Algorithmique et structure de données

Algorithme glouton et arbre couvrant

Rohan Fossé

10 Novembre 2021

Algorithmes gloutons

Problème d'optimisation

Un problème d'optimisation a les caractéristiques suivantes :

- Une solution est un **sous-ensemble** d'une des données du problème ;
- Il existe en général plusieurs solutions **admissibles** ;
- A chaque solution (admissible) est associée une **valeur** (en général un coût ou un gain).

Le problème d'optimisation consiste en non seulement à trouver une solution admissible, mais à trouver une solution de valeur **minimale** (pour un coût) ou **maximale** pour un gain.

Problème d'optimisation

Problème d'optimisation

Un problème d'optimisation a les caractéristiques suivantes :

- Une solution est un **sous-ensemble** d'une des données du problème ;
- Il existe en général plusieurs solutions **admissibles** ;
- A chaque solution (admissible) est associée une **valeur** (en général un coût ou un gain).

Le problème d'optimisation consiste en non seulement à trouver une solution admissible, mais à trouver une solution de valeur **minimale** (pour un coût) ou **maximale** pour un gain.

un exemple : rendu de monnaie

- **Problème :** on dispose de pièces de valeurs $v_1, v_2, .., v_n$;
- **Solution :** une suite de (valeurs de) pièces ayant pour total S ;
- **Meilleure solution :** celle utilisant le moins possible de pièces.

Algorithme gloutons

Il s'agit d'une *stratégie particulière* pour résoudre un problème d'optimisation.

Algorithme glouton
Un algorithme glouton est un algorithme qui construit une solution :

- élément par élément **sans jamais revenir en arrière** ;
- en se basant sur des considérations locales.

Algorithme gloutons

Il s'agit d'une *stratégie particulière* pour résoudre un problème d'optimisation.

Algorithme glouton

Un algorithme glouton est un algorithme qui construit une solution :

- élément par élément **sans jamais revenir en arrière** ;
- en se basant sur des considérations locales.

Attention

Il n'existe **pas toujours** un algorithme glouton pour résoudre un problème d'optimisation.

Algorithme glouton pour le rendu de monnaie

Algorithme glouton

1. Tant que S > 0 ;
2. Choisir la pièce de plus grande valeur v inférieur à S ;
3. Recommencer avec S - v.

Algorithme glouton pour le rendu de monnaie

Algorithme glouton

1. Tant que S > 0 ;
2. Choisir la pièce de plus grande valeur v inférieur à S ;
3. Recommencer avec S - v.

Exemple correct

S=16 avec pièces de 10, 5, 2, 1 :

- 16 - 10 = 6
- 6 - 5 = 1
- 1 - 1 = 0

3 pièces

Algorithme glouton pour le rendu de monnaie

Algorithme glouton

1. Tant que S > 0 ;
2. Choisir la pièce de plus grande valeur v inférieur à S ;
3. Recommencer avec S - v.

Exemple correct

S=16 avec pièces de 10, 5, 2, 1 :

- 16 - 10 = 6
- 6 - 5 = 1
- 1 - 1 = 0

3 pièces

Exemple incorrect

S=16 avec pièces de 9, 8 et 1 :

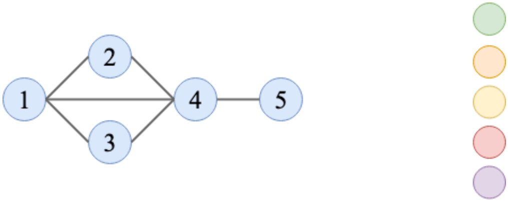
- 16 - 9 = 7
- 7 - 1 = 6
- ...

8 pièces alors que 8 + 8 = 16 en 2 pièces

Idée de l'algorithme

On se donne un graphe G = (V , E). Pour tout $v \in V$:

- On regarde l'ensemble des couleurs déjà attribuées aux voisins de s.
- On en déduit le plus petit entier naturel qui n'appartient pas à cet ensemble.
- On attribue cette couleur à s.

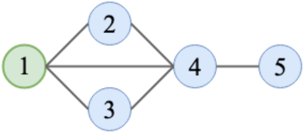


Coloration d'un graphe à l'aide d'un algorithme glouton

Idée de l'algorithme

On se donne un graphe G = (V , E). Pour tout $v \in V$:

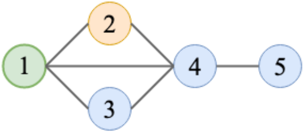
- On regarde l'ensemble des couleurs déjà attribuées aux voisins de s.
- On en déduit le plus petit entier naturel qui n'appartient pas à cet ensemble.
- On attribue cette couleur à s.



Idée de l'algorithme

On se donne un graphe G = (V , E). Pour tout $v \in V$:

- On regarde l'ensemble des couleurs déjà attribuées aux voisins de s.
- On en déduit le plus petit entier naturel qui n'appartient pas à cet ensemble.
- On attribue cette couleur à s.

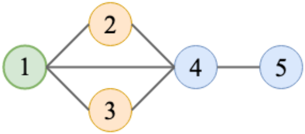


Coloration d’un graphe à l’aide d’un algorithme glouton

Idée de l’algorithme

On se donne un graphe G = (V , E). Pour tout $v \in V$:

- On regarde l’ensemble des couleurs déjà attribuées aux voisins de s.
- On en déduit le plus petit entier naturel qui n’appartient pas à cet ensemble.
- On attribue cette couleur à s.

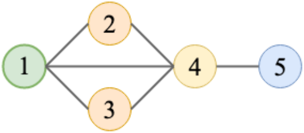


Coloration d'un graphe à l'aide d'un algorithme glouton

Idée de l'algorithme

On se donne un graphe G = (V , E). Pour tout $v \in V$:

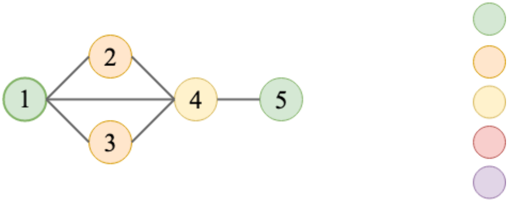
- On regarde l'ensemble des couleurs déjà attribuées aux voisins de s.
- On en déduit le plus petit entier naturel qui n'appartient pas à cet ensemble.
- On attribue cette couleur à s.



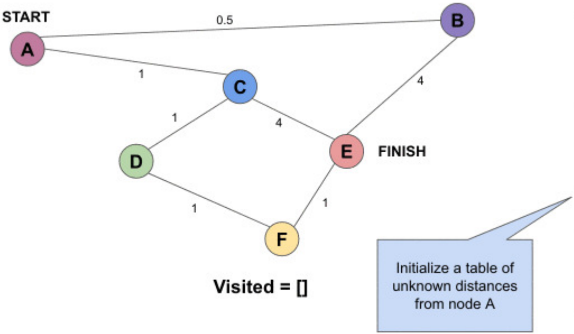
Idée de l'algorithme

On se donne un graphe G = (V , E). Pour tout $v \in V$:

- On regarde l'ensemble des couleurs déjà attribuées aux voisins de s.
- On en déduit le plus petit entier naturel qui n'appartient pas à cet ensemble.
- On attribue cette couleur à s.



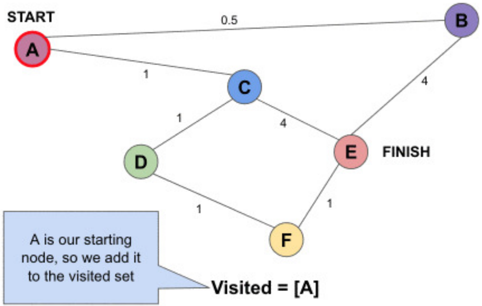
Rappel de l'algorithme de Dijsktra



	Distance	Last Node
A	0	
B	?	?
C	?	?
D	?	?
E	?	?
F	?	?

<u>Source :</u> *https ://themergedsort.com/ ?p=261*

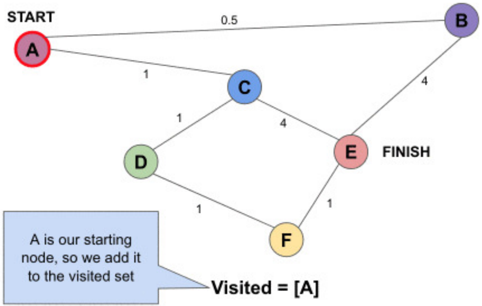
Algorithme de Dijsktra



	Distance	Last Node
A	0	
B	?	?
C	?	?
D	?	?
E	?	?
F	?	?

Source : *https ://themergedsort.com/ ?p=261*

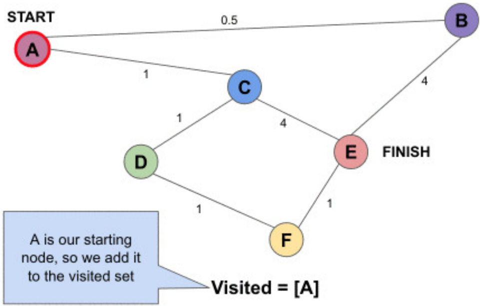
Algorithme de Dijsktra



	Distance	Last Node
A	0	
B	?	?
C	?	?
D	?	?
E	?	?
F	?	?

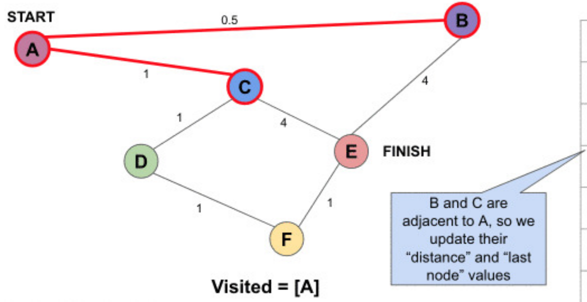
Source : *https ://themergedsort.com/ ?p=261*

Algorithme de Dijkstra



	Distance	Last Node
A	0	
B	?	?
C	?	?
D	?	?
E	?	?
F	?	?

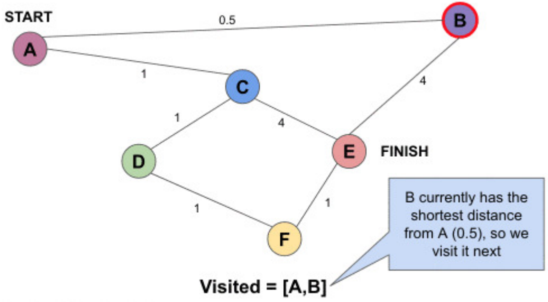
Source : *https ://themergedsort.com/ ?p=261*



	Distance	Last Node
A	0	
B	**0.5**	**A**
C	**1**	**A**
D	?	?
E	?	?
F	?	?

Source : *https ://themergedsort.com/ ?p=261*

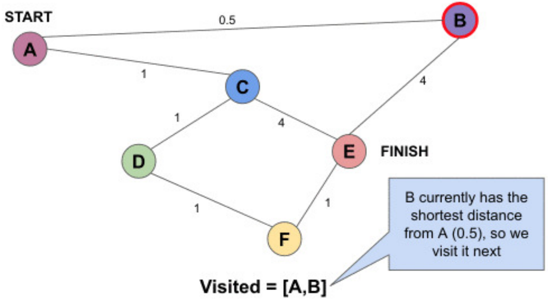
Algorithme de Dijsktra



	Distance	Last Node
A	0	
B	0.5	A
C	1	A
D	?	?
E	?	?
F	?	?

Source : *https ://themergedsort.com/ ?p=261*

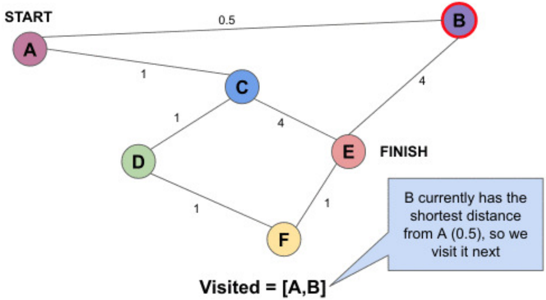
Algorithme de Dijsktra



	Distance	Last Node
A	0	
B	0.5	A
C	1	A
D	?	?
E	?	?
F	?	?

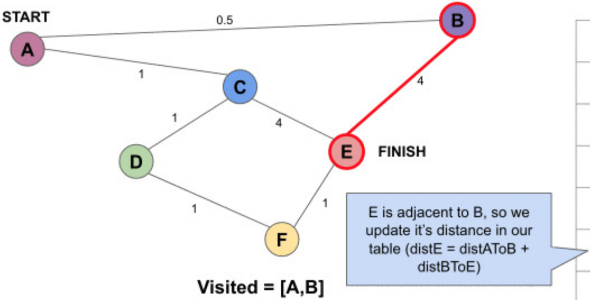
Source : *https ://themergedsort.com/ ?p=261*

Algorithme de Dijsktra



	Distance	Last Node
A	0	
B	0.5	A
C	1	A
D	?	?
E	?	?
F	?	?

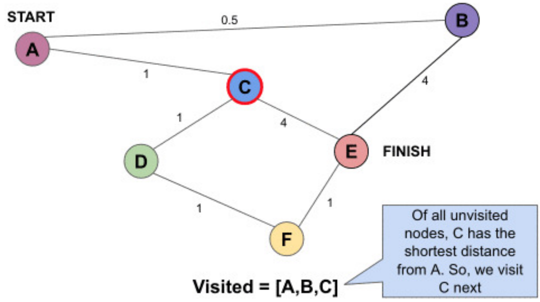
Source : *https ://themergedsort.com/ ?p=261*



	Distance	Last Node
A	0	
B	0.5	A
C	1	A
D	?	?
E	**4.5**	**B**
F	?	?

Source : *https ://themergedsort.com/ ?p=261*

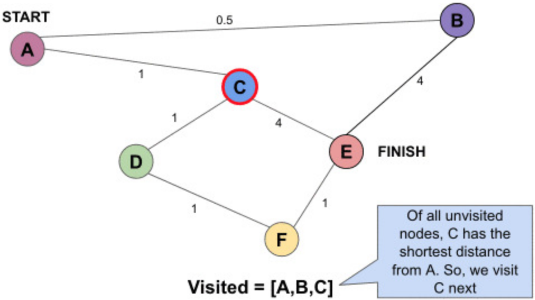
Algorithme de Dijsktra



	Distance	Last Node
A	0	
B	0.5	A
C	1	A
D	?	?
E	4.5	B
F	?	?

Source : *https ://themergedsort.com/ ?p=261*

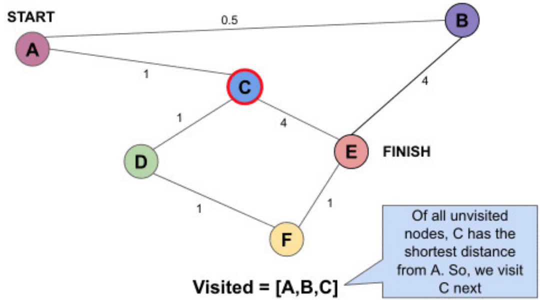
Algorithme de Dijsktra



	Distance	Last Node
A	0	
B	0.5	A
C	1	A
D	?	?
E	4.5	B
F	?	?

Source : *https ://themergedsort.com/ ?p=261*

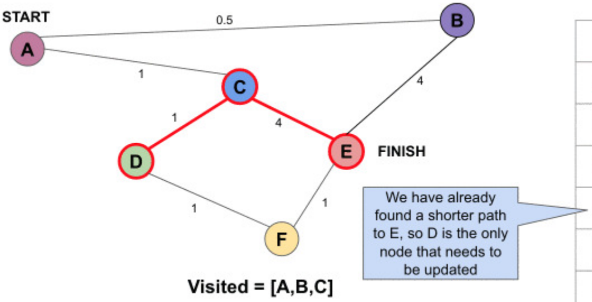
Algorithme de Dijsktra



	Distance	Last Node
A	0	
B	0.5	A
C	1	A
D	?	?
E	4.5	B
F	?	?

<u>Source :</u> *https ://themergedsort.com/ ?p=261*

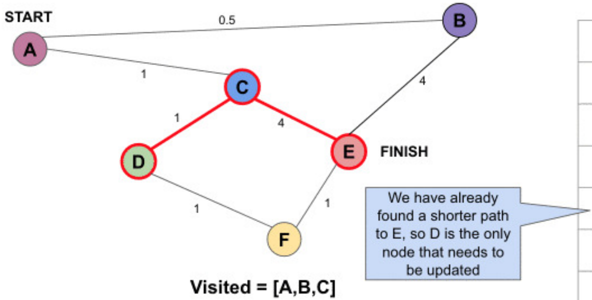
Algorithme de Dijsktra



	Distance	Last Node
A	0	
B	0.5	A
C	1	A
D	**2**	**C**
E	4.5	B
F	?	?

Source : *https ://themergedsort.com/ ?p=261*

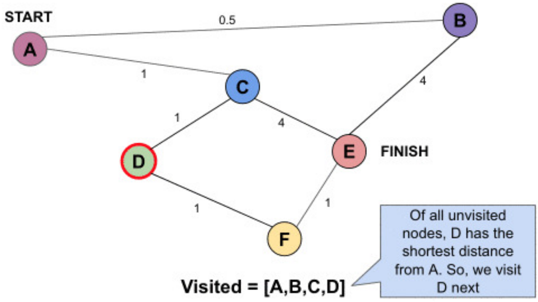
Algorithme de Dijkstra



	Distance	Last Node
A	0	
B	0.5	A
C	1	A
D	**2**	**C**
E	4.5	B
F	?	?

Source : *https ://themergedsort.com/ ?p=261*

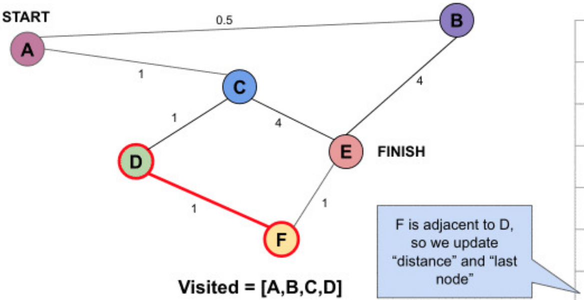
Algorithme de Dijsktra



	Distance	Last Node
A	0	
B	0.5	A
C	1	A
D	2	C
E	4.5	B
F	3	D

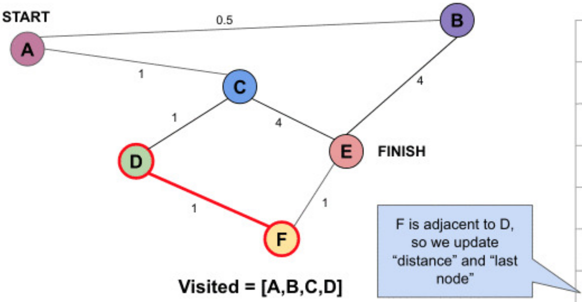
Source : *https ://themergedsort.com/ ?p=261*

Algorithme de Dijsktra



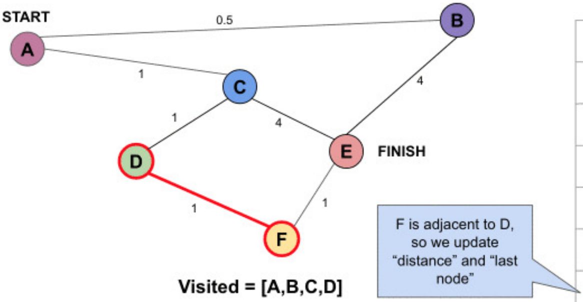
	Distance	Last Node
A	0	
B	0.5	A
C	1	A
D	2	C
E	4.5	B
F	**3**	**D**

Source : *https ://themergedsort.com/ ?p=261*



	Distance	Last Node
A	0	
B	0.5	A
C	1	A
D	2	C
E	4.5	B
F	**3**	**D**

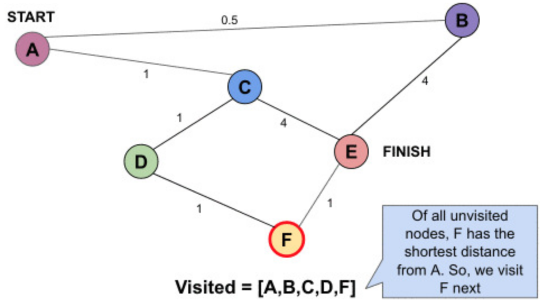
Source : *https ://themergedsort.com/ ?p=261*



	Distance	Last Node
A	0	
B	0.5	A
C	1	A
D	2	C
E	4.5	B
F	**3**	**D**

Source : *https ://themergedsort.com/ ?p=261*

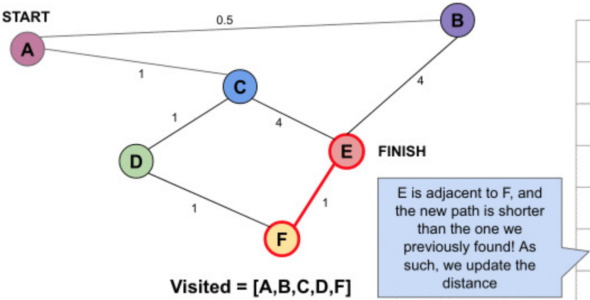
Algorithme de Dijsktra



	Distance	Last Node
A	0	
B	0.5	A
C	1	A
D	2	C
E	4.5	B
F	**3**	**D**

Source : *https ://themergedsort.com/ ?p=261*

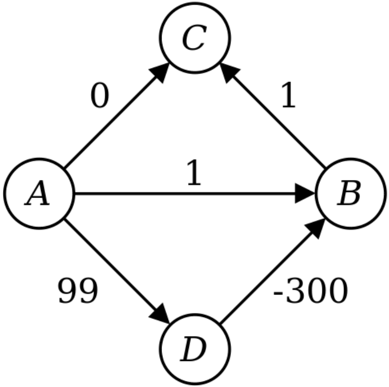
Algorithme de Dijsktra



	Distance	Last Node
A	0	
B	0.5	A
C	1	A
D	2	C
E	**4**	**F**
F	3	D

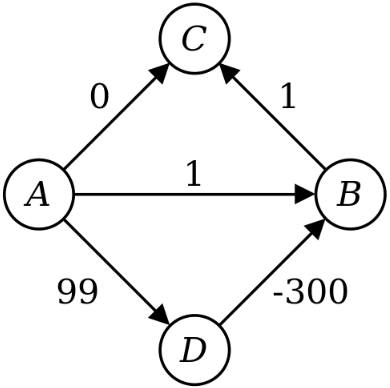
Source : *https ://themergedsort.com/ ?p=261*

Considérons le graphe suivant :



- Tout d'abord, vous fixez d(A) à 0 et les autres distances à ∞.

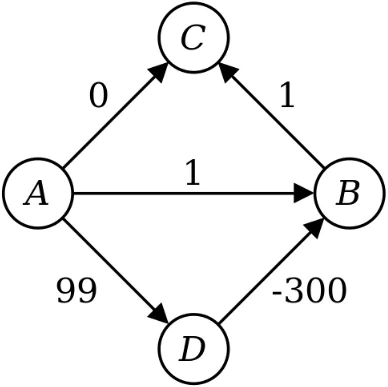
Considérons le graphe suivant :



- Tout d'abord, vous fixez d(A) à 0 et les autres distances à ∞.
- Vous développez ensuite le nœud A, en fixant d(B) à 1, d(C) à 0 et d(D) à 99.

Problème avec Dijsktra

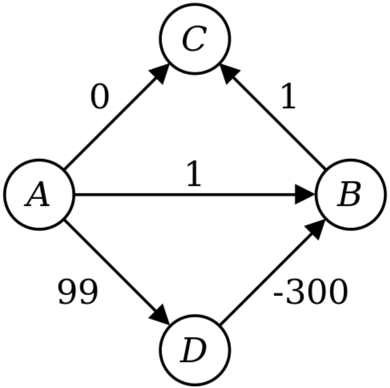
Considérons le graphe suivant :



- Tout d'abord, vous fixez d(A) à 0 et les autres distances à ∞.
- Vous développez ensuite le nœud A, en fixant d(B) à 1, d(C) à 0 et d(D) à 99.
- Ensuite, vous développez le nœud C ;

Problème avec Dijsktra

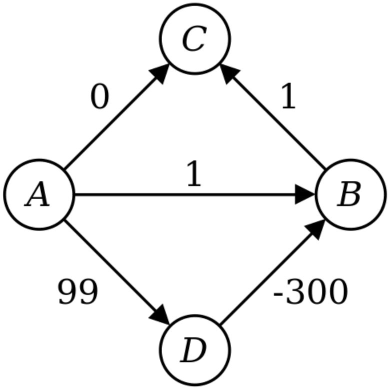
Considérons le graphe suivant :



- Tout d'abord, vous fixez d(A) à 0 et les autres distances à ∞.
- Vous développez ensuite le nœud A, en fixant d(B) à 1, d(C) à 0 et d(D) à 99.
- Ensuite, vous développez le nœud C ;
- Vous développez ensuite le nœud B, ce qui n'a aucun effet.

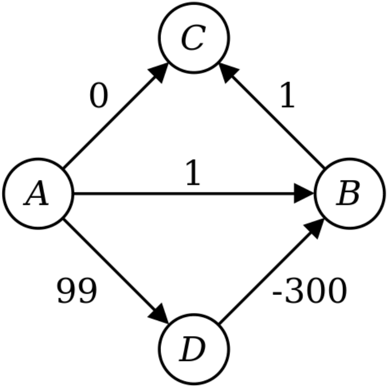
Problème avec Dijsktra

Considérons le graphe suivant :



- Tout d'abord, vous fixez d(A) à 0 et les autres distances à ∞.
- Vous développez ensuite le nœud A, en fixant d(B) à 1, d(C) à 0 et d(D) à 99.
- Ensuite, vous développez le nœud C ;
- Vous développez ensuite le nœud B, ce qui n'a aucun effet.
- Enfin, vous développez D, ce qui fait passer d(B) à -201.

Considérons le graphe suivant :



Remarquez cependant qu'à la fin, d(C) est toujours égal à 0, même si le plus court chemin vers C a une longueur de -200. Cela signifie que l'algorithme ne calcule pas les distances correctes vers tous les nœuds.

De plus, même si vous stockez des pointeurs indiquant comment aller de chaque nœud au nœud de départ A, vous finirez par prendre le mauvais chemin de retour de C à A.

Algorithme de Bellman-Ford

Algorithme de Bellman-Ford

Histoire

L'algorithme de Bellman-Ford est un algorithme qui calcule des plus courts chemins depuis un sommet source donné dans un graphe orienté pondéré. Il porte le nom de ses inventeurs *Richard Bellman* et *Lester Randolph Ford junior*.

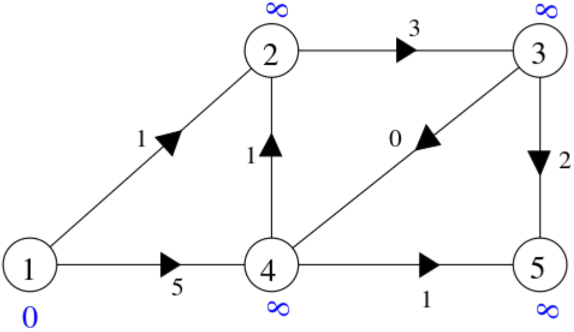
Histoire

L'algorithme de Bellman-Ford est un algorithme qui calcule des plus courts chemins depuis un sommet source donné dans un graphe orienté pondéré. Il porte le nom de ses inventeurs *Richard Bellman* et *Lester Randolph Ford junior*.

Idée de l'algorithme

L'algorithme de Bellman-Ford permet de trouver le plus court chemin d'un sommet donné vers tous les autres sommets. De façon plus précise, à l'initialisation, chaque sommet à une étiquette initialisée à ∞ sauf le sommet qui a une étiquette à 0. Puis, à chaque itération , on calcule le plus court chemin de à tout autre sommet en au plus k arcs.

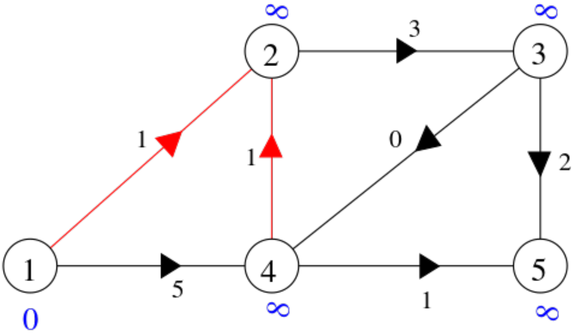
Déroulement sur un exemple



	1	2	3	4	5
V	0	∞	∞	∞	∞

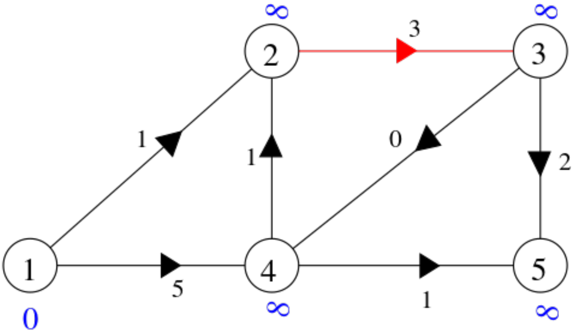
	1	2	3	4	5
P	1	0	0	0	0

Déroulement sur un exemple



	1	2	3	4	5
V	0	∞	∞	∞	∞
	0	1			

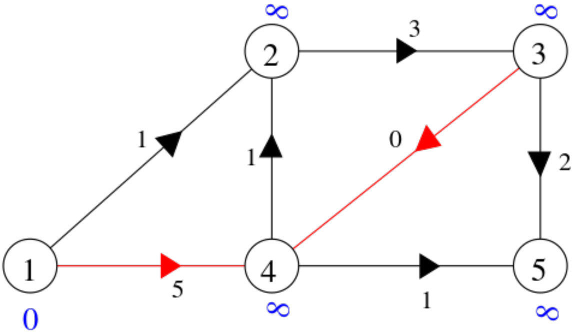
	1	2	3	4	5
P	1	0	0	0	0
	1	1			



	1	2	3	4	5
V	0	∞	∞	∞	∞
	0	1	∞		

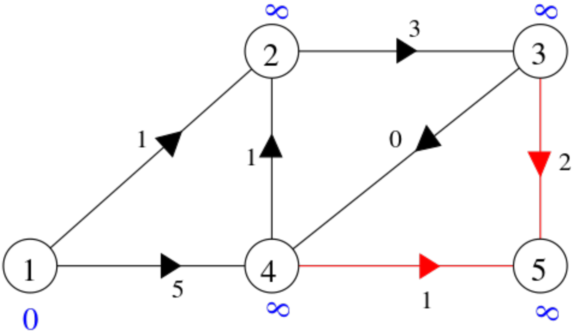
	1	2	3	4	5
P	1	0	0	0	0
	1	1	0		

Déroulement sur un exemple



	1	2	3	4	5
V	0	∞	∞	∞	∞
	0	1	∞	5	

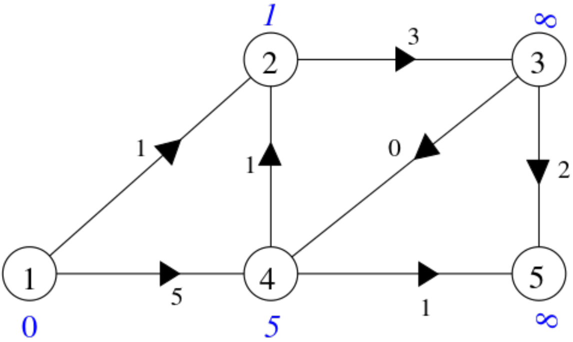
	1	2	3	4	5
P	1	0	0	0	0
	1	1	0	1	



	1	2	3	4	5
V	0	∞	∞	∞	∞
	0	1	∞	5	∞

	1	2	3	4	5
P	1	0	0	0	0
	1	1	0	1	0

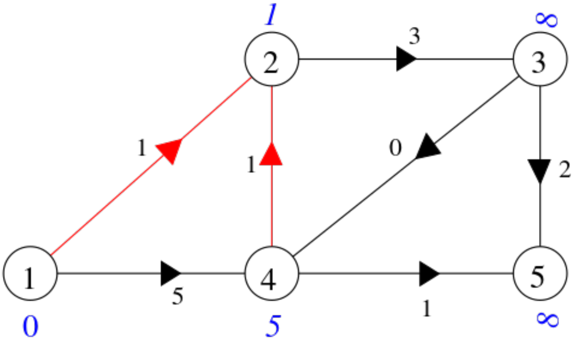
Déroulement sur un exemple



	1	2	3	4	5
V	0	∞	∞	∞	∞
	0	1	∞	5	∞

	1	2	3	4	5
P	1	0	0	0	0
	1	1	0	1	0

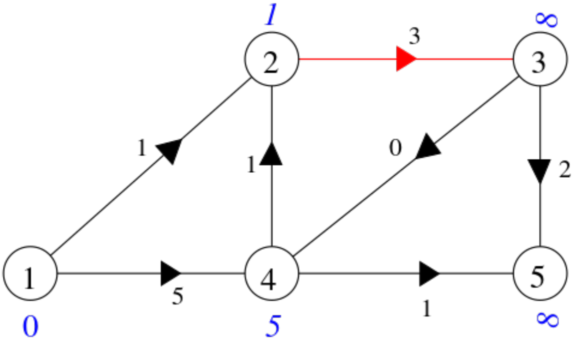
Déroulement sur un exemple



	1	2	3	4	5
V	0	∞	∞	∞	∞
	0	1	∞	5	∞
	0	1			

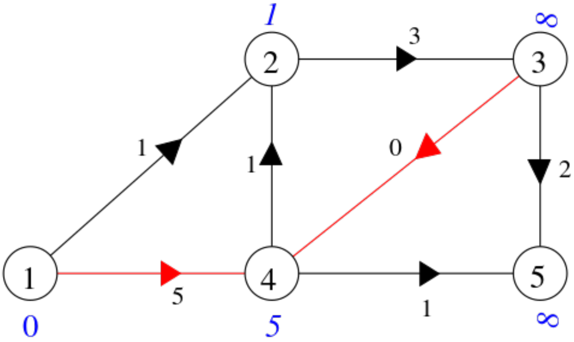
	1	2	3	4	5
P	1	0	0	0	0
	1	1	0	1	0
	1	1			

Déroulement sur un exemple



	1	2	3	4	5
V	0	∞	∞	∞	∞
	0	1	∞	5	∞
	0	1	4		

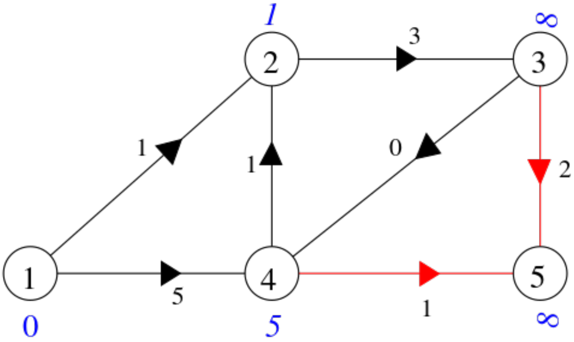
	1	2	3	4	5
P	1	0	0	0	0
	1	1	0	1	0
	1	1	2		



	1	2	3	4	5
V	0	∞	∞	∞	∞
	0	1	∞	5	∞
	0	1	4	5	

	1	2	3	4	5
P	1	0	0	0	0
	1	1	0	1	0
	1	1	2	1	

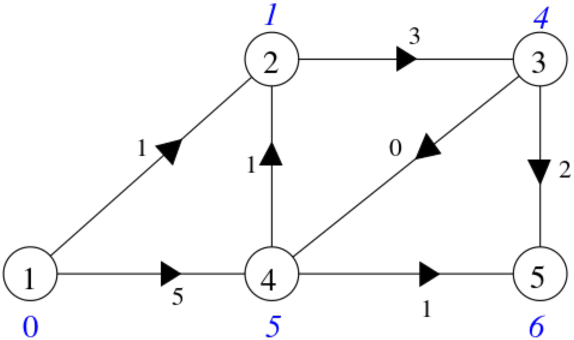
Déroulement sur un exemple



	1	2	3	4	5
V	0	∞	∞	∞	∞
	0	1	∞	5	∞
	0	1	4	5	6

	1	2	3	4	5
P	1	0	0	0	0
	1	1	0	1	0
	1	1	2	1	4

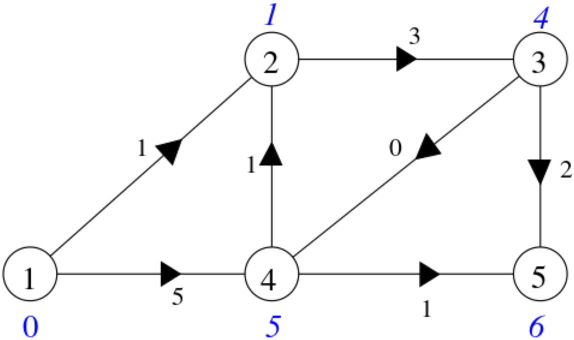
Déroulement sur un exemple



	1	2	3	4	5
V	0	∞	∞	∞	∞
	0	1	∞	5	∞
	0	1	4	5	6

	1	2	3	4	5
P	1	0	0	0	0
	1	1	0	1	0
	1	1	2	1	4

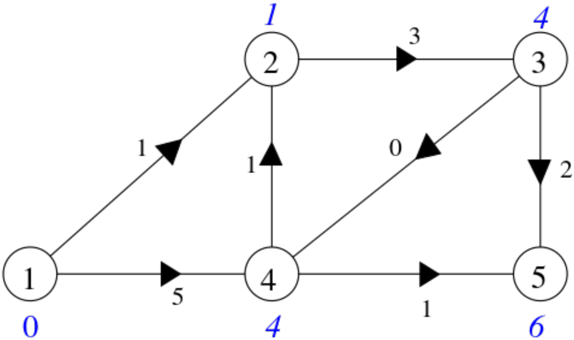
Déroulement sur un exemple



	1	2	3	4	5
V	0	∞	∞	∞	∞
	0	1	∞	5	∞
	0	1	4	5	6
	0	1	4	4	6

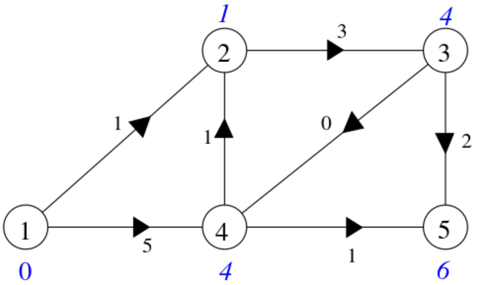
	1	2	3	4	5
P	1	0	0	0	0
	1	1	0	1	0
	1	1	2	1	4
	1	1	2	3	4

Déroulement sur un exemple



	1	2	3	4	5
V	0	∞	∞	∞	∞
	0	1	∞	5	∞
	0	1	4	5	6
	0	1	4	4	6

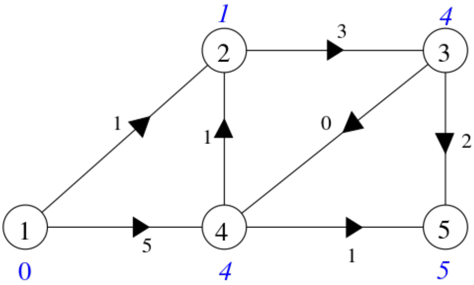
	1	2	3	4	5
P	1	0	0	0	0
	1	1	0	1	0
	1	1	2	1	4
	1	1	2	3	4



	1	2	3	4	5
V	0	∞	∞	∞	∞
	0	1	∞	5	∞
	0	1	4	5	6
	0	1	4	4	6
	0	1	4	4	5

	1	2	3	4	5
P	1	0	0	0	0
	1	1	0	1	0
	1	1	2	1	4
	1	1	2	3	4
	1	1	2	3	4

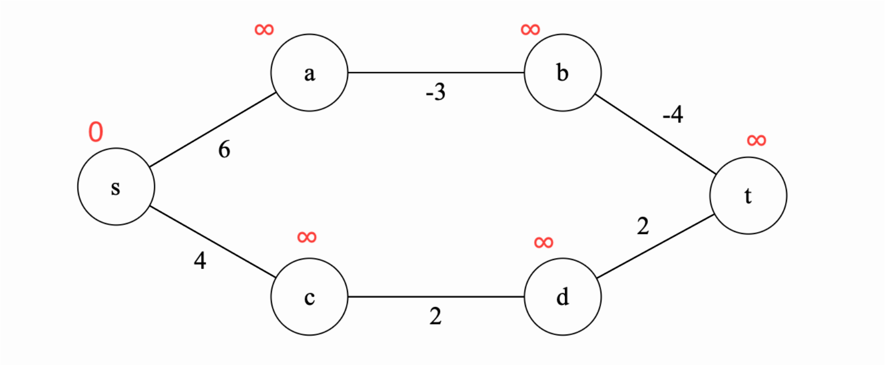
Déroulement sur un exemple



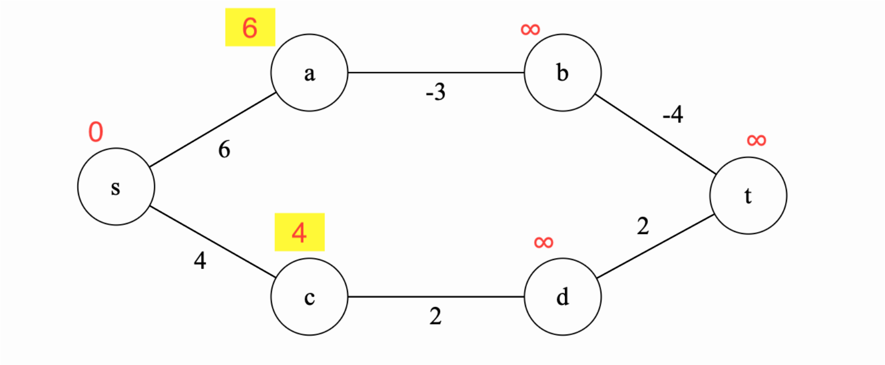
V	1	2	3	4	5
	0	∞	∞	∞	∞
	0	1	∞	5	∞
	0	1	4	5	6
	0	1	4	4	6
	0	1	4	4	5

P	1	2	3	4	5
	1	0	0	0	0
	1	1	0	1	0
	1	1	2	1	4
	1	1	2	3	4
	1	1	2	3	4

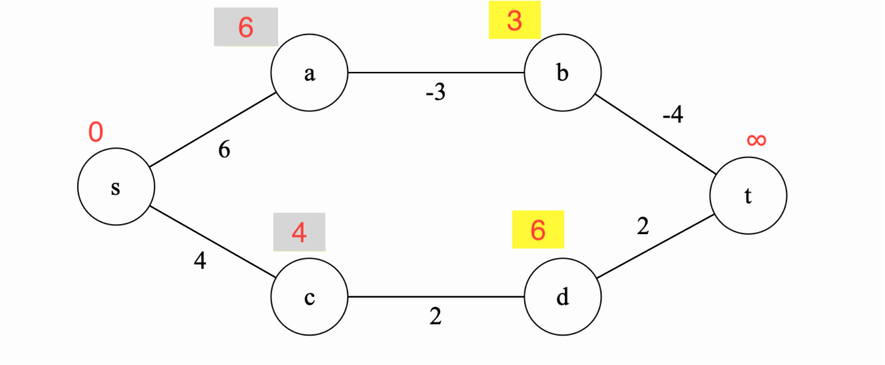
A vous de jouer !



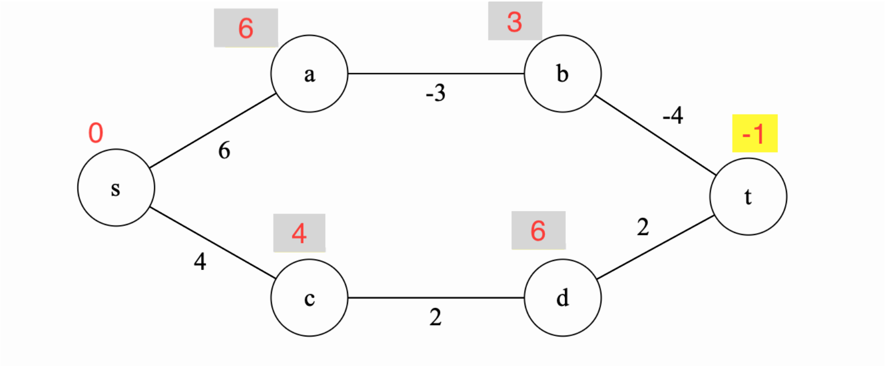
A vous de jouer !



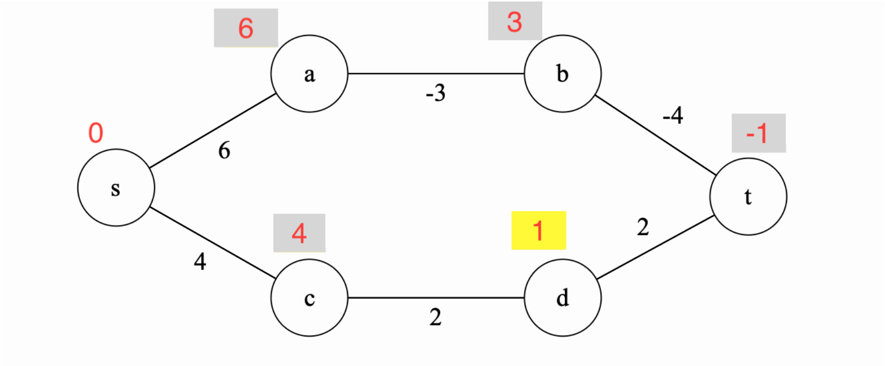
A vous de jouer !



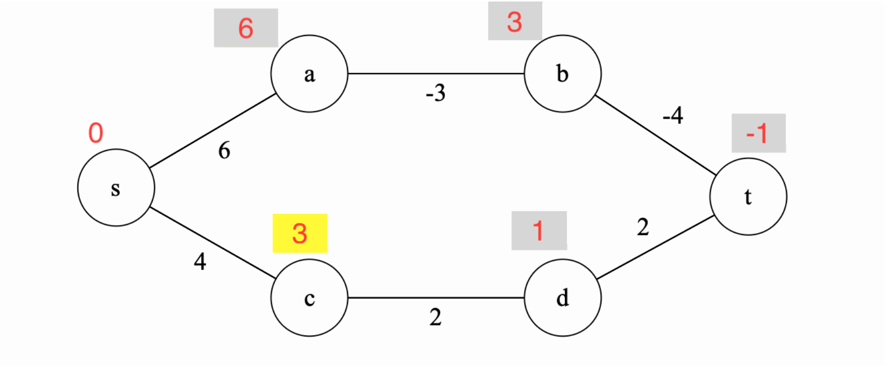
A vous de jouer !



A vous de jouer !



A vous de jouer !



Problème de l’arbre couvrant

Rappel des définitions

Arbre

un arbre est un graphe connexe et sans cycle.

Arbre

un arbre est un graphe connexe et sans cycle.

Attention, la notion est différente des arbres utilisés habituellement en algo :

- Aucune contrainte d'arité ;
- Pas de notion de racine

Rappel des définitions

Arbre

un arbre est un graphe connexe et sans cycle.

Attention, la notion est différente des arbres utilisés habituellement en algo :

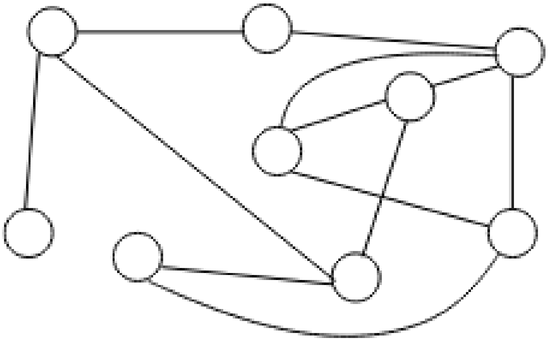
- Aucune contrainte d'arité ;
- Pas de notion de racine

Propriétés

- Entre deux sommets donnés d'un arbre, il existe toujours exactement une chaîne (élémentaire) ;
- Un arbre à n sommets comporte $n-1$ arêtes.

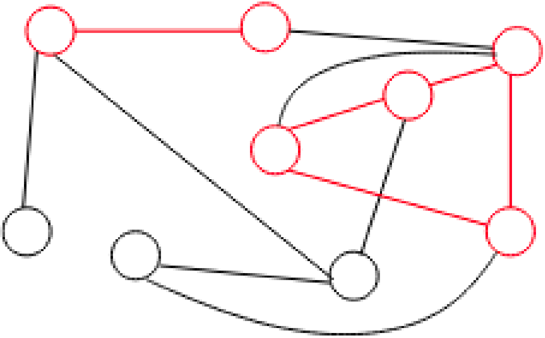
Sous-graphe

Si G = (V, E), un sous-graphe de G est un graphe H = (V' ,E') avec $V' \subseteq X$ et $E' \subseteq E$.



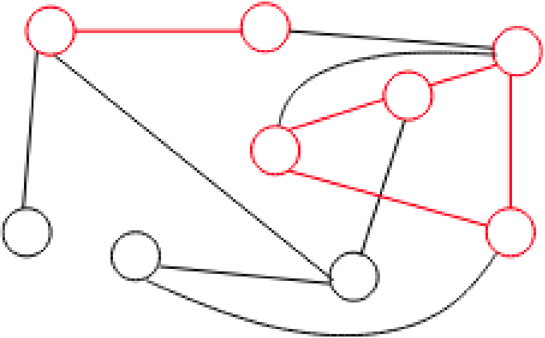
Sous-graphe

Si G = (V, E), un sous-graphe de G est un graphe H = (V' ,E') avec $V' \subseteq X$ et $E' \subseteq E$.



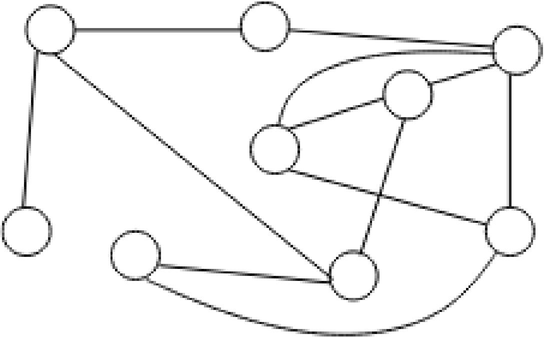
Sous-graphe

Si G = (V, E), un sous-graphe de G est un graphe H = (V' ,E') avec $V' \subseteq X$ et $E' \subseteq E$.

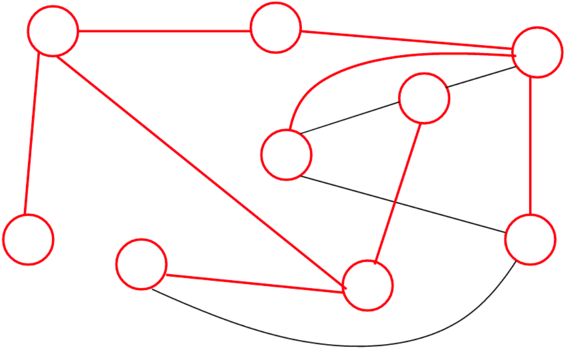


Un sous-graphe de G est dit couvrant s'il contient tous les sommets de G. Attention, un sous-graphe couvrant n'est pas forcément connexe.

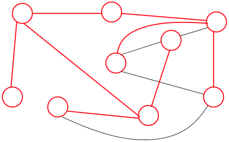
Tout graphe connexe admet un arbre couvrant.



Tout graphe connexe admet un arbre couvrant.



Caractérisation d'un arbre couvrant



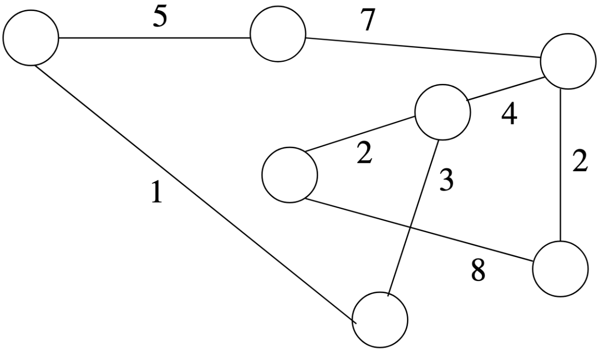
Si A est un sous-graphe couvrant d'un graphe G ayant V sommets, les caractérisations suivantes sont équivalentes :

- A est un arbre couvrant de G ;
- A est sans cycle et possède V - 1 arêtes ;
- A est connexe
- On ne peut pas ajouter une arête à A sans créer un cycle ;
- On ne peut pas retirer une arête à A sans briser sa connexité.

Dans un graphe pondéré

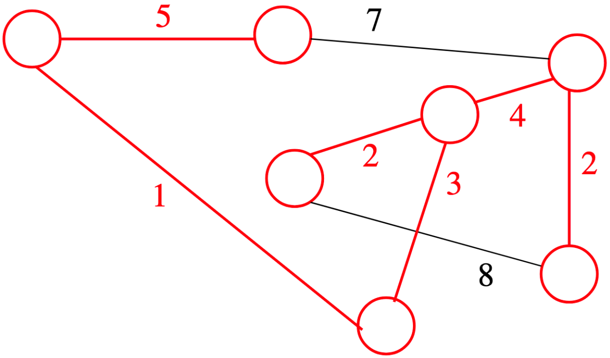
Arbre couvrant de poids minimal

Le poids d'un sous-graphe est la somme des poids des arêtes. Nous cherchons un arbre couvrant de poids minimal (en anglais *Minimum Spanning Tree* (MST))



Arbre couvrant de poids minimal

Le poids d'un sous-graphe est la somme des poids des arêtes. Nous cherchons un arbre couvrant de poids minimal (en anglais *Minimum Spanning Tree* (MST))



Application

Problème

Problème proposé (et résolu) en 1926 par *Otakar Boruvka* pour la construction de réseaux électriques efficaces.

- Les sommets de G représentent des lieux à connecter : villes, ordinateurs, composants électroniques, etc.
- Les arêtes de G représentent les liens (routes, câbles,..) possibles entre ces lieux, avec les coûts effectifs de création de ces liens.
- L'arbre couvrant minimal est le réseau le moins coûteux ne laissant aucun lieu isolé.

Application

Problème

Problème proposé (et résolu) en 1926 par *Otakar Boruvka* pour la construction de réseaux électriques efficaces.

- Les sommets de G représentent des lieux à connecter : villes, ordinateurs, composants électroniques, etc.
- Les arêtes de G représentent les liens (routes, câbles,..) possibles entre ces lieux, avec les coûts effectifs de création de ces liens.
- L'arbre couvrant minimal est le réseau le moins coûteux ne laissant aucun lieu isolé.

Attention : on minimise le coût global du réseau, pas les longeurs des chemins dans l'arbre. Il existe des variantes du problème contraignant la forme de l'arbre obtenu : degré borné pour chaque sommet, diamètre borné, etc..

Algorithmes de détermination d'un arbre couvrant

Deux idées duales :

Première idée	**Deuxième idée**
$A = (V,\varnothing)$ (le graphe vide)	$A = (V,E)$ (le graphe G)

Deux idées duales :

Première idée

$A = (V,\emptyset)$ (le graphe vide)

tant que

faire

Ajouter cette arête à A

Deuxième idée

$A = (V,E)$ (le graphe G)

tant que

faire

Retirer cette arête de A

Deux idées duales :

Première idée

A = (V,∅) (le graphe vide)

tant que

faire

Choisir une arête de G qui ne crée pas de cycle.

Ajouter cette arête à A

Deuxième idée

A = (V,E) (le graphe G)

tant que

faire

Choisir une arête de A qui n'est pas indispensable à la connexité

Retirer cette arête de A

Deux idées duales :

Première idée

A = (V,∅) (le graphe vide)

tant que

faire

Choisir une arête de G qui ne crée pas de cycle.

Ajouter cette arête à A

Deuxième idée

A = (V,E) (le graphe G)

tant que

faire

Choisir une arête de A qui n'est pas indispensable à la connexité

Retirer cette arête de A

Par construction, le graphe A obtenu est un arbre couvrant (en supposant que G soit connexe).

Algorithmes de détermination d'un arbre couvrant

Deux idées duales :

Première idée

$A = (V, \emptyset)$ (le graphe vide)

tant que

faire

Choisir une arête de G qui ne crée pas de cycle.

Ajouter cette arête à A

Deuxième idée

$A = (V, E)$ (le graphe G)

tant que

faire

Choisir une arête de A qui n'est pas indispensable à la connexité

Retirer cette arête de A

Par construction, le graphe A obtenu est un arbre couvrant (en supposant que G soit connexe).

Critères de choix

- Ne pas créer de cycle : assez facile ;
- vérifier la connexité : moins facile et plus coûteux dans le cas général.

Choisir une arête

Toute arête qui ne crée pas de cycle convient. Pour que l'arbre soit minimal, il faut aussi tenir compte des poids.
La encore, deux choix sont possibles :

Choisir une arête

Toute arête qui ne crée pas de cycle convient. Pour que l'arbre soit minimal, il faut aussi tenir compte des poids.
La encore, deux choix sont possibles :

Connexité d'abord : Algorithme de Prim
On choisit l'arête de poids minimal **parmi celles incidentes à** A.
En cours d'algorithme A est un arbre. il suffit qu'une extrémité de l'arête choisie hors de A.

Choisir une arête

Toute arête qui ne crée pas de cycle convient. Pour que l'arbre soit minimal, il faut aussi tenir compte des poids.
La encore, deux choix sont possibles :

Connexité d'abord : Algorithme de Prim

On choisit l'arête de poids minimal **parmi celles incidentes à** A.
En cours d'algorithme A est un arbre. il suffit qu'une extrémité de l'arête choisie hors de A.

Minimalité d'abord : Algorithme de Kruskal

On choisit l'arête de poids minimal dans tout le graphe. En cours d'algorithme A est une **forêt**.
Il faut mémoriser si deux sommets sont dans des composantes connexes distinctes.

Choisir une arête

Toute arête qui ne crée pas de cycle convient. Pour que l'arbre soit minimal, il faut aussi tenir compte des poids.
La encore, deux choix sont possibles :

Connexité d'abord : Algorithme de Prim

On choisit l'arête de poids minimal **parmi celles incidentes à** A.
En cours d'algorithme A est un arbre. il suffit qu'une extrémité de l'arête choisie hors de A.

Minimalité d'abord : Algorithme de Kruskal

On choisit l'arête de poids minimal dans tout le graphe. En cours d'algorithme A est une **forêt**.
Il faut mémoriser si deux sommets sont dans des composantes connexes distinctes.

Utilisation des algorithmes gloutons

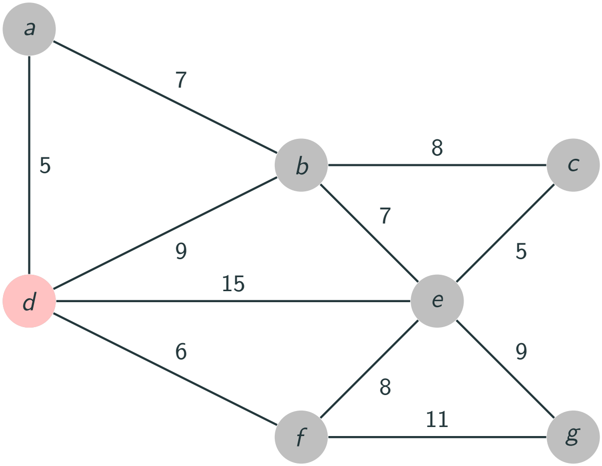
Algorithme de Prim

Histoire

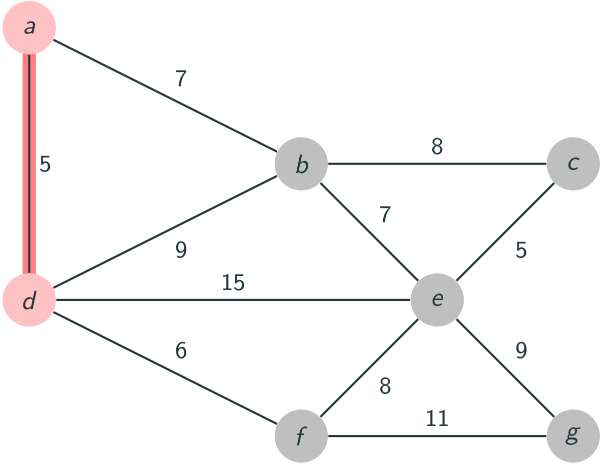
L'algorithme a été développé en 1930 par le mathématicien tchèque *Vojtech Jarnik*, puis a été redécouvert et republié par *Robert C. Prim* et *Edsger W. Dijkstra* en 1959.

```
A := (∅, ∅)
ajouterSommet( choisirSommet(G), A )
while nbSommets(A) < V
    // Recherche de l'arête min incidente à A
    m := +∞
    foreach x ∈ A do
        foreach y ∈ ensVoisins(x) do
            if y ∉ ensSommets(A) et poids(x, y) < m
                m := poids(x, y)
                ymin := y
    ajouterSommet( ymin, A )
    ajouterArete( (x, ymin), A )
return A
```

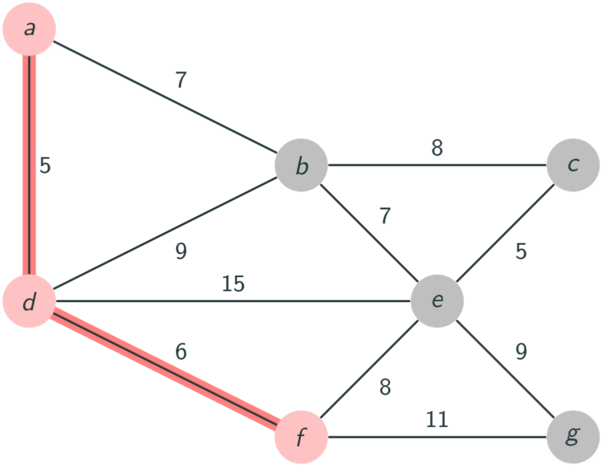
Déroulement sur un exemple



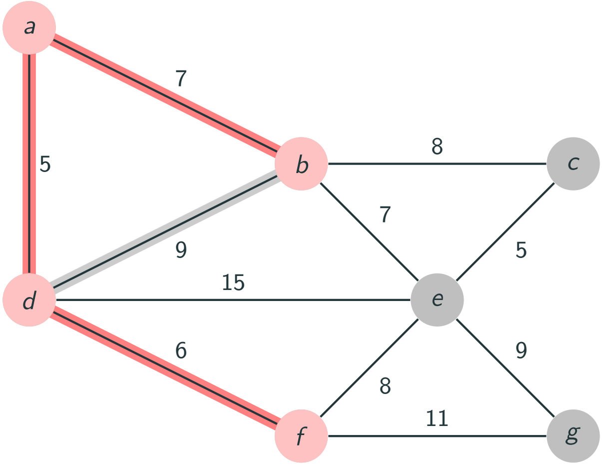
Déroulement sur un exemple



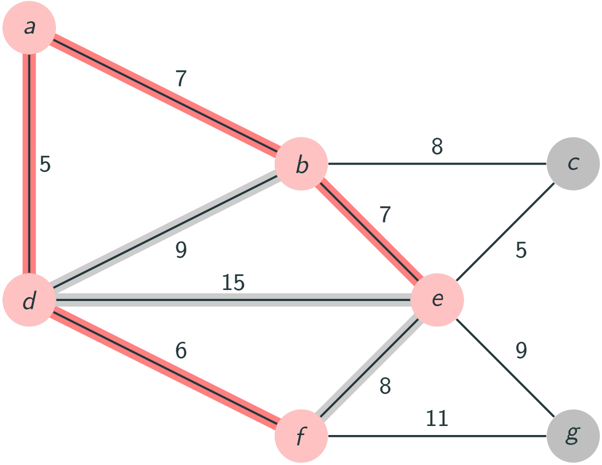
Déroulement sur un exemple



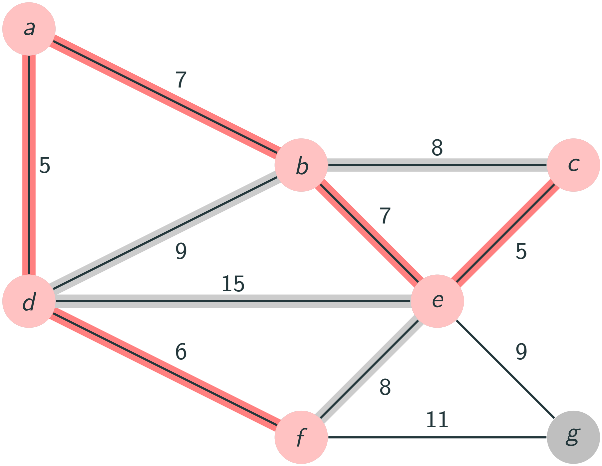
Déroulement sur un exemple



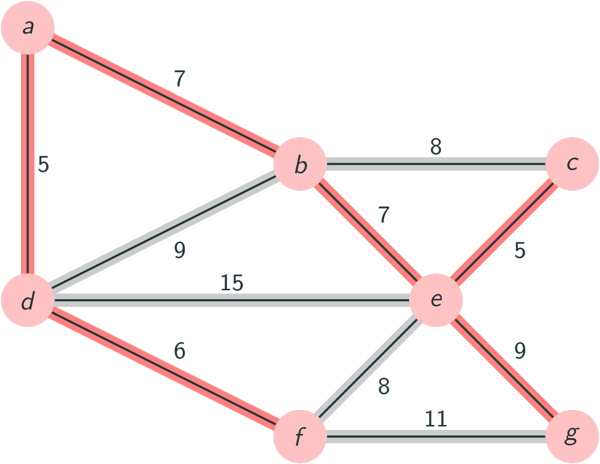
Déroulement sur un exemple



Déroulement sur un exemple



Déroulement sur un exemple



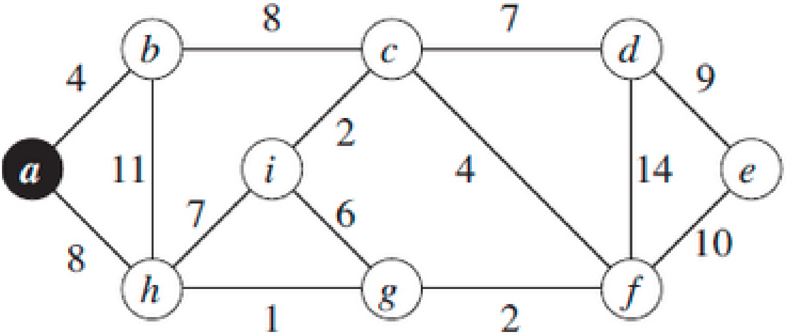
Caractère glouton

On retrouve les caractéristiques d'un algorithme glouton :

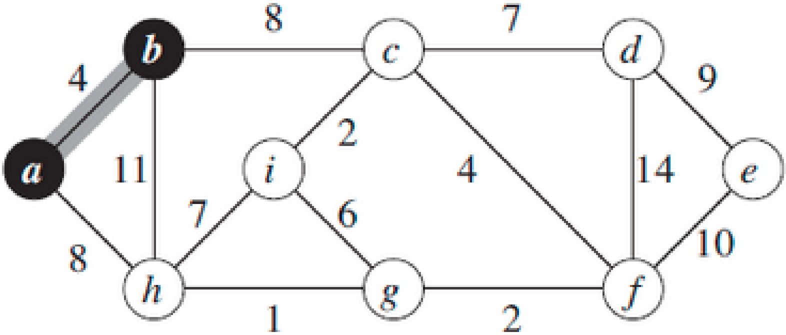
- La solution construite est un ensemble (d'arêtes) ;
- Les solutions admissibles sont les arbres couvrants ;
- On recherche une solution de poids total minimal ;
- L'algorithme de Prim procède uniquement par ajout d'arêtes (et de sommets) dans A ;
- Le choix de la prochaine arête n'est basé que sur son poids.

L'algorithme a donc coût en $\mathcal{O}(V \times C$ où C est le coût du choix de la prochaine arête.

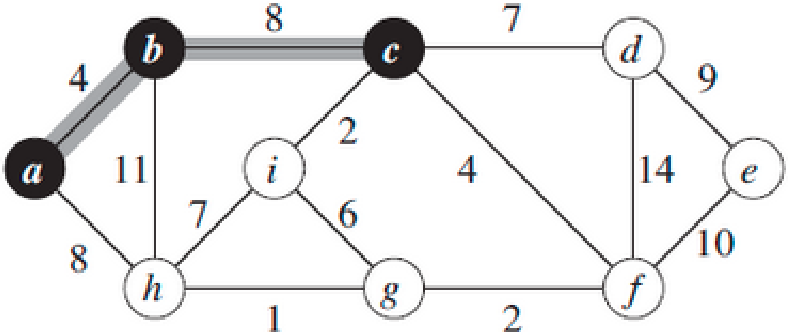
A vous de jouer !



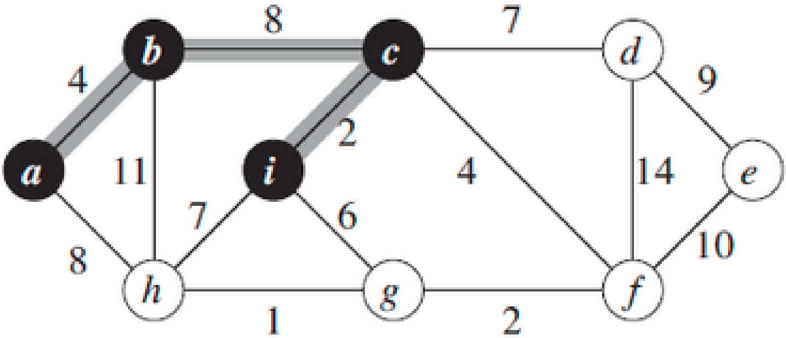
A vous de jouer !



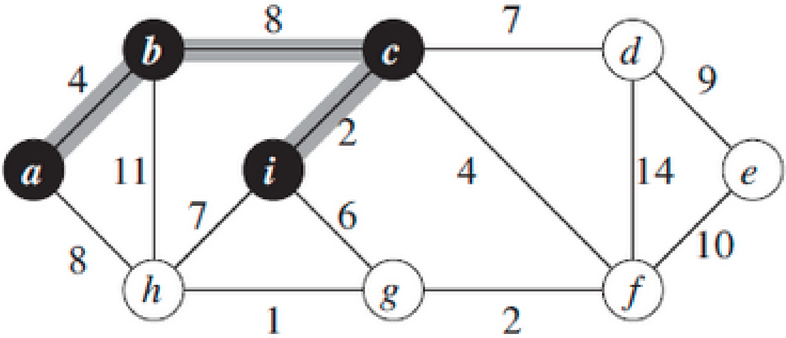
A vous de jouer !



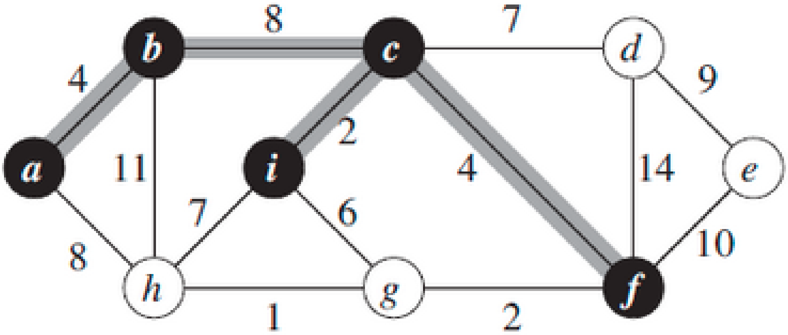
A vous de jouer !



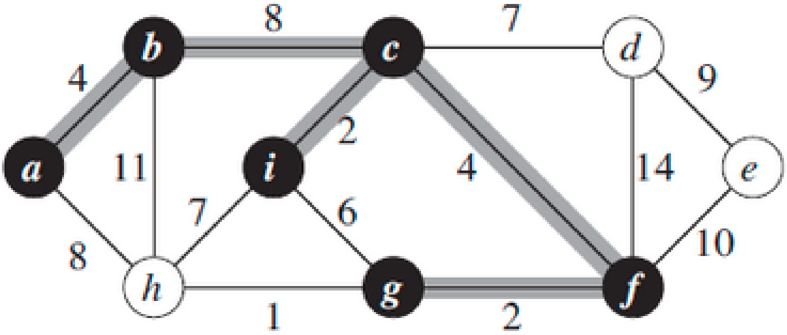
A vous de jouer !



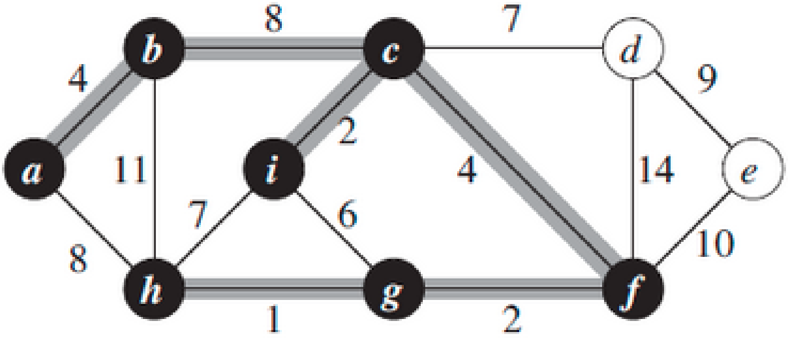
A vous de jouer !



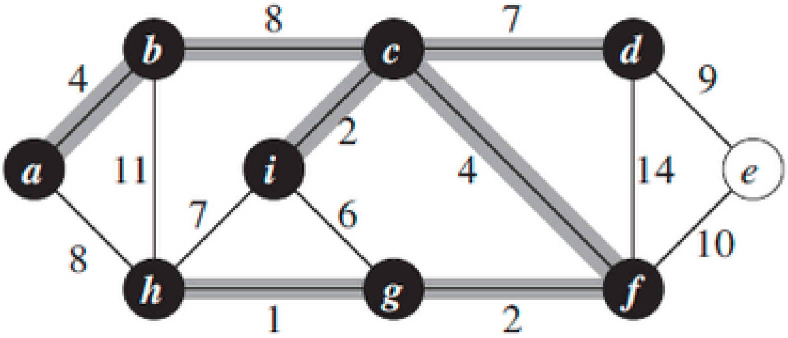
A vous de jouer !



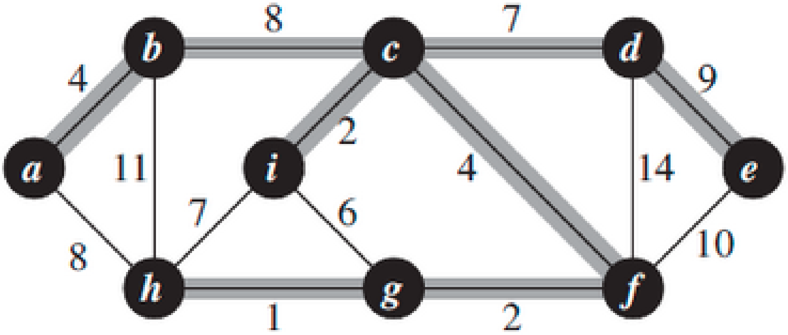
A vous de jouer !



A vous de jouer !



A vous de jouer !



Algorithme de Kruskal

Algorithme de Kruskal

Algorithme

L'algorithme construit un arbre couvrant minimum en sélectionnant des arêtes par poids croissant.
Plus précisément, l'algorithme considère toutes les arêtes du graphe par poids croissant (en pratique, on trie d'abord les arêtes du graphe par poids croissant) et pour chacune d'elles, il la sélectionne si elle ne crée pas un cycle.

Algorithme de Kruskal

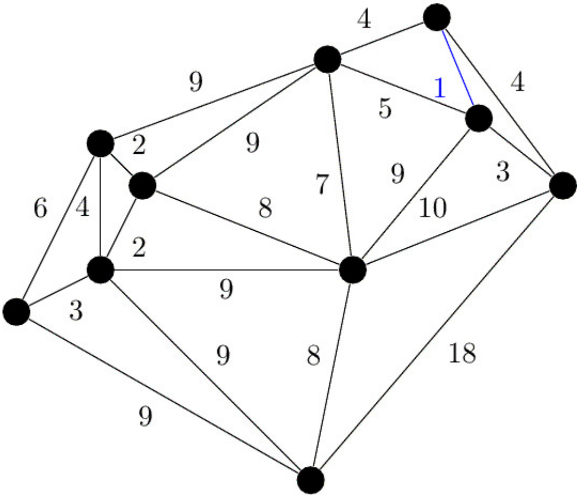
Algorithme

L'algorithme construit un arbre couvrant minimum en sélectionnant des arêtes par poids croissant.

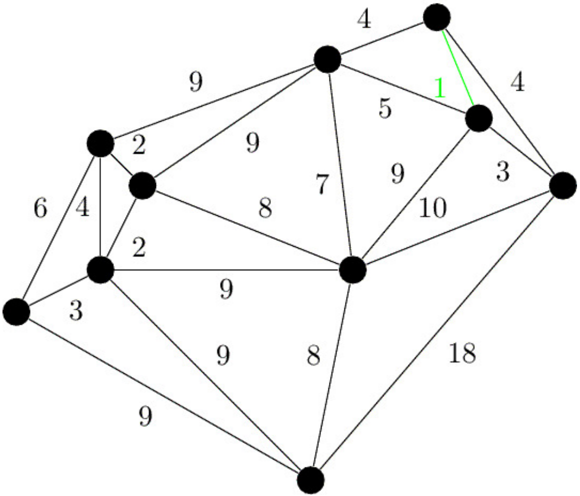
Plus précisément, l'algorithme considère toutes les arêtes du graphe par poids croissant (en pratique, on trie d'abord les arêtes du graphe par poids croissant) et pour chacune d'elles, il la sélectionne si elle ne crée pas un cycle.

foreach *arête* (x, y) *(par poids croissant)* **do**
  if *x et y sont dans des composantes connexes différentes*
    ajouterArete((x, y), *Sol*)

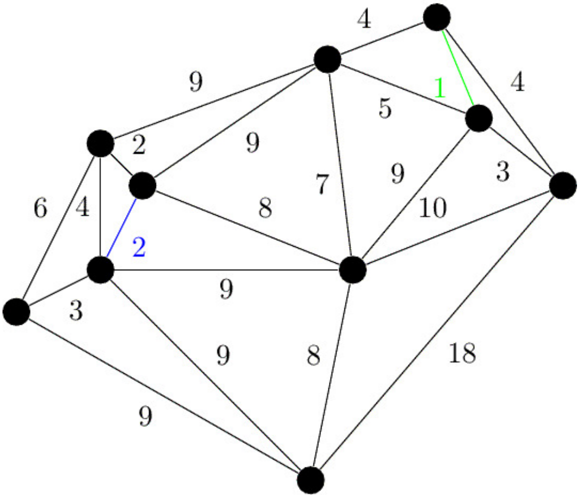
Déroulement sur un exemple



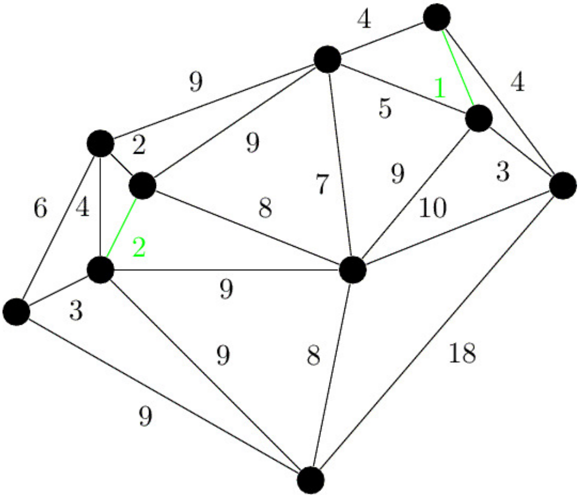
Déroulement sur un exemple



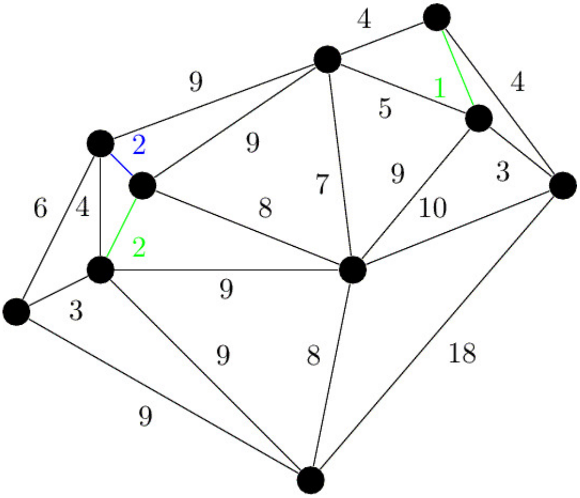
Déroulement sur un exemple



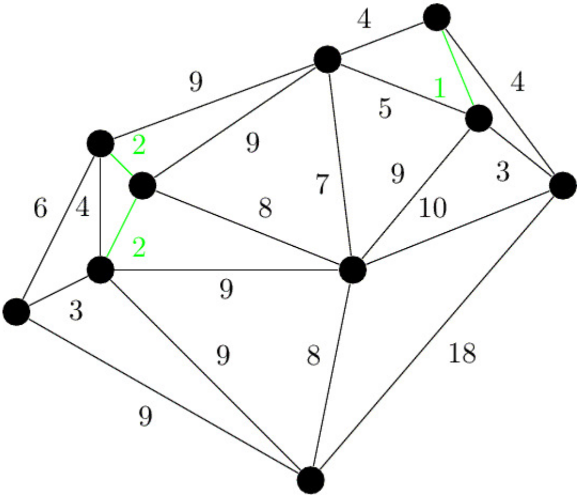
Déroulement sur un exemple



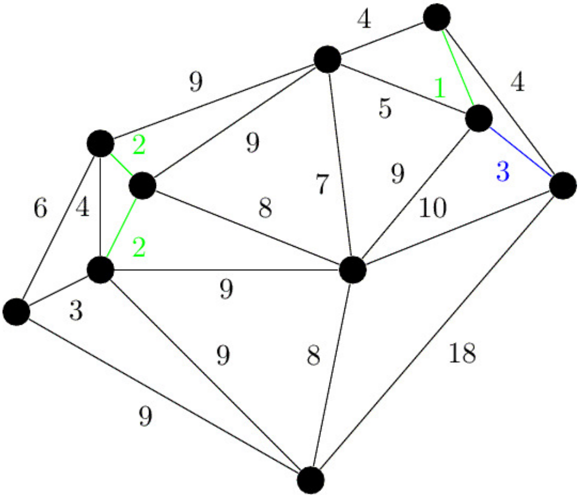
Déroulement sur un exemple



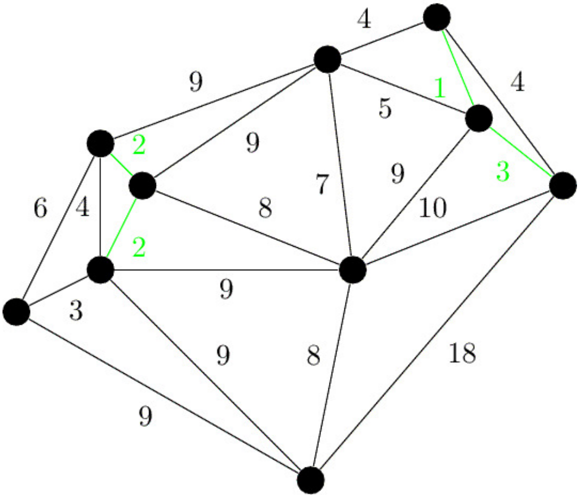
Déroulement sur un exemple



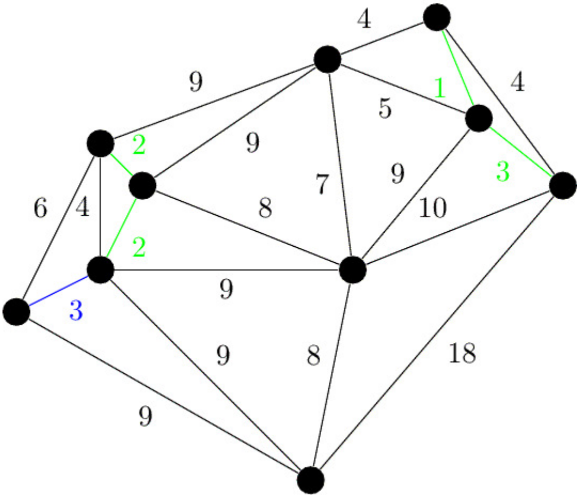
Déroulement sur un exemple



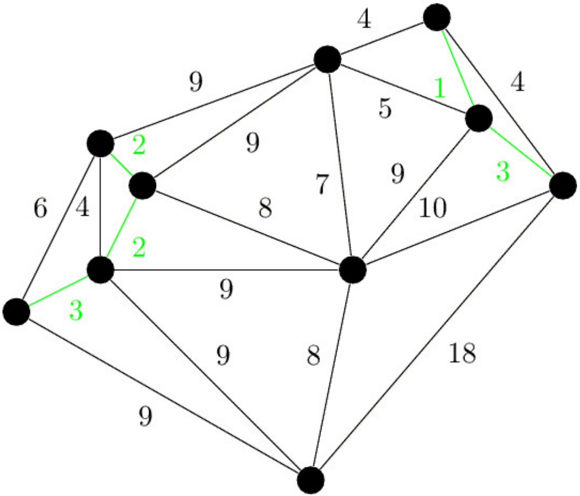
Déroulement sur un exemple



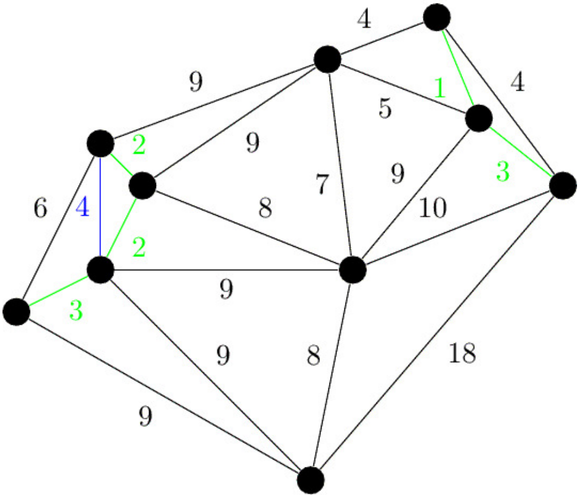
Déroulement sur un exemple



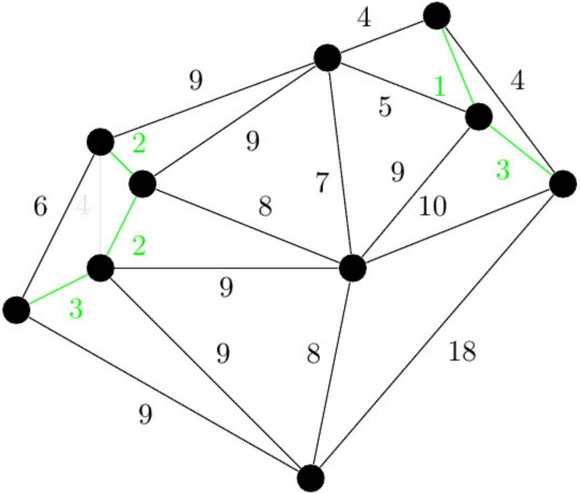
Déroulement sur un exemple



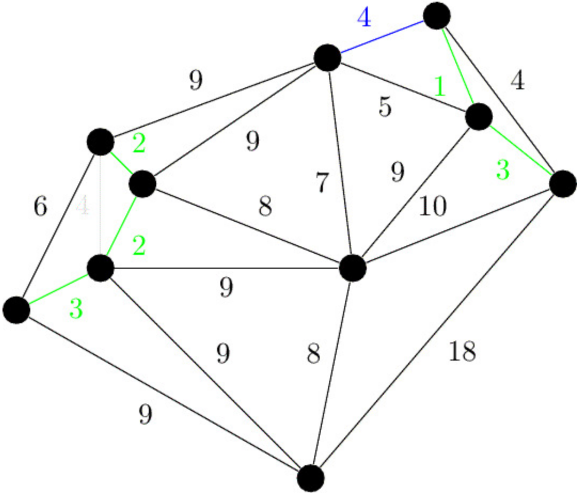
Déroulement sur un exemple



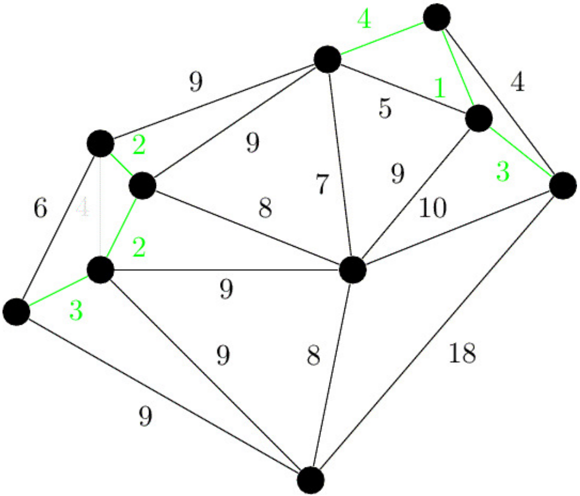
Déroulement sur un exemple



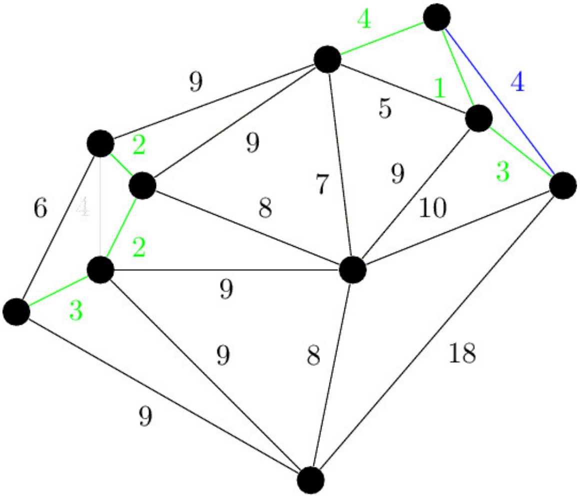
Déroulement sur un exemple



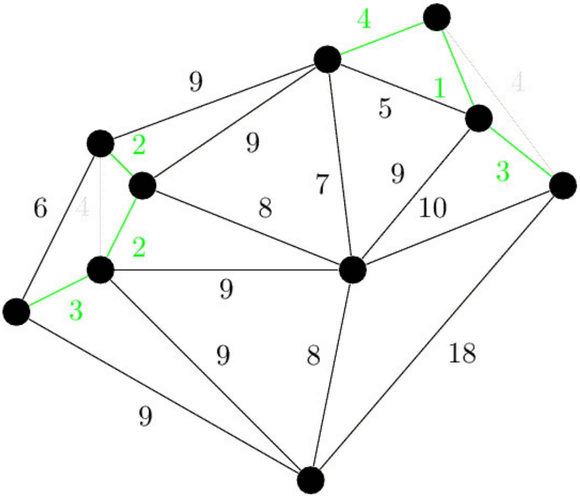
Déroulement sur un exemple



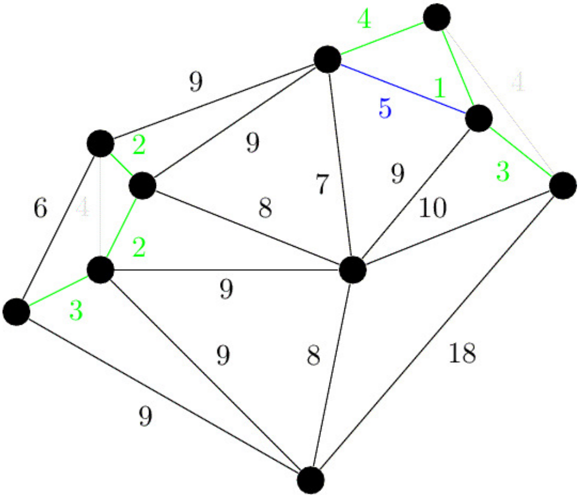
Déroulement sur un exemple



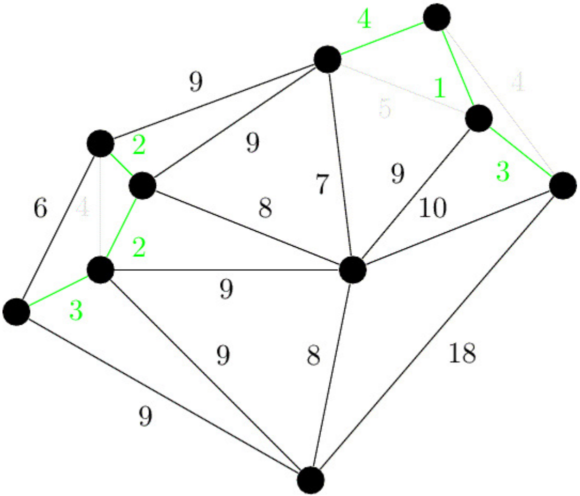
Déroulement sur un exemple



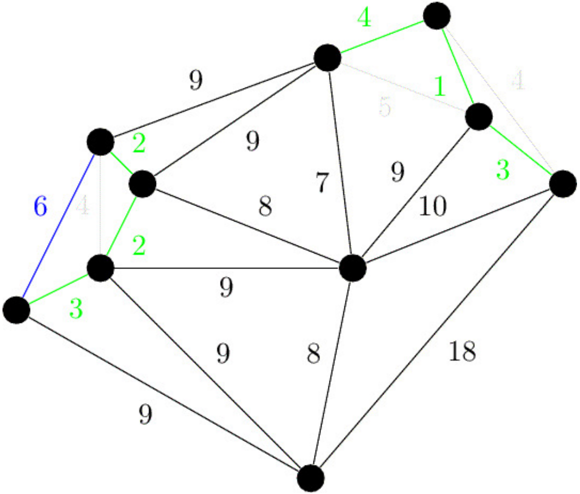
Déroulement sur un exemple

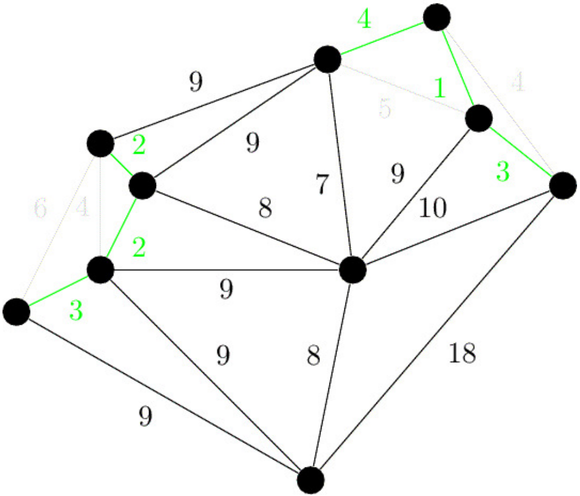


Déroulement sur un exemple

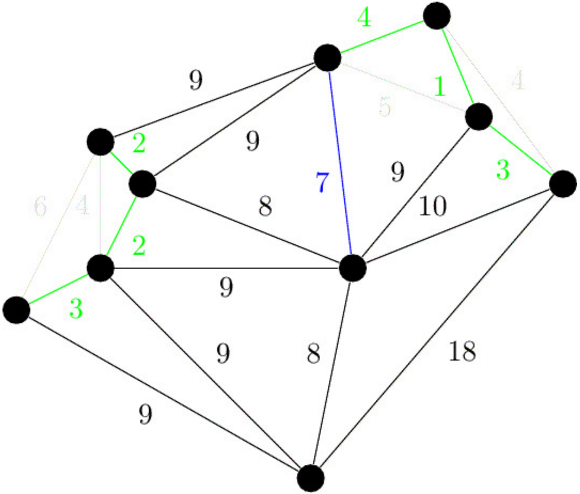


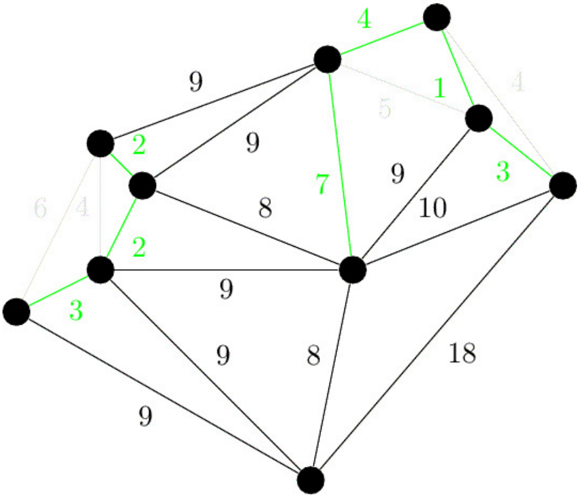
Déroulement sur un exemple



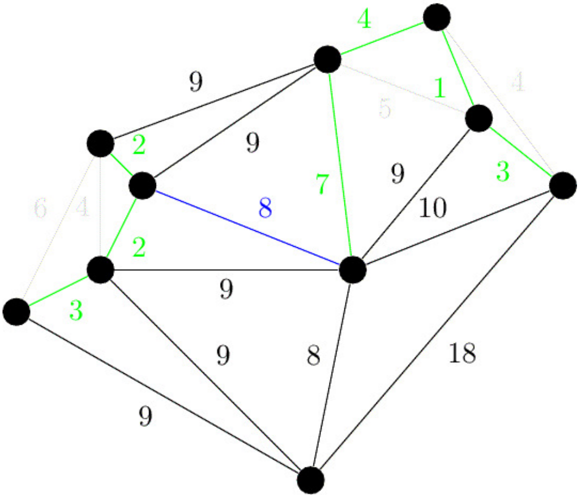

4
9
1
4
5
2
9
7
9
3
6
4
8
10
2
9
3
9
8
18
9

Déroulement sur un exemple

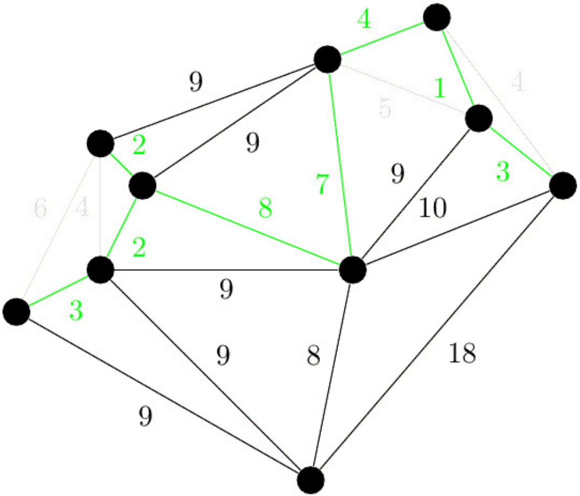



4
9
1
4
5
2
9
9
3
7
6
4
8
10
2
9
3
9
8
18
9

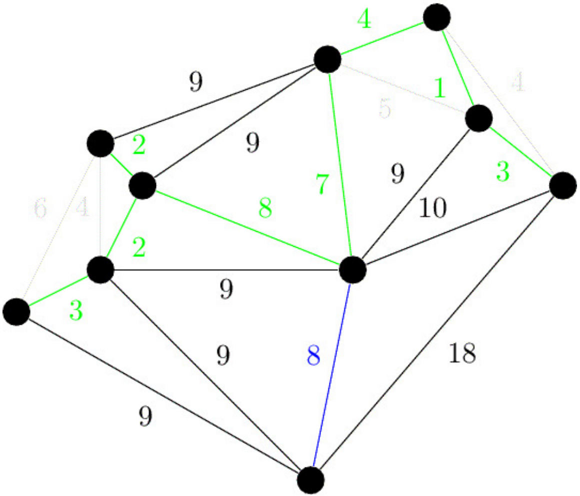
Déroulement sur un exemple

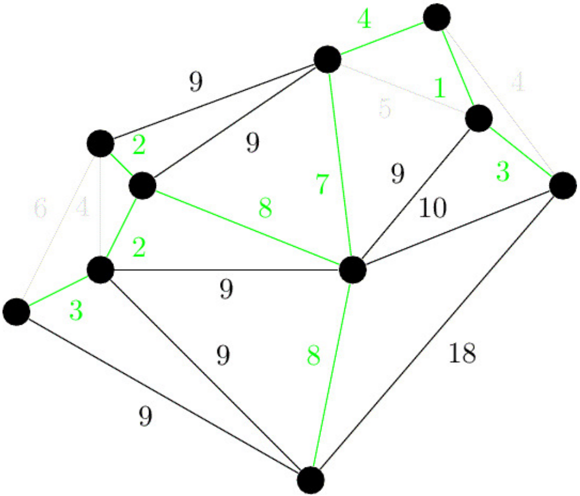


Déroulement sur un exemple

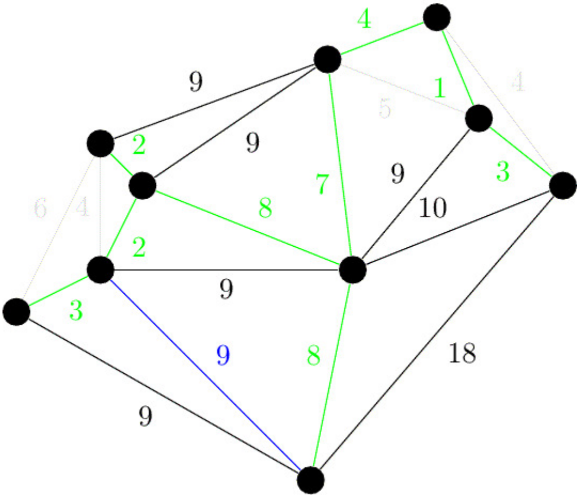


Déroulement sur un exemple

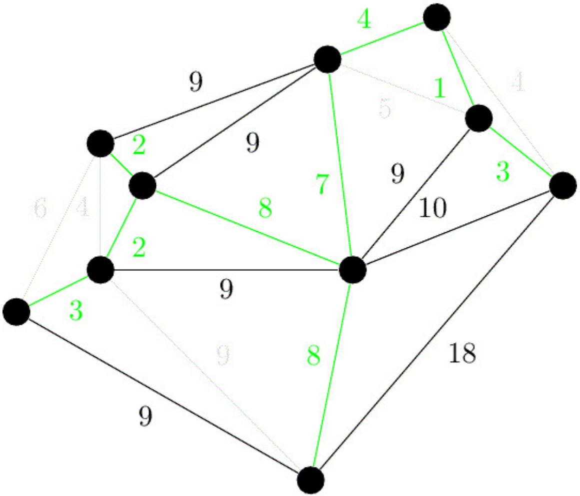



4
9
1
5
4
2
9
7
9
3
8
10
6
4
2
9
3
9
8
18
9

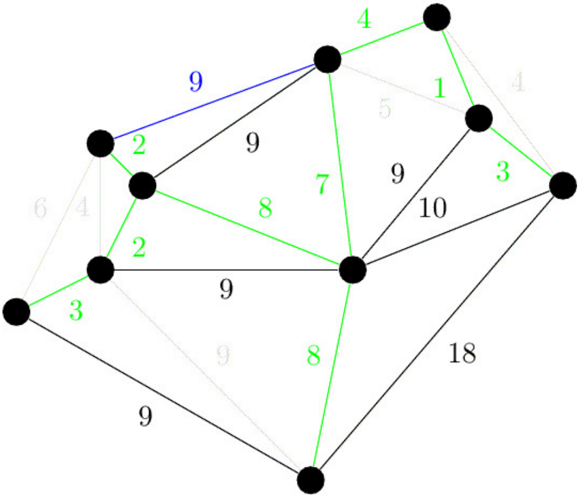
Déroulement sur un exemple



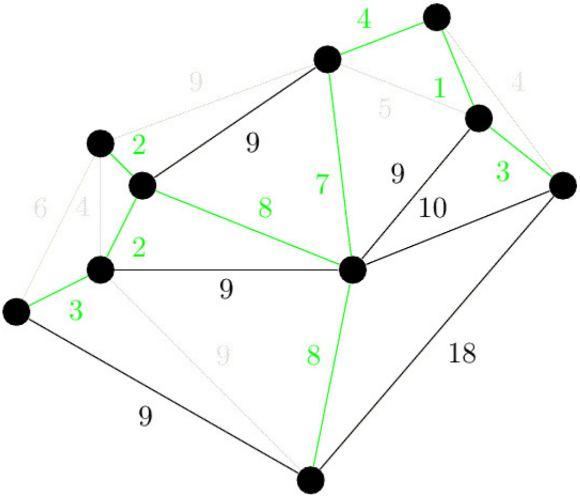
Déroulement sur un exemple



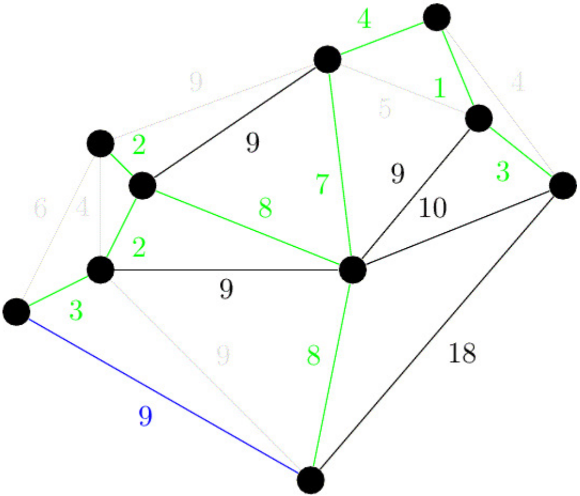
Déroulement sur un exemple



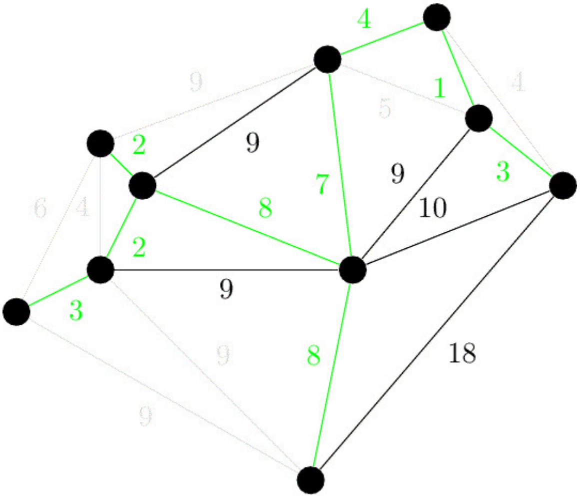
Déroulement sur un exemple

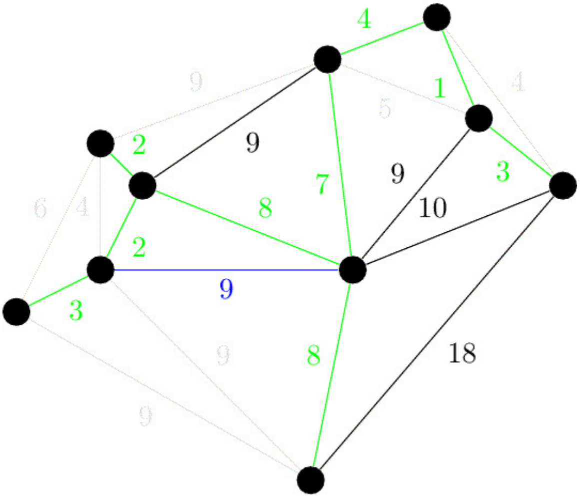


Déroulement sur un exemple

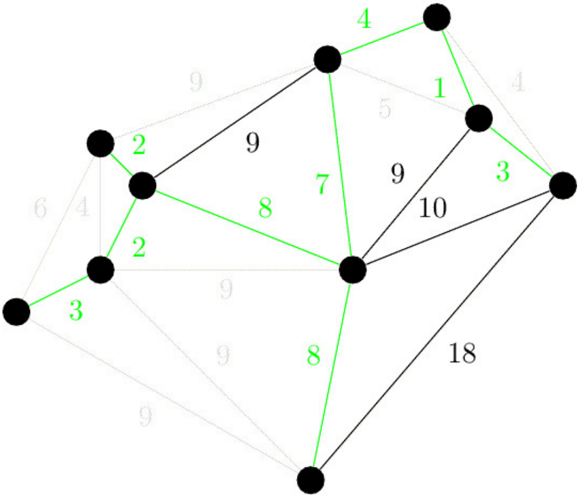


Déroulement sur un exemple

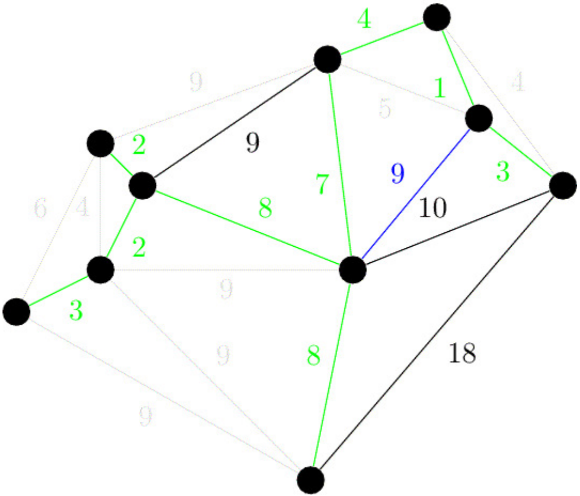



4
9
1
5
4
2
9
7
9
3
6
4
8
10
2
9
3
9
8
18
9

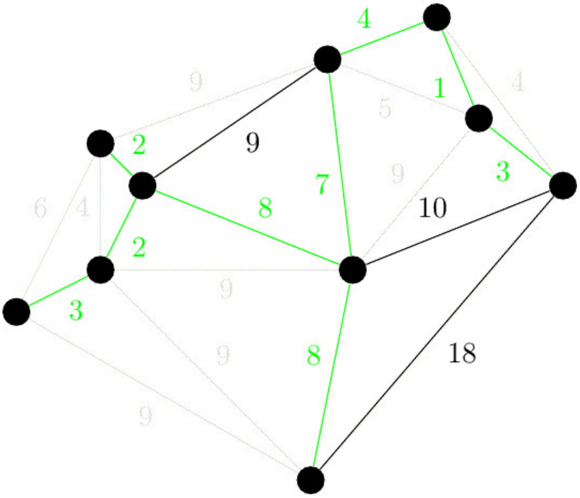
Déroulement sur un exemple



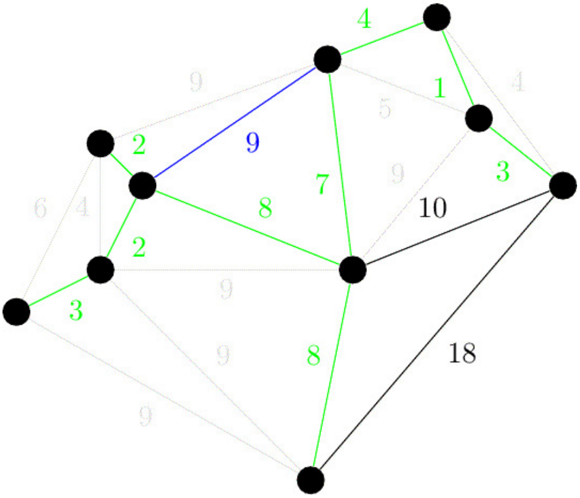
Déroulement sur un exemple



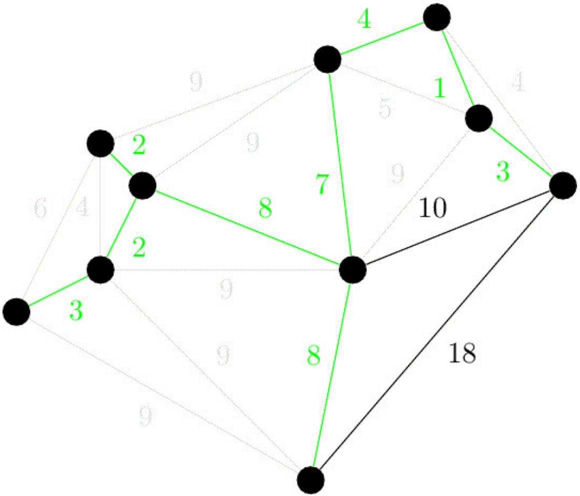
Déroulement sur un exemple



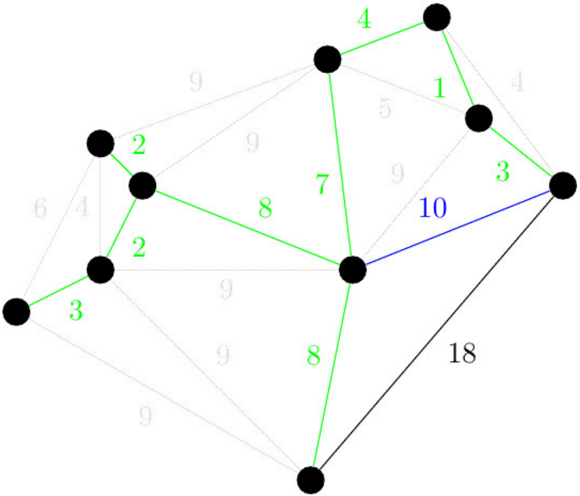
Déroulement sur un exemple



Déroulement sur un exemple



Déroulement sur un exemple



Conclusion du cours

- Présentation des algorithmes gloutons ;
- Faiblesse de l'algorithme de Dijsktra et algorithme de Bellman-Ford ;
- Définition d'un arbre couvrant ;
- Algorithme de Prim ;
- Algorithme de Kruskal.