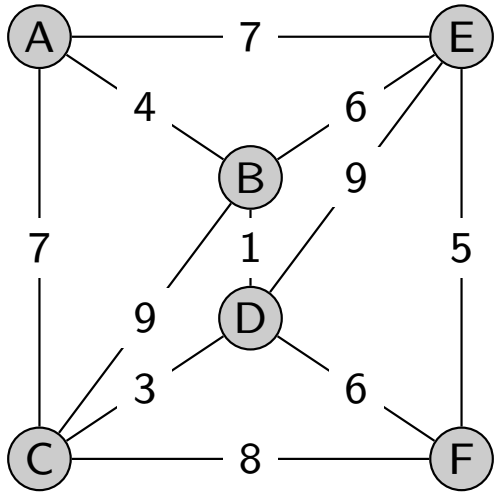
Problème d'optimisation

9 Mars 2015

5.1 Arbre couvrant

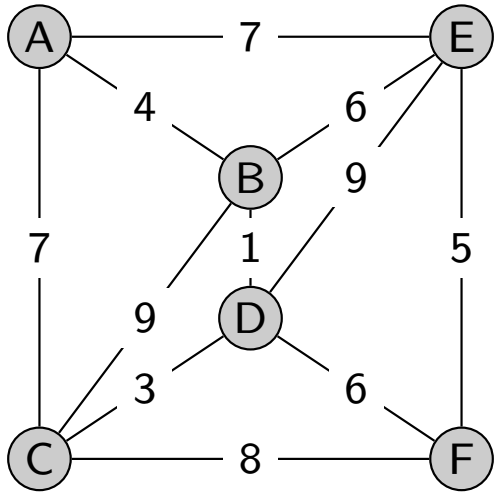
Position du problème

On va étudier deux algorithmes classiques permettant de trouver un arbre couvrant de poids minimum dans un graphe valué G donné. On supposera G est non orienté et que les valuations des arêtes sont toutes positives.



Position du problème

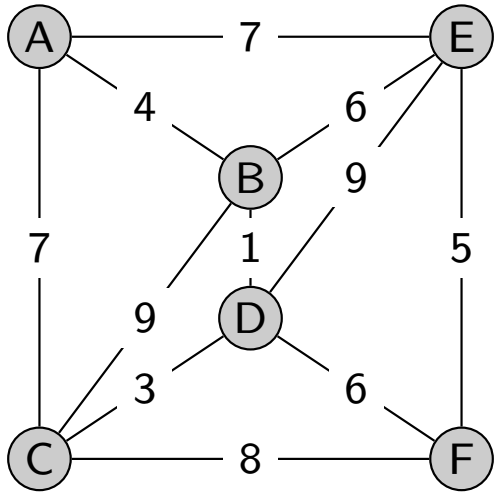
On va étudier deux algorithmes classiques permettant de trouver un arbre couvrant de poids minimum dans un graphe valué G donné. On supposera G est non orienté et que les valuations des arêtes sont toutes positives.



On va construire l'arbre couvrant de proche en proche. Deux approches possible:

Position du problème

On va étudier deux algorithmes classiques permettant de trouver un arbre couvrant de poids minimum dans un graphe valué G donné. On supposera G est non orienté et que les valuations des arêtes sont toutes positives.

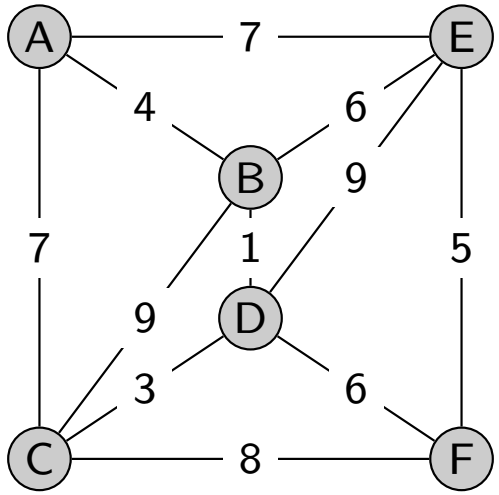


On va construire l'arbre couvrant de proche en proche. Deux approches possible:

- Approche locale: à chaque étapes, parmi les sommets connectés, on rajoute l'arête optimale qui relie un sommet déjà connecté à un sommet non connecté. (*algorithme de Prim*);

Position du problème

On va étudier deux algorithmes classiques permettant de trouver un arbre couvrant de poids minimum dans un graphe valué G donné. On supposera G est non orienté et que les valuations des arêtes sont toutes positives.

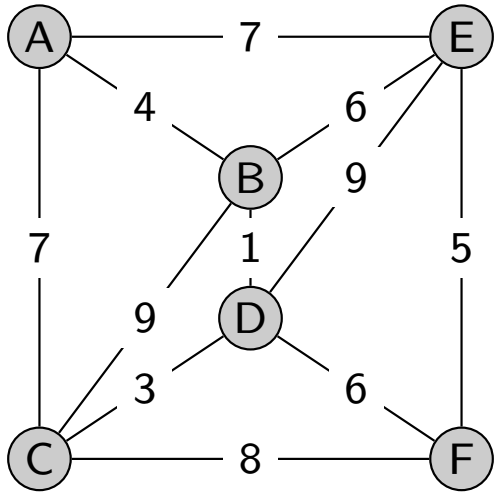


On va construire l'arbre couvrant de proche en proche. Deux approches possible:

- Approche locale: à chaque étapes, parmi les sommets connectés, on rajoute l'arête optimale qui relie un sommet déjà connecté à un sommet non connecté. (*algorithme de Prim*);
- Approche globale: on choisit l'arête optimale de façon que l'arête rajoutée ne relie pas des sommets déjà connectés (*algorithme de Kruskal*).

Position du problème

On va étudier deux algorithmes classiques permettant de trouver un arbre couvrant de poids minimum dans un graphe valué G donné. On supposera G est non orienté et que les valuations des arêtes sont toutes positives.

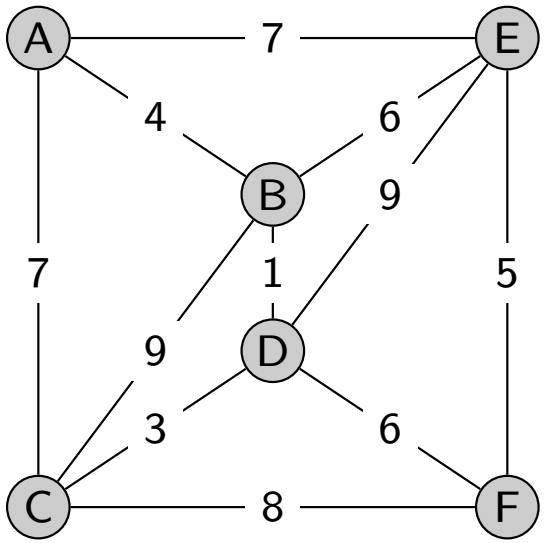


On va construire l'arbre couvrant de proche en proche. Deux approches possible:

- Approche locale: à chaque étapes, parmi les sommets connectés, on rajoute l'arête optimale qui relie un sommet déjà connecté à un sommet non connecté. (*algorithme de Prim*);
- Approche globale: on choisit l'arête optimale de façon que l'arête rajoutée ne relie pas des sommets déjà connectés (*algorithme de Kruskal*).

Ce sont deux exemples d'algorithmes *gloutons* : à chaque étape on fait le meilleur choix instantané dans le but d'obtenir la meilleure solution globale.

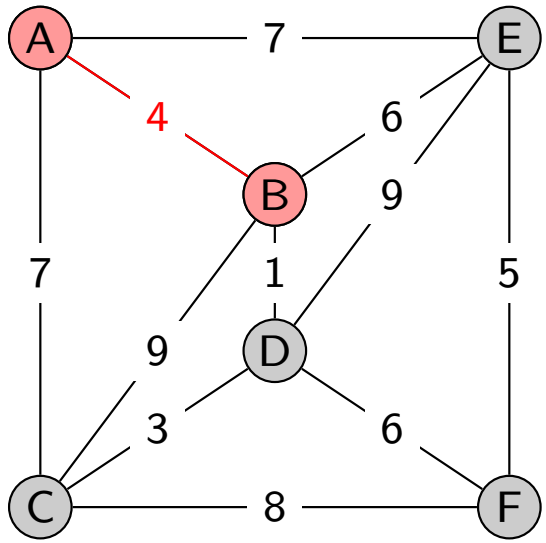
Algorithme de Prim



Data: $G = (S, A, \lambda)$ graphe valué et $s \in S$
Data: Un arbre $T = (S, A')$

Poids$\leftarrow$ 0 (Poids total de l'arbre couvrant);
$A' \leftarrow \emptyset$;
$L \leftarrow \{s\}$ (sommets marqués);
while $L \neq S$ **do**
  $a \leftarrow$ arête de poids minimal joignant
  $x \in L$ et $y \in S \setminus L$;
  $L \leftarrow L \cup \{y\}$;
  $A' \leftarrow A' \cup \{a\}$;
  Poids$\leftarrow$Poids$+\lambda(a)$;
end

Algorithme de Prim

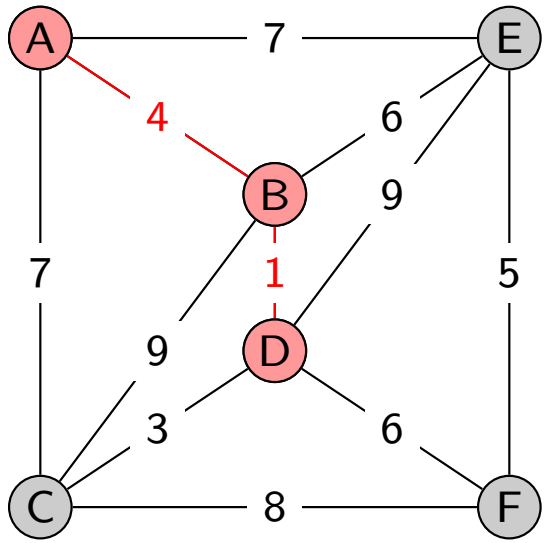


Sommets marqués	arêtes sortantes	poids minimal
A	$\{A, B\}$ $\{A, C\}$ $\{A, E\}$	**4**

Data: $G = (S, A, \lambda)$ graphe valué et $s \in S$
Data: Un arbre $T = (S, A')$

Poids$\leftarrow$ 0 (Poids total de l'arbre couvrant);
$A' \leftarrow \emptyset$;
$L \leftarrow \{s\}$ (sommets marqués);
while $L \neq S$ **do**
  $a \leftarrow$ arête de poids minimal joignant
  $x \in L$ et $y \in S \setminus L$;
  $L \leftarrow L \cup \{y\}$;
  $A' \leftarrow A' \cup \{a\}$;
  Poids$\leftarrow$Poids$+\lambda(a)$;
end

Algorithme de Prim

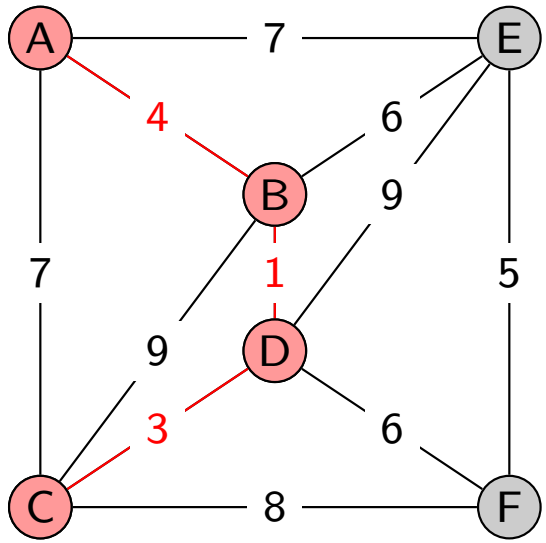


Sommets marqués	arêtes sortantes	poids minimal
A	$\{A,B\}$ $\{A,C\}$ $\{A,E\}$	4
A,B	$\{A,C\}$ $\{A,E\}$ $\{B,C\}$ $\{B,E\}$ $\{B,D\}$	1

Data: $G = (S, A, \lambda)$ graphe valué et $s \in S$
Data: Un arbre $T = (S, A')$

Poids$\leftarrow$ 0 (Poids total de l'arbre couvrant);
$A' \leftarrow \emptyset$;
$L \leftarrow \{s\}$ (sommets marqués);
while $L \neq S$ **do**
  $a \leftarrow$ arête de poids minimal joignant
  $x \in L$ et $y \in S \setminus L$;
  $L \leftarrow L \cup \{y\}$;
  $A' \leftarrow A' \cup \{a\}$;
  Poids$\leftarrow$Poids$+\lambda(a)$;
end

Algorithme de Prim

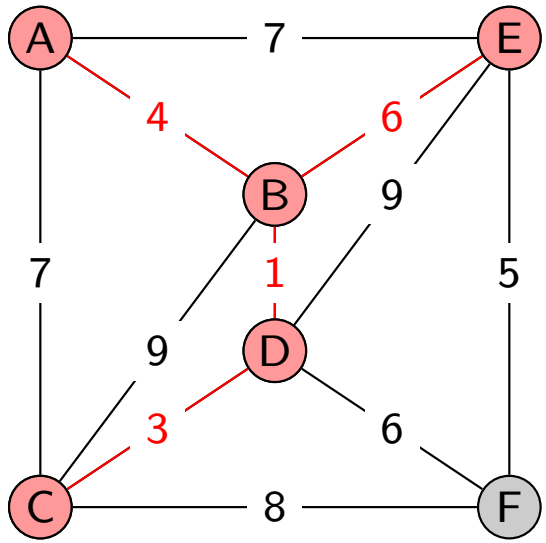


Sommets marqués	arêtes sortantes	poids minimal
A	$\{A,B\}$ $\{A,C\}$ $\{A,E\}$	4
A,B	$\{A,C\}$ $\{A,E\}$ $\{B,C\}$ $\{B,E\}$ $\{B,D\}$	1
A,B,D	$\{A,C\}$ $\{A,E\}$ $\{B,C\}$ $\{B,E\}$ $\{C,D\}$ $\{D,E\}$ $\{D,F\}$	3

Data: $G = (S, A, \lambda)$ graphe valué et $s \in S$
Data: Un arbre $T = (S, A')$

Poids$\leftarrow$ 0 (Poids total de l'arbre couvrant);
$A' \leftarrow \emptyset$;
$L \leftarrow \{s\}$ (sommets marqués);
while $L \neq S$ **do**
  $a \leftarrow$ arête de poids minimal joignant
  $x \in L$ et $y \in S \setminus L$;
  $L \leftarrow L \cup \{y\}$;
  $A' \leftarrow A' \cup \{a\}$;
  Poids$\leftarrow$Poids$+\lambda(a)$;
end

Algorithme de Prim

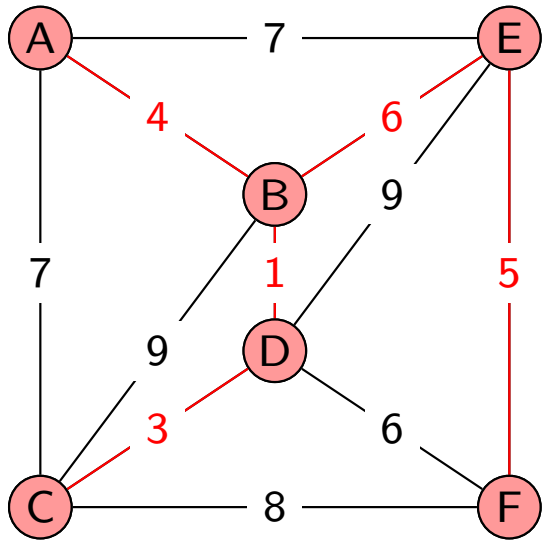


Sommets marqués	arêtes sortantes	poids minimal
A	$\{A,B\}$ $\{A,C\}$ $\{A,E\}$	4
A,B	$\{A,C\}$ $\{A,E\}$ $\{B,C\}$ $\{B,E\}$ $\{B,D\}$	1
A,B,D	$\{A,C\}$ $\{A,E\}$ $\{B,C\}$ $\{B,E\}$ $\{C,D\}$ $\{D,E\}$ $\{D,F\}$	3
A,B,D,C	$\{A,E\}$ $\{B,E\}$ $\{D,E\}$ $\{C,F\}$ $\{D,F\}$	6

Data: $G = (S, A, \lambda)$ graphe valué et $s \in S$
Data: Un arbre $T = (S, A')$

Poids$\leftarrow$ 0 (Poids total de l'arbre couvrant);
$A' \leftarrow \emptyset$;
$L \leftarrow \{s\}$ (sommets marqués);
while $L \neq S$ **do**
  $a \leftarrow$ arête de poids minimal joignant
  $x \in L$ et $y \in S \setminus L$;
  $L \leftarrow L \cup \{y\}$;
  $A' \leftarrow A' \cup \{a\}$;
  Poids$\leftarrow$Poids$+\lambda(a)$;
end

Algorithme de Prim

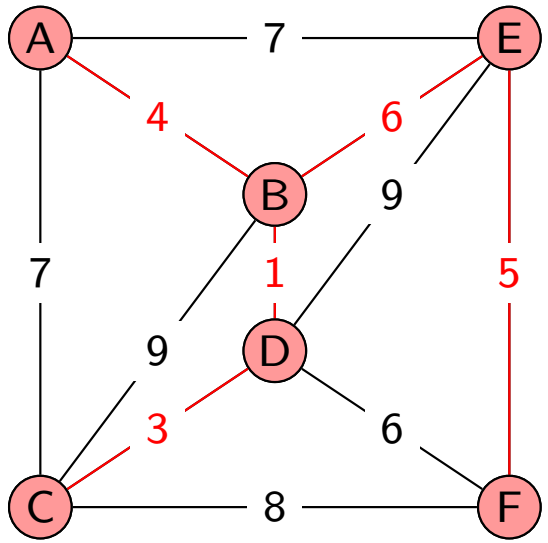


Sommets marqués	arêtes sortantes	poids minimal
A	$\{A,B\}$ $\{A,C\}$ $\{A,E\}$	4
A,B	$\{A,C\}$ $\{A,E\}$ $\{B,C\}$ $\{B,E\}$ $\{B,D\}$	1
A,B,D	$\{A,C\}$ $\{A,E\}$ $\{B,C\}$ $\{B,E\}$ $\{C,D\}$ $\{D,E\}$ $\{D,F\}$	3
A,B,D,C	$\{A,E\}$ $\{B,E\}$ $\{D,E\}$ $\{C,F\}$ $\{D,F\}$	6
A,B,D,C,E	$\{C,F\}$ $\{D,F\}$ $\{E,F\}$	5

Data: $G = (S, A, \lambda)$ graphe valué et $s \in S$
Data: Un arbre $T = (S, A')$

Poids$\leftarrow$ 0 (Poids total de l'arbre couvrant);
$A' \leftarrow \emptyset$;
$L \leftarrow \{s\}$ (sommets marqués);
while $L \neq S$ **do**
  $a \leftarrow$ arête de poids minimal joignant
  $x \in L$ et $y \in S \setminus L$;
  $L \leftarrow L \cup \{y\}$;
  $A' \leftarrow A' \cup \{a\}$;
  Poids$\leftarrow$Poids$+\lambda(a)$;
end

Algorithme de Prim



Sommets marqués	arêtes sortantes	poids minimal
A	$\{A,B\}$ $\{A,C\}$ $\{A,E\}$	4
A,B	$\{A,C\}$ $\{A,E\}$ $\{B,C\}$ $\{B,E\}$ $\{B,D\}$	1
A,B,D	$\{A,C\}$ $\{A,E\}$ $\{B,C\}$ $\{B,E\}$ $\{C,D\}$ $\{D,E\}$ $\{D,F\}$	3
A,B,D,C	$\{A,E\}$ $\{B,E\}$ $\{D,E\}$ $\{C,F\}$ $\{D,F\}$	6
A,B,D,C,E	$\{C,F\}$ $\{D,F\}$ $\{E,F\}$	5

Poids total: 19

Data: $G = (S, A, \lambda)$ graphe valué et $s \in S$
Data: Un arbre $T = (S, A')$

Poids$\leftarrow$ 0 (Poids total de l'arbre couvrant);
$A' \leftarrow \emptyset$;
$L \leftarrow \{s\}$ (sommets marqués);
while $L \neq S$ **do**
  $a \leftarrow$ arête de poids minimal joignant
  $x \in L$ et $y \in S \setminus L$;
  $L \leftarrow L \cup \{y\}$;
  $A' \leftarrow A' \cup \{a\}$;
  Poids$\leftarrow$Poids$+\lambda(a)$;
end

Algorithme de Prim

Terminaison: A chaque coup on marque un sommet.

Data: $G = (S, A, \lambda)$ graphe valué et $s \in S$
Data: Un arbre $T = (S, A')$

Poids$\leftarrow$ 0 (Poids total de l'arbre couvrant);
$A' \leftarrow \emptyset$;
$L \leftarrow \{s\}$ (sommets marqués);
while $L \neq S$ **do**
    $a \leftarrow$ arête de poids minimal joignant
    $x \in L$ et $y \in S \setminus L$;
    $L \leftarrow L \cup \{y\}$;
    $A' \leftarrow A' \cup \{a\}$;
    Poids$\leftarrow$Poids$+\lambda(a)$;
end

Algorithme de Prim

Terminaison: A chaque coup on marque un sommet.

Correction: Si on cherche un arbre à coût minimal contenant un sous-arbre G' imposé, alors il existe parmi les solutions optimales contenant G', une solution qui contient une des arêtes de coût minimal adjacente à G' et ne formant pas de cycle avec G' (c'est-à-dire une extrémité dans A et l'autre à l'extérieur de G').

Data: $G = (S, A, \lambda)$ graphe valué et $s \in S$
Data: Un arbre $T = (S, A')$

Poids$\leftarrow$ 0 (Poids total de l'arbre couvrant);
$A' \leftarrow \emptyset$;
$L \leftarrow \{s\}$ (sommets marqués);
while $L \neq S$ **do**
    $a \leftarrow$ arête de poids minimal joignant
    $x \in L$ et $y \in S \setminus L$;
    $L \leftarrow L \cup \{y\}$;
    $A' \leftarrow A' \cup \{a\}$;
    Poids$\leftarrow$Poids$+\lambda(a)$;
end

Algorithme de Prim

Terminaison: A chaque coup on marque un sommet.

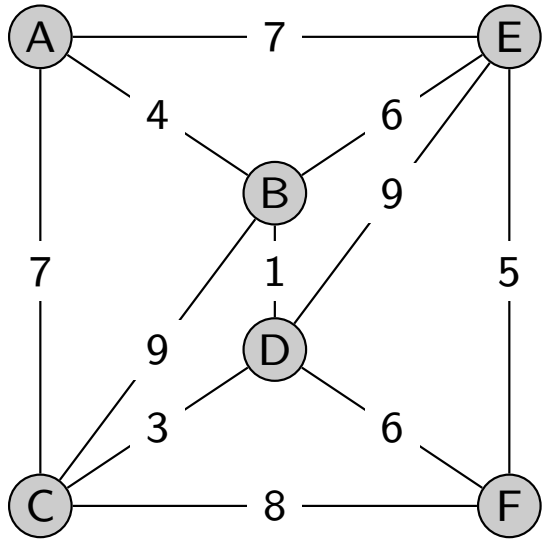
Correction: Si on cherche un arbre à coût minimal contenant un sous-arbre G' imposé, alors il existe parmi les solutions optimales contenant G', une solution qui contient une des arêtes de coût minimal adjacente à G' et ne formant pas de cycle avec G' (c'est-à-dire une extrémité dans A et l'autre à l'extérieur de G').

Complexité: $O(|S||A|)$

Data: $G = (S, A, \lambda)$ graphe valué et $s \in S$
Data: Un arbre $T = (S, A')$

Poids$\leftarrow$ 0 (Poids total de l'arbre couvrant);
$A' \leftarrow \emptyset$;
$L \leftarrow \{s\}$ (sommets marqués);
while $L \neq S$ **do**
    $a \leftarrow$ arête de poids minimal joignant
    $x \in L$ et $y \in S \setminus L$;
    $L \leftarrow L \cup \{y\}$;
    $A' \leftarrow A' \cup \{a\}$;
    Poids$\leftarrow$Poids$+\lambda(a)$;
end

Algorithme de Kruskal

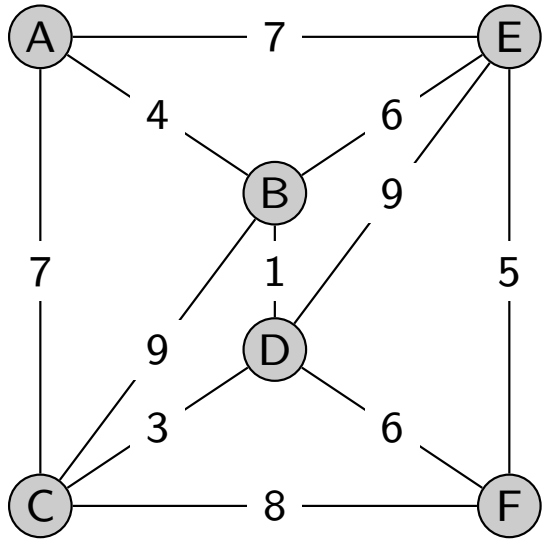


E(A)	
E(B)	
E(C)	
E(D)	
E(E)	
E(F)	
arête considéré	
poids	

Data: $G = (S, A, \lambda)$ graphe valué
Data: Un arbre $T = (S, A')$

Poids$\leftarrow$ 0 (Poids total de l'arbre couvrant);
$A' \leftarrow \emptyset$;
for $s \in S$ **do**
  $E(s) \leftarrow \{s\}$ ($E(s)$: sommets reliés à s)
end
for $\{s, s'\} \in A$ *dans l'ordre croissant* **do**
  if $E(s) \neq E(s')$ **then**
    $A' \leftarrow A' \cup \{s, s'\}$
    Poids$\leftarrow$Poids$+\lambda(\{s, s'\})$;
    $F \leftarrow E(s) \cup E(s')$;
    for $z \in F$ **do**
      $E(z) \leftarrow F$;
    end
  end
end

Algorithme de Kruskal

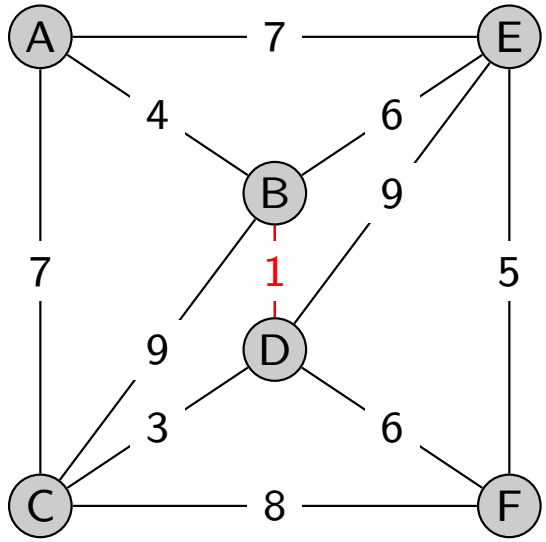


E(A)	A
E(B)	B
E(C)	C
E(D)	D
E(E)	E
E(F)	F
arête considéré	
poids	

Data: $G = (S, A, \lambda)$ graphe valué
Data: Un arbre $T = (S, A')$

Poids$\leftarrow$ 0 (Poids total de l'arbre couvrant);
$A' \leftarrow \emptyset$;
for $s \in S$ **do**
  $E(s) \leftarrow \{s\}$ ($E(s)$: sommets reliés à s)
end
for $\{s, s'\} \in A$ *dans l'ordre croissant* **do**
  if $E(s) \neq E(s')$ **then**
    $A' \leftarrow A' \cup \{s, s'\}$
    Poids$\leftarrow$Poids$+\lambda(\{s, s'\})$;
    $F \leftarrow E(s) \cup E(s')$;
    for $z \in F$ **do**
      $E(z) \leftarrow F$;
    end
  end
end

Algorithme de Kruskal

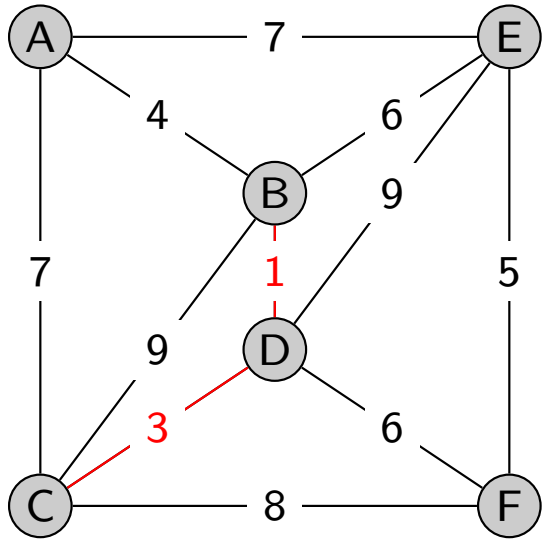


E(A)	A
E(B)	B,D
E(C)	C
E(D)	D,B
E(E)	E
E(F)	F
arête considéré	{B,D}
poids	1

Data: $G = (S, A, \lambda)$ graphe valué
Data: Un arbre $T = (S, A')$

Poids$\leftarrow$ 0 (Poids total de l'arbre couvrant);
$A' \leftarrow \emptyset$;
for $s \in S$ **do**
  $E(s) \leftarrow \{s\}$ ($E(s)$: sommets reliés à s)
end
for $\{s, s'\} \in A$ *dans l'ordre croissant* **do**
  if $E(s) \neq E(s')$ **then**
    $A' \leftarrow A' \cup \{s, s'\}$
    Poids$\leftarrow$Poids$+\lambda(\{s, s'\})$;
    $F \leftarrow E(s) \cup E(s')$;
    for $z \in F$ **do**
      $E(z) \leftarrow F$;
    end
  end
end

Algorithme de Kruskal

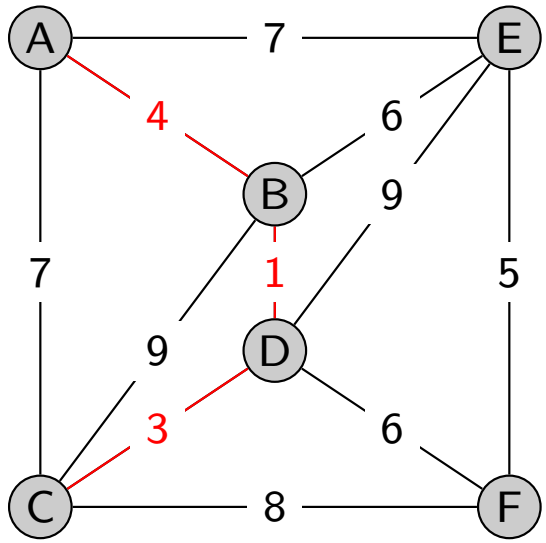


E(A)	A
E(B)	B,D,C
E(C)	C,B,D
E(D)	D,B,C
E(E)	E
E(F)	F
arête considéré	{C,D}
poids	4

Data: $G = (S, A, \lambda)$ graphe valué
Data: Un arbre $T = (S, A')$

Poids← 0 (Poids total de l'arbre couvrant);
$A' \leftarrow \emptyset$;
for $s \in S$ **do**
  $E(s) \leftarrow \{s\}$ ($E(s)$: sommets reliés à s)
end
for $\{s, s'\} \in A$ *dans l'ordre croissant* **do**
  if $E(s) \neq E(s')$ **then**
    $A' \leftarrow A' \cup \{s, s'\}$
    Poids←Poids+$\lambda(\{s, s'\})$;
    $F \leftarrow E(s) \cup E(s')$;
    for $z \in F$ **do**
      $E(z) \leftarrow F$;
    end
  end
end

Algorithme de Kruskal

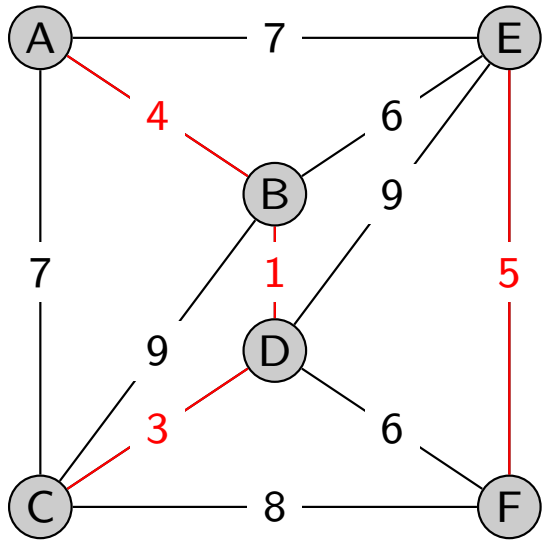


E(A)	A,B,C,D
E(B)	B,D,C,A
E(C)	C,B,D,A
E(D)	D,B,C,A
E(E)	E
E(F)	F
arête considéré	{A,B}
poids	8

Data: $G = (S, A, \lambda)$ graphe valué
Data: Un arbre $T = (S, A')$

Poids$\leftarrow$ 0 (Poids total de l'arbre couvrant);
$A' \leftarrow \emptyset$;
for $s \in S$ **do**
  $E(s) \leftarrow \{s\}$ ($E(s)$: sommets reliés à s)
end
for $\{s, s'\} \in A$ *dans l'ordre croissant* **do**
  if $E(s) \neq E(s')$ **then**
    $A' \leftarrow A' \cup \{s, s'\}$
    Poids$\leftarrow$Poids$+\lambda(\{s, s'\})$;
    $F \leftarrow E(s) \cup E(s')$;
    for $z \in F$ **do**
      $E(z) \leftarrow F$;
    end
  end
end

Algorithme de Kruskal

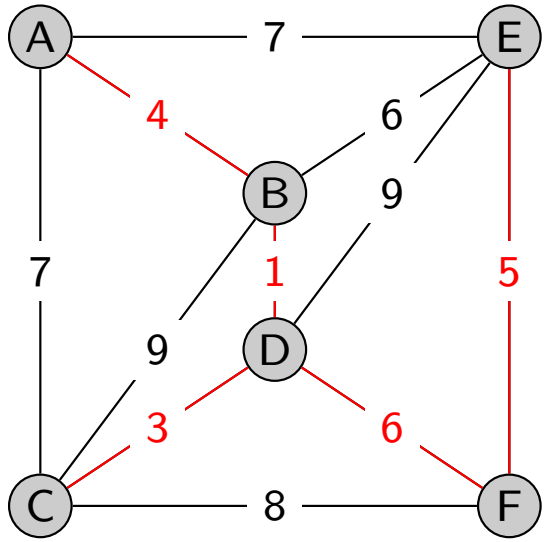


E(A)	A,B,C,D
E(B)	B,D,C,A
E(C)	C,B,D,A
E(D)	D,B,C,A
E(E)	E,F
E(F)	F,E
arête considéré	{E,F}
poids	13

Data: $G = (S, A, \lambda)$ graphe valué
Data: Un arbre $T = (S, A')$

Poids← 0 (Poids total de l'arbre couvrant);
$A' \leftarrow \emptyset$;
for $s \in S$ **do**
  $E(s) \leftarrow \{s\}$ ($E(s)$: sommets reliés à s)
end
for $\{s, s'\} \in A$ *dans l'ordre croissant* **do**
  if $E(s) \neq E(s')$ **then**
    $A' \leftarrow A' \cup \{s, s'\}$
    Poids←Poids+$\lambda(\{s, s'\})$;
    $F \leftarrow E(s) \cup E(s')$;
    for $z \in F$ **do**
      $E(z) \leftarrow F$;
    end
  end
end

Algorithme de Kruskal



E(A)	A,B,C,D,E,F
E(B)	B,D,C,A,E,F
E(C)	C,B,D,A,E,F
E(D)	D,B,C,A,E,F
E(E)	E,F,A,B,C,D
E(F)	F,E,A,B,C,D
arête considéré	{D,F}
poids	19

Data: $G = (S, A, \lambda)$ graphe valué
Data: Un arbre $T = (S, A')$

Poids$\leftarrow$ 0 (Poids total de l'arbre couvrant);
$A' \leftarrow \emptyset$;
for $s \in S$ **do**
  $E(s) \leftarrow \{s\}$ ($E(s)$: sommets reliés à s)
end
for $\{s, s'\} \in A$ *dans l'ordre croissant* **do**
  if $E(s) \neq E(s')$ **then**
    $A' \leftarrow A' \cup \{s, s'\}$
    Poids$\leftarrow$Poids$+\lambda(\{s, s'\})$;
    $F \leftarrow E(s) \cup E(s')$;
    for $z \in F$ **do**
      $E(z) \leftarrow F$;
    end
  end
end

Algorithme de Kruskal

Terminaison: Une fois que l'on a énuméré toutes les arêtes l'algorithme s'arête.

Data: $G = (S, A, \lambda)$ graphe valué
Data: Un arbre $T = (S, A')$

Poids$\leftarrow$ 0 (Poids total de l'arbre couvrant);
$A' \leftarrow \emptyset$;
for $s \in S$ **do**
  $E(s) \leftarrow \{s\}$ ($E(s)$: sommets reliés à s)
end
for $\{s, s'\} \in A$ *dans l'ordre croissant* **do**
  if $E(s) \neq E(s')$ **then**
    $A' \leftarrow A' \cup \{s, s'\}$;
    Poids$\leftarrow$Poids$+\lambda(\{s, s'\})$;
    $F \leftarrow E(s) \cup E(s')$;
    for $z \in F$ **do**
      $E(z) \leftarrow F$;
    end
  end
end

Algorithme de Kruskal

Terminaison: Une fois que l'on a énuméré toutes les arêtes l'algorithme s'arête.

Correction: Parmi les arbres de recouvrement minimaux du graphe $G = (S, A)$ pour lesquels le sous-ensemble d'arêtes A' est imposé, il en existe au moins un qui contient une des plus petites arêtes de $A \setminus A'$ qui ne crée pas de cycle lorsqu'on l'ajoute à A'.

Data: $G = (S, A, \lambda)$ graphe valué
Data: Un arbre $T = (S, A')$

Poids$\leftarrow$ 0 (Poids total de l'arbre couvrant);
$A' \leftarrow \emptyset$;
for $s \in S$ **do**
  $E(s) \leftarrow \{s\}$ ($E(s)$: sommets reliés à s)
end
for $\{s, s'\} \in A$ *dans l'ordre croissant* **do**
  if $E(s) \neq E(s')$ **then**
    $A' \leftarrow A' \cup \{s, s'\}$;
    Poids$\leftarrow$Poids$+\lambda(\{s, s'\})$;
    $F \leftarrow E(s) \cup E(s')$;
    for $z \in F$ **do**
      $E(z) \leftarrow F$;
    end
  end
end

Algorithme de Kruskal

Terminaison: Une fois que l'on a énuméré toutes les arêtes l'algorithme s'arête.

Correction: Parmi les arbres de recouvrement minimaux du graphe $G = (S, A)$ pour lesquels le sous-ensemble d'arêtes A' est imposé, il en existe au moins un qui contient une des plus petites arêtes de $A \setminus A'$ qui ne crée pas de cycle lorsqu'on l'ajoute à A'.

Complexité: $O(|S||A|)$

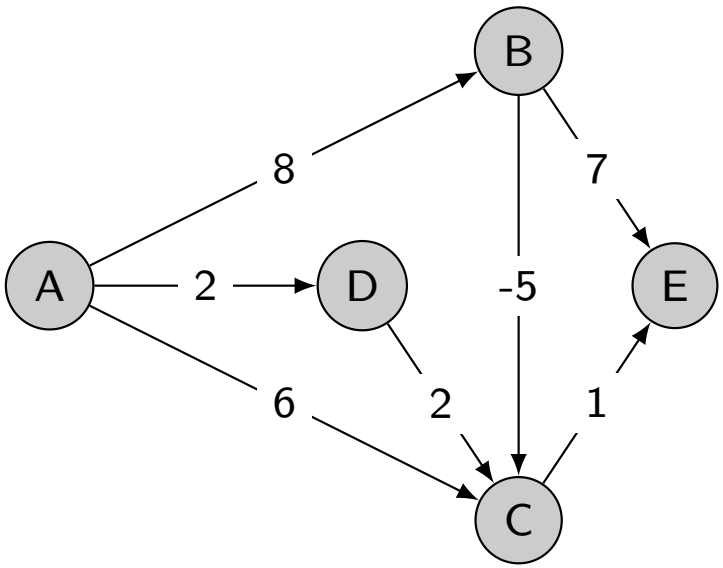
Data: $G = (S, A, \lambda)$ graphe valué
Data: Un arbre $T = (S, A')$

Poids$\leftarrow$ 0 (Poids total de l'arbre couvrant);
$A' \leftarrow \emptyset$;
for $s \in S$ **do**
  $E(s) \leftarrow \{s\}$ ($E(s)$: sommets reliés à s)
end
for $\{s, s'\} \in A$ *dans l'ordre croissant* **do**
  if $E(s) \neq E(s')$ **then**
    $A' \leftarrow A' \cup \{s, s'\}$;
    Poids$\leftarrow$Poids$+\lambda(\{s, s'\})$;
    $F \leftarrow E(s) \cup E(s')$;
    for $z \in F$ **do**
      $E(z) \leftarrow F$;
    end
  end
end

5.2 Problème de plus court chemin

Principe

Cas général.



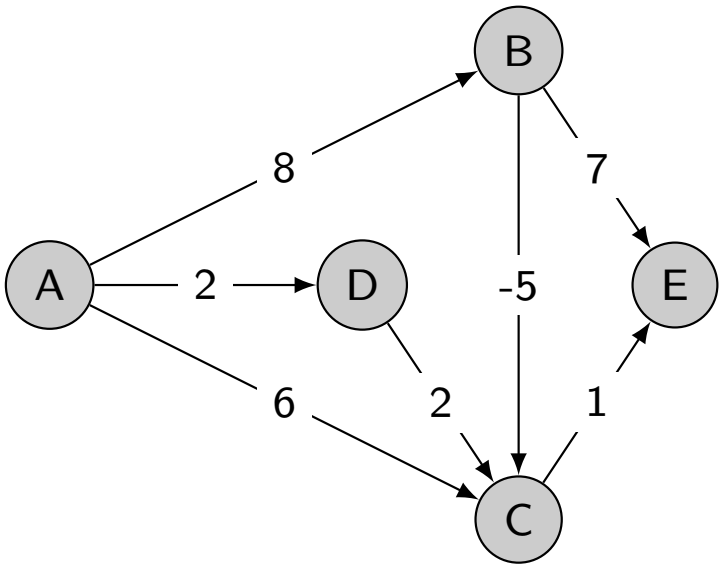
- *Dist*: tableau donnant$d(s,x)$ pour tout $x \in S$;
- *Pred*: table des prédécesseurs de l'arbre des plus court chemins issus de x.

Principe des trois algorithmes suivants:

- on initialise les tableaux *Dist* et *Pred*
- on calcule $Dist(s)$ et $Pred(s)$ par approximations successives: on essaye d'améliorer les valeurs obtenues précédemment;
- amélioration au niveau local : pour $s \in S$ et un successeur s' de s, on compare $Dist(s')$ obtenue à l'étape précédente avec $Dist(s) + W(s,s')$ on a alors deux cas:
  - si elle est plus petite, alors $Dist(s') \leftarrow Dist(s) + W(s,s')$ et $Pred(s') \leftarrow s$;
  - sinon on ne fait rien.

Algorithme de Bellman-Ford

Cas général.



Data: W matrice des distances et $s \in S$
Data: *Dist* tableau des distance à s

Pred tableau des prédécesseur initialisé à $\varnothing$;
Dist tableau des distances initialisé à $+\infty$;
$Dist(s) \leftarrow 0$;
$k \leftarrow 1$;
while $k \leq |S|$ *et il y'a eu des modifications à l'étape précédente* **do**
  for $x \in S$ **do**
    for y *successeur de* x **do**
      if $Dist(x) + W(x,y) < Dist(y)$ **then**
        $Dist(y) \leftarrow Dist(x) + W(x,y)$;
        $Pred(y) \leftarrow x$;
      end
    end
  end
  $k \leftarrow k + 1$;
end

Algorithme de Bellman-Ford

Cas général.

Terminaison: Dans chaque boucles on passe au plus $|S|$ fois.

Data: W matrice des distances et $s \in S$
Data: $Dist$ tableau des distance à s

$Pred$ tableau des prédécesseur initialisé à $\varnothing$;
$Dist$ tableau des distances initialisé à $+\infty$;
$Dist(s) \leftarrow 0$;
$k \leftarrow 1$;
while $k \leq n$ *et il y'a eu des modifications à l'étape précédente* **do**
  for $x \in S$ **do**
    for y *successeur de* x **do**
      if $Dist(x) + W(x,y) < Dist(y)$
      then
        $Dist(y) \leftarrow Dist(x) + W(x,y)$;
        $Pred(y) \leftarrow x$;
      end
    end
  end
  $k \leftarrow k + 1$;
end

Algorithme de Bellman-Ford

Cas général.

Terminaison: Dans chaque boucles on passe au plus $|S|$ fois.

Correction: On montre par récurrence que si un plus cout chemin élémentaire comporte k arc alors après k passages dans la boucle $Dist(x) = d(s, x)$.

Data: W matrice des distances et $s \in S$
Data: $Dist$ tableau des distance à s

$Pred$ tableau des prédécesseur initialisé à $\varnothing$;
$Dist$ tableau des distances initialisé à $+\infty$;
$Dist(s) \leftarrow 0$;
$k \leftarrow 1$;
while $k \leq n$ *et il y'a eu des modifications à l'étape précédente* **do**
  for $x \in S$ **do**
    for y *successeur de* x **do**
      if $Dist(x) + W(x, y) < Dist(y)$
      then
        $Dist(y) \leftarrow Dist(x) + W(x, y)$;
        $Pred(y) \leftarrow x$;
      end
    end
  end
  $k \leftarrow k + 1$;
end

Algorithme de Bellman-Ford

Cas général.

Terminaison: Dans chaque boucles on passe au plus $|S|$ fois.

Correction: On montre par récurrence que si un plus cout chemin élémentaire comporte k arc alors après k passages dans la boucle $Dist(x) = d(s, x)$.

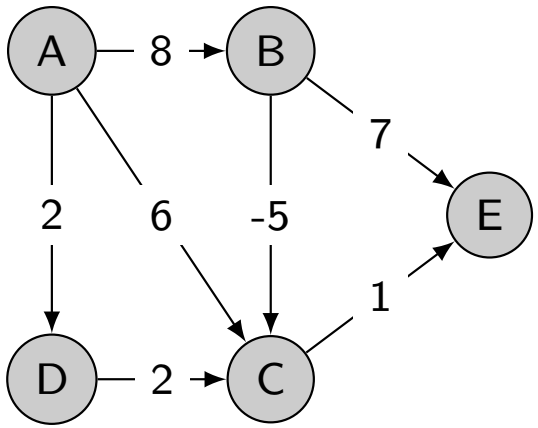
Complexité: $O(|S|^3)$

Data: W matrice des distances et $s \in S$
Data: $Dist$ tableau des distance à s

$Pred$ tableau des prédécesseur initialisé à $\varnothing$;
$Dist$ tableau des distances initialisé à $+\infty$;
$Dist(s) \leftarrow 0$;
$k \leftarrow 1$;
while $k \leq n$ *et il y'a eu des modifications à l'étape précédente* **do**
  for $x \in S$ **do**
    for y *successeur de* x **do**
      if $Dist(x) + W(x, y) < Dist(y)$
      then
        $Dist(y) \leftarrow Dist(x) + W(x, y)$;
        $Pred(y) \leftarrow x$;
      end
    end
  end
  $k \leftarrow k + 1$;
end

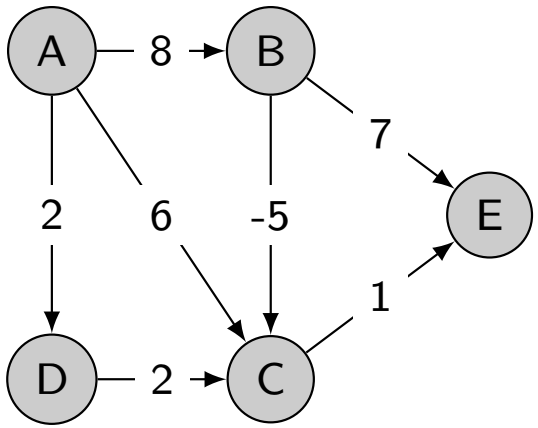
Algorithme de Bellman

Cas où le graphe n'admet pas de circuit.



Algorithme de Bellman

Cas où le graphe n'admet pas de circuit.

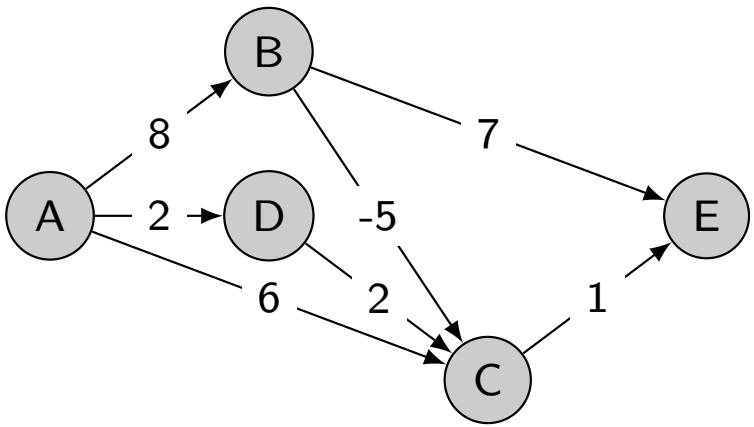


Data: Un graphe orienté $G = (S, A)$
Result: Une numérotation des sommet
$r : S \to \mathbb{N}$

$k \leftarrow 1$;
while $k < n$ **do**
  for $x \in S$ *dont tous les prédécesseurs sont numéroté* **do**
    $r(x) \leftarrow k$;
  end
  $k \leftarrow k + 1$;
end

Algorithme de Bellman

Cas où le graphe n'admet pas de circuit.

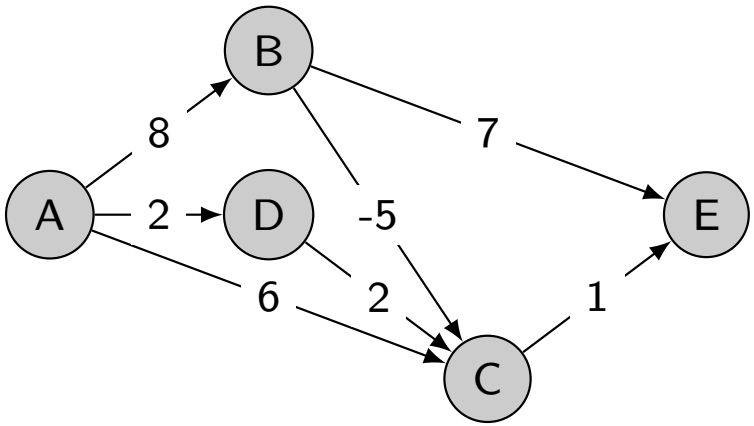


Data: Un graphe orienté $G = (S, A)$
Result: Une numérotation des sommet
$r : S \to \mathbb{N}$

$k \leftarrow 1$;
while $k < n$ **do**
  for $x \in S$ *dont tous les prédécesseurs sont numéroté* **do**
    $r(x) \leftarrow k$;
  end
  $k \leftarrow k + 1$;
end

Algorithme de Bellman

Cas où le graphe n'admet pas de circuit

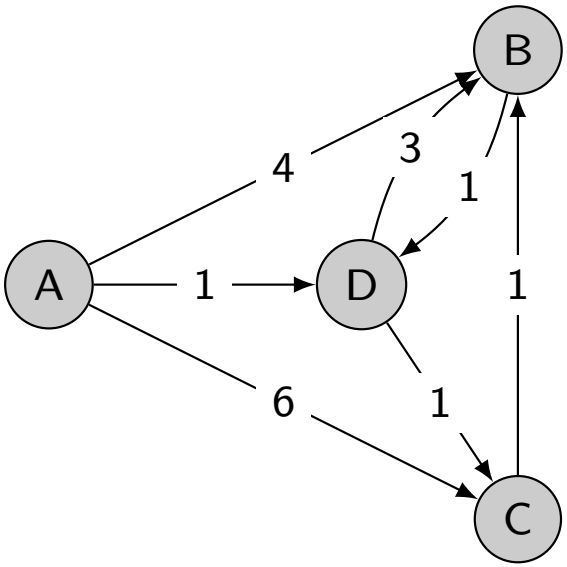


Data: W matrice des distances et $s \in S$
et $r : S \to \mathbb{N}$ une fonction rang
Data: *Dist* tableau des distance à s

Pred tableau des prédécesseur initialisé à $\varnothing$;
Dist tableau des distances initialisé à $+\infty$;
W matrice des poids des arcs;
$Dist(s) \leftarrow 0$;
for $k = r(s) + 1$ *jusqu'à* $\max(r)$ **do**
  for $y \in S$ *tel que* $r(y) = k$ **do**
    for x *prédecesseur de* y **do**
      if $Dist(x) + W(x, y) < Dist(y)$
      then
        $Dist(y) \leftarrow Dist(x) + W(x, y)$;
        $Pred(y) \leftarrow x$;
      end
    end
  end
end

Algorithme de Dijkstra-Moore

Cas où les valuations sont positives.



Data: W matrice des distances et $s \in S$
Data: $Dist$ tableau des distance à s

$Pred$ tableau des prédécesseur initialisé à $\varnothing$;
$Dist$ tableau des distances initialisé à $+\infty$;
$Dist(s) \leftarrow 0$;
$D \leftarrow \varnothing$ (listes des sommets déjà traités);
$k \leftarrow 1$;
while $D \neq S$ **do**
  $x \leftarrow$ sommet de $S \setminus D$ tel que $Dist(x)$ minimal;
  $D \leftarrow \{x\} \cup D$;
  for $y \notin D$ *et* y *successeur de* x **do**
    if $Dist(x) + W(x, y) < Dist(y)$ **then**
      $Dist(y) \leftarrow Dist(x) + W(x, y)$;
      $Pred(y) \leftarrow x$;
    end
  end
end

Résumé

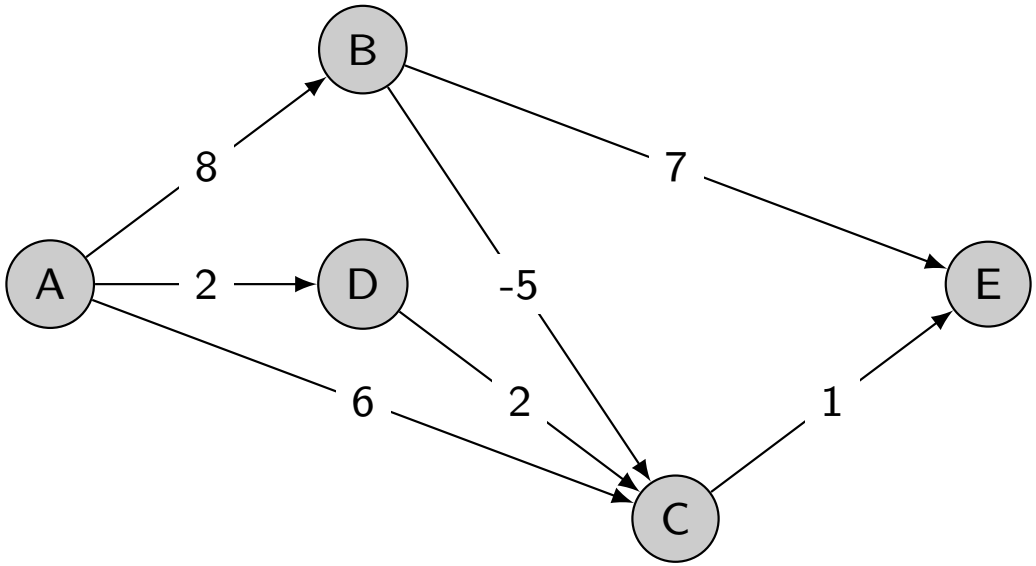
Algorithme	Type de graphe	Complexité
Bellman-Ford	tout type de graphe	$O(n^3)$
Bellman	graphe sans circuit	$O(n^2)$
Dijkstra-Moore	graphe de valuation positive	$O(n^2)$

Un exemple d'ordonnancement

Tâches	Opérations et contraintes	Durée en jour
A	Début du projet	1
B	commence au plus tôt 7 jour après la fin de la tâche A commence au plus tard 5 jour après le début de la tâche C	3
C	commence au plus tôt 6 jour après le début de la tâche A commence au plus tôt 1 jour après la fin de la tâche D	1
D	commence au plus tôt 1 jour après la fin de la tâche A	1
E	commence après la fin de la tâche C commence 4 jours après la fin de la tâche B Fin du projet	0

Un exemple d'ordonnancement

Si on traduit les contraintes sur un graphe, on obtient:

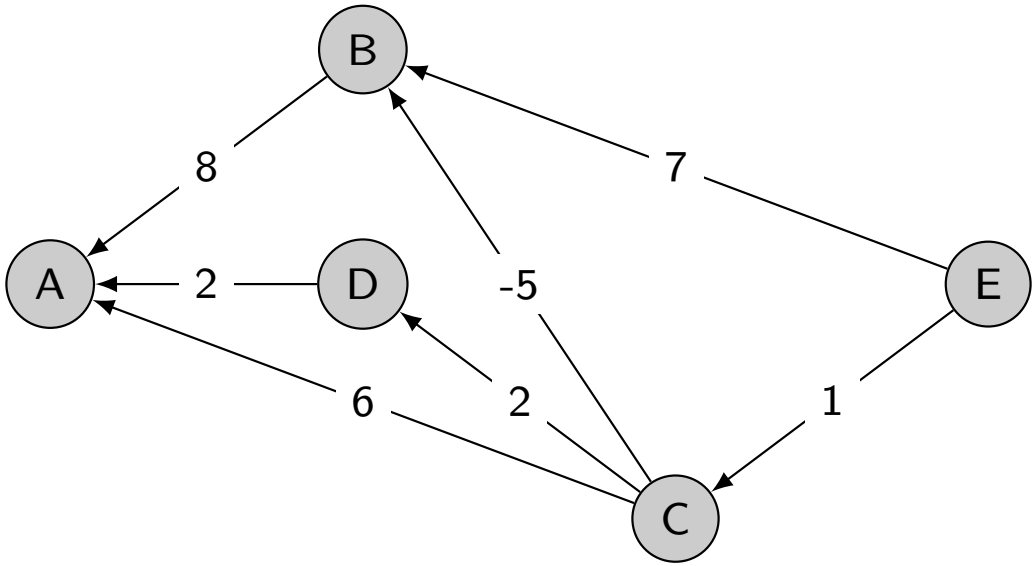


Si on recherche l'ordonnancement au plus tôt, cela revient à chercher les chemins maximaux. On décompose en niveau et on utilise Bellmon simplifié. Pour l'exemple on obtient:

	A	B	C	D	E
Dist	0	8	2	6	15
Pred	$\varnothing$	A	A	A	B

Un exemple d'ordonnancement

Si l'on veut réaliser le projet en 17 jours au plus et que l'on cherche l'ordonnancement au plus tard on inverse les arêtes et on applique Bellman pour rechercher les chemins les plus longs (attention les niveaux peuvent changer).



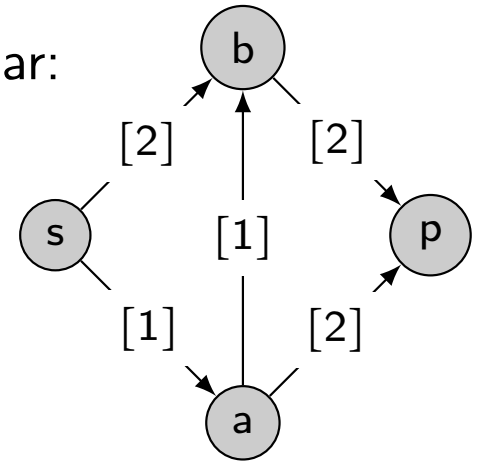
	A	B	C	D	E
Dist	15	7	1	3	0
Pred	B	E	E	C	$\varnothing$

Ainsi la tâche A doit être faite dans $[0, 2]$, la tâche B doit être faite dans $[8, 10]$, la tâche C doit être faite dans $[2, 16]$, la tâche D doit être faite dans $[6, 15]$ et la tâche E doit être faite dans $[15, 17]$.

5.3 Flots dans les transports

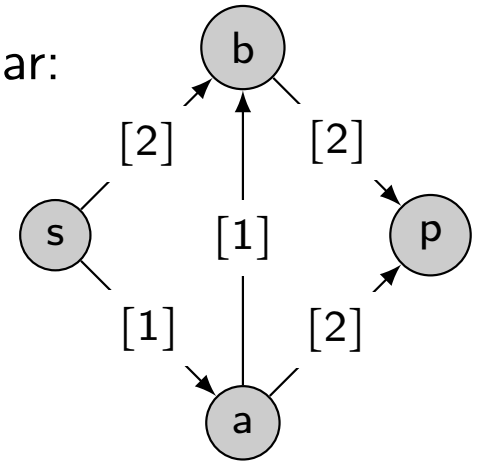
Problématique

- Un *réseau de transport* $R = (S, A, s, p, c)$ est définit par:
  - un graphe orienté $G = (S, A)$ sans circuit,
  - un sommet $s \in S$ appelé source;
  - un sommet $p \in S$ appelé puits;
  - une fonction capacité $c : A \rightarrow]0, +\infty[$;
  - il existe au moins un chemin de s à p.



Problématique

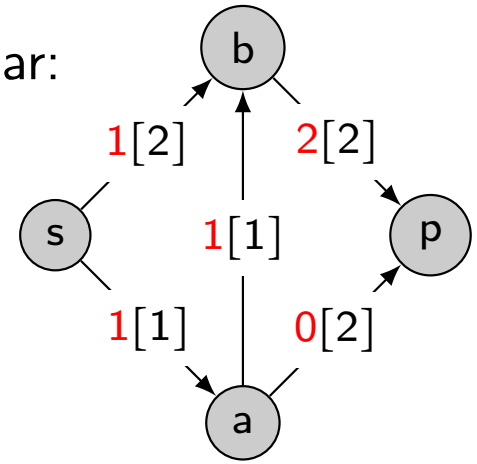
- Un *réseau de transport* $R = (S, A, s, p, c)$ est définit par:
  - un graphe orienté $G = (S, A)$ sans circuit,
  - un sommet $s \in S$ appelé source;
  - un sommet $p \in S$ appelé puits;
  - une fonction capacité $c : A \rightarrow]0, +\infty[$;
  - il existe au moins un chemin de s à p.



- Exemple : Réseau routier, réseau de distribution d'eau, électricité, etc...

Problématique

- Un *réseau de transport* $R = (S, A, s, p, c)$ est définit par:
  - un graphe orienté $G = (S, A)$ sans circuit,
  - un sommet $s \in S$ appelé source;
  - un sommet $p \in S$ appelé puits;
  - une fonction capacité $c : A \to]0, +\infty[$;
  - il existe au moins un chemin de s à p.



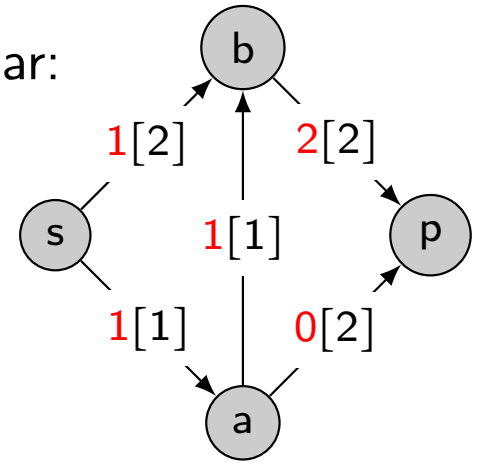
- Exemple : Réseau routier, réseau de distribution d'eau, électricité, etc...

- Un *flot* sur R est une fonction $f : A \to \mathbb{R}_+$ telle que
  - pour tout arcs $a \in A$, $f(a) \leq c(a)$;
  - pour tout $s \in S$ et $x \in S \setminus \{s, p\}$, on a

$$\underbrace{\sum_{z \in \mathrm{Pred}(x)} f(z, x)}_{\text{flot entrant dans } x} = \underbrace{\sum_{y \in \mathrm{Succ}(x)} f(x, y)}_{\text{flot sortant de } x}$$

Problématique

- Un *réseau de transport* $R = (S, A, s, p, c)$ est définit par:
  - un graphe orienté $G = (S, A)$ sans circuit,
  - un sommet $s \in S$ appelé source;
  - un sommet $p \in S$ appelé puits;
  - une fonction capacité $c : A \to]0, +\infty[$;
  - il existe au moins un chemin de s à p.



- Exemple : Réseau routier, réseau de distribution d'eau, électricité, etc...

- Un *flot* sur R est une fonction $f : A \to \mathbb{R}_+$ telle que
  - pour tout arcs $a \in A$, $f(a) \leq c(a)$;
  - pour tout $s \in S$ et $x \in S \setminus \{s, p\}$, on a

$$\underbrace{\sum_{z \in \mathrm{Pred}(x)} f(z, x)}_{\text{flot entrant dans } x} = \underbrace{\sum_{y \in \mathrm{Succ}(x)} f(x, y)}_{\text{flot sortant de } x}$$

- On appelle *valeur* du flot la quantité

$$v(f) = \sum_{y \in \mathrm{Succ}(s)} f(s, y) = \sum_{z \in \mathrm{Pred}(p)} f(z, p)$$

Un arc $a \in A$ est *saturé* par le flot f si $f(a) = c(a)$.

Résultats

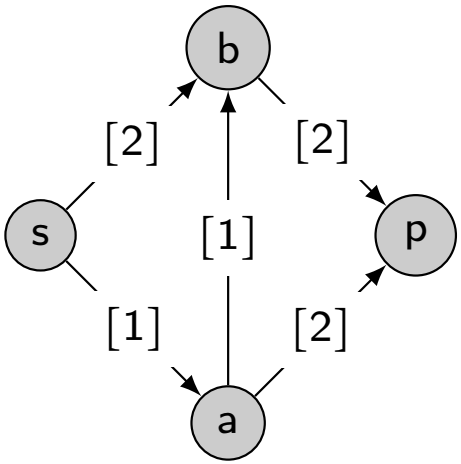
- Une *coupe* sur un réseau de transport R est un sous-ensemble X de S tel que $s \in X$ et $p \notin X$.
  On définit alors la *capacité* d'une coupe comme la somme des capacités des arcs allant de X à $S \setminus X$ noté par

$$C(X) = \sum_{(x,y) \in A, x \in X, y \in \overline{X}} c(x, y)$$

Résultats

- Une *coupe* sur un réseau de transport R est un sous-ensemble X de S tel que $s \in X$ et $p \notin X$.
  On définit alors la *capacité* d'une coupe comme la somme des capacités des arcs allant de X à $S \setminus X$ noté par

$$C(X) = \sum_{(x,y)\in A, x\in X, y\in \overline{X}} c(x,y)$$



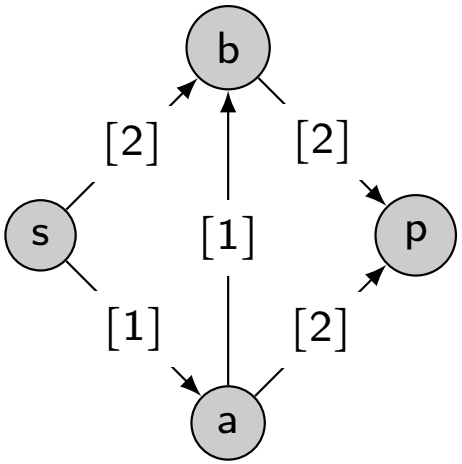
Les coupes possibles de l'exemple sont:

X	$\overline{X}$	$C(X)$
$\{s\}$	$\{a,b,p\}$	3
$\{s,a\}$	$\{b,p\}$	5
$\{s,b\}$	$\{a,p\}$	3
$\{s,a,b\}$	$\{p\}$	4

Résultats

- Une *coupe* sur un réseau de transport R est un sous-ensemble X de S tel que $s \in X$ et $p \notin X$.
  On définit alors la *capacité* d'une coupe comme la somme des capacités des arcs allant de X à $S \setminus X$ noté par

$$C(X) = \sum_{(x,y)\in A, x\in X, y\in \overline{X}} c(x,y)$$



Les coupes possibles de l'exemple sont:

X	$\overline{X}$	$C(X)$
$\{s\}$	$\{a,b,p\}$	3
$\{s,a\}$	$\{b,p\}$	5
$\{s,b\}$	$\{a,p\}$	3
$\{s,a,b\}$	$\{p\}$	4

Théorème

Soit $R = (S, A, s, p, c)$ un réseau de transport, il existe un flot maximal f tel que

$$v(f) = \min_{\text{coupe } X} C(X).$$

Algorithme de Ford-Fulkerson

Data: Un réseau de transport $R = (S, A, s, p, c)$
Result: Un flot maximal

$f \leftarrow$ flot de départ (éventuellement nul);
while *il existe un chemin* γ *augmentant* **do**
  augmenter f le long du chemin γ;
end