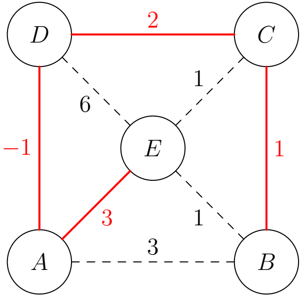
- Le poids de l'arbre de recouvrement est $-1 + 2 + 1 + 3 = 5$

Arbre couvrant optimal



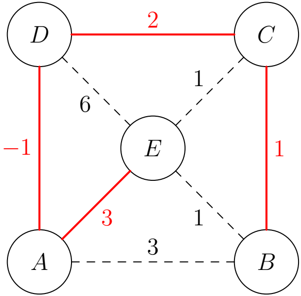
- Le poids de l'arbre de recouvrement est $-1 + 2 + 1 + 3 = 5$
- On cherche une méthode pour déterminer un arbre de recouvrement de poids minimal ou maximal

Arbre couvrant optimal



- Le poids de l'arbre de recouvrement est $-1+2+1+3=5$
- On cherche une méthode pour déterminer un arbre de recouvrement de poids minimal ou maximal
- Algorithme de Kruskal qui opère sur les arcs

Arbre couvrant optimal



- Le poids de l'arbre de recouvrement est $-1+2+1+3=5$
- On cherche une méthode pour déterminer un arbre de recouvrement de poids minimal ou maximal
- Algorithme de Kruskal qui opère sur les arcs
- Algorithme de Prim qui opère sur les sommets

Algorithme de Kruskal

L'algorithme de Kruskal consiste

Algorithme de Kruskal

L'algorithme de Kruskal consiste

- à balayer les arêtes triées dans l'ordre

Algorithme de Kruskal

L'algorithme de Kruskal consiste

- à balayer les arêtes triées dans l'ordre
  - croissant pour un arbre de poids minimal

Algorithme de Kruskal

L'algorithme de Kruskal consiste

- à balayer les arêtes triées dans l'ordre
  - croissant pour un arbre de poids minimal
  - décroissant pour un arbre de poids maximal

Algorithme de Kruskal

L'algorithme de Kruskal consiste

- à balayer les arêtes triées dans l'ordre
  - croissant pour un arbre de poids minimal
  - décroissant pour un arbre de poids maximal
- à sélectionner l'arête si les sommets ne sont pas déjà connectés

Algorithme de Kruskal

L'algorithme de Kruskal consiste

- à balayer les arêtes triées dans l'ordre
  - croissant pour un arbre de poids minimal
  - décroissant pour un arbre de poids maximal
- à sélectionner l'arête si les sommets ne sont pas déjà connectés
- bien sur sans créer de cycle

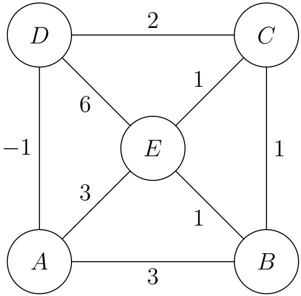
Algorithme de Kruskal

L'algorithme de Kruskal consiste

- à balayer les arêtes triées dans l'ordre
  - croissant pour un arbre de poids minimal
  - décroissant pour un arbre de poids maximal
- à sélectionner l'arête si les sommets ne sont pas déjà connectés
- bien sur sans créer de cycle

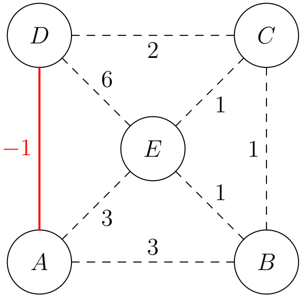
Regardons sur un exemple comment obtenir un arbre de recouvrement minimal

Algorithme de Kruskal

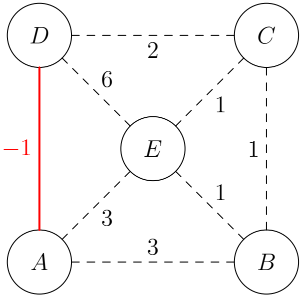


On commence par choisir l'arête de poids la plus faible

Algorithme de Kruskal

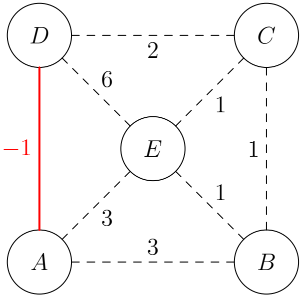


Algorithme de Kruskal



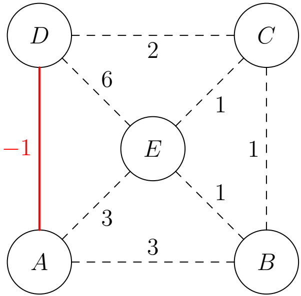
- Il y a plusieurs solutions pour le choix de la seconde arête

Algorithme de Kruskal



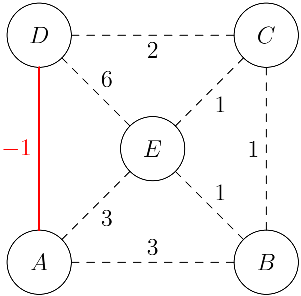
- Il y a plusieurs solutions pour le choix de la seconde arête
- Quelque soit le choix on aboutira à une solution

Algorithme de Kruskal



- Il y a plusieurs solutions pour le choix de la seconde arête
- Quelque soit le choix on aboutira à une solution
- En général, le choix est clair d'après le contexte

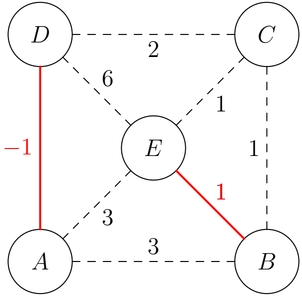
Algorithme de Kruskal



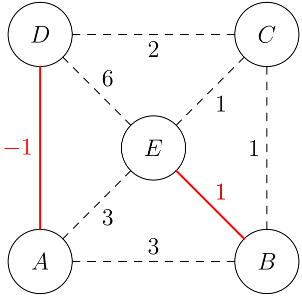
- Il y a plusieurs solutions pour le choix de la seconde arête
- Quelque soit le choix on aboutira à une solution
- En général, le choix est clair d'après le contexte

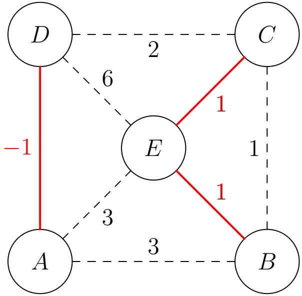
C'est l'heuristique !

Algorithme de Kruskal

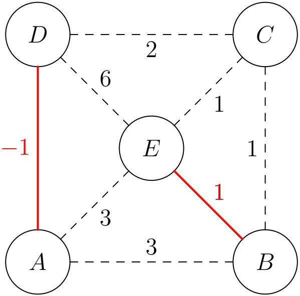


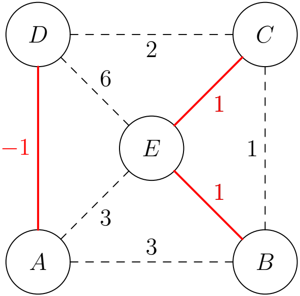
Algorithme de Kruskal





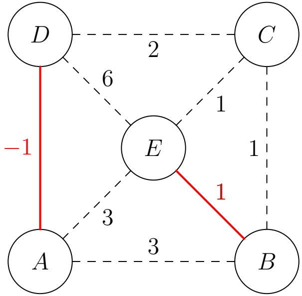
Algorithme de Kruskal

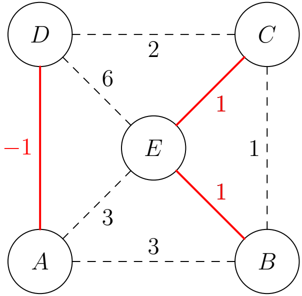




- On ne peut pas choisir l'arête $B - -C$ sinon nous aurions un cycle

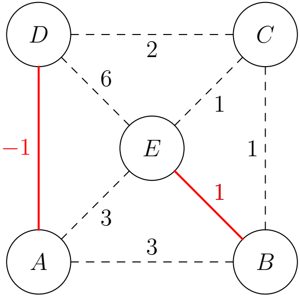
Algorithme de Kruskal

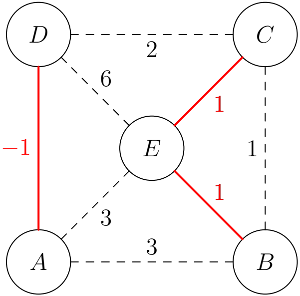




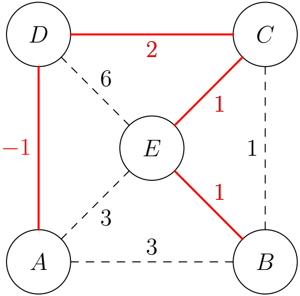
- On ne peut pas choisir l'arête $B - -C$ sinon nous aurions un cycle
- On choisit l'arête $D - -C$

Algorithme de Kruskal





- On ne peut pas choisir l'arête $B--C$ sinon nous aurions un cycle
- On choisit l'arête $D--C$

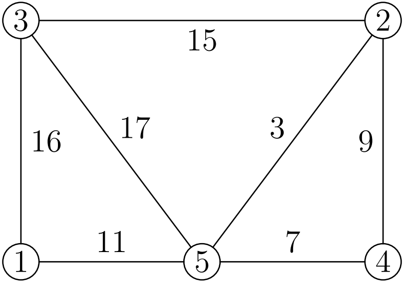


Algorithme de Kruskal

Entrées : $\mathscr{G} = (\mathcal{S}, \mathcal{A}, p)$: graphe pondéré
début
    $E \leftarrow \emptyset$;
    pour $e \in \mathcal{A}$ **faire**
        si $E \cup \{e\}$ *n'est pas un cycle* **alors**
            $E \leftarrow E \cup \{e\}$
        fin
    fin
    Retourner $(\mathcal{S}, E)$;
fin

Algorithme de Kruskal

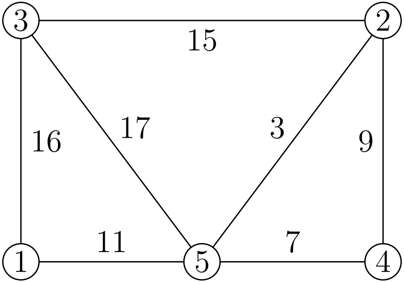
Arbre couvrant de poids maximal avec l'algorithme de Kruskal



Algorithme de Kruskal

Arbre couvrant de poids maximal avec l'algorithme de Kruskal

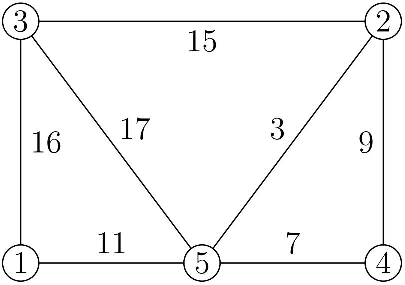
- arête $5, 3$
  $\Longrightarrow$ poids $= 0 + 17$



Algorithme de Kruskal

Arbre couvrant de poids maximal avec l'algorithme de Kruskal

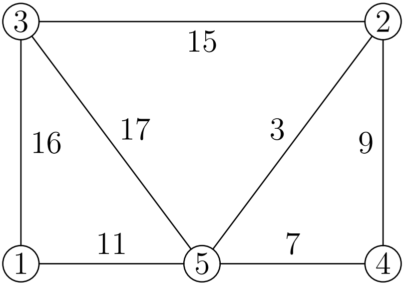
- arête $5, 3$
  $\implies$ poids $= 0 + 17$
- arête $3, 1$
  $\implies$ poids $= 17 + 16$



Algorithme de Kruskal

Arbre couvrant de poids maximal avec l'algorithme de Kruskal

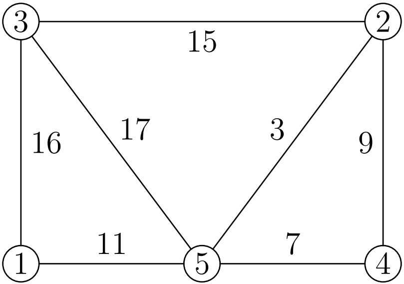
- arête $5, 3$
  $\implies$ poids $= 0 + 17$
- arête $3, 1$
  $\implies$ poids $= 17 + 16$
- arête $2, 3$
  $\implies$ poids $= 33 + 15$



Algorithme de Kruskal

Arbre couvrant de poids maximal avec l'algorithme de Kruskal

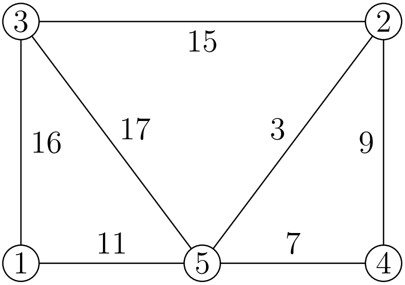
- arête $5, 3$
  $\Longrightarrow$ poids $= 0 + 17$
- arête $3, 1$
  $\Longrightarrow$ poids $= 17 + 16$
- arête $2, 3$
  $\Longrightarrow$ poids $= 33 + 15$
- arête $1, 5$:
  1 et 5 déjà connectés



Algorithme de Kruskal

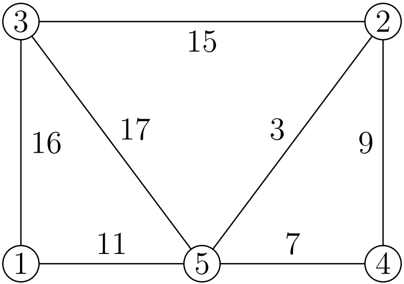
Arbre couvrant de poids maximal avec l'algorithme de Kruskal

- arête $5, 3$
  $\implies$ poids $= 0 + 17$
- arête $3, 1$
  $\implies$ poids $= 17 + 16$
- arête $2, 3$
  $\implies$ poids $= 33 + 15$
- arête $1, 5$:
  1 et 5 déjà connectés
- arête $4, 2$
  $\implies$ poids $= 48 + 9$



Arbre couvrant de poids maximal avec l'algorithme de Kruskal

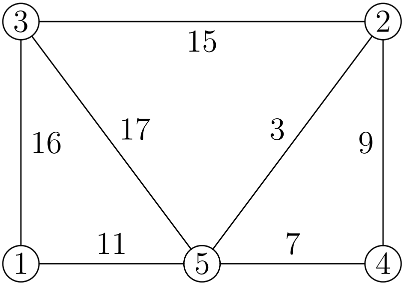
- arête $5, 3$
  $\Longrightarrow$ poids $= 0 + 17$
- arête $3, 1$
  $\Longrightarrow$ poids $= 17 + 16$
- arête $2, 3$
  $\Longrightarrow$ poids $= 33 + 15$
- arête $1, 5$:
  1 et 5 déjà connectés
- arête $4, 2$
  $\Longrightarrow$ poids $= 48 + 9$
- arête $5, 4$: 5 et 4 déjà connectés



Algorithme de Kruskal

Arbre couvrant de poids maximal avec l'algorithme de Kruskal

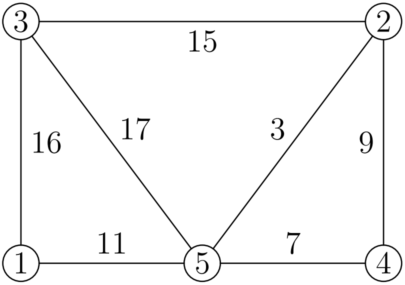
- arête $5, 3$
  $\implies$ poids $= 0 + 17$
- arête $3, 1$
  $\implies$ poids $= 17 + 16$
- arête $2, 3$
  $\implies$ poids $= 33 + 15$
- arête $1, 5$:
  1 et 5 déjà connectés
- arête $4, 2$
  $\implies$ poids $= 48 + 9$
- arête $5, 4$: 5 et 4 déjà connectés
- arête $2, 5$: 2 et 5 déjà connectés



Algorithme de Kruskal

Arbre couvrant de poids maximal avec l'algorithme de Kruskal

- arête $5, 3$
  $\implies$ poids $= 0 + 17$
- arête $3, 1$
  $\implies$ poids $= 17 + 16$
- arête $2, 3$
  $\implies$ poids $= 33 + 15$
- arête $1, 5$:
  1 et 5 déjà connectés
- arête $4, 2$
  $\implies$ poids $= 48 + 9$

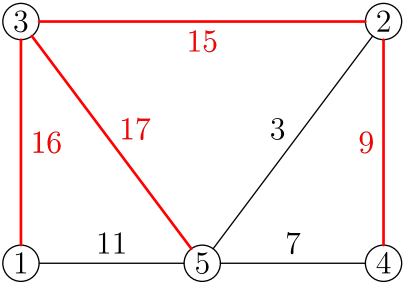


- arête $5, 4$: 5 et 4 déjà connectés
- arête $2, 5$: 2 et 5 déjà connectés
- Poids total de l'arbre : 57

Algorithme de Kruskal

Arbre couvrant de poids maximal avec l'algorithme de Kruskal

- arête $5, 3$
  $\Longrightarrow$ poids $= 0 + 17$
- arête $3, 1$
  $\Longrightarrow$ poids $= 17 + 16$
- arête $2, 3$
  $\Longrightarrow$ poids $= 33 + 15$
- arête $1, 5$:
  1 et 5 déjà connectés
- arête $4, 2$
  $\Longrightarrow$ poids $= 48 + 9$
- arête $5, 4$: 5 et 4 déjà connectés
- arête $2, 5$: 2 et 5 déjà connectés
- Poids total de l'arbre : 57



Algorithme de Prim

- L'algorithme de Prim est un algorithme glouton

Algorithme de Prim

- L'algorithme de Prim est un algorithme glouton
- Il fonctionne sur un graphe connexe non orienté et pondéré

Algorithme de Prim

- L'algorithme de Prim est un algorithme glouton
- Il fonctionne sur un graphe connexe non orienté et pondéré

Principe :

Algorithme de Prim

- L'algorithme de Prim est un algorithme glouton
- Il fonctionne sur un graphe connexe non orienté et pondéré

Principe :

- On part d'un sommet

Algorithme de Prim

- L'algorithme de Prim est un algorithme glouton
- Il fonctionne sur un graphe connexe non orienté et pondéré

Principe :

- On part d'un sommet
- À chaque étape,

Algorithme de Prim

- L'algorithme de Prim est un algorithme glouton
- Il fonctionne sur un graphe connexe non orienté et pondéré

Principe :

- On part d'un sommet
- À chaque étape,
  - on ajoute une arête de poids minimum adjacente à l'arbre en construction

Algorithme de Prim

- L'algorithme de Prim est un algorithme glouton
- Il fonctionne sur un graphe connexe non orienté et pondéré

Principe :

- On part d'un sommet
- À chaque étape,
  - on ajoute une arête de poids minimum adjacente à l'arbre en construction
  - on recommence jusqu'à ce que l'arbre « recouvre » le graphe

Algorithme de Prim

- L'algorithme de Prim est un algorithme glouton
- Il fonctionne sur un graphe connexe non orienté et pondéré

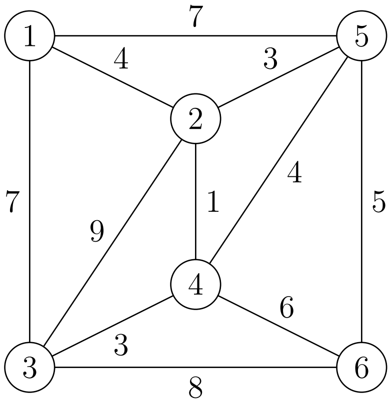
Principe :

- On part d'un sommet
- À chaque étape,
  - on ajoute une arête de poids minimum adjacente à l'arbre en construction
  - on recommence jusqu'à ce que l'arbre « recouvre » le graphe

Regardons un exemple

Algorithme de Prim

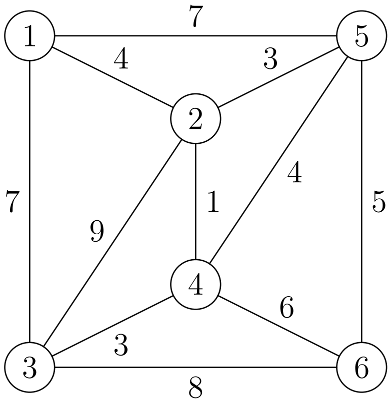
Arbre couvrant de poids minimal avec l'algorithme de Prim



Algorithme de Prim

Arbre couvrant de poids minimal avec l'algorithme de Prim

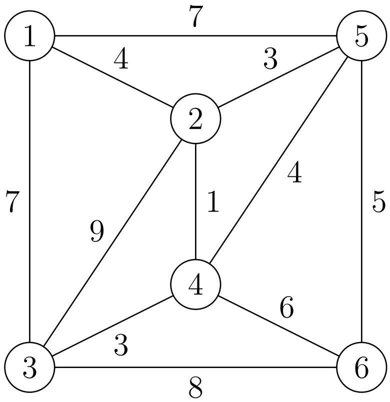
- Sommet marqué : 1



Algorithme de Prim

Arbre couvrant de poids minimal avec l'algorithme de Prim

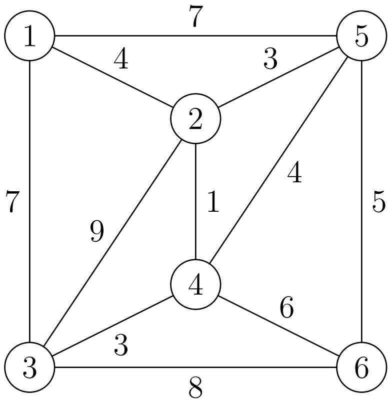
- Sommet marqué : 1
- arêtes sortantes : $\{1;2\}$, $\{1;3\}$, $\{1;5\}$



Algorithme de Prim

Arbre couvrant de poids minimal avec l'algorithme de Prim

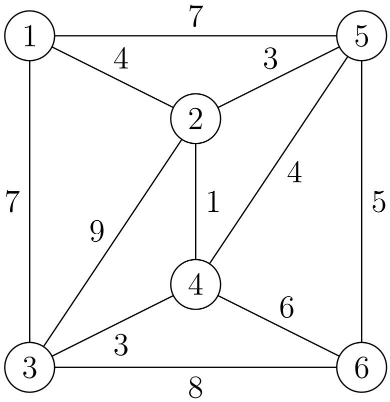
- Sommet marqué : 1
- arêtes sortantes : $\{1; 2\}$, $\{1; 3\}$, $\{1; 5\}$



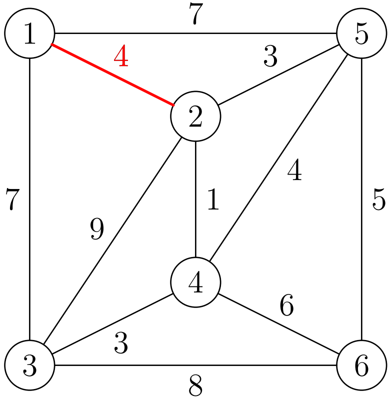
Algorithme de Prim

Arbre couvrant de poids minimal avec l'algorithme de Prim

- Sommet marqué : 1
- arêtes sortantes : $\{1; 2\}$, $\{1; 3\}$, $\{1; 5\}$
- Poids total : 4



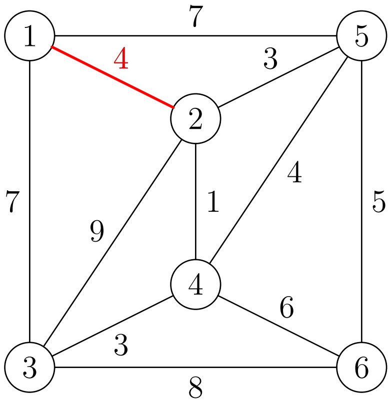
Arbre couvrant de poids minimal avec l'algorithme de Prim



Algorithme de Prim

Arbre couvrant de poids minimal avec l'algorithme de Prim

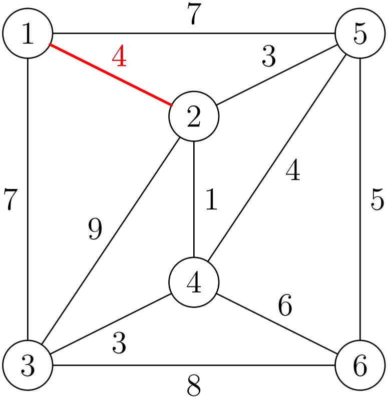
- Sommets marqués : 1, 2



Algorithme de Prim

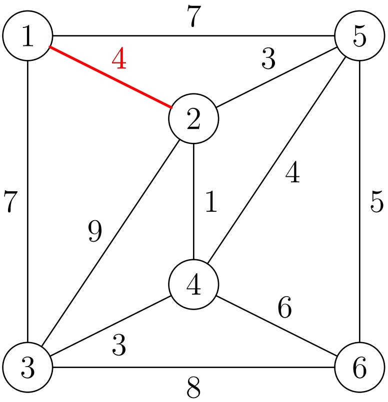
Arbre couvrant de poids minimal avec l'algorithme de Prim

- Sommets marqués : 1, 2
- arêtes sortantes : $\{1;3\}$, $\{1;5\}$, $\{2;3\}$, $\{2;5\}$, $\{2;4\}$



Arbre couvrant de poids minimal avec l'algorithme de Prim

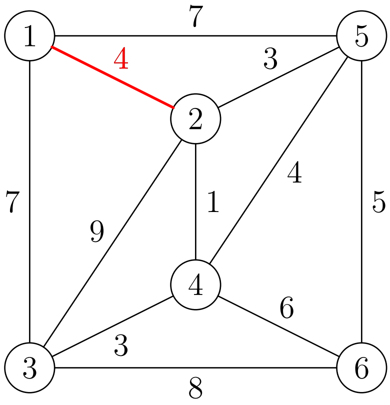
- Sommets marqués : 1, 2
- arêtes sortantes : $\{1; 3\}$, $\{1; 5\}$, $\{2; 3\}$, $\{2; 5\}$, $\{2; 4\}$



Algorithme de Prim

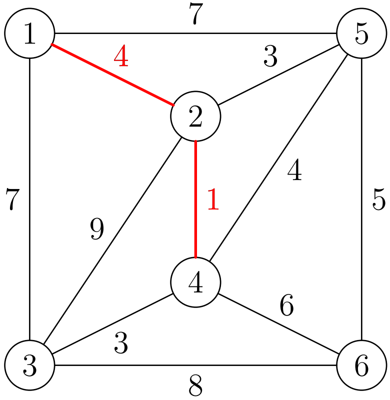
Arbre couvrant de poids minimal avec l'algorithme de Prim

- Sommets marqués : 1, 2
- arêtes sortantes : $\{1;3\}$, $\{1;5\}$, $\{2;3\}$, $\{2;5\}$, $\{2;4\}$
- Poids total : $4 + 1 = 5$



Algorithme de Prim

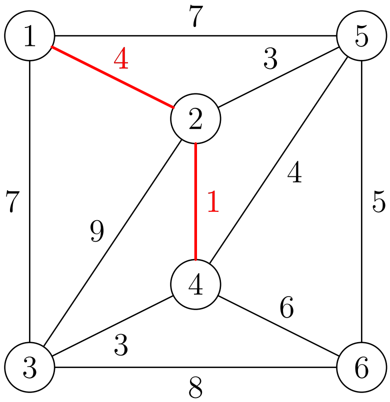
Arbre couvrant de poids minimal avec l'algorithme de Prim



Algorithme de Prim

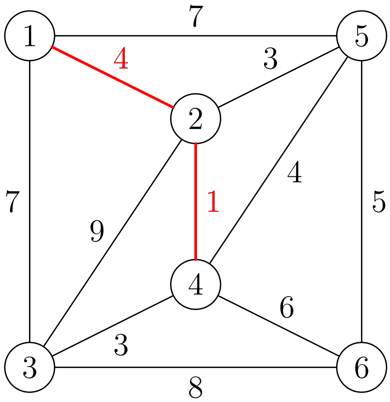
Arbre couvrant de poids minimal avec l'algorithme de Prim

- Sommets marqués : 1, 2, 4



Arbre couvrant de poids minimal avec l'algorithme de Prim

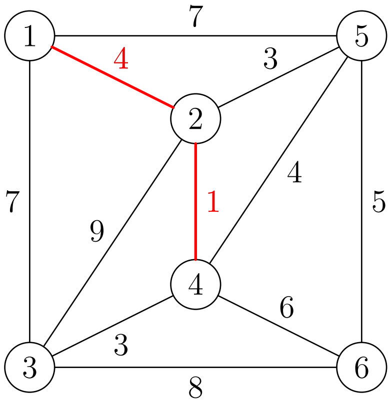
- Sommets marqués : 1, 2, 4
- arêtes sortantes : $\{1;3\}$, $\{2;3\}$, $\{1;5\}$, $\{2;5\}$, $\{4,3\}$, $\{4;5\}$, $\{4;6\}$



Algorithme de Prim

Arbre couvrant de poids minimal avec l'algorithme de Prim

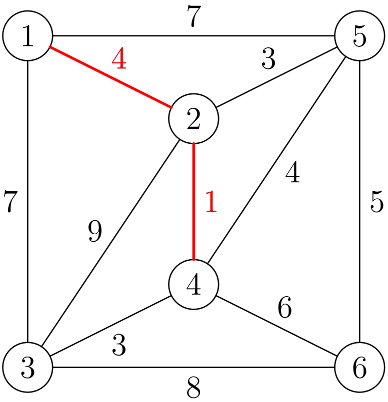
- Sommets marqués : 1, 2, 4
- arêtes sortantes : $\{1;3\}$, $\{2;3\}$, $\{1;5\}$, $\{2;5\}$, $\{4,3\}$, $\{4;5\}$, $\{4;6\}$



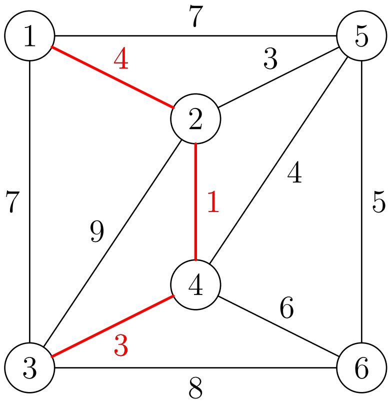
Algorithme de Prim

Arbre couvrant de poids minimal avec l'algorithme de Prim

- Sommets marqués : 1, 2, 4
- arêtes sortantes : $\{1;3\}$, $\{2;3\}$, $\{1;5\}$, $\{2;5\}$, $\{4,3\}$, $\{4;5\}$, $\{4;6\}$
- Poids total : $5 + 3 = 8$



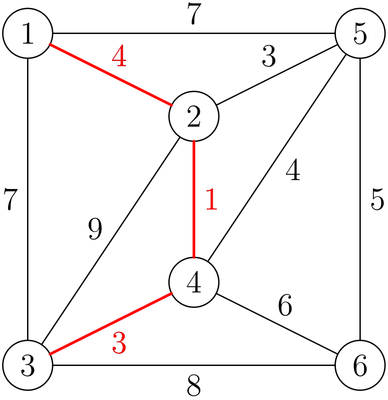
Arbre couvrant de poids minimal avec l'algorithme de Prim



Algorithme de Prim

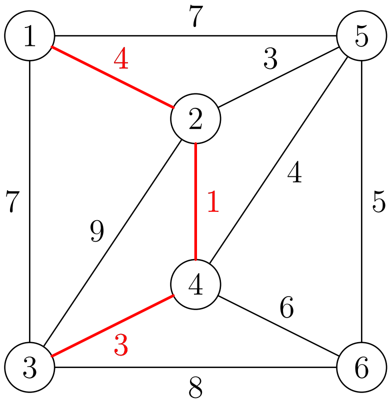
Arbre couvrant de poids minimal avec l'algorithme de Prim

- Sommets marqués : 1, 2, 4, 3



Arbre couvrant de poids minimal avec l'algorithme de Prim

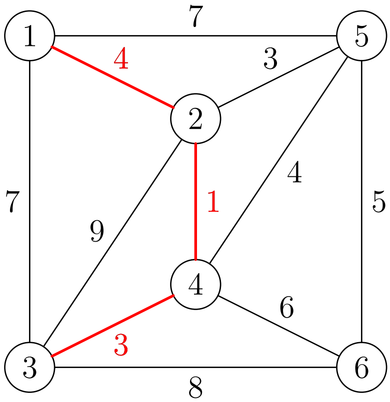
- Sommets marqués : 1, 2, 4, 3
- arêtes sortantes : $\{1;5\}$, $\{2;5\}$, $\{4;5\}$, $\{3,6\}$, $\{4;6\}$



Algorithme de Prim

Arbre couvrant de poids minimal avec l'algorithme de Prim

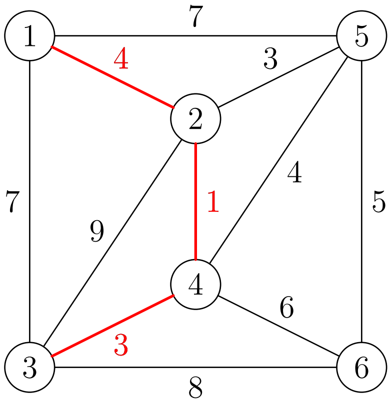
- Sommets marqués : 1, 2, 4, 3
- arêtes sortantes : $\{1;5\}$, $\{2;5\}$, $\{4;5\}$, $\{3,6\}$, $\{4;6\}$



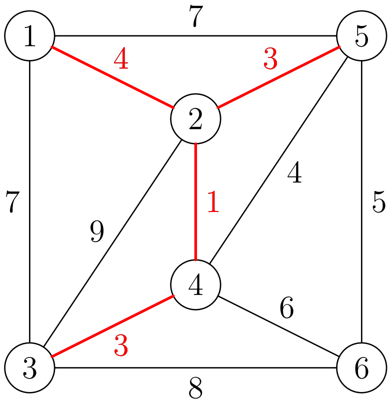
Algorithme de Prim

Arbre couvrant de poids minimal avec l'algorithme de Prim

- Sommets marqués : 1, 2, 4, 3
- arêtes sortantes : $\{1; 5\}$, $\{2; 5\}$, $\{4; 5\}$, $\{3, 6\}$, $\{4; 6\}$
- Poids total : $8 + 3 = 11$



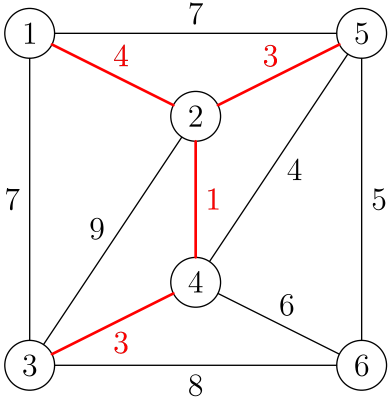
Arbre couvrant de poids minimal avec l'algorithme de Prim



Algorithme de Prim

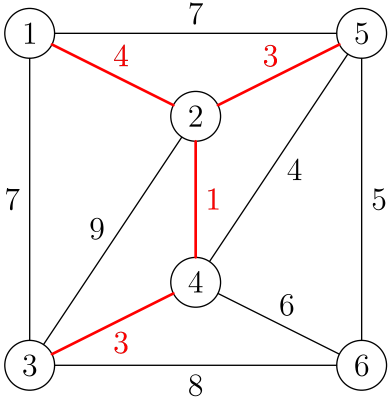
Arbre couvrant de poids minimal avec l'algorithme de Prim

- Sommets marqués : 1, 2, 4, 3



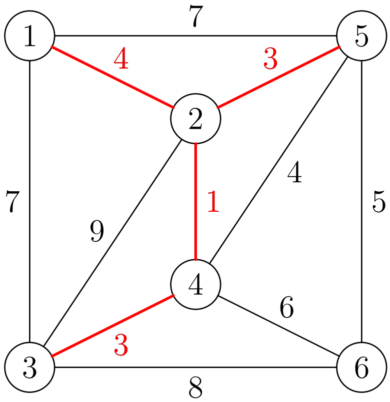
Arbre couvrant de poids minimal avec l'algorithme de Prim

- Sommets marqués : 1, 2, 4, 3
- arêtes sortantes : $\{3, 6\}$, $\{4; 6\}$, $\{5; 6\}$



Arbre couvrant de poids minimal avec l'algorithme de Prim

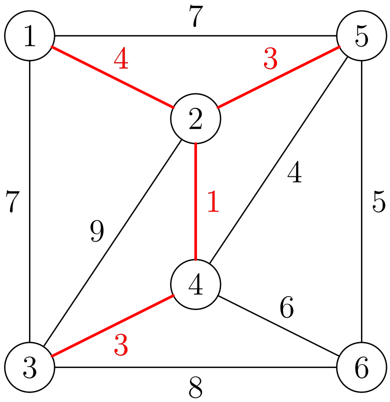
- Sommets marqués : 1, 2, 4, 3
- arêtes sortantes : $\{3,6\}$, $\{4;6\}$, $\{5;6\}$



Algorithme de Prim

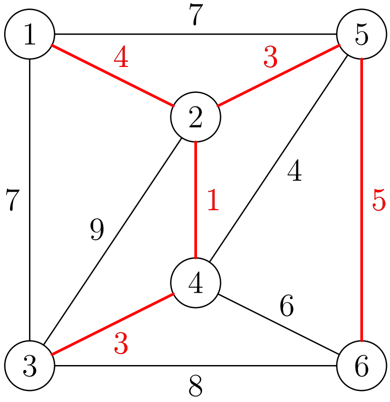
Arbre couvrant de poids minimal avec l'algorithme de Prim

- Sommets marqués : 1, 2, 4, 3
- arêtes sortantes : $\{3, 6\}$, $\{4; 6\}$, $\{5; 6\}$
- Poids total : $11 + 5 = 16$

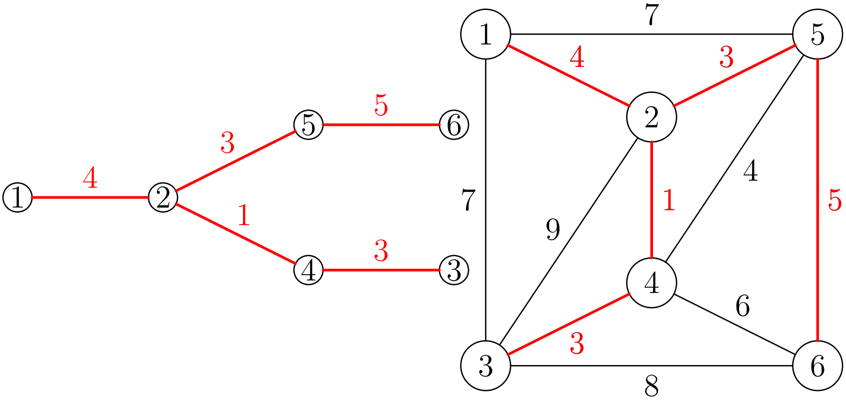


Algorithme de Prim

- Tous les sommets sont marqués
- Poids total de l'arbre : 16



Algorithme de Prim



Sommaire

- Les flots constituent une partie intéressante pour l'informatique

Flots et réseaux de transports

- Les flots constituent une partie intéressante pour l'informatique
- Ils sont utilisés dans la gestion des réseaux

Flots et réseaux de transports

- Les flots constituent une partie intéressante pour l'informatique
- Ils sont utilisés dans la gestion des réseaux
- mais également dans beaucoup d'autres domaines

Flots et réseaux de transports

- Les flots constituent une partie intéressante pour l'informatique
- Ils sont utilisés dans la gestion des réseaux
- mais également dans beaucoup d'autres domaines
- Par exemple, la gestion du trafic routier ...

Flots et réseaux de transports

- Les flots constituent une partie intéressante pour l'informatique
- Ils sont utilisés dans la gestion des réseaux
- mais également dans beaucoup d'autres domaines
- Par exemple, la gestion du trafic routier ...

Pour étudier ce problème il faut d'abord définir ce qu'est un réseau de transport.

Flots et réseaux de transports

Définition

Un *réseau de transport* est un graphe orienté et pondéré pour lequel :

Définition

Un *réseau de transport* est un graphe orienté et pondéré pour lequel :

1. les pondérations sont strictement positives ;

Définition

Un *réseau de transport* est un graphe orienté et pondéré pour lequel :

1. les pondérations sont strictement positives ;
2. il existe deux sommets particuliers :

Flots et réseaux de transports

Définition

Un *réseau de transport* est un graphe orienté et pondéré pour lequel :

1. les pondérations sont strictement positives ;
2. il existe deux sommets particuliers :
   - la *source* qui n'a pas de prédécesseur

Définition

Un *réseau de transport* est un graphe orienté et pondéré pour lequel :

1. les pondérations sont strictement positives ;
2. il existe deux sommets particuliers :
   - la *source* qui n'a pas de prédécesseur
   - le *puits* qui n'a pas de successeur

Définition

Un *réseau de transport* est un graphe orienté et pondéré pour lequel :

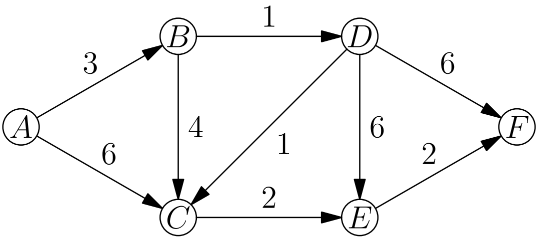
1. les pondérations sont strictement positives ;
2. il existe deux sommets particuliers :
   - la *source* qui n'a pas de prédécesseur
   - le *puits* qui n'a pas de successeur
3. le graphe non orienté induit est connexe

Flots et réseaux de transports

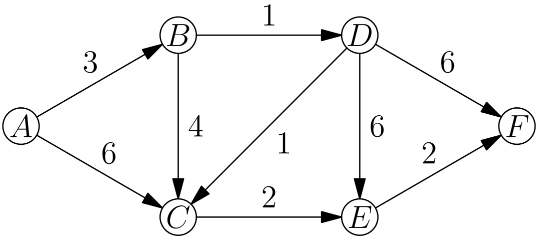
Définition

Un *réseau de transport* est un graphe orienté et pondéré pour lequel :

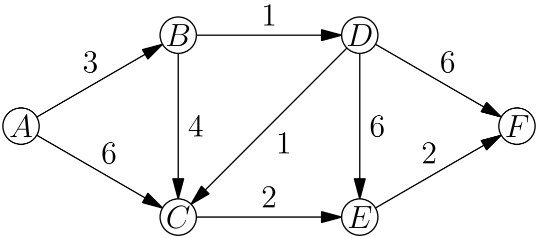
1. les pondérations sont strictement positives ;
2. il existe deux sommets particuliers :
   - la *source* qui n'a pas de prédécesseur
   - le *puits* qui n'a pas de successeur
3. le graphe non orienté induit est connexe



Flots et réseaux de transports

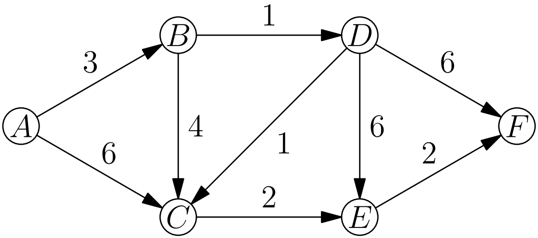


Flots et réseaux de transports



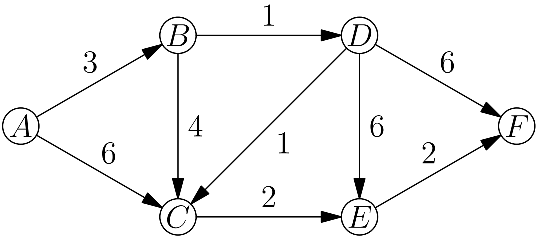
- la source est A

Flots et réseaux de transports



- la source est A
- le puit est F

Flots et réseaux de transports

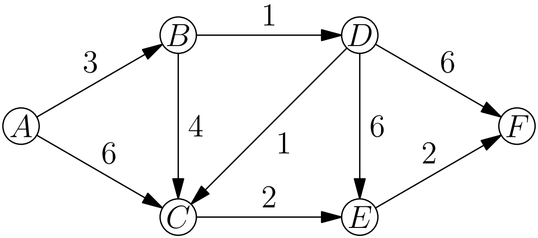


- la source est A
- le puit est F

Vobulaire – Notation

- les pondérations sont appelées *capacités*

Flots et réseaux de transports

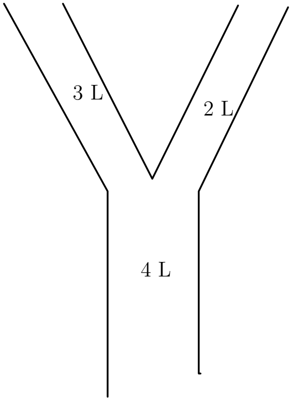


- la source est A
- le puit est F

Vobulaire – Notation

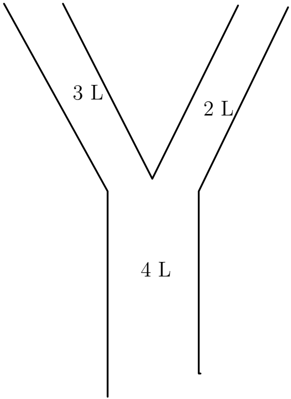
- les pondérations sont appelées *capacités*
- un réseau de transport sera noté $(\mathscr{G}; c; s; p)$ où
  - $\mathscr{G}$ est le graphe orienté
  - s la source et p le puits
  - c la fonction des capacités

Regardons ce nœud de plomberie



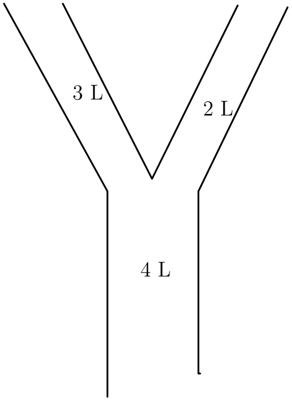
Regardons ce nœud de plomberie

- Principe du flux : tout ce qui sort est égal à ce qui entre



Regardons ce nœud de plomberie

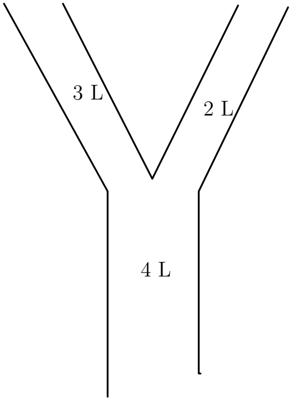
- Principe du flux : tout ce qui sort est égal à ce qui entre
- il ne sera possible que de faire passer 4 litres . . .



Flots et réseaux de transports

Regardons ce nœud de plomberie

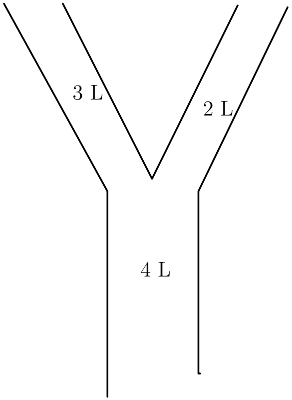
- Principe du flux : tout ce qui sort est égal à ce qui entre
- il ne sera possible que de faire passer 4 litres . . .
- même si la capacité d'entrée est de 5 litres



Flots et réseaux de transports

Regardons ce nœud de plomberie

- Principe du flux : tout ce qui sort est égal à ce qui entre
- il ne sera possible que de faire passer 4 litres . . .
- même si la capacité d'entrée est de 5 litres



La problématique est la même pour

- la bande passante dans un réseau
- le trafic routier
- réseau électrique

Définition (Flot)

Soit $\mathcal{R} = (\mathscr{G}; c; s; p)$ un réseau de transport.
$\mathcal{S}$: ensemble des sommets
$\mathcal{A}$: ensemble des arcs

Un *flot* dans $\mathcal{R}$ est une applications $\varphi : \mathcal{A} \to \mathbb{R}$ telle que :

Définition (Flot)

Soit $\mathcal{R} = (\mathscr{G}; c; s; p)$ un réseau de transport.
$\mathcal{S}$: ensemble des sommets
$\mathcal{A}$: ensemble des arcs

Un *flot* dans $\mathcal{R}$ est une applications $\varphi : \mathcal{A} \to \mathbb{R}$ telle que :

- $\forall a \in \mathcal{A},\ \varphi(a) \leqslant c(a)$

Définition (Flot)

Soit $\mathcal{R} = (\mathscr{G}; c; s; p)$ un réseau de transport.
$\mathcal{S}$: ensemble des sommets
$\mathcal{A}$: ensemble des arcs

Un *flot* dans $\mathcal{R}$ est une applications $\varphi : \mathcal{A} \to \mathbb{R}$ telle que :

- $\forall a \in \mathcal{A},\ \varphi(a) \leqslant c(a)$
- $\forall s \in \mathcal{S}$,
$$\sum_{a \in \mathcal{A}^+(s)} \varphi(a) = \sum_{a \in \mathcal{A}^-(s)} \varphi(a)$$
avec
$\mathcal{A}^+(s)$: arcs différents du puits et de la source de but s
$\mathcal{A}^-(s)$: arcs différents du puits et de la source d'origine s

Cela signifie qu'à chaque nœud,

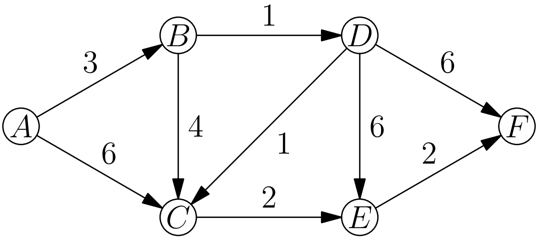
Flots et réseaux de transports

Cela signifie qu'à chaque nœud,

- le flux entrant est égal au flux sortant
- en restant inférieur à la capacité

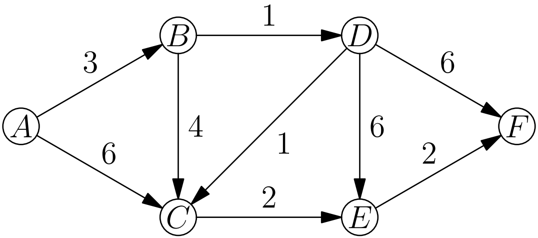
Cela signifie qu'à chaque nœud,

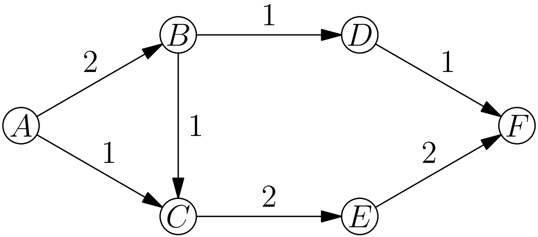
- le flux entrant est égal au flux sortant
- en restant inférieur à la capacité



Cela signifie qu'à chaque nœud,

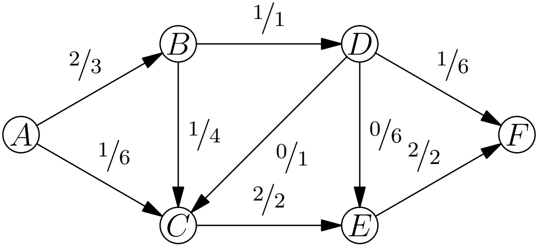
- le flux entrant est égal au flux sortant
- en restant inférieur à la capacité



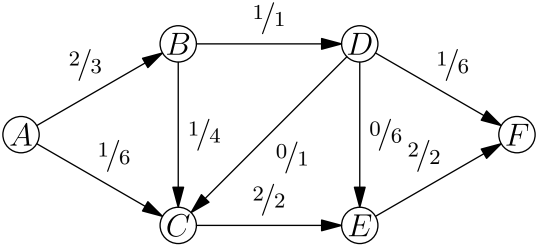


Flots et réseaux de transports

Pour plus de lisibilité, on écrit :

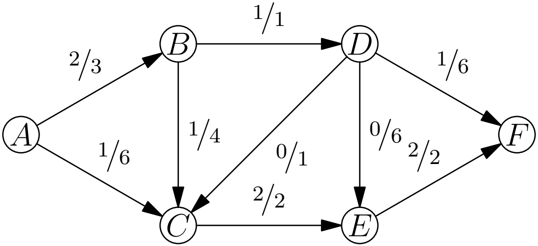


Pour plus de lisibilité, on écrit :



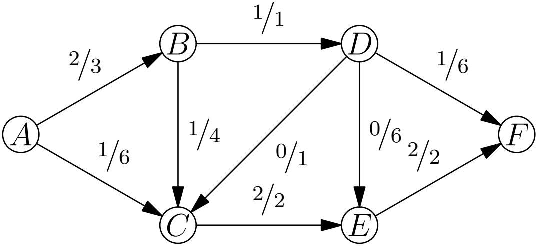
- La *valeur du flot* est la somme des valeurs du flot entrant dans le puits

Pour plus de lisibilité, on écrit :



- La *valeur du flot* est la somme des valeurs du flot entrant dans le puits
- Ici, la valeur du flot est 3

Pour plus de lisibilité, on écrit :



- La *valeur du flot* est la somme des valeurs du flot entrant dans le puits
- Ici, la valeur du flot est 3
- On cherche souvent à déterminer le flot qui rend la valeur de flot maximale

Définition (Coupe)

Un *coupe* sur un réseau de transport est la donnée de deux ensembles disjoints

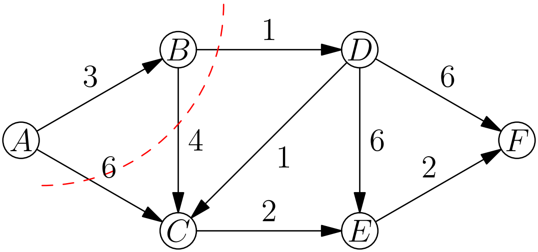
- l'un contenant la source
- l'autre contenant le puits

Coupe et réseaux de transports

Définition (Coupe)

Un *coupe* sur un réseau de transport est la donnée de deux ensembles disjoints

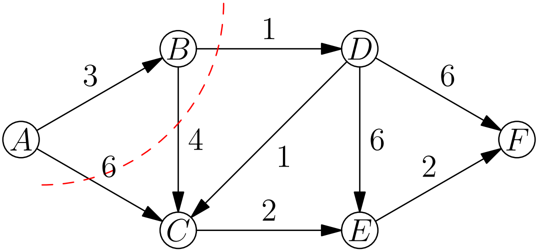
- l'un contenant la source
- l'autre contenant le puits



Représentation de la coupe :

$$(\{A, B\}\,;\{C, D, E, F\})$$

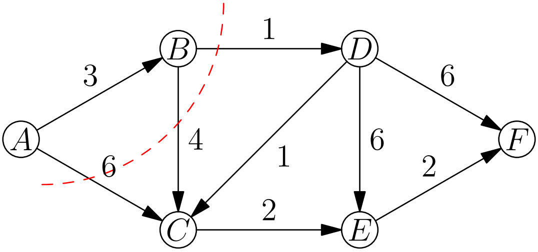
Coupe et réseaux de transports



Définition

Si $C = (S, P)$ est une coupe ($s \in S$ et $p \in P$), la *valeur de la coupe* est la somme des capacités des arc joignant un sommet de S à un sommet de P

Coupe et réseaux de transports



Définition

Si $C = (S, P)$ est une coupe ($s \in S$ et $p \in P$), la *valeur de la coupe* est la somme des capacités des arc joignant un sommet de S à un sommet de P

- La valeur de la coupe est $6 + 4 + 1 = 11$
- Elle ne dépend que des capacités du réseau de transport

Théorème (Flot maximal et coupe minimale)

Dans un réseau de transport :

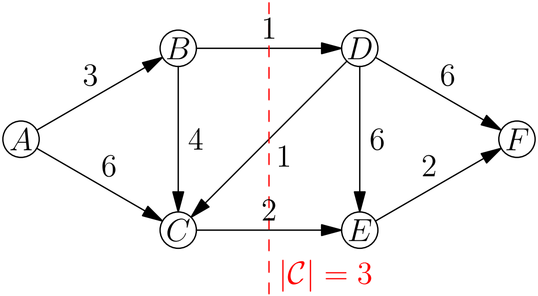
1. *la valeur d'un flot est inférieur à la valeur d'une coupe*

Théorème (Flot maximal et coupe minimale)

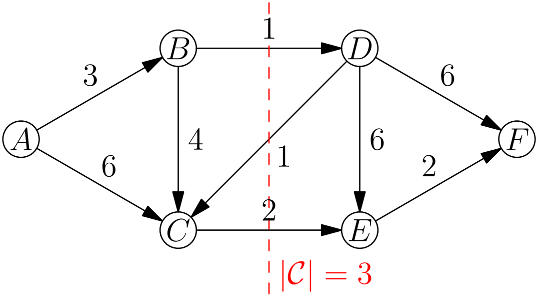
Dans un réseau de transport :

1. *la valeur d'un flot est inférieur à la valeur d'une coupe*
2. *Un flot est maximum si il est égal à la valeur d'une coupe minimale*

Coupe et réseaux de transports

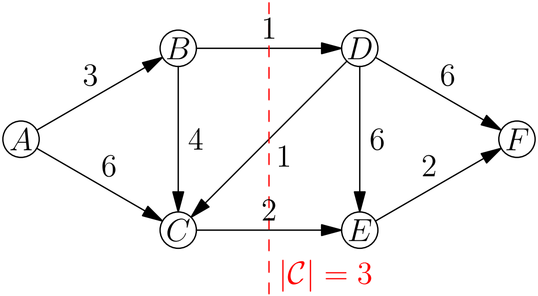


Coupe et réseaux de transports



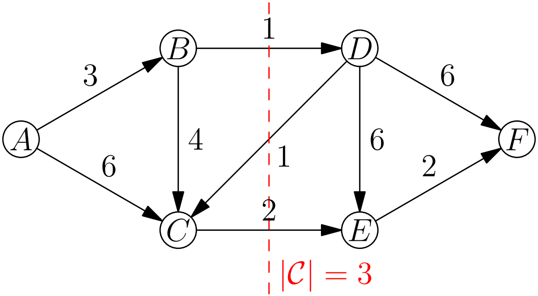
- La coupe a pour valeur 3

Coupe et réseaux de transports



- La coupe a pour valeur 3
- on avait trouvé un flot de valeur 3

Coupe et réseaux de transports



- La coupe a pour valeur 3
- on avait trouvé un flot de valeur 3
- il s'agit du flot maximum

Graphe d'écart

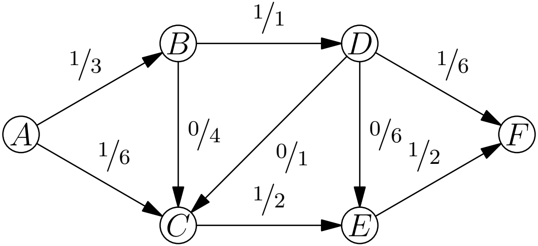
Graphe d'écart

Définition (Graphe d'écart)

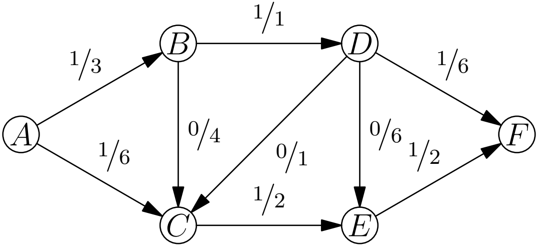
Soit $\mathcal{R} = (\mathscr{G}, c, s, p)$ un réseau de transport. On note $\mathcal{S}$ l'ensemble de ses sommets et $\mathcal{A}$ celui de ses arcs.
Pour un flot φ sur $\mathcal{R}$, le graphe d'écart est un graphe tel que :

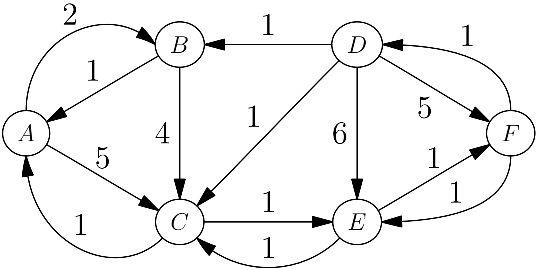
- ses sommets sont les sommets de $\mathscr{G}$ et ont le même rôle que dans $\mathscr{G}$;
- les arêtes de $\mathscr{G}$ indiquent de combien on peut augmenter le flot pour arriver à saturation ;
  on ajoute alors les arc dans le sens contraire pondérés du flot si ce flot est non nul.

Graphe d'écart

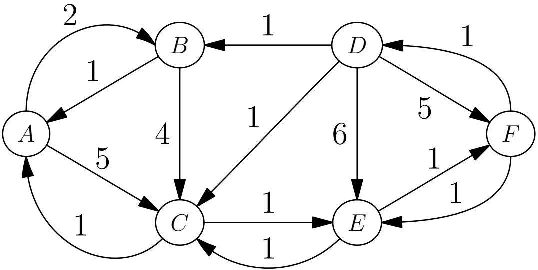


Graphe d'écart

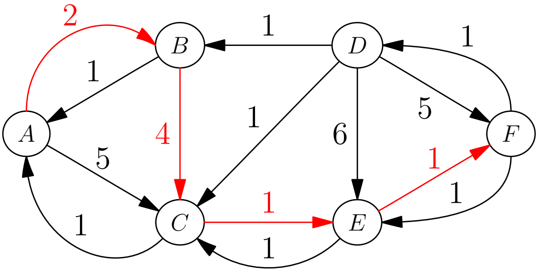




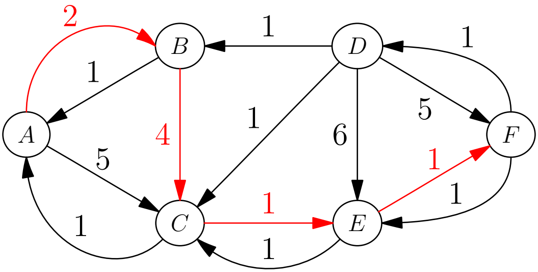
Graphe d'écart



Graphe d'écart

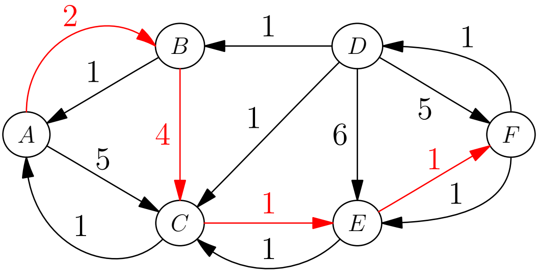


Graphe d'écart



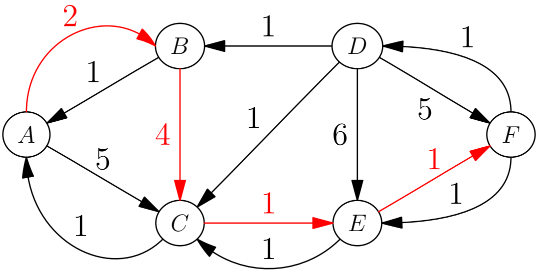
- Le chemin $A - B - C - E - F$ va de la source au puits ;

Graphe d'écart



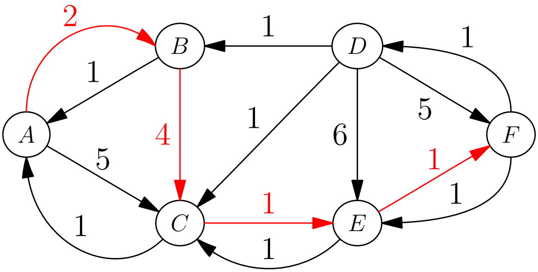
- Le chemin $A - B - C - E - F$ va de la source au puits ;
- possibilité d'amélioration du flot ;

Graphe d'écart



- Le chemin $A - B - C - E - F$ va de la source au puits ;
- possibilité d'amélioration du flot ;
- augmentation du minimum des capacités sur ce chemin (ici 1)

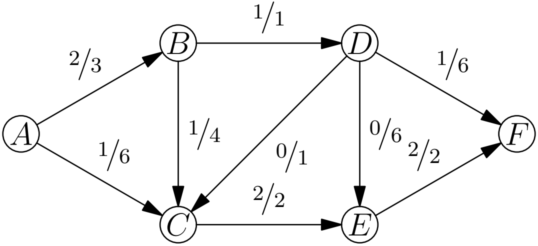
Graphe d'écart



- Le chemin $A - B - C - E - F$ va de la source au puits ;
- possibilité d'amélioration du flot ;
- augmentation du minimum des capacités sur ce chemin (ici 1)

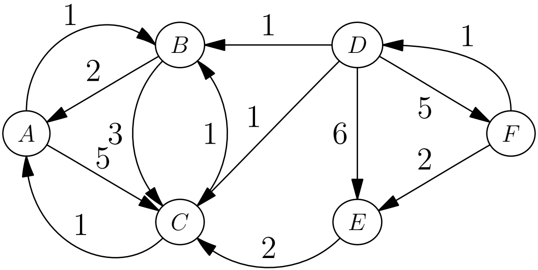
Chemin augmentant

Graphe d'écart



Le flot après amélioration

Graphe d'écart



Dans le nouveau graphe d'écart, il n'y a pas de chemin améliorant

- le flot est maximal

Algorithme de Ford et Fulkerson

Théorème

Si f est un flot dans un réseau de transport, les trois conditions suivantes sont équivalentes :

1. *f est un flot maximum ;*
2. *le graphe d'écart de f ne contient aucun chemin améliorant ;*
3. *il existe une coupe $\mathcal{C}$ dont la capacité vaut $|f|$.*

- On part d'un flot quelconque (éventuellement nul) ;

Algorithme de Ford et Fulkerson

- On part d'un flot quelconque (éventuellement nul) ;
- on fabrique le réseau résiduel ;

Algorithme de Ford et Fulkerson

- On part d'un flot quelconque (éventuellement nul) ;
- on fabrique le réseau résiduel ;
- on cherche un chemin améliorant ;

Algorithme de Ford et Fulkerson

- On part d'un flot quelconque (éventuellement nul) ;
- on fabrique le réseau résiduel ;
- on cherche un chemin améliorant ;
- on itère jusqu'à ce qu'on ne trouve plus de tel chemin.