Arbres couvrants

Florent Foucaud - Malika More - Thibault Ralet
Carine Simon - Thierry Trévisan

1A - BUT Info - UCA

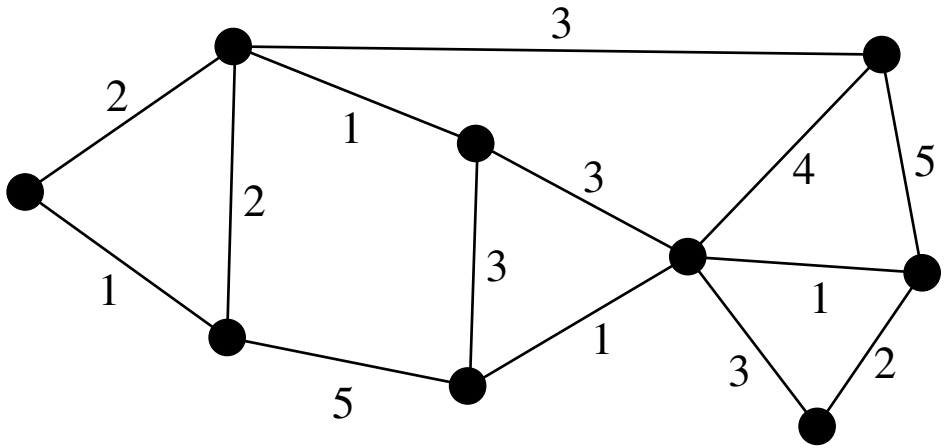
R207 Graphes

Année 2021-2022

Plan du cours

Un (projet de) réseau entre des centres de calcul

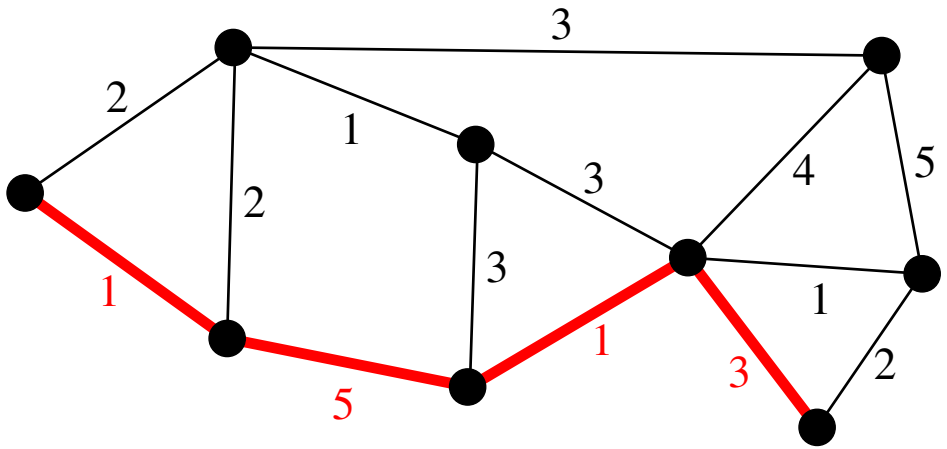
On souhaite construire les connexions à très haut débit indiquées sur le schéma



Un (projet de) réseau entre des centres de calcul

On souhaite construire les connexions à très haut débit indiquées sur le schéma

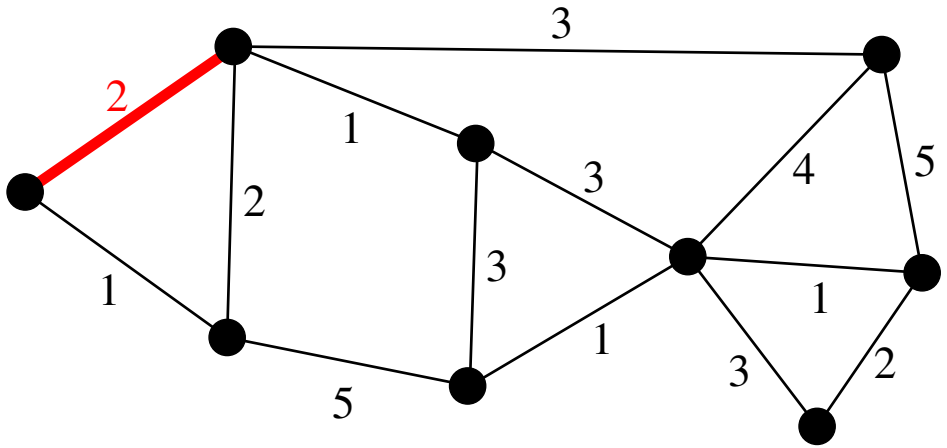
- Deux centres éloignés pourront communiquer par l'intermédiaire d'autres centres



Un (projet de) réseau entre des centres de calcul

On souhaite construire les connexions à très haut débit indiquées sur le schéma

- La construction de chaque connexion a un coût

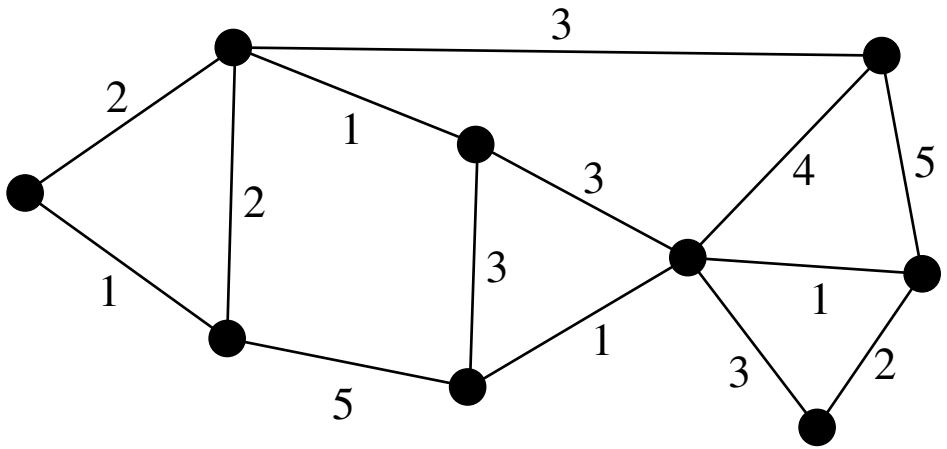


Un (projet de) réseau entre des centres de calcul

On souhaite construire les connexions à très haut débit indiquées sur le schéma

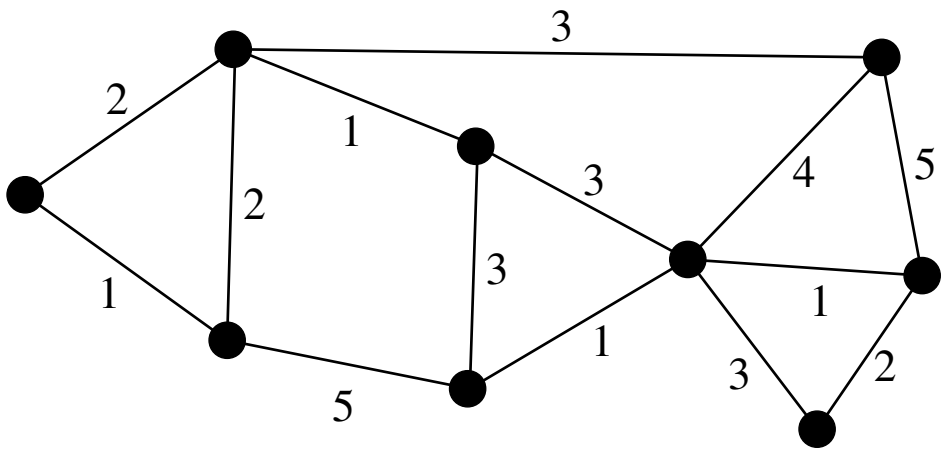
- L'ensemble est jugé trop cher

$$1 + 2 + 2 + 1 + 5 + 3 + 3 + 3 + 1 + 4 + 1 + 3 + 5 + 2 = 36$$



Il faut revoir le projet

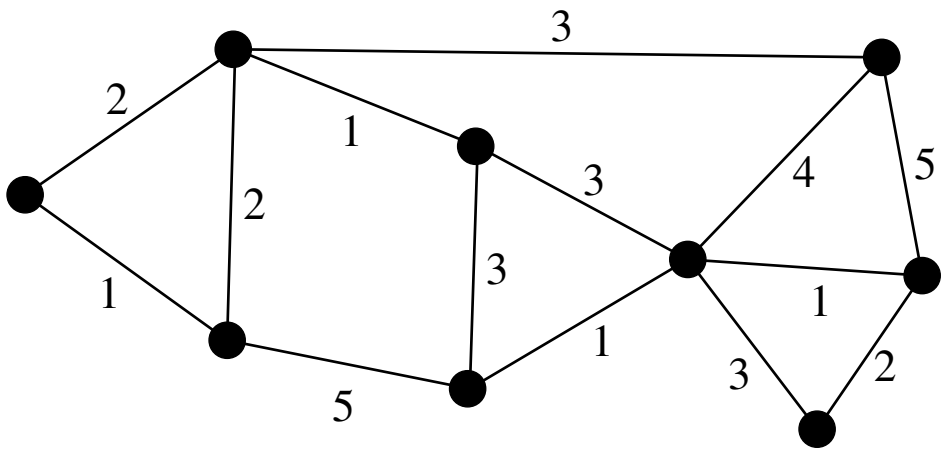
On décide de ne construire que certaines connexions



Il faut revoir le projet

On décide de ne construire que certaines connexions

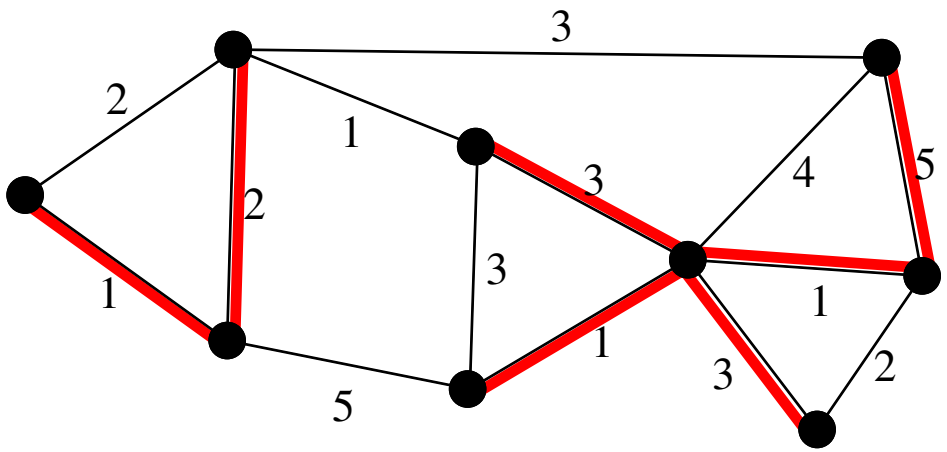
- Tous les centres doivent pouvoir communiquer



Il faut revoir le projet

On décide de ne construire que certaines connexions

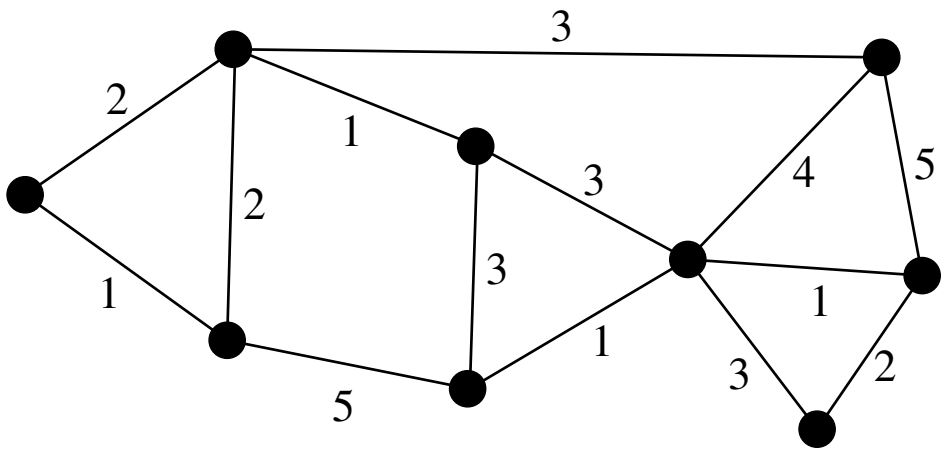
- Tous les centres doivent pouvoir communiquer



Il faut revoir le projet

On décide de ne construire que certaines connexions

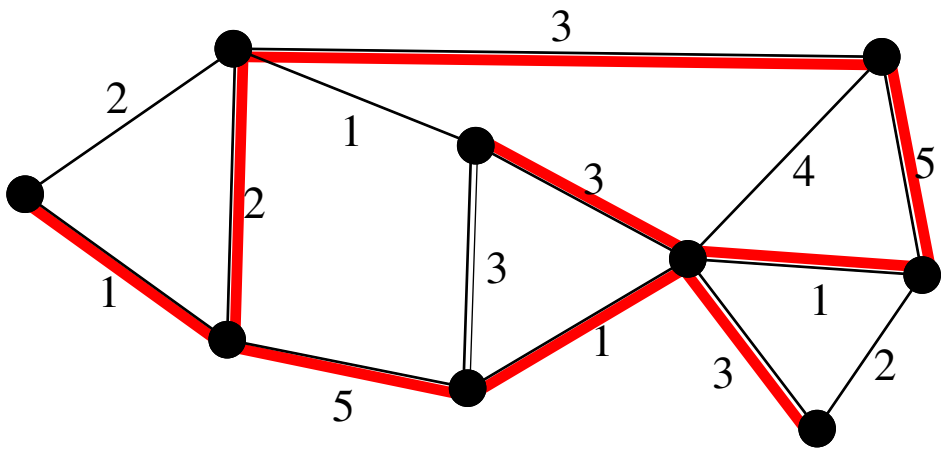
- Il ne faut pas construire de connexions dont on pourrait se passer



Il faut revoir le projet

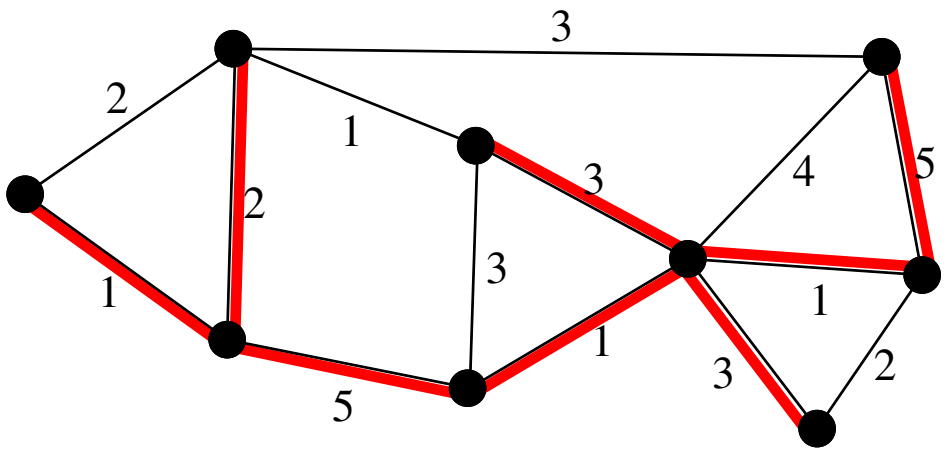
On décide de ne construire que certaines connexions

- Il ne faut pas construire de connexions dont on pourrait se passer



Il faut revoir le projet

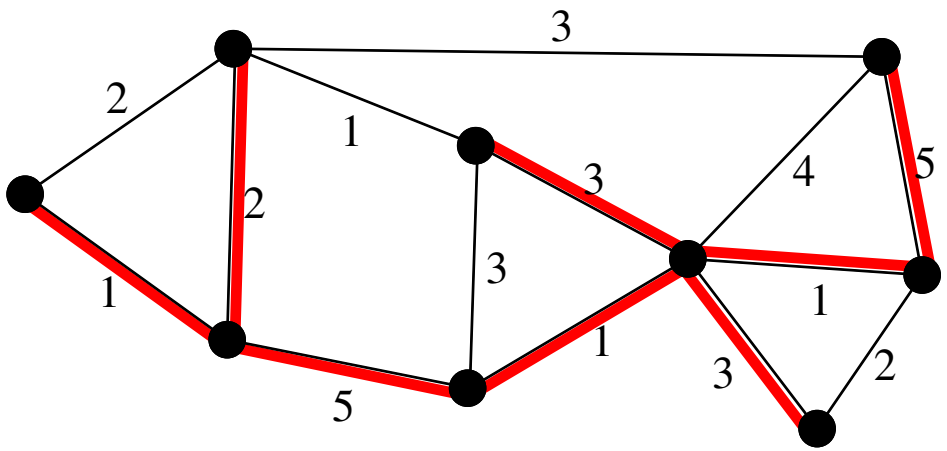
On décide de ne construire que certaines connexions



Il faut revoir le projet

On décide de ne construire que certaines connexions

- Le coût total du projet doit être le plus faible possible



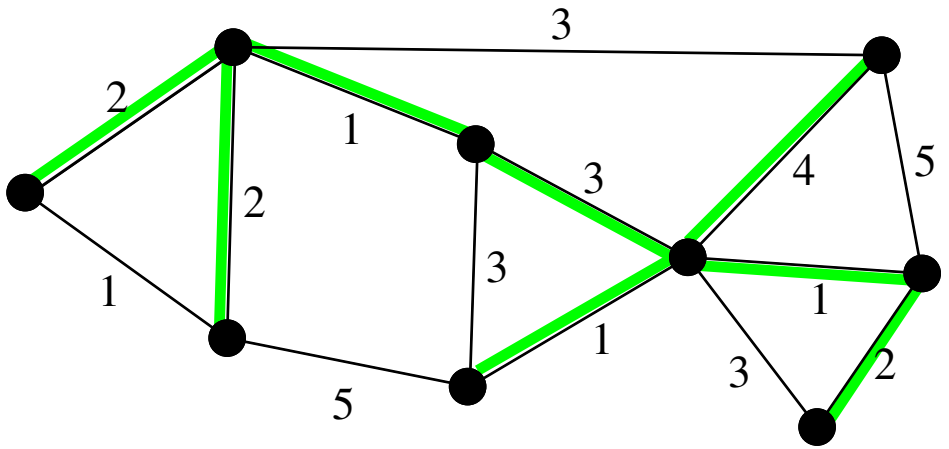
$1 + 2 + 5 + 1 + 3 + 5 + 1 + 3 = 21$

Peut-on faire mieux ?

Il faut revoir le projet

On décide de ne construire que certaines connexions

- Le coût total du projet doit être le plus faible possible



$2 + 2 + 1 + 3 + 1 + 4 + 1 + 2 = 16$

Peut-on faire mieux ?

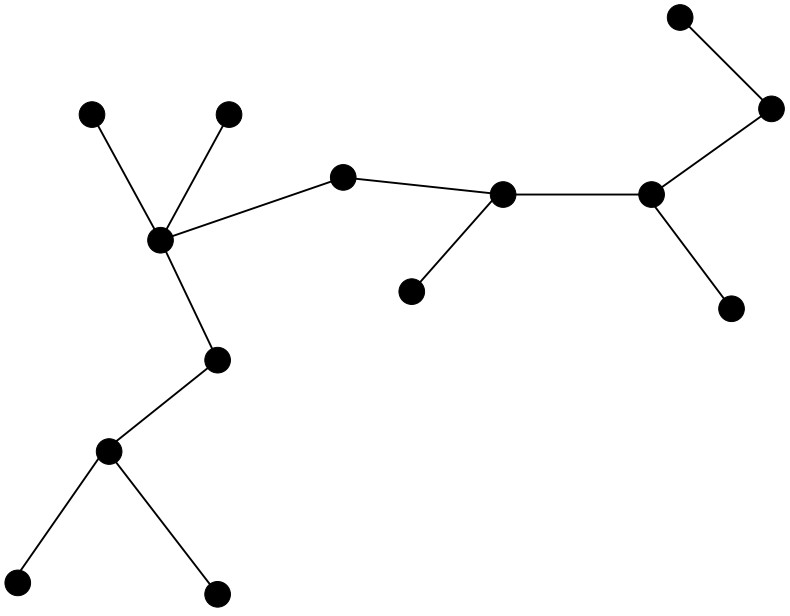
Construction récursive

- Un graphe avec un seul sommet et zéro arête est un arbre

- À partir d'un arbre avec au moins 1 sommet
- On accroche une feuille à n'importe quel sommet
- On obtient un arbre avec un sommet et une arête de plus

Construction récursive

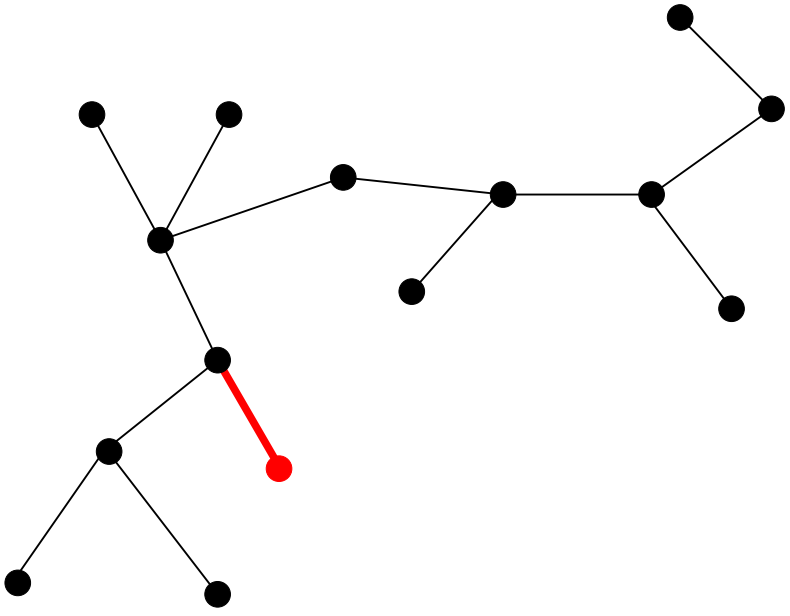
- Un graphe avec un seul sommet et zéro arête est un arbre



- À partir d'un arbre avec au moins 1 sommet
- On accroche une feuille à n'importe quel sommet
- On obtient un arbre avec un sommet et une arête de plus

Construction récursive

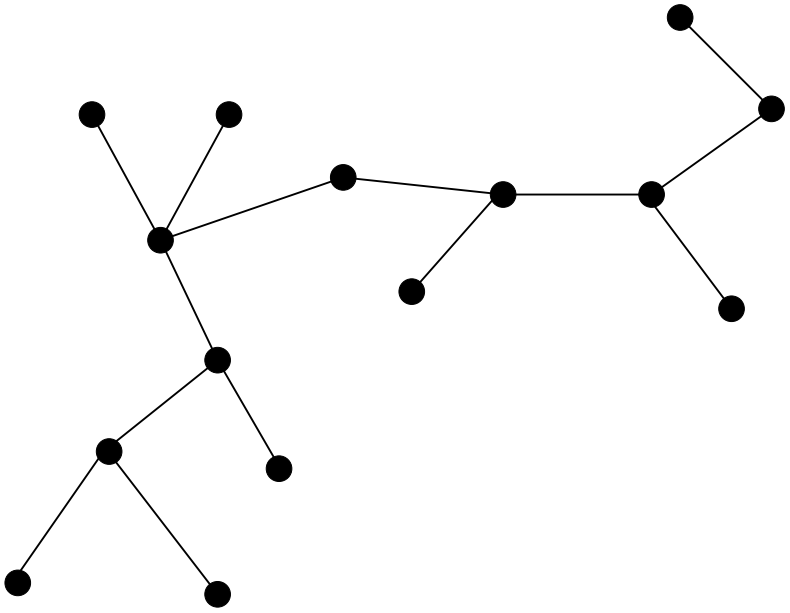
- Un graphe avec un seul sommet et zéro arête est un arbre



- À partir d'un arbre avec au moins 1 sommet
- On accroche une feuille à n'importe quel sommet
- On obtient un arbre avec un sommet et une arête de plus

Construction récursive

- Un graphe avec un seul sommet et zéro arête est un arbre



- À partir d'un arbre avec au moins 1 sommet
- On accroche une feuille à n'importe quel sommet

- On obtient un arbre avec un sommet et une arête de plus

Remarques

1. Un arbre à n sommets possède exactement $n - 1$ arêtes.
2. Un arbre avec au moins deux sommets a forcément au moins une feuille.

Remarques

1. Un arbre à n sommets possède exactement $n - 1$ arêtes.
2. Un arbre avec au moins deux sommets a forcément au moins une feuille.

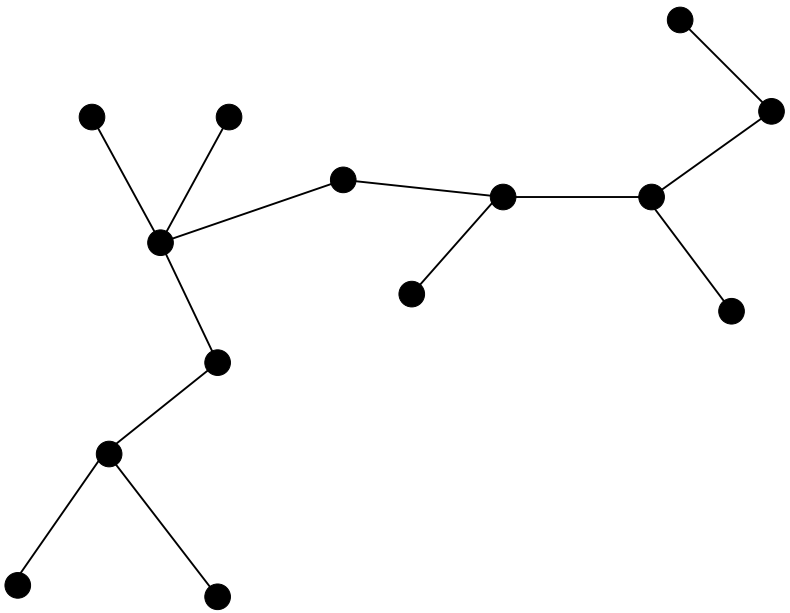
Remarques

1. Un arbre à n sommets possède exactement $n - 1$ arêtes.
2. Un arbre avec au moins deux sommets a forcément au moins une feuille.
   - Prenons une chaîne sans répétition la plus longue possible.
   - Regardons une extrémité de la chaîne, disons v.
   - Alors v a au moins un voisin (dans la chaîne).
   - Supposons que v ait un autre voisin, disons w.

Remarques

1. Un arbre à n sommets possède exactement $n - 1$ arêtes.
2. Un arbre avec au moins deux sommets a forcément au moins une feuille.
   - Ou bien w est dans la chaîne, et alors il y aurait un cycle, mais dans un arbre c'est impossible.
   - Ou bien w n'est pas dans la chaîne, mais alors on pourrait allonger la chaîne, et c'est impossible aussi.
   - Donc w n'existe pas.
   - Càd v n'a qu'un seul voisin, c'est une feuille.

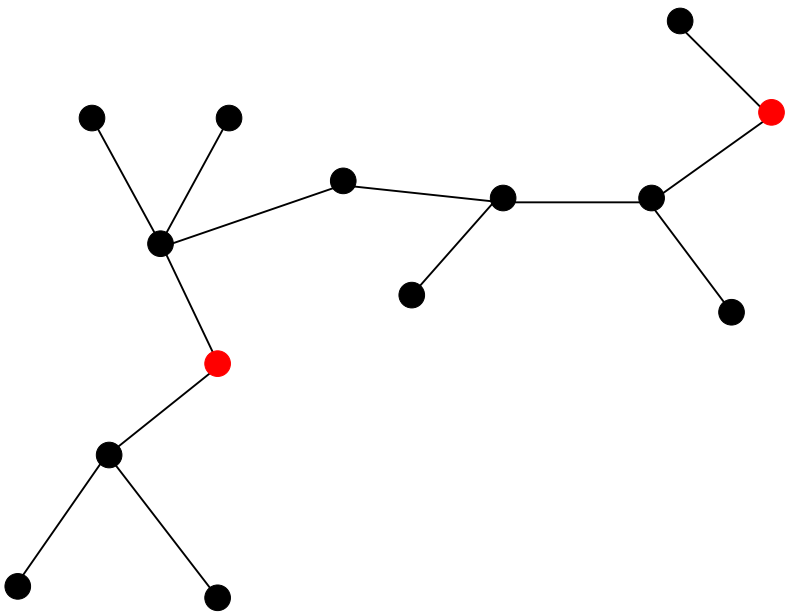
Propriétés caractéristiques



1. deux sommets quelconques d'un arbre sont reliés par une unique chaîne
2. enlever une arête quelconque à un arbre le déconnecte
3. ajouter une arête quelconque à un arbre crée un cycle

Un graphe ayant ces propriétés est forcément un arbre.

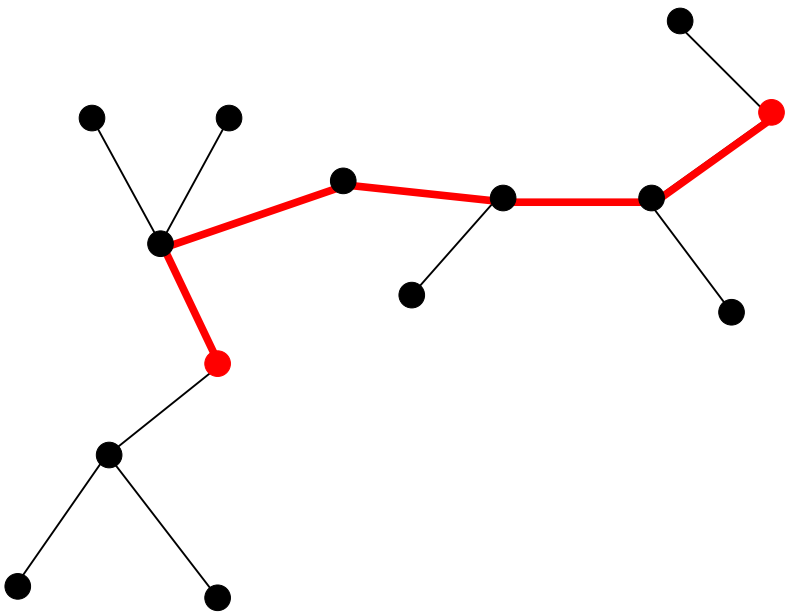
Propriétés caractéristiques



1. deux sommets quelconques d'un arbre sont reliés par une unique chaîne
2. enlever une arête quelconque à un arbre le déconnecte
3. ajouter une arête quelconque à un arbre crée un cycle

Un graphe ayant ces propriétés est forcément un arbre.

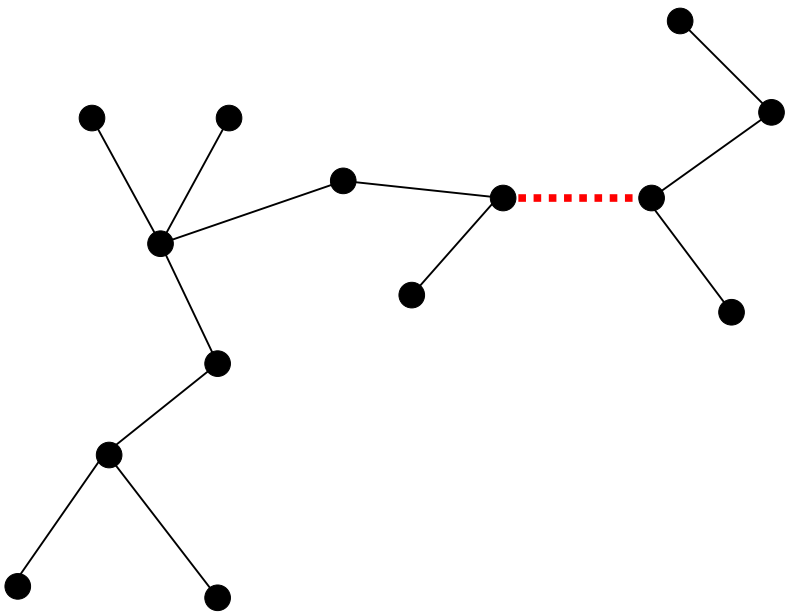
Propriétés caractéristiques



1. deux sommets quelconques d'un arbre sont reliés par une unique chaîne
2. enlever une arête quelconque à un arbre le déconnecte
3. ajouter une arête quelconque à un arbre crée un cycle

Un graphe ayant ces propriétés est forcément un arbre.

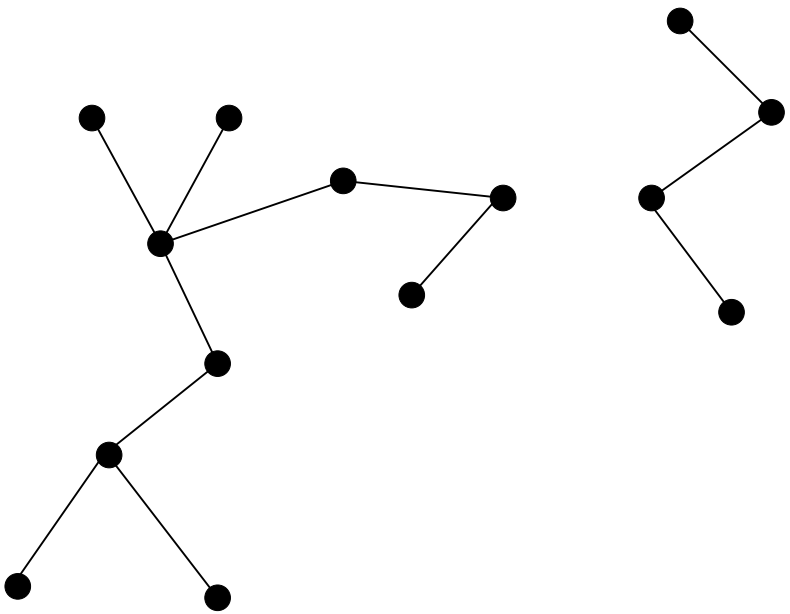
Propriétés caractéristiques



1. deux sommets quelconques d'un arbre sont reliés par une unique chaîne
2. enlever une arête quelconque à un arbre le déconnecte
3. ajouter une arête quelconque à un arbre crée un cycle

Un graphe ayant ces propriétés est forcément un arbre.

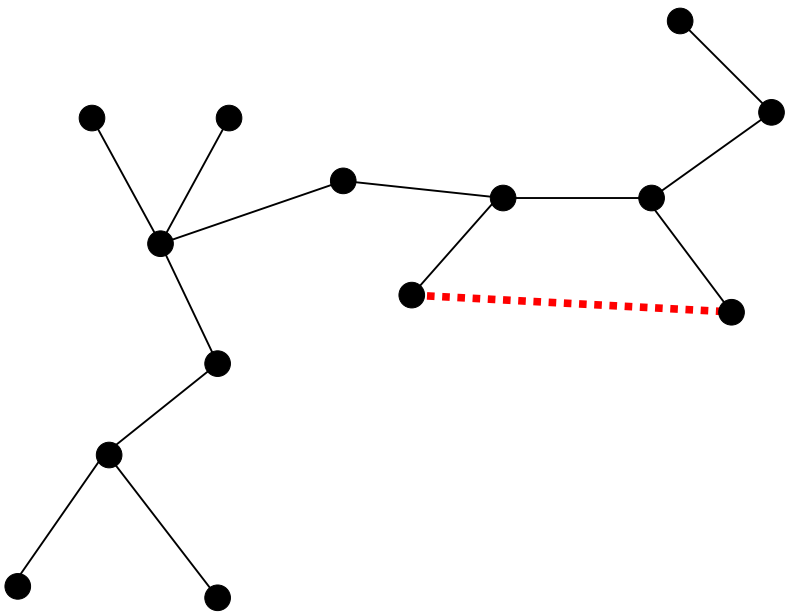
Propriétés caractéristiques



1. deux sommets quelconques d'un arbre sont reliés par une unique chaîne
2. enlever une arête quelconque à un arbre le déconnecte
3. ajouter une arête quelconque à un arbre crée un cycle

Un graphe ayant ces propriétés est forcément un arbre.

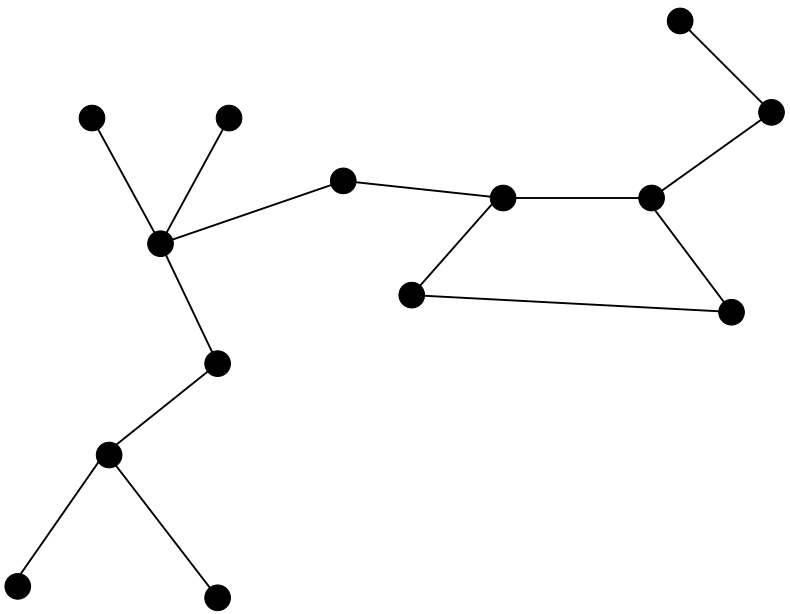
Propriétés caractéristiques



1. deux sommets quelconques d'un arbre sont reliés par une unique chaîne
2. enlever une arête quelconque à un arbre le déconnecte
3. ajouter une arête quelconque à un arbre crée un cycle

Un graphe ayant ces propriétés est forcément un arbre.

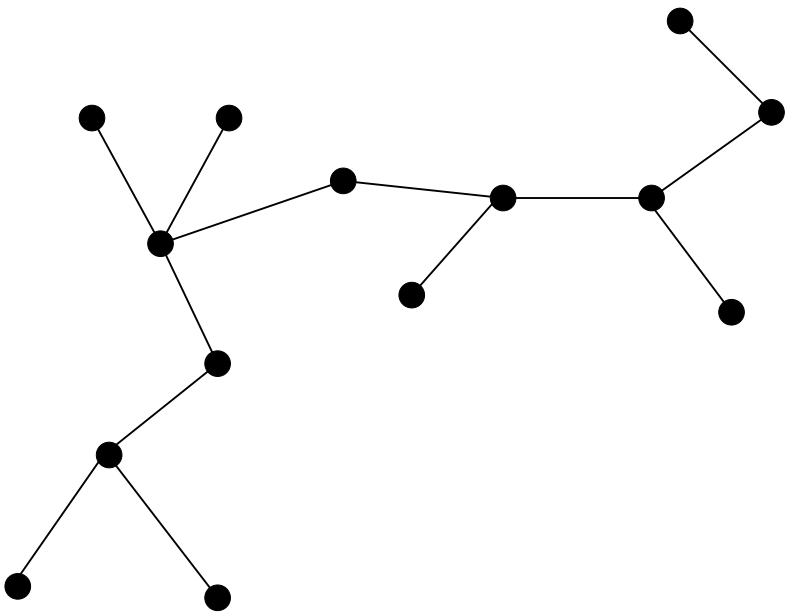
Propriétés caractéristiques



1. deux sommets quelconques d'un arbre sont reliés par une unique chaîne
2. enlever une arête quelconque à un arbre le déconnecte
3. ajouter une arête quelconque à un arbre crée un cycle

Un graphe ayant ces propriétés est forcément un arbre.

Propriétés caractéristiques

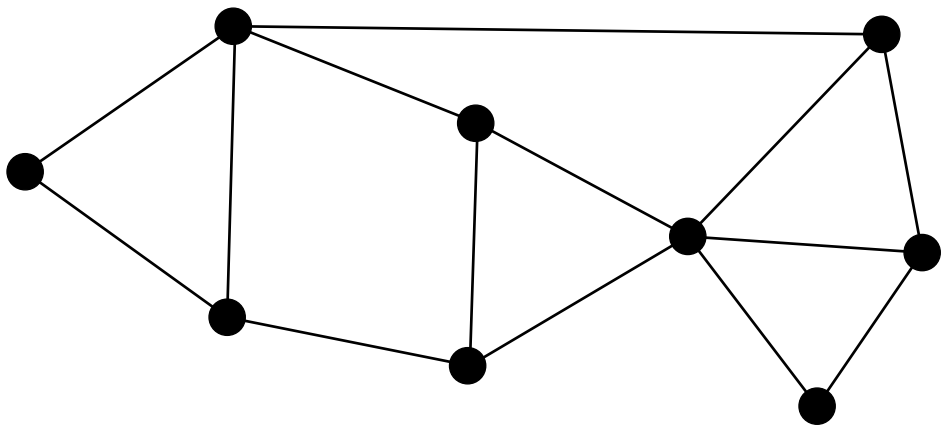


1. deux sommets quelconques d'un arbre sont reliés par une unique chaîne
2. enlever une arête quelconque à un arbre le déconnecte
3. ajouter une arête quelconque à un arbre crée un cycle

Un graphe ayant ces propriétés est forcément un arbre.

Définition

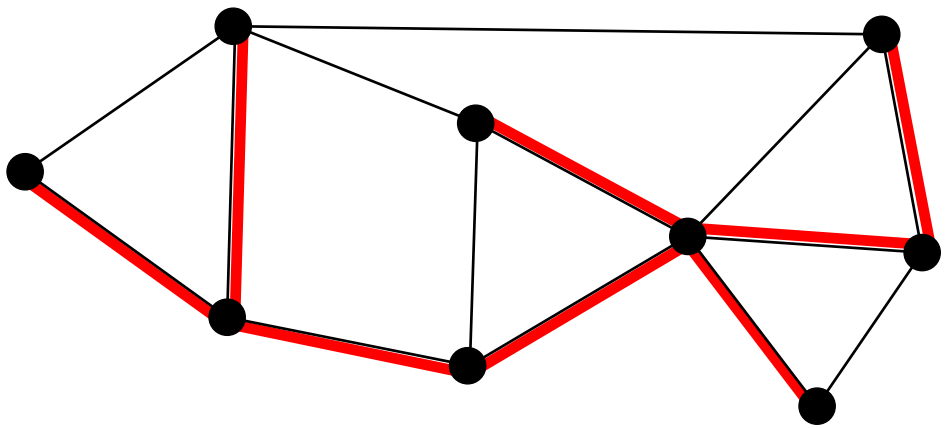
- graphe non orienté G
- arbre T $\begin{cases} \text{sommets : tous les sommets de } G \\ \text{arêtes : certaines arêtes de } G \end{cases}$



T arbre couvrant de G

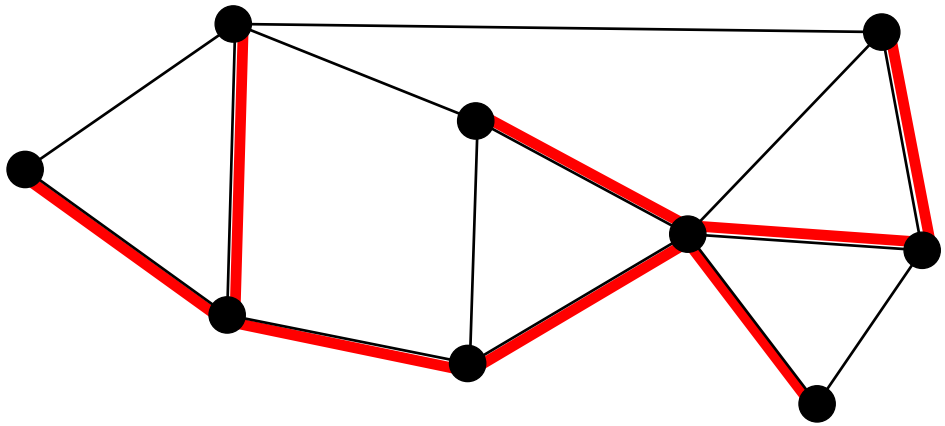
Définition

- graphe non orienté G
- arbre T $\begin{cases} \text{sommets : tous les sommets de } G \\ \text{arêtes : certaines arêtes de } G \end{cases}$



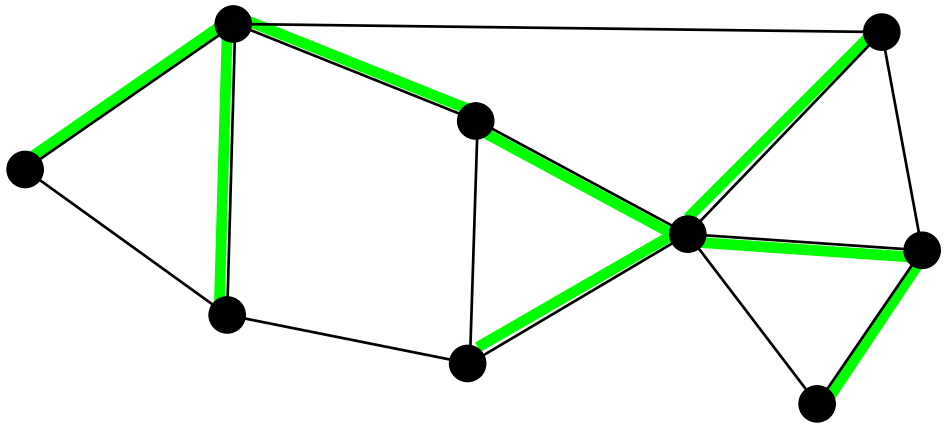
T arbre couvrant de G

Remarques



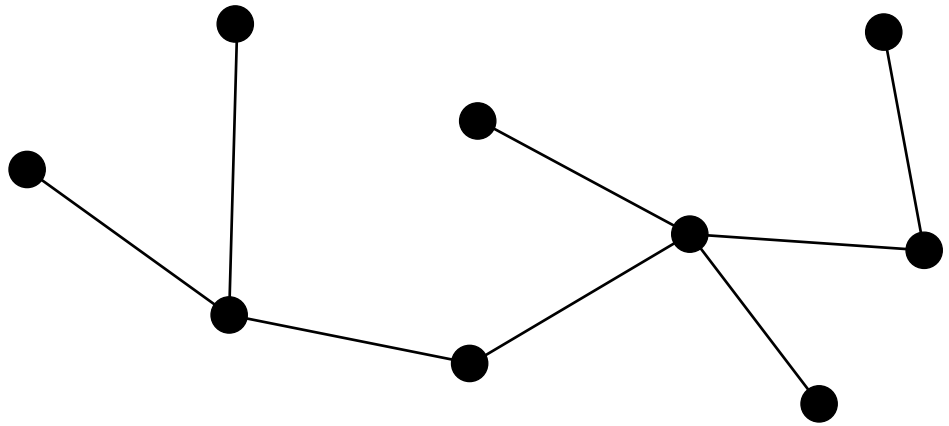
1. Un graphe peut avoir plusieurs arbres couvrants.
2. Un arbre n'a qu'un seul arbre couvrant, lui-même.
3. Un graphe non connexe n'a aucun arbre couvrant.
   (Autrement dit, un graphe qui a un arbre couvrant est forcément connexe.)
4. Un graphe connexe a forcément (au moins) un arbre couvrant.

Remarques



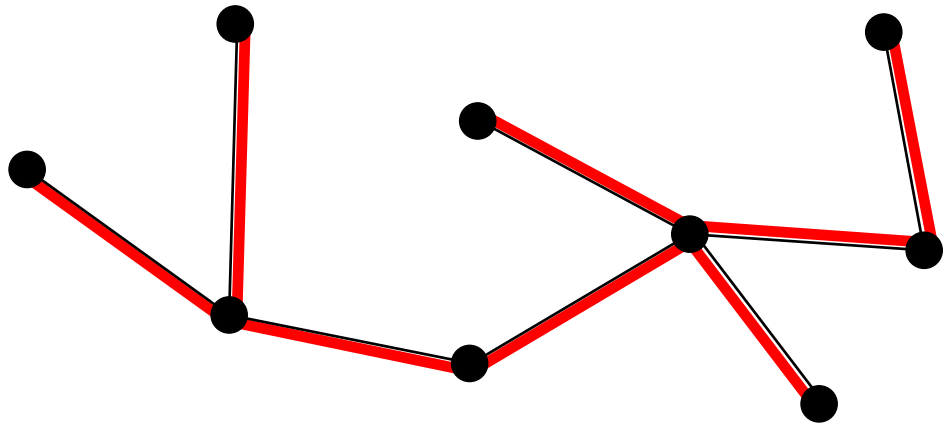
1. Un graphe peut avoir plusieurs arbres couvrants.
2. Un arbre n'a qu'un seul arbre couvrant, lui-même.
3. Un graphe non connexe n'a aucun arbre couvrant.
   (Autrement dit, un graphe qui a un arbre couvrant est forcément connexe.)
4. Un graphe connexe a forcément (au moins) un arbre couvrant.

Remarques



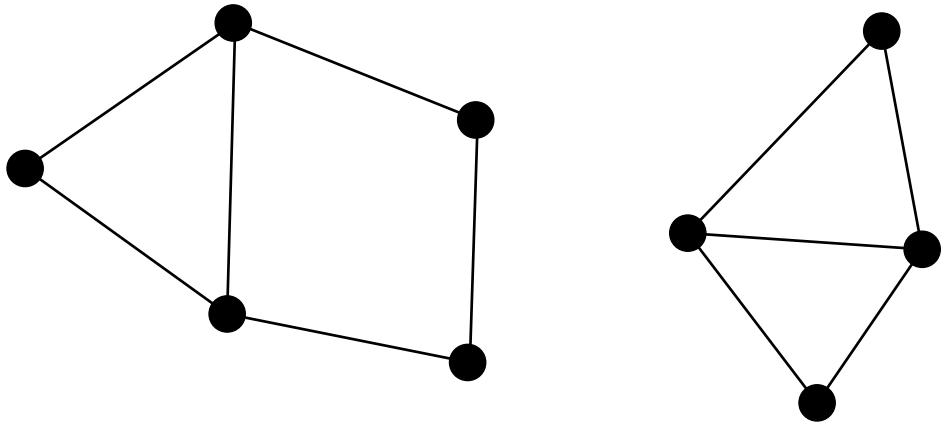
1. Un graphe peut avoir plusieurs arbres couvrants.
2. Un arbre n'a qu'un seul arbre couvrant, lui-même.
3. Un graphe non connexe n'a aucun arbre couvrant.
   (Autrement dit, un graphe qui a un arbre couvrant est forcément connexe.)
4. Un graphe connexe a forcément (au moins) un arbre couvrant.

Remarques



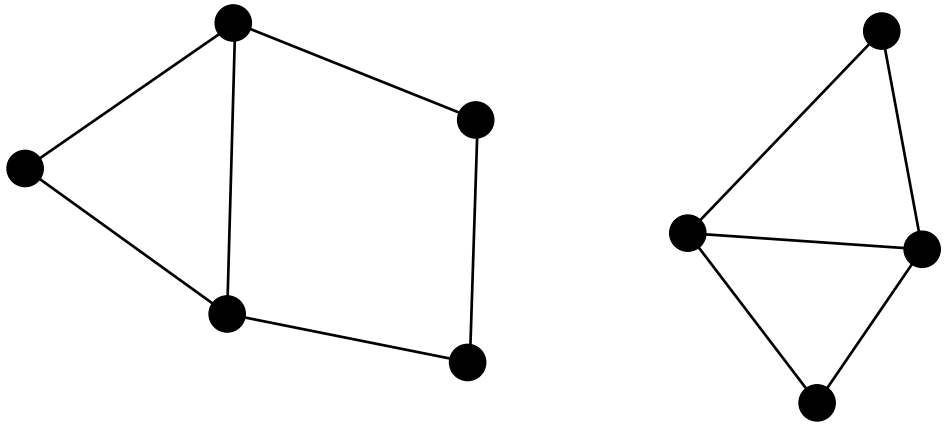
1. Un graphe peut avoir plusieurs arbres couvrants.
2. Un arbre n'a qu'un seul arbre couvrant, lui-même.
3. Un graphe non connexe n'a aucun arbre couvrant.
   (Autrement dit, un graphe qui a un arbre couvrant est forcément connexe.)
4. Un graphe connexe a forcément (au moins) un arbre couvrant.

Remarques



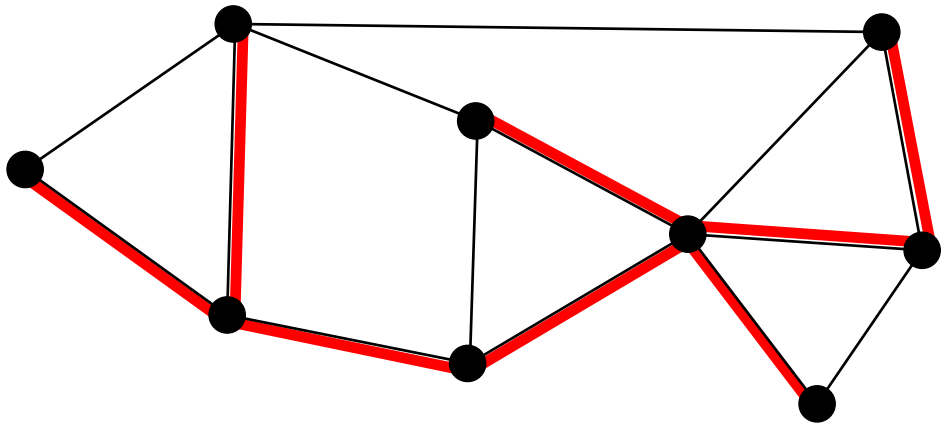
1. Un graphe peut avoir plusieurs arbres couvrants.
2. Un arbre n'a qu'un seul arbre couvrant, lui-même.
3. Un graphe non connexe n'a aucun arbre couvrant.
   (Autrement dit, un graphe qui a un arbre couvrant est forcément connexe.)
4. Un graphe connexe a forcément (au moins) un arbre couvrant.

Remarques



1. Un graphe peut avoir plusieurs arbres couvrants.
2. Un arbre n'a qu'un seul arbre couvrant, lui-même.
3. Un graphe non connexe n'a aucun arbre couvrant.
   (Autrement dit, un graphe qui a un arbre couvrant est forcément connexe.)
4. Un graphe connexe a forcément (au moins) un arbre couvrant.

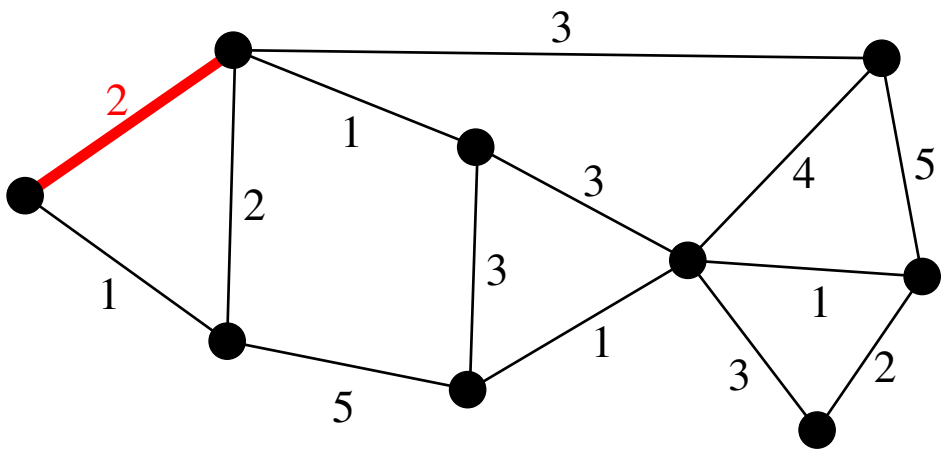
Remarques



1. Un graphe peut avoir plusieurs arbres couvrants.
2. Un arbre n'a qu'un seul arbre couvrant, lui-même.
3. Un graphe non connexe n'a aucun arbre couvrant.
   (Autrement dit, un graphe qui a un arbre couvrant est forcément connexe.)
4. Un graphe connexe a forcément (au moins) un arbre couvrant.

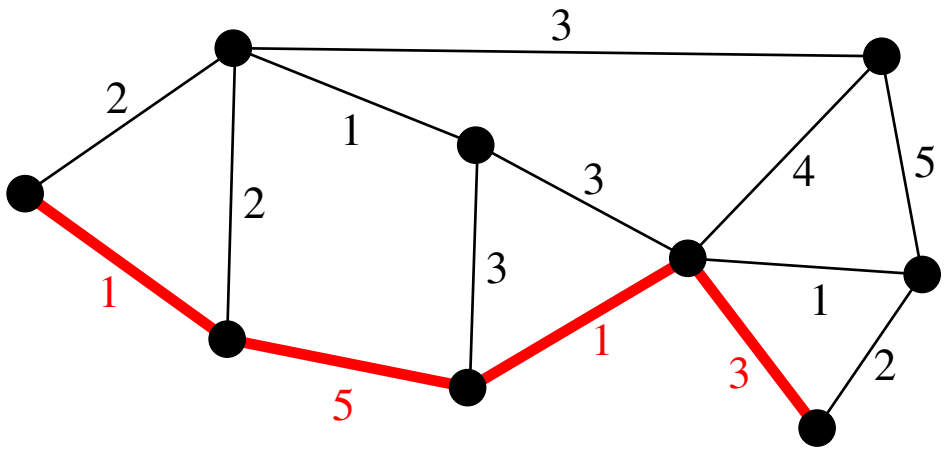
Définition

- Chaque arête a un poids (> 0)
- Le poids d'une chaîne est la somme des poids des arêtes qui la composent.



Définition

- Chaque arête a un poids (> 0)
- Le poids d'une chaîne est la somme des poids des arêtes qui la composent.

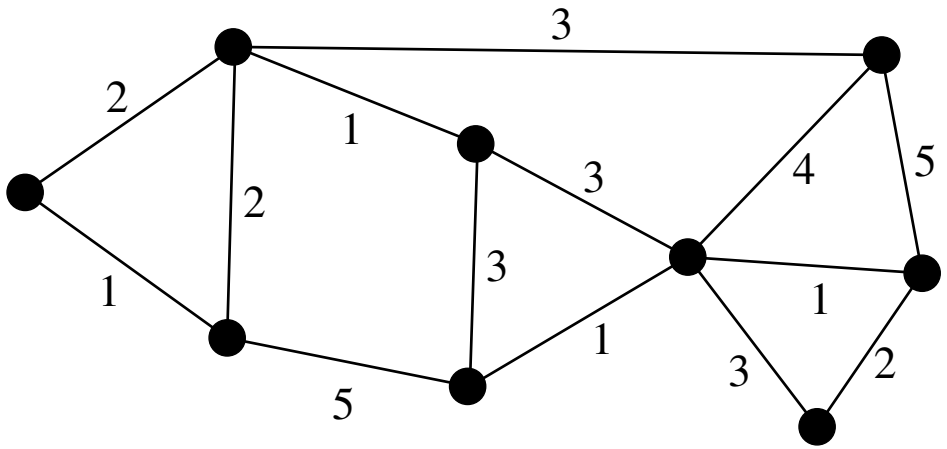


$$1 + 5 + 1 + 3 = 10$$

Question

Étant donné un graphe valué connexe

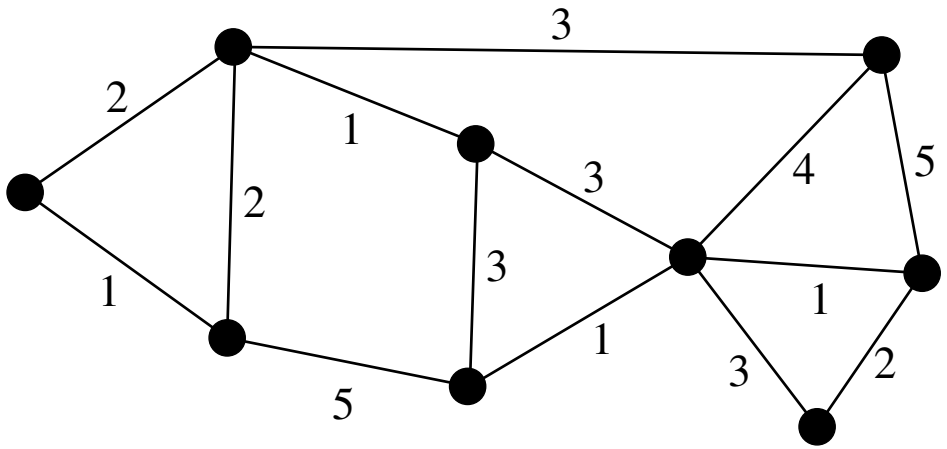
On veut construire un arbre couvrant de poids total le plus petit possible



Question

Étant donné un graphe valué connexe

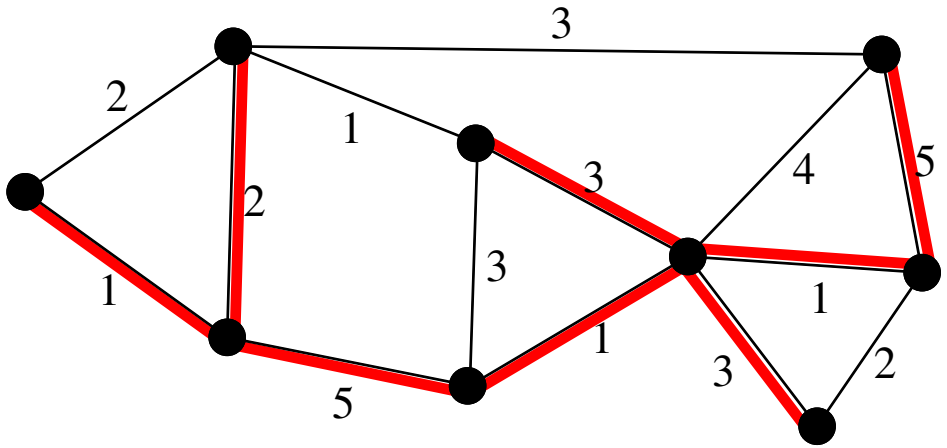
On veut construire un arbre couvrant
de poids total le plus petit possible



Question

Étant donné un graphe valué connexe

On veut construire un arbre couvrant
de poids total le plus petit possible



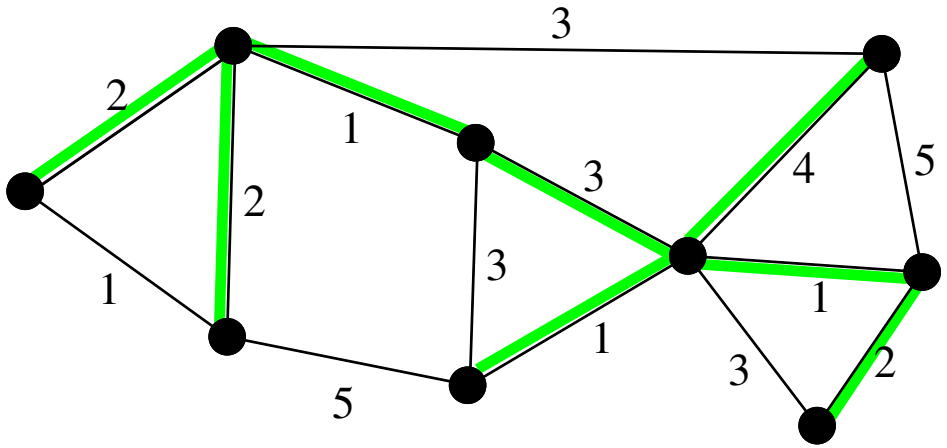
$1 + 2 + 5 + 1 + 3 + 5 + 1 + 3 = 21$

Peut-on faire mieux ?

Question

Étant donné un graphe valué connexe

On veut construire un arbre couvrant
de poids total le plus petit possible



$2 + 2 + 1 + 3 + 1 + 4 + 1 + 2 = 16$

Peut-on faire mieux ?

Algorithme glouton

Définition (algorithme glouton / greedy algorithm)

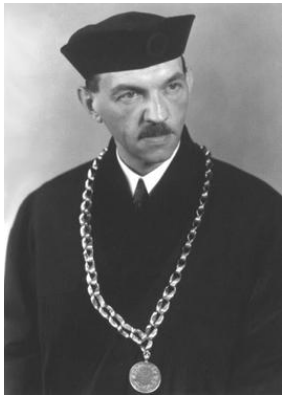
C'est un type d'algorithme qui construit une solution **pas à pas**, en prenant la meilleure décision **à chaque étape**

$\rightarrow$ (optimum **local** mais pas forcément **global**).

Le Glouton ou Carcajou, un animal du grand nord

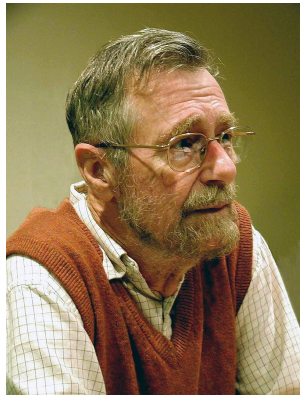
("Wolverine" en anglais)

Algorithme de Jarník-Prim-Dijkstra (1930/1957/1959)



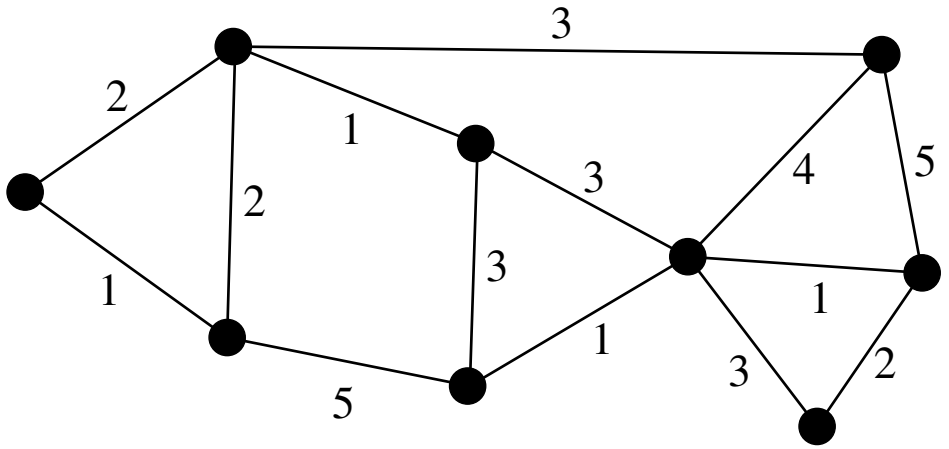
Vojtěch Jarník
(1897-1970)

Robert C. Prim
(1921-)



Edsger W. Dijkstra
(1930-2002)

Idée générale

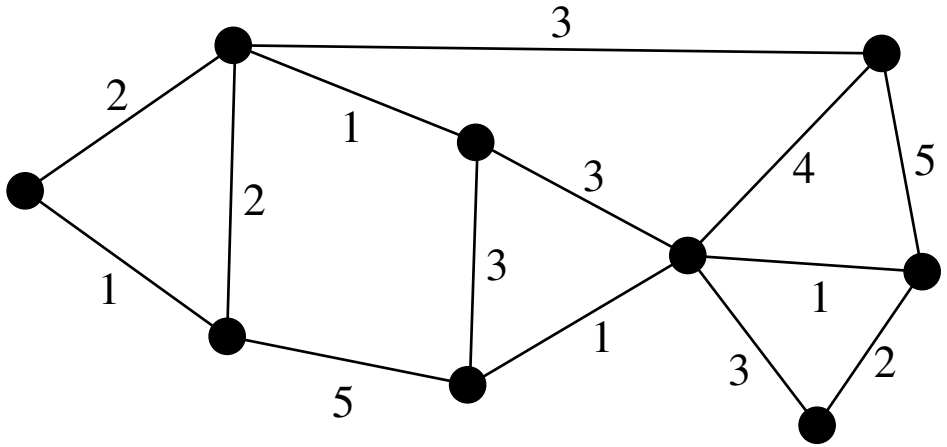


- On part d'un arbre initial réduit à un seul sommet du graphe.
- À chaque itération, on agrandit l'arbre en lui ajoutant le sommet libre accessible de plus petit poids possible.
- On stoppe quand l'arbre est couvrant.

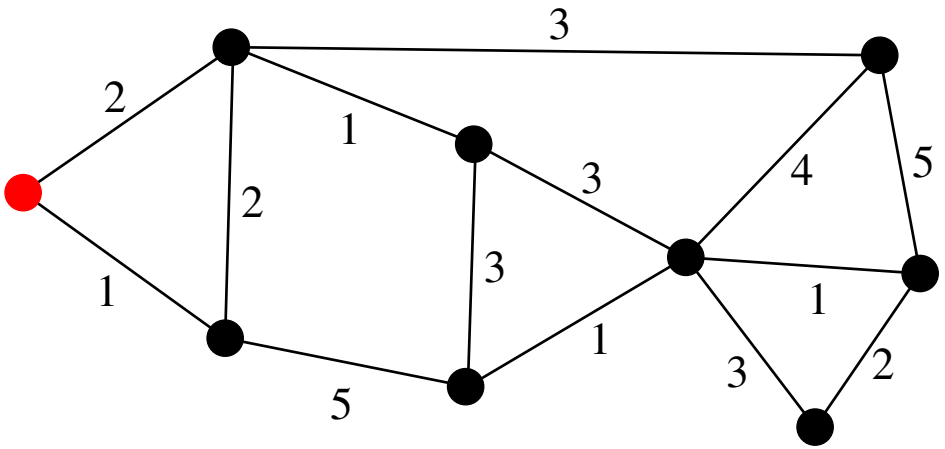
Méthode

- Initialiser T avec
  $\begin{cases} \text{sommets : un sommet de } G \text{ qu'on choisit} \\ \text{arêtes : aucune} \end{cases}$
- Répéter :
  - Trouver toutes les arêtes de G qui relient un sommet de T et un sommet extérieur à T
  - Parmi celles-ci, choisir une arête de poids le plus petit possible
  - Ajouter à T cette arête et le sommet correspondant
- S'arrêter dès que tous les sommets de G sont dans T
- Retourner T

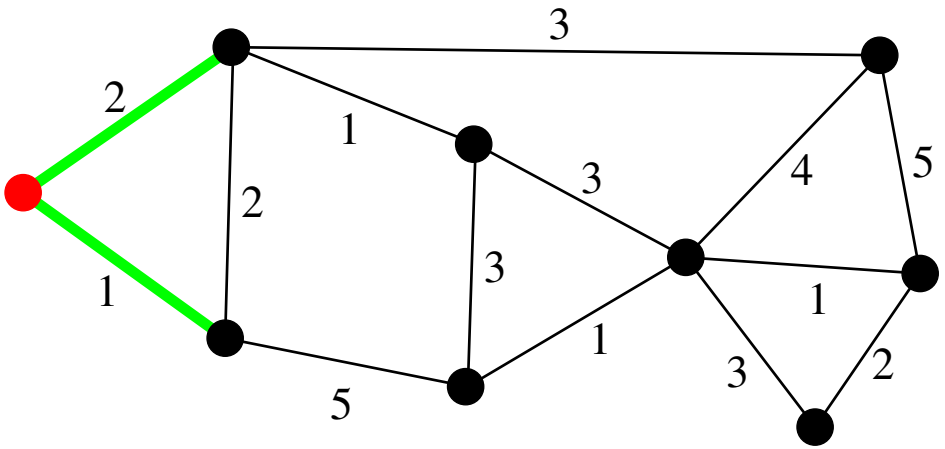
Exemple



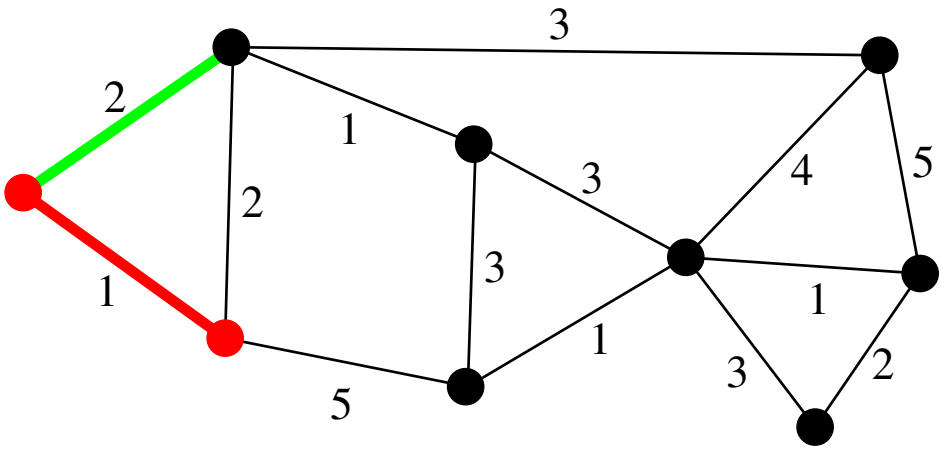
Exemple



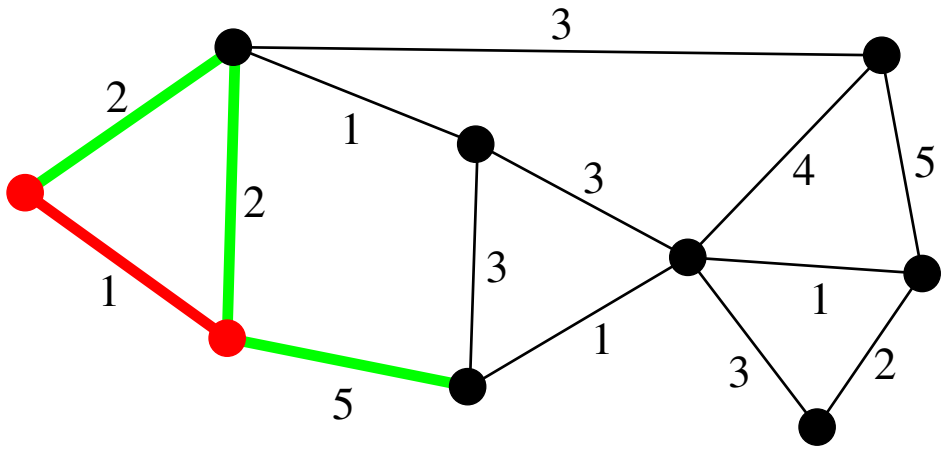
Exemple



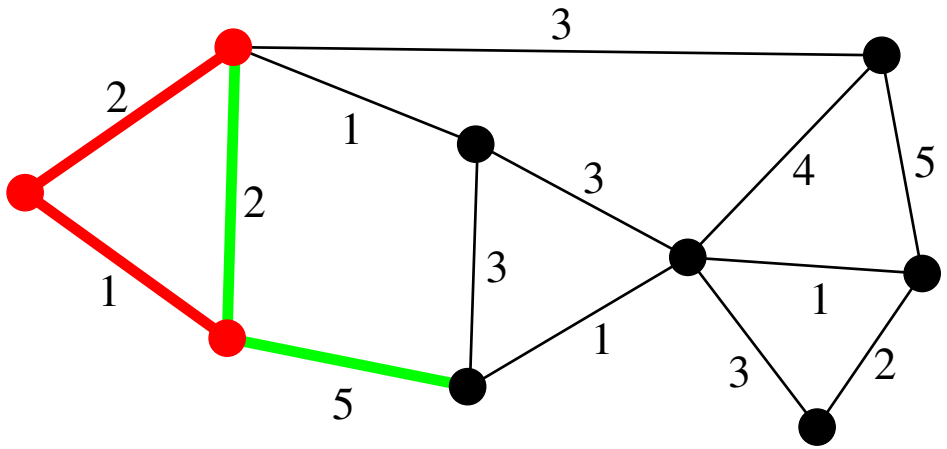
Exemple



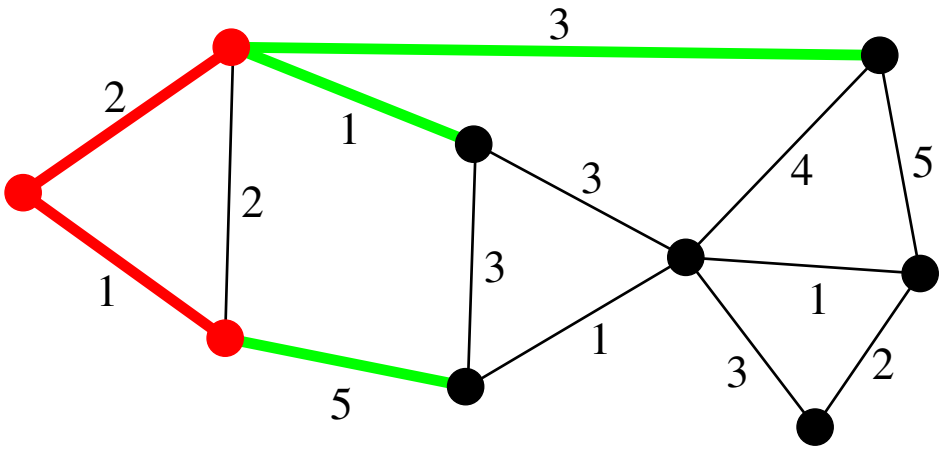
Exemple



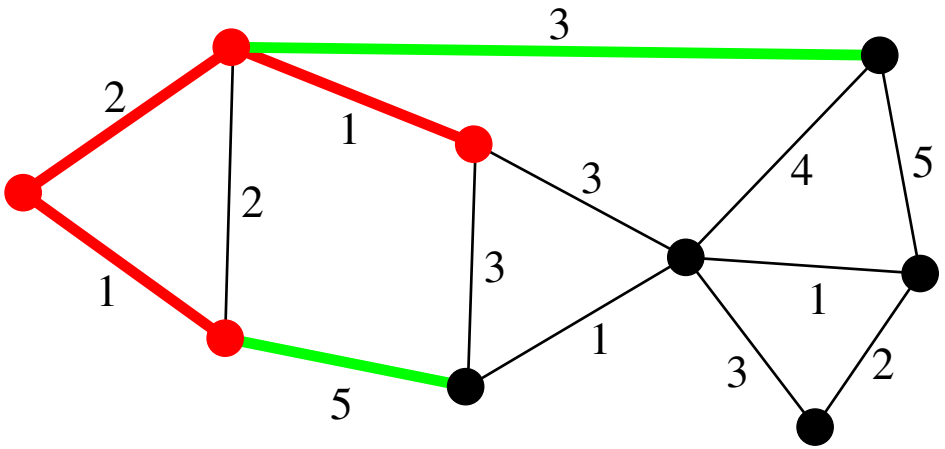
Exemple



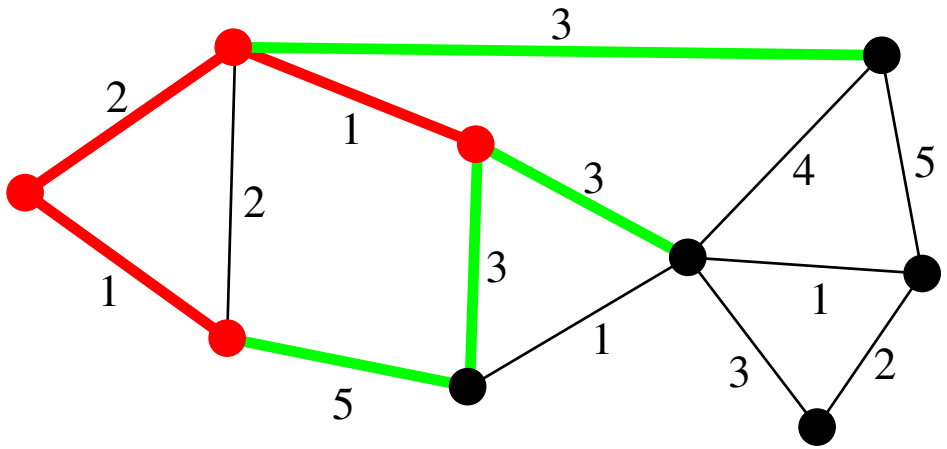
Exemple



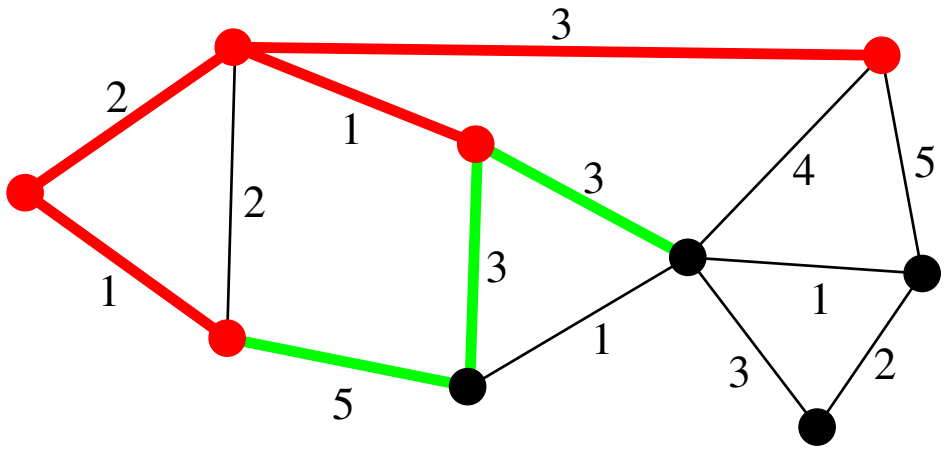
Exemple



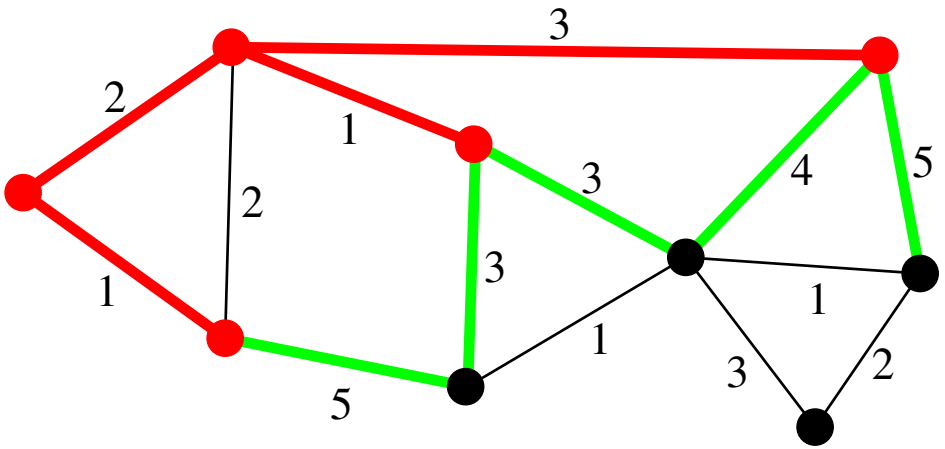
Exemple



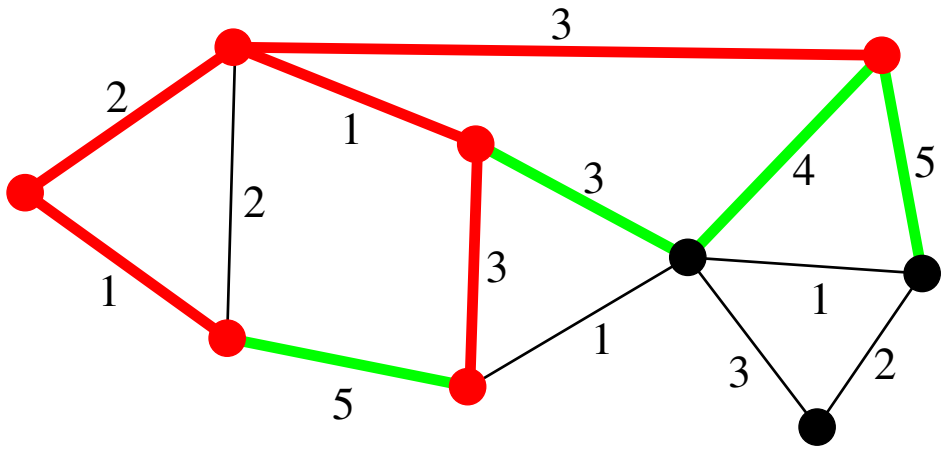
Exemple



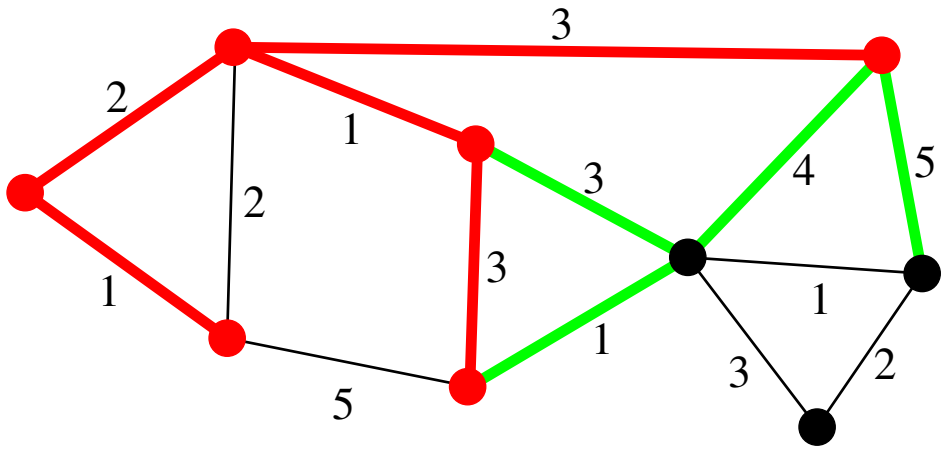
Exemple



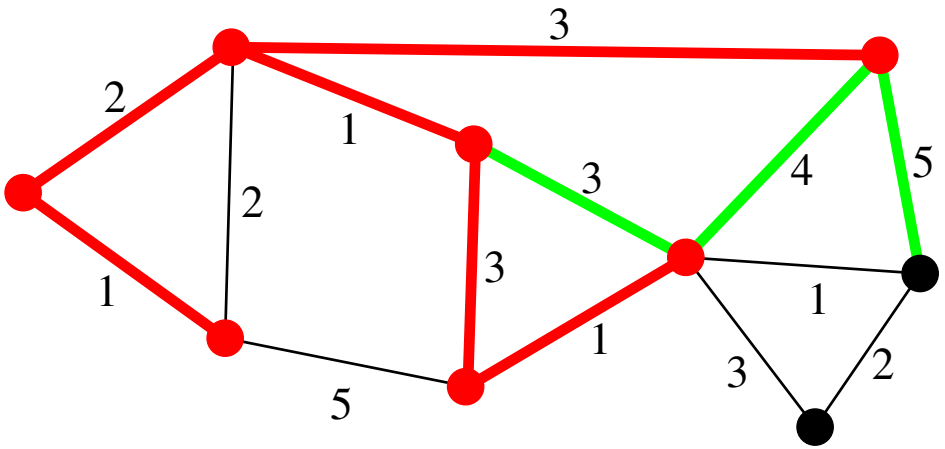
Exemple



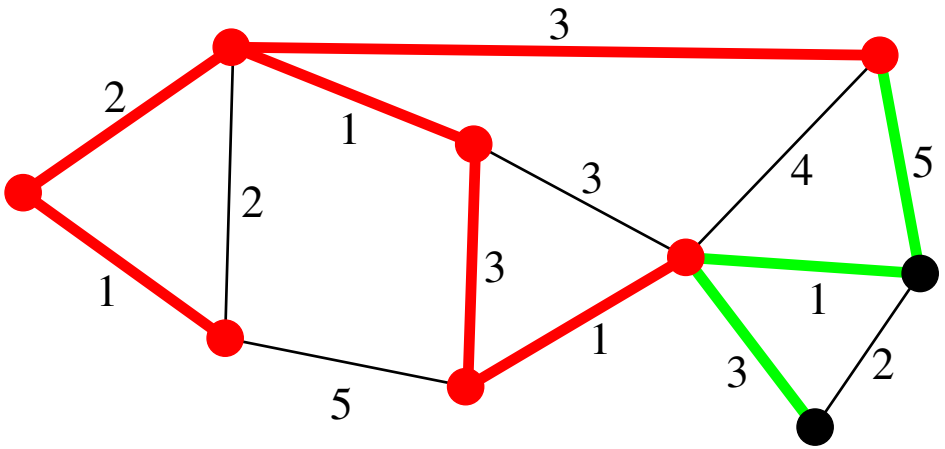
Exemple



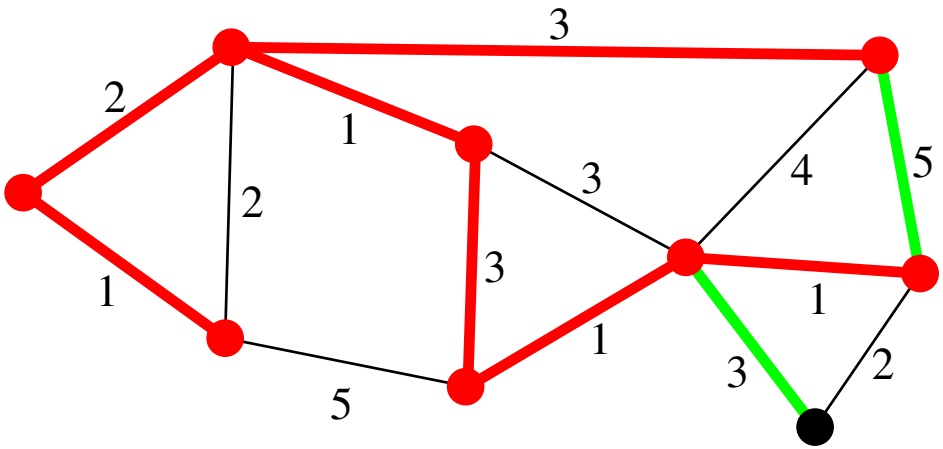
Exemple



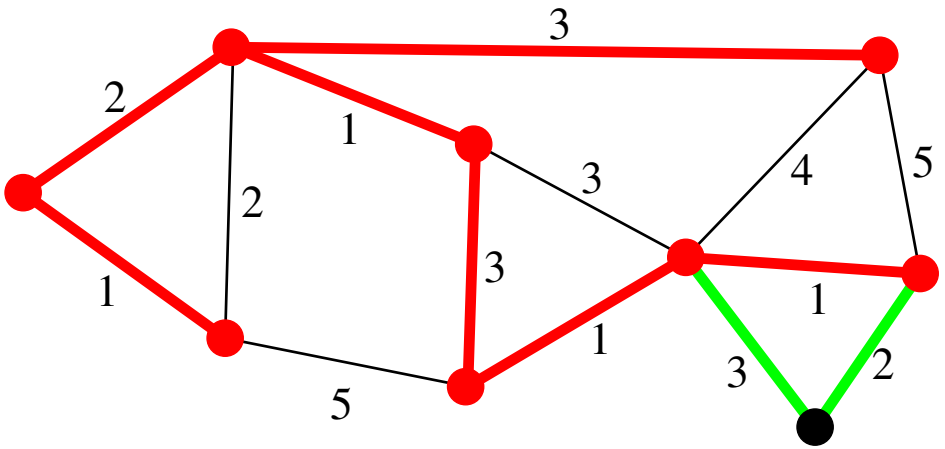
Exemple



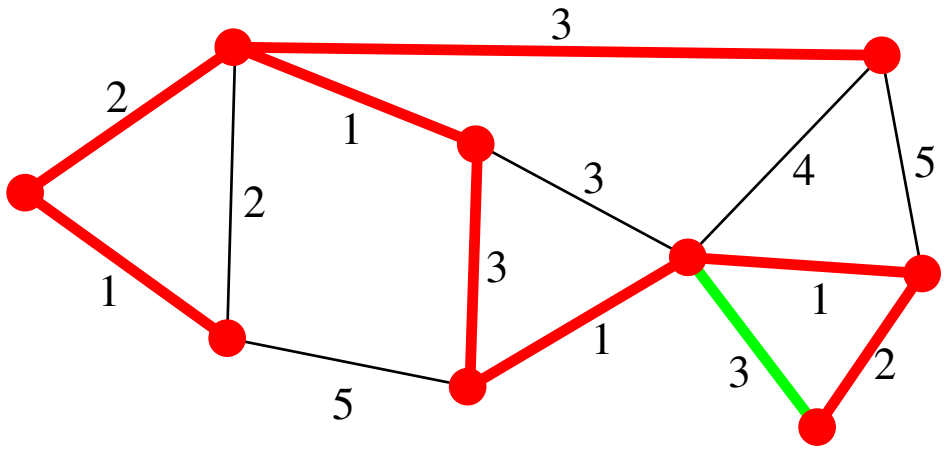
Exemple



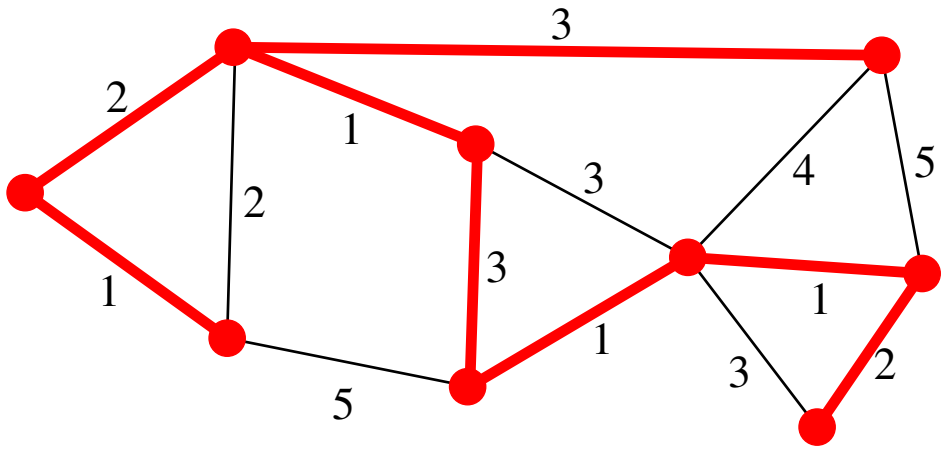
Exemple



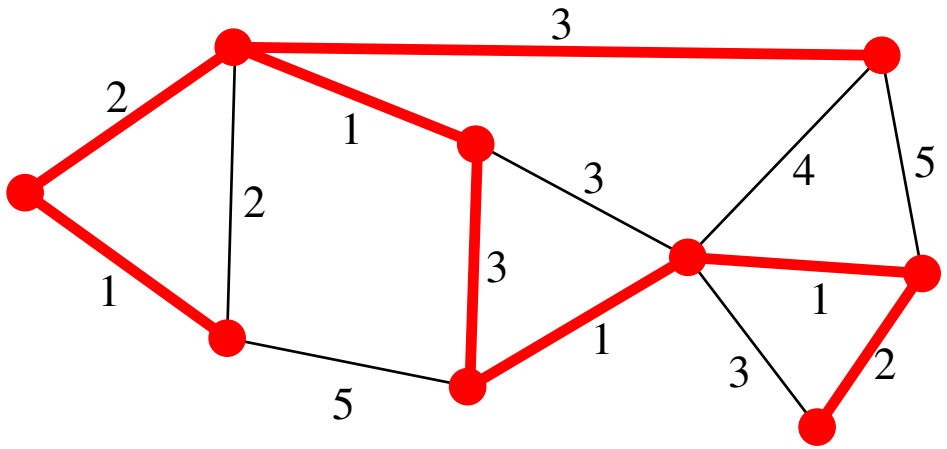
Exemple



Exemple



Exemple



$1 + 2 + 1 + 3 + 3 + 1 + 1 + 2 = 14$

On obtient un arbre couvrant de poids 14. Peut-on faire mieux ?

Algorithme de Prim

- Initialiser T avec
  $\begin{cases} \text{sommets : un sommet de } G \text{ qu'on choisit} \\ \text{arêtes : aucune} \end{cases}$
- Répéter :
  - Trouver toutes les arêtes de G qui relient un sommet de T et un sommet extérieur à T
  - Parmi celles-ci, choisir une arête de poids le plus petit possible
  - Ajouter à T cette arête et le sommet correspondant
- S'arrêter dès que tous les sommets de G sont dans T
- Retourner T

Pourquoi l'algorithme de Prim construit-il un arbre couvrant de poids minimal ?

Il faut une preuve !

Preuve : Terminaison

L'algorithme est bien défini car si la condition d'arrêt « *Tous les sommets de G sont dans T* » n'est pas réalisée, alors il existe au moins une arête qui relie un sommet v extérieur à T à un sommet de T (car G est connexe). Ce qui signifie qu'il sera possible d'ajouter une arête dans la boucle.

L'algorithme se termine car on ajoute à chaque passage un sommet et une arête, donc au bout d'un nombre fini d'itérations, la condition d'arrêt « *Tous les sommets de G sont dans T* » est réalisée.

Preuve : Correction

L'algorithme renvoie un arbre couvrant

Au départ, T est réduit à un sommet, c'est donc un arbre.

À chaque passage dans la boucle, on ajoute une arête et une feuille à l'arbre T, donc l'objet construit demeure encore un arbre.

Finalement, quand on sort de la boucle on retourne T qui est bien un arbre.

Comme l'arbre T obtenu à l'issue de l'algorithme a pour sommets tous les sommets de G et pour arêtes certaines arêtes de G, c'est bien un arbre couvrant de G.

Preuve : Correction

Il reste à vérifier que l'arbre couvrant T est de poids minimal.

On va utiliser la propriété ci-dessous, et montrer qu'elle reste vraie à chaque passage dans la boucle :

L'ensemble des arêtes choisies par l'algorithme de Prim est contenu dans un arbre couvrant de G de poids minimal.

Initialement, l'ensemble des arêtes choisies par l'agorithme de Prim est vide, donc la propriété est trivialement vérifiée.

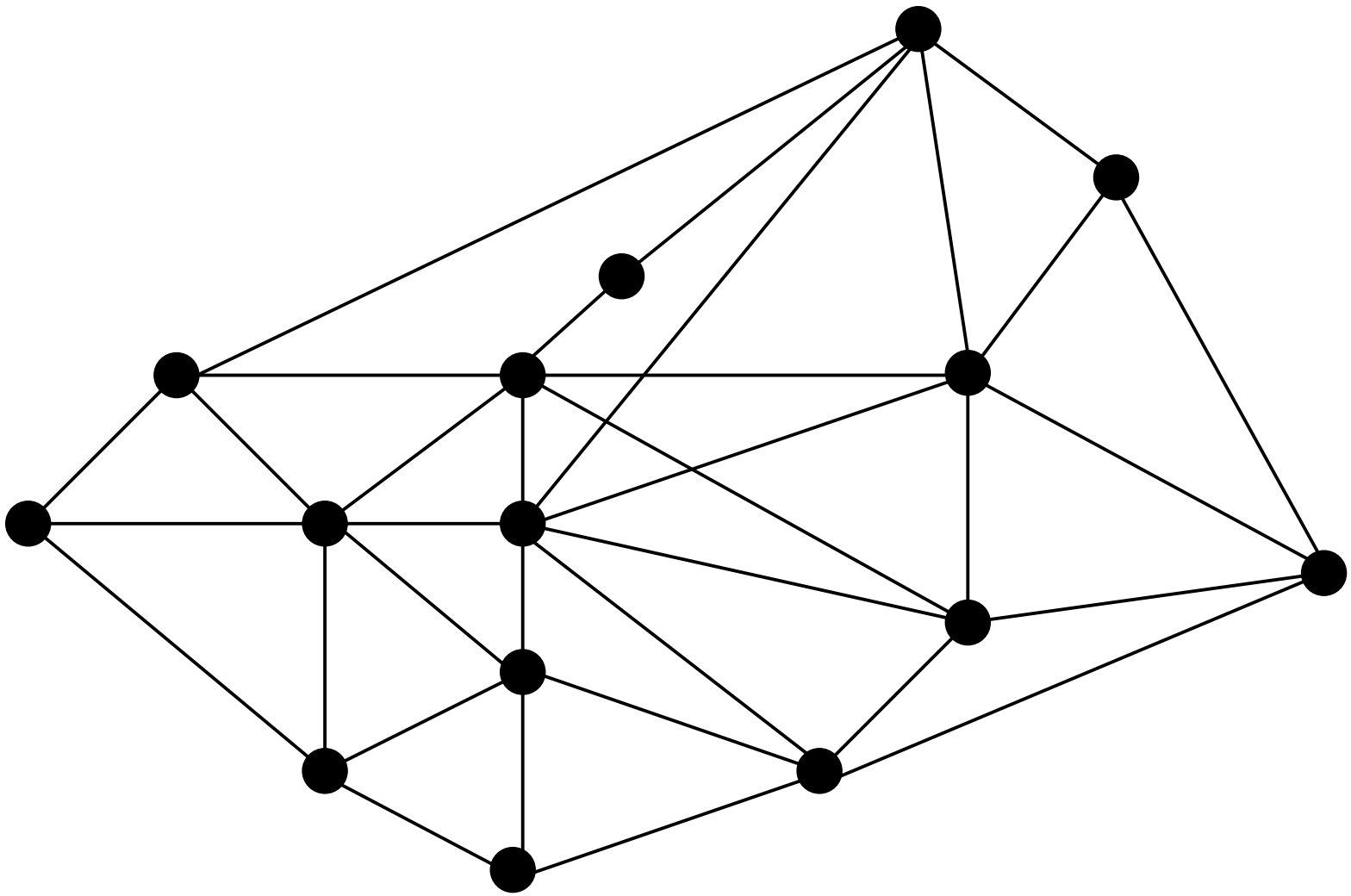
Preuve : Correction

Nous allons vérifier l'hérédité de la propriété :

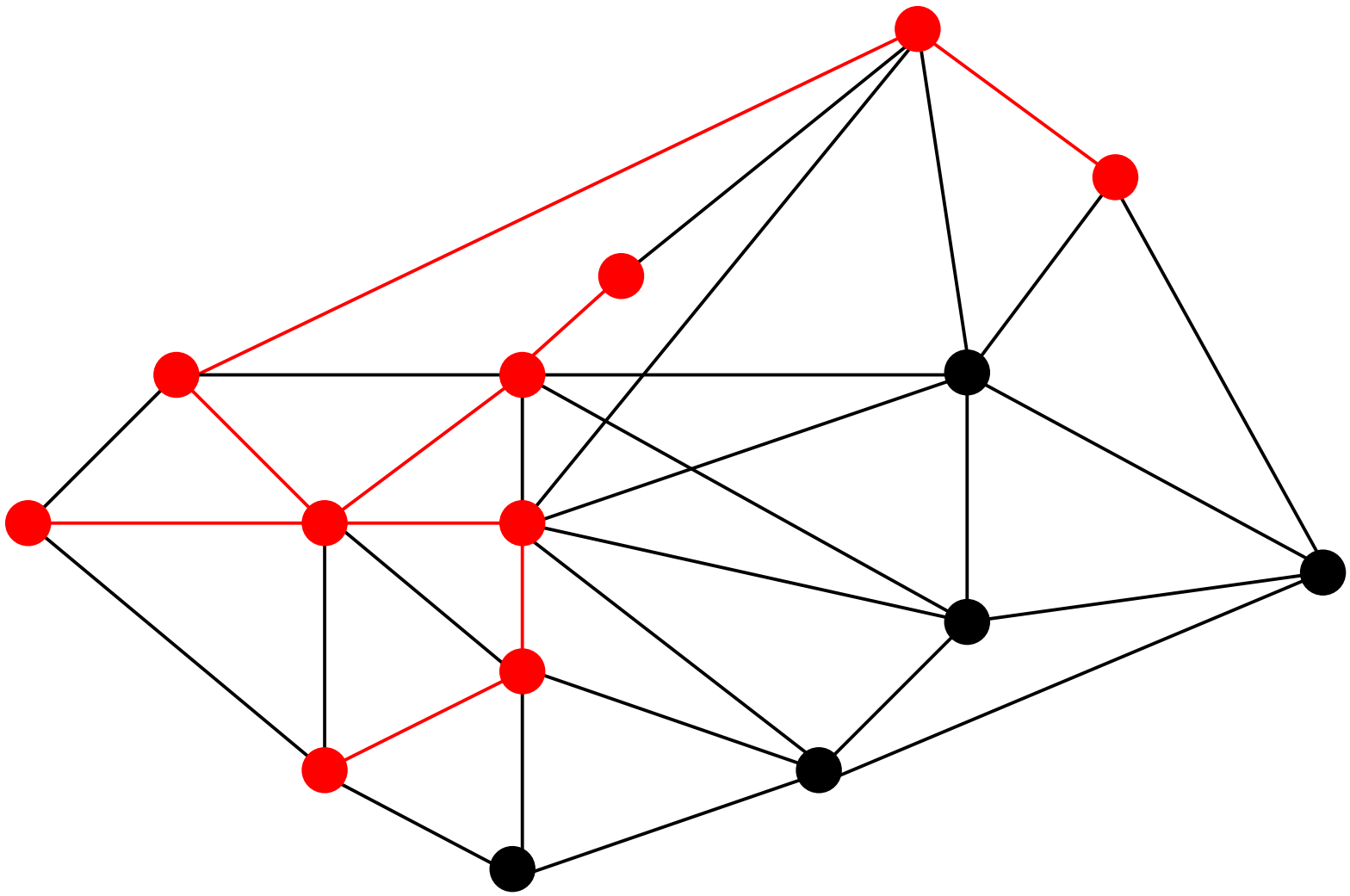
L'ensemble des arêtes choisies par l'algorithme de Prim est contenu dans un arbre couvrant de G de poids minimal.

Ceci nous va nous assurer qu'à la fin de l'exécution de l'algorithme, l'ensemble des arêtes composant T est contenu dans un arbre couvrant de G de poids minimal. Comme T constitue lui-même un arbre couvrant de G, c'est celui qu'on cherche.

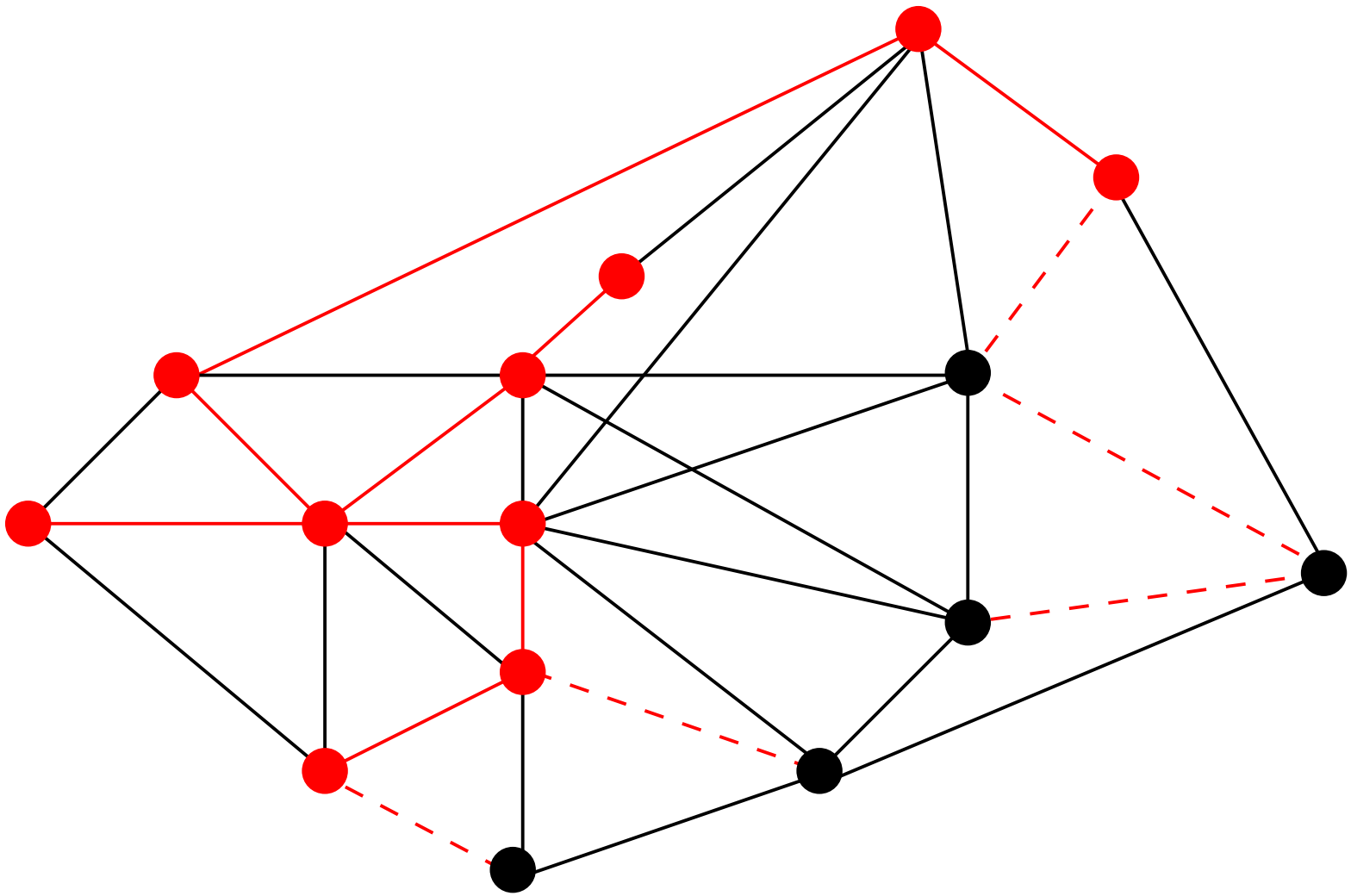
Preuve : Correction



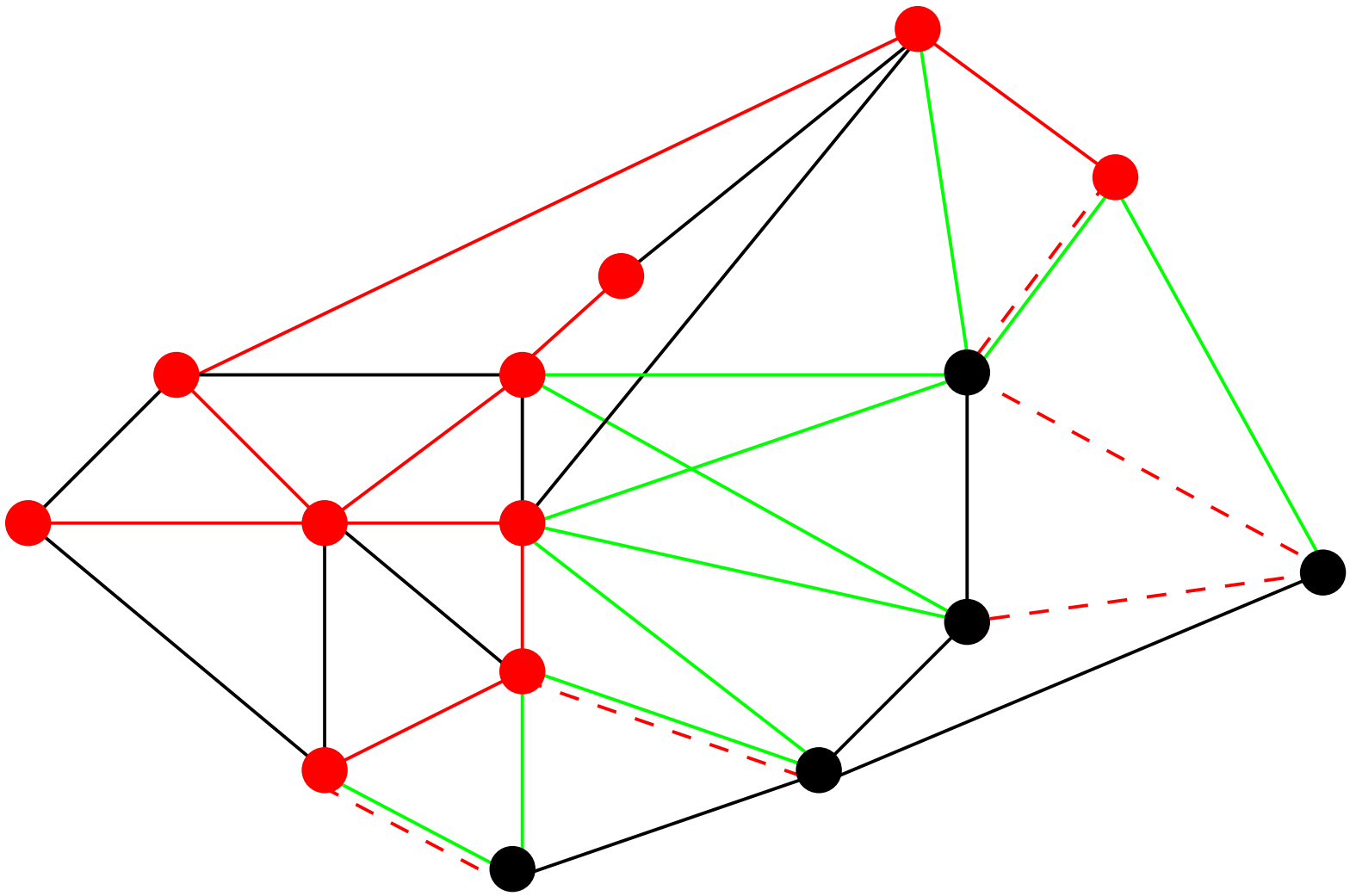
Preuve : Correction



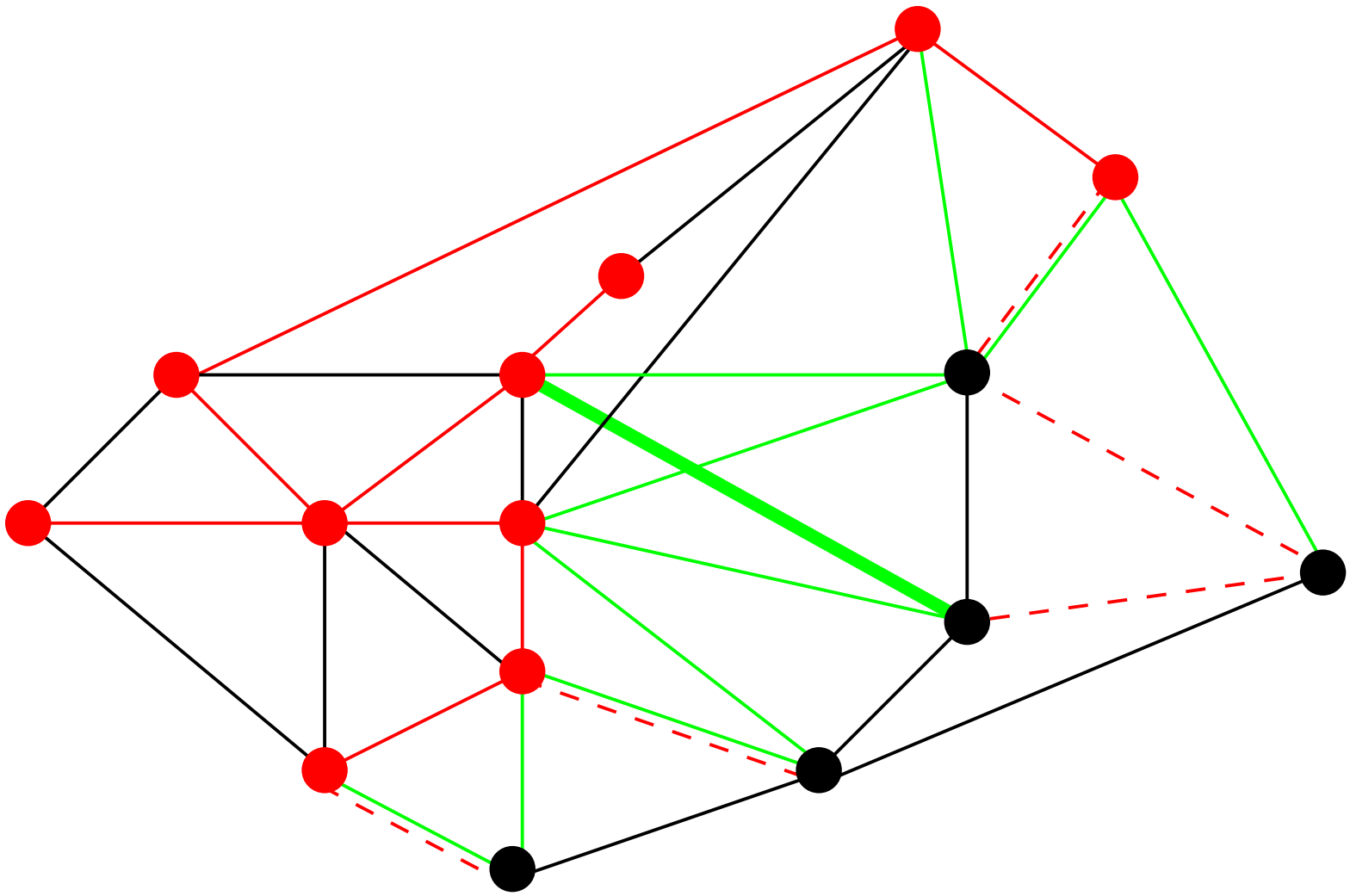
Preuve : Correction



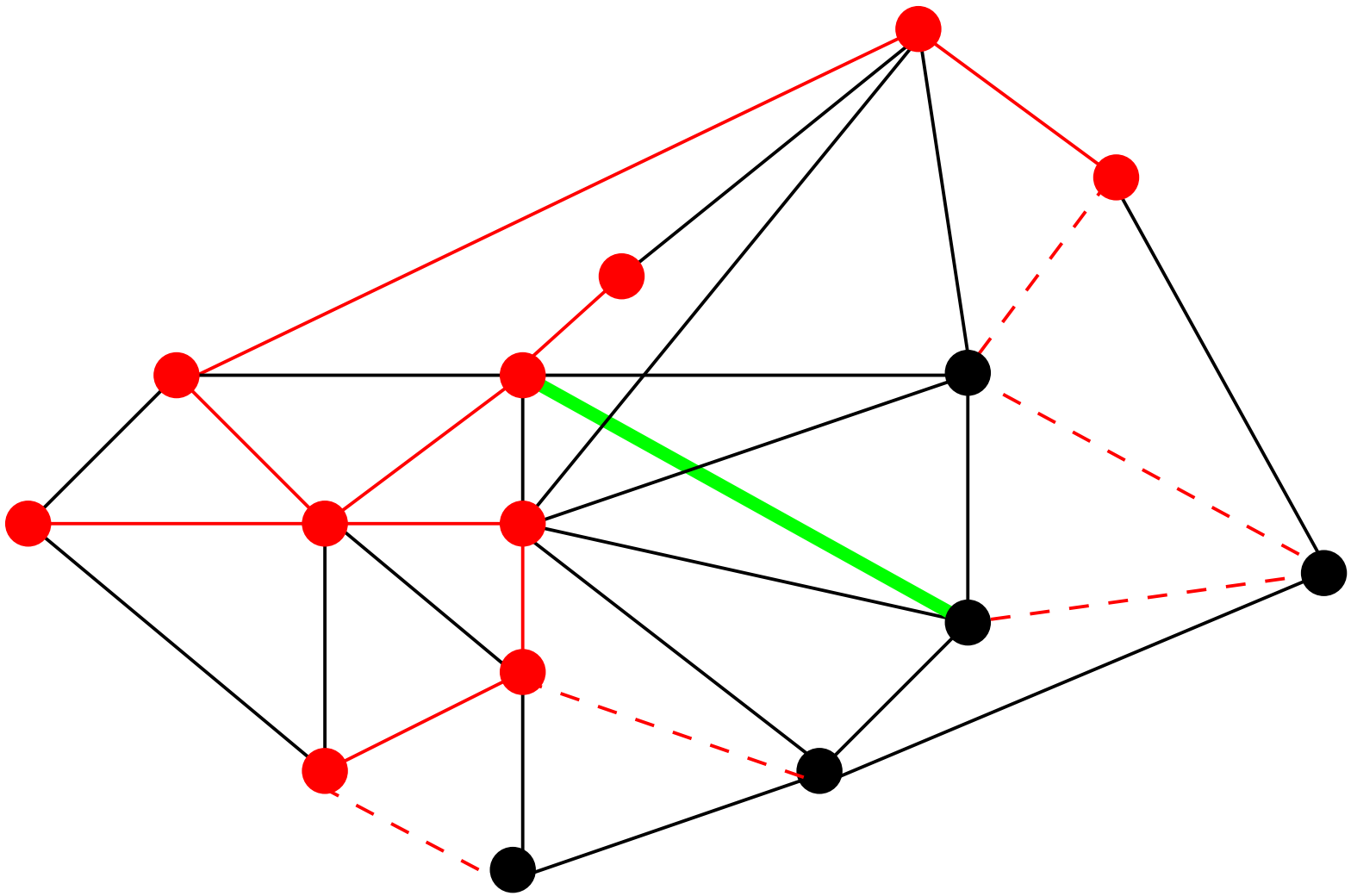
Preuve : Correction



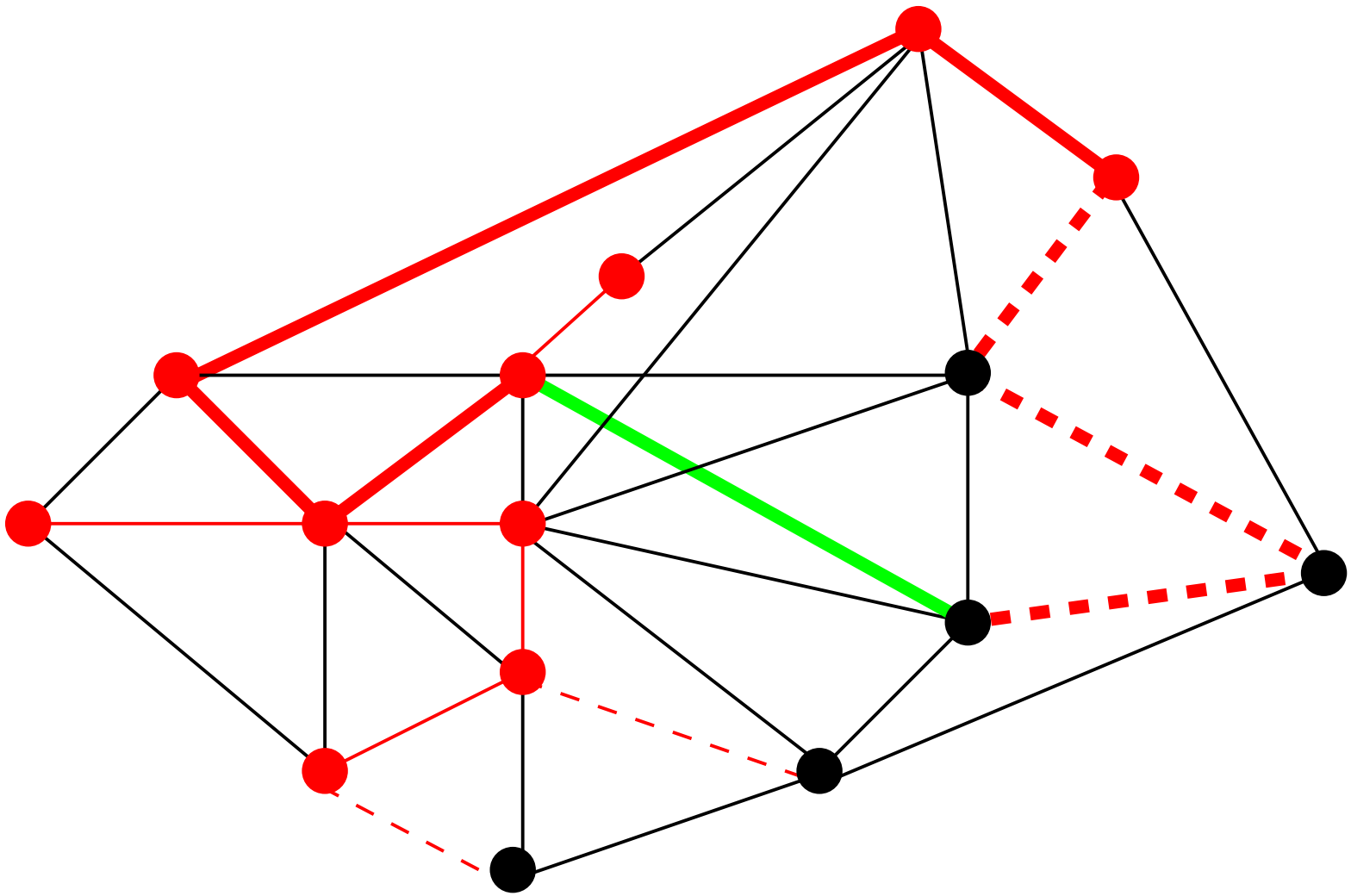
Preuve : Correction



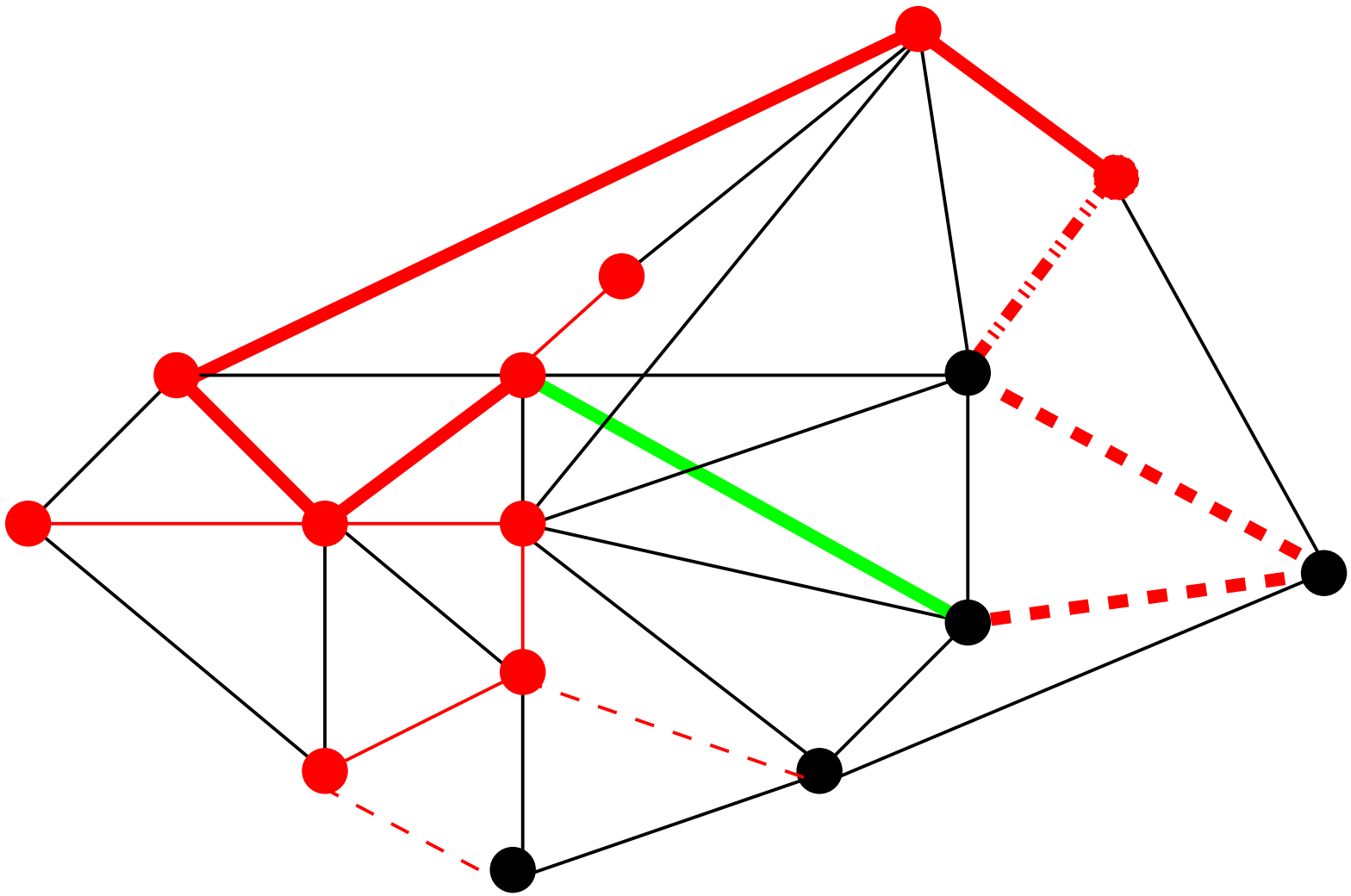
Preuve : Correction



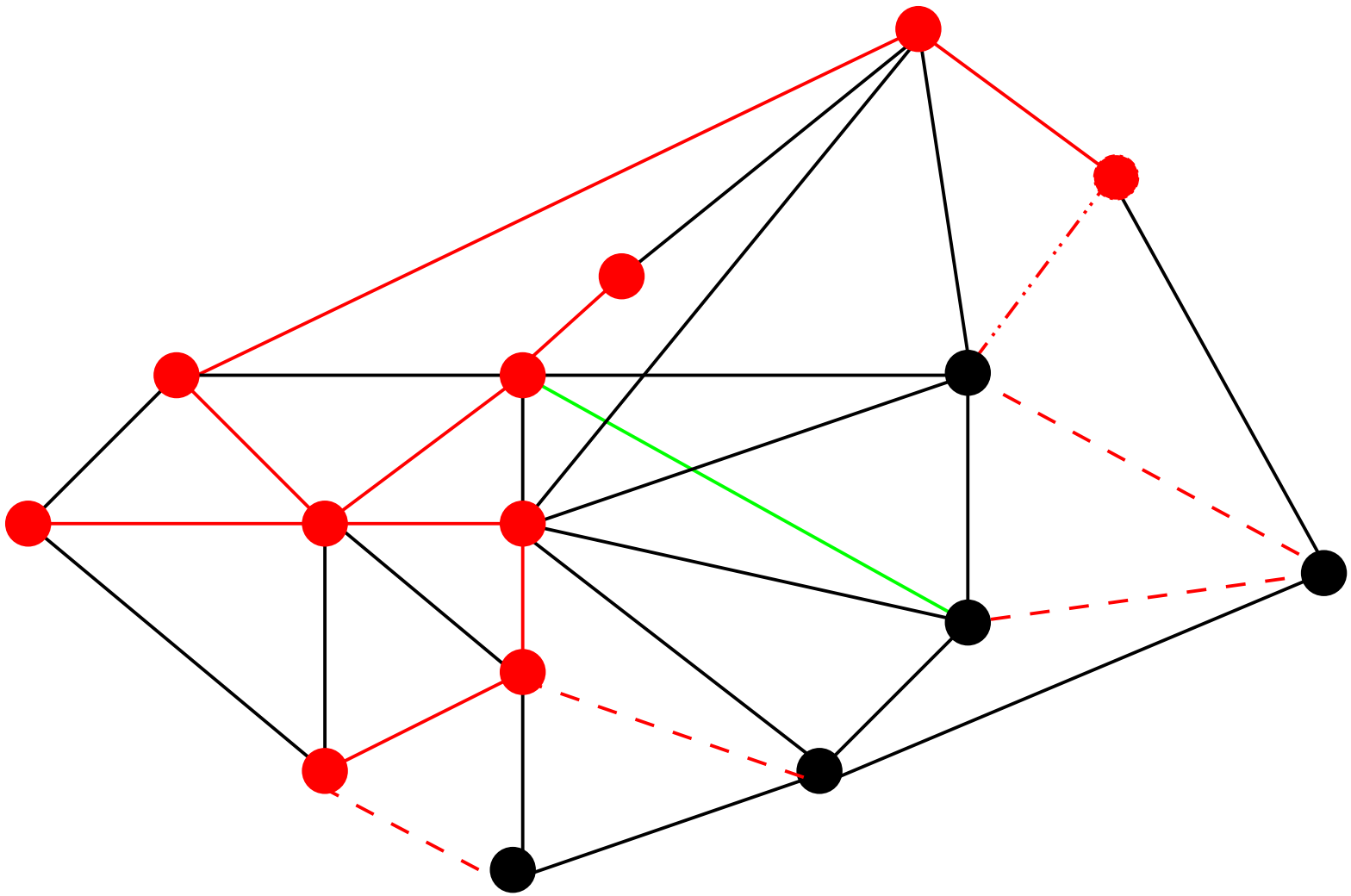
Preuve : Correction



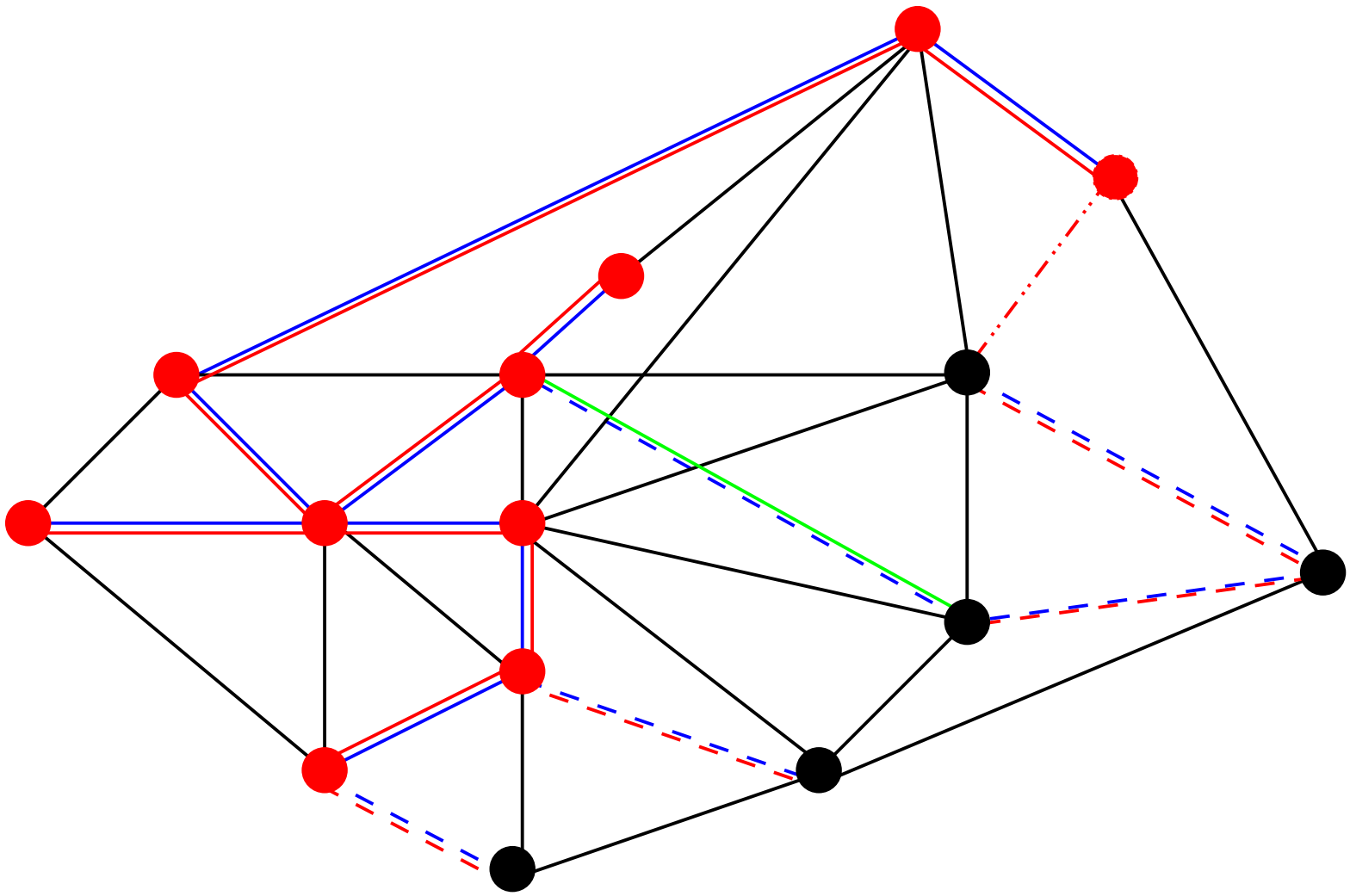
Preuve : Correction



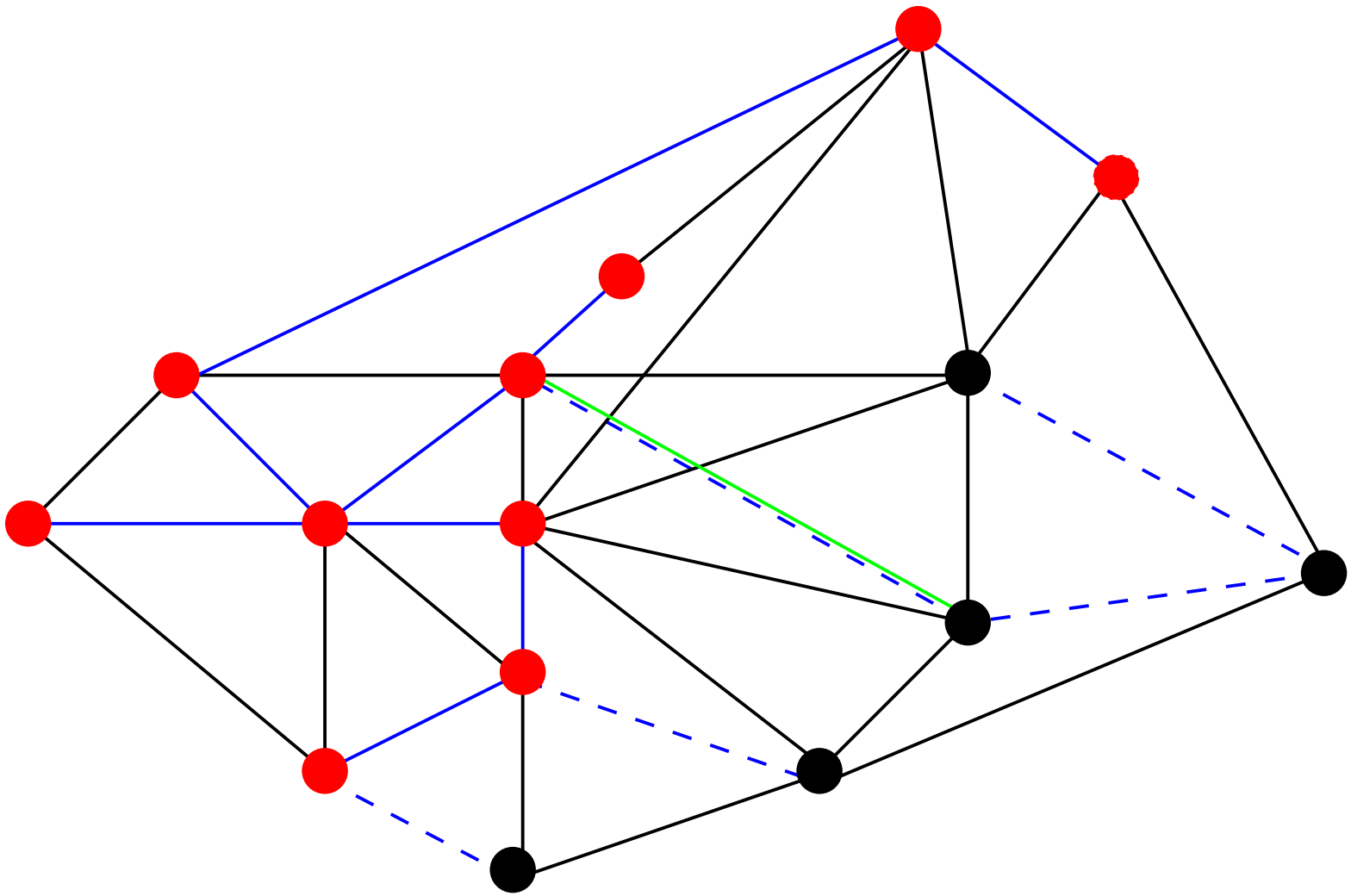
Preuve : Correction



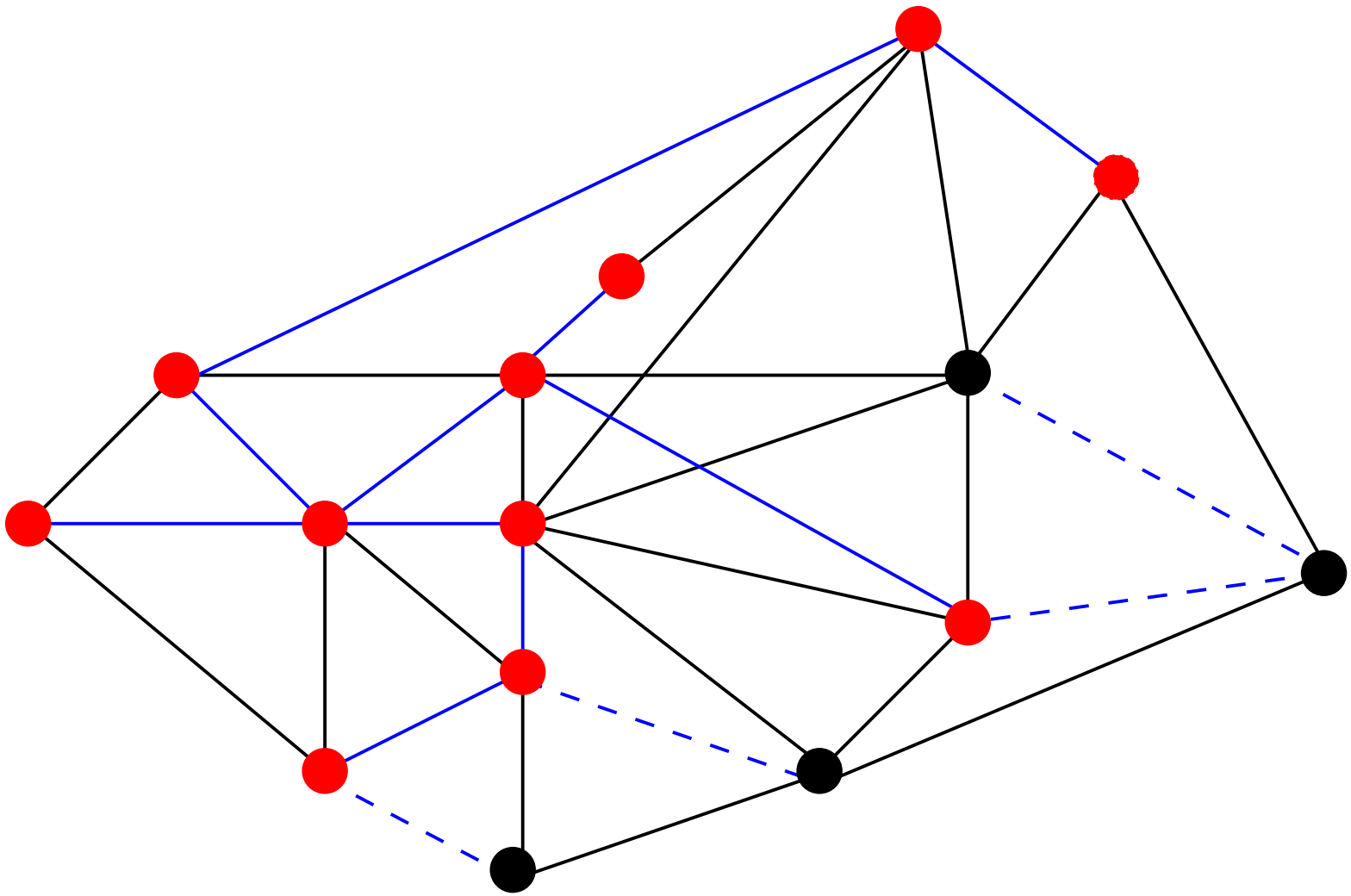
Preuve : Correction



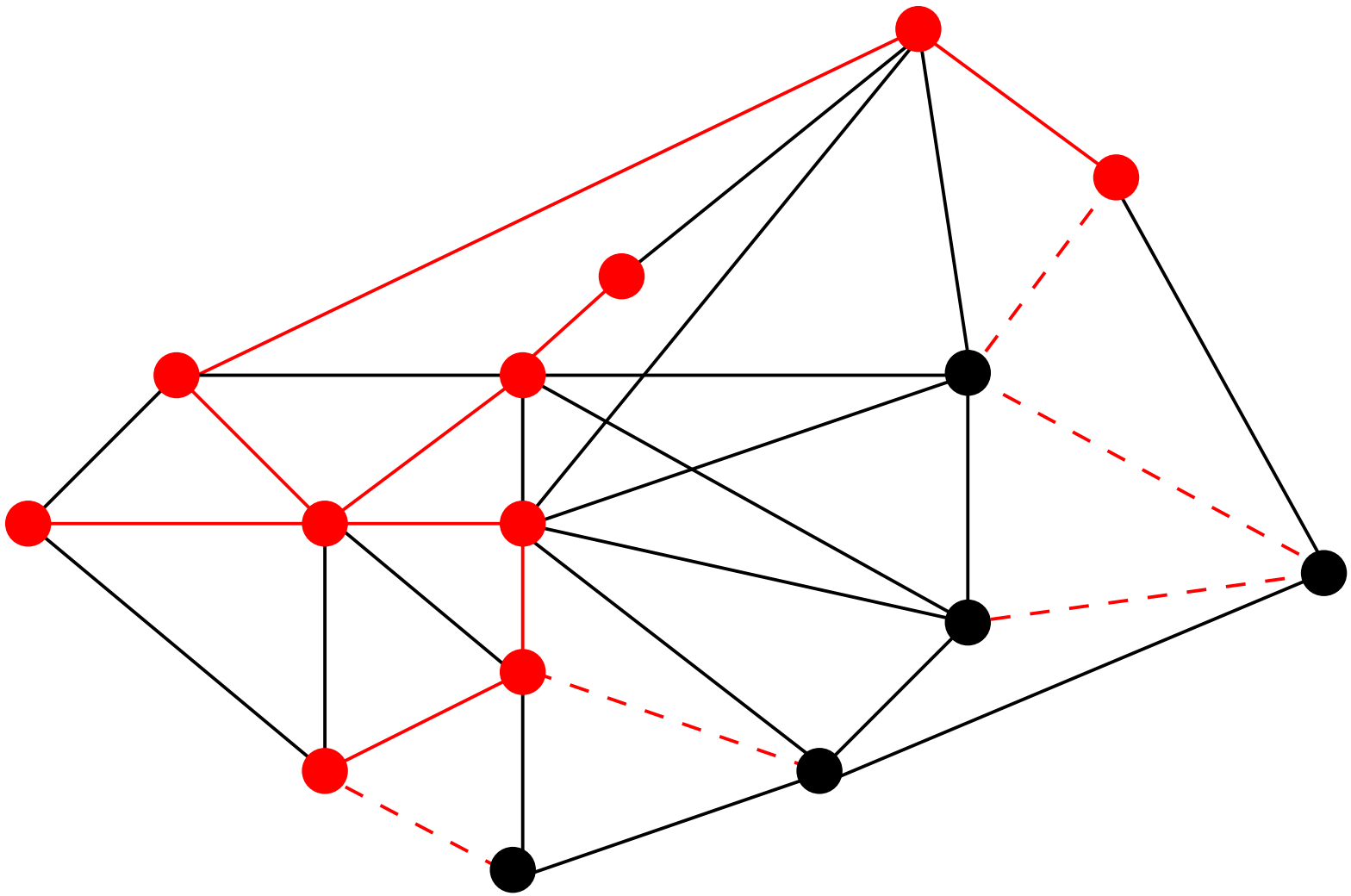
Preuve : Correction



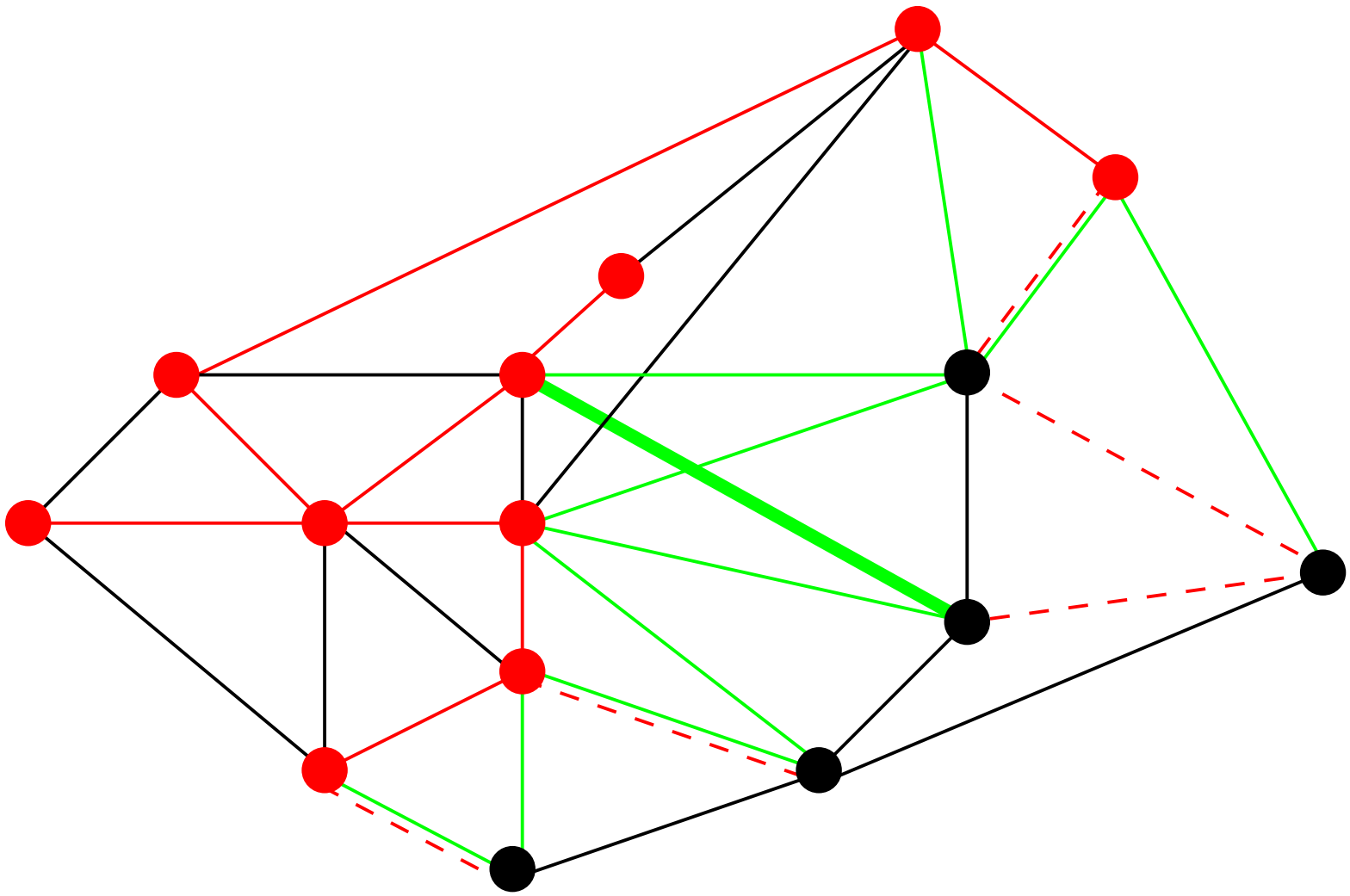
Preuve : Correction



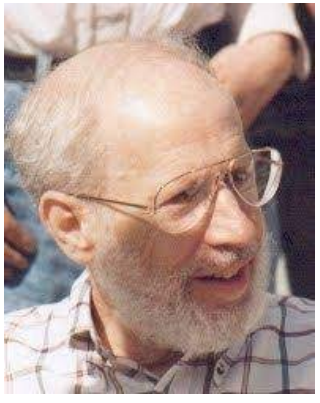
Preuve : Correction



Preuve : Correction

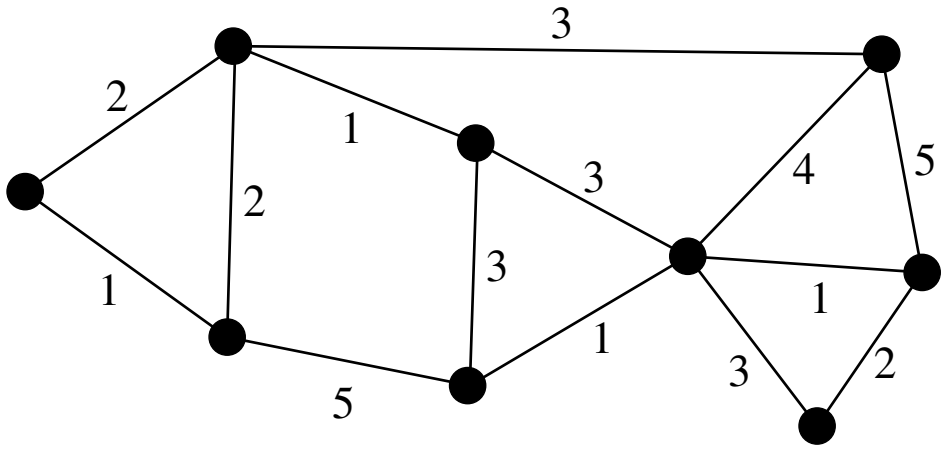


Algorithme de Kruskal (1956)



Joseph B. Kruskal, Jr
(1928-2010)

Idée générale

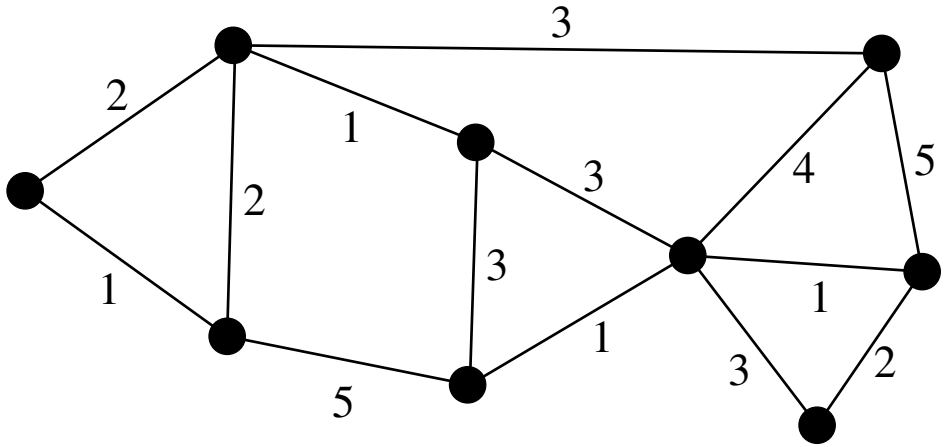


- On part d'une forêt d'arbres constitués de chacun des sommets isolés du graphe.
- À chaque itération, on ajoute à cette forêt l'arête de poids le plus faible ne créant pas de cycle avec les arêtes déjà choisies.
- On stoppe quand on obtient un arbre.

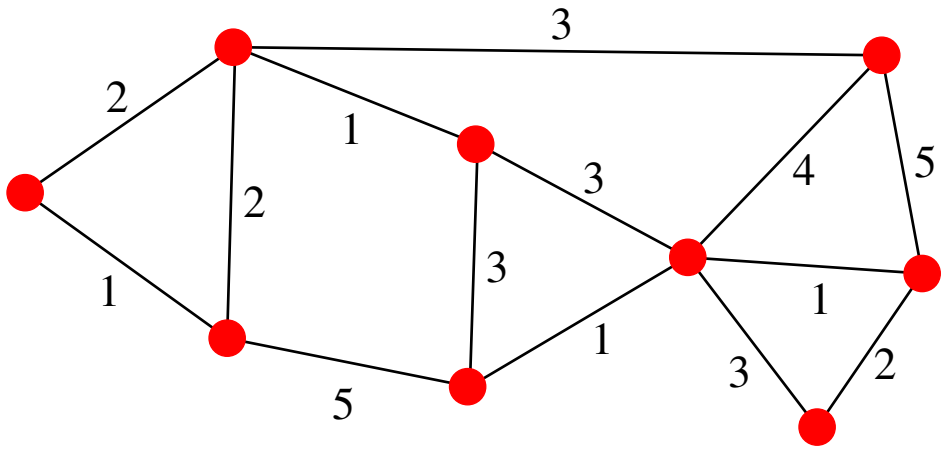
Méthode

- Initialiser T avec
  $\begin{cases} \text{sommets : tous les sommets de } G \\ \text{arêtes : aucune} \end{cases}$
- Traiter les arêtes de G l'une après l'autre par poids croissant :
  - Si une arête permet de connecter deux composantes connexes de T,
    - alors l'ajouter à T
    - sinon ne rien faire
  - Passer à l'arête suivante
- S'arrêter dès que T est connexe
- Retourner T

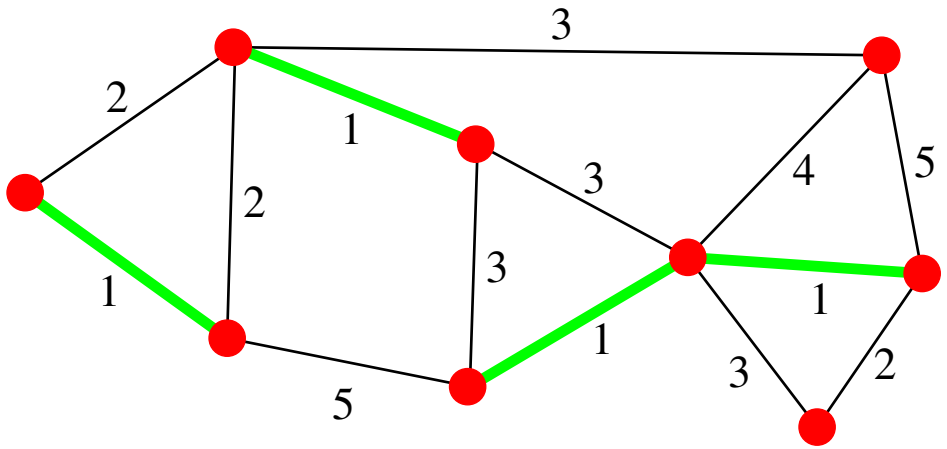
Exemple



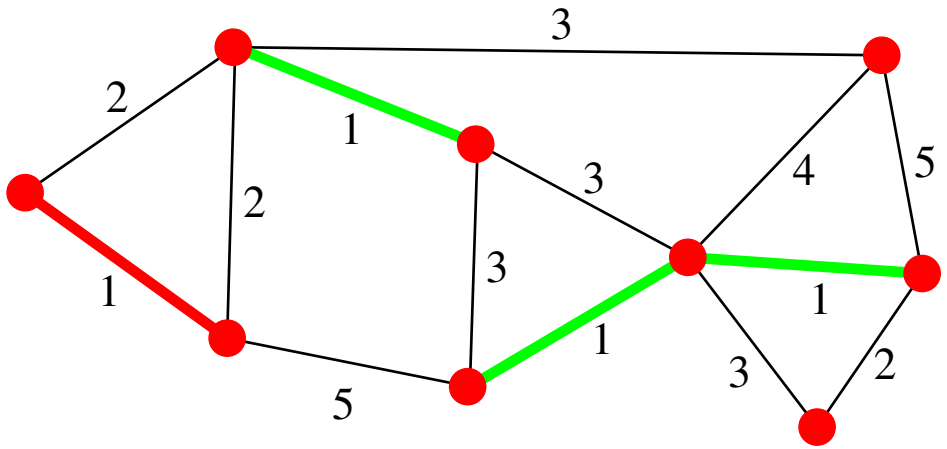
Exemple



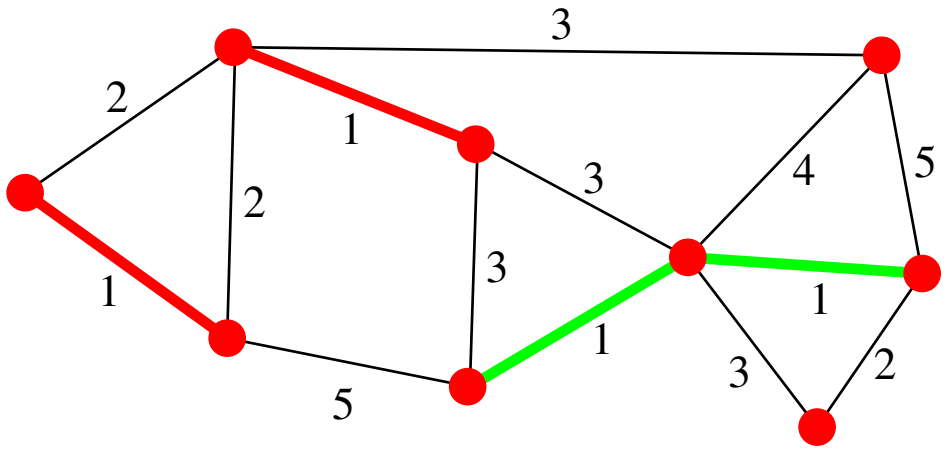
Exemple



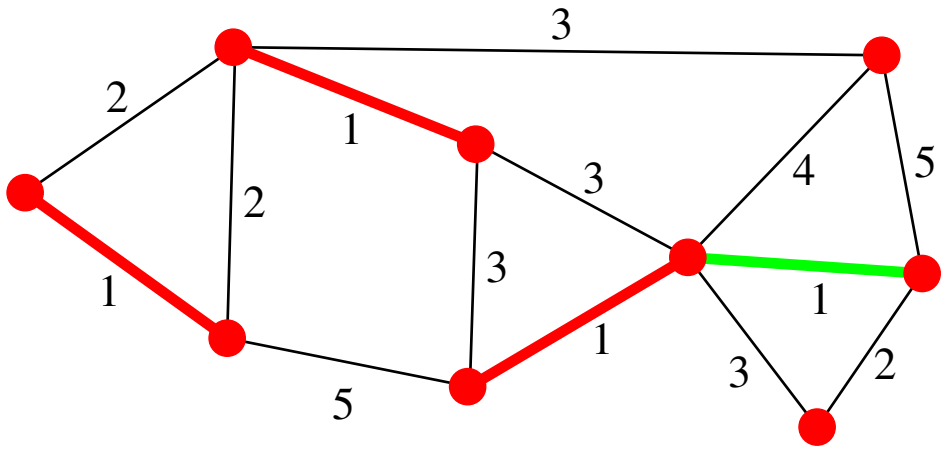
Exemple



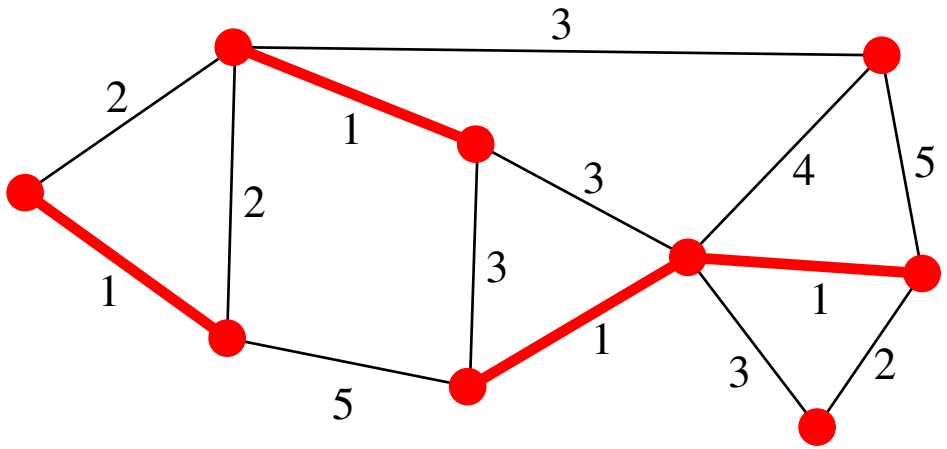
Exemple



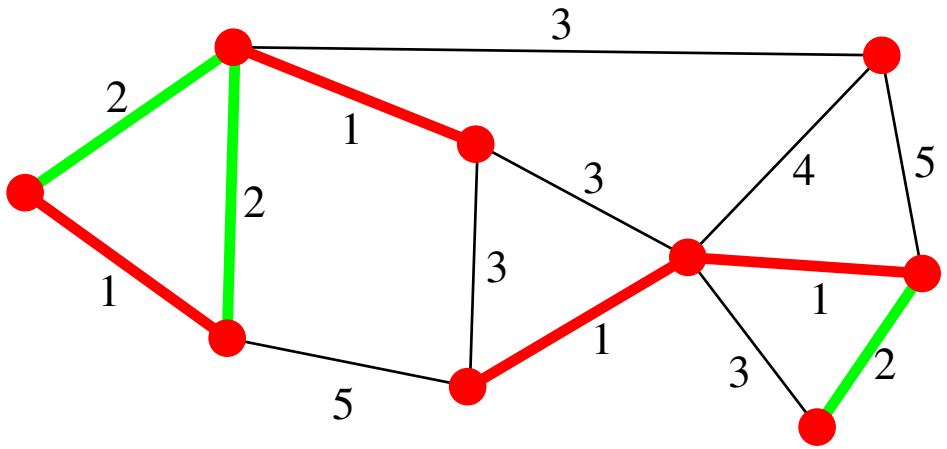
Exemple



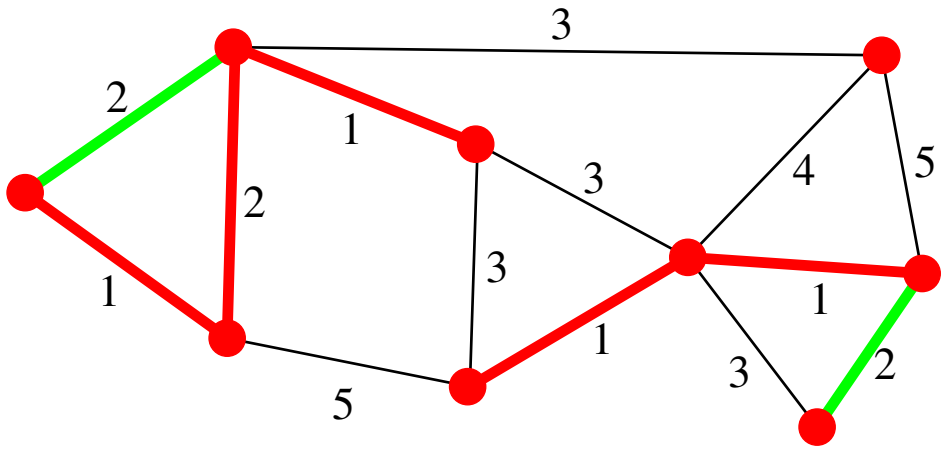
Exemple



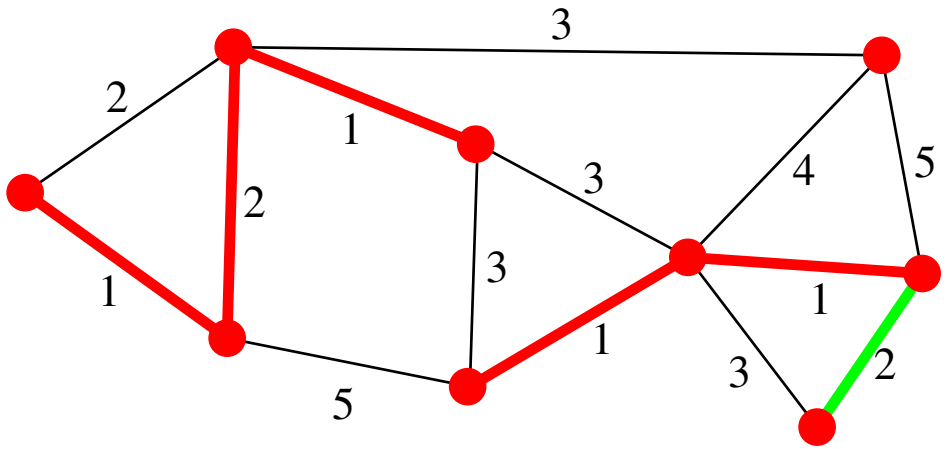
Exemple



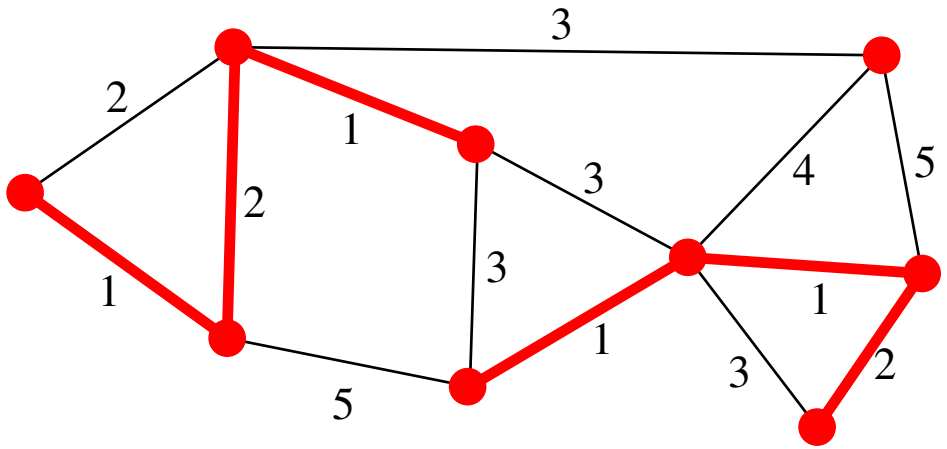
Exemple



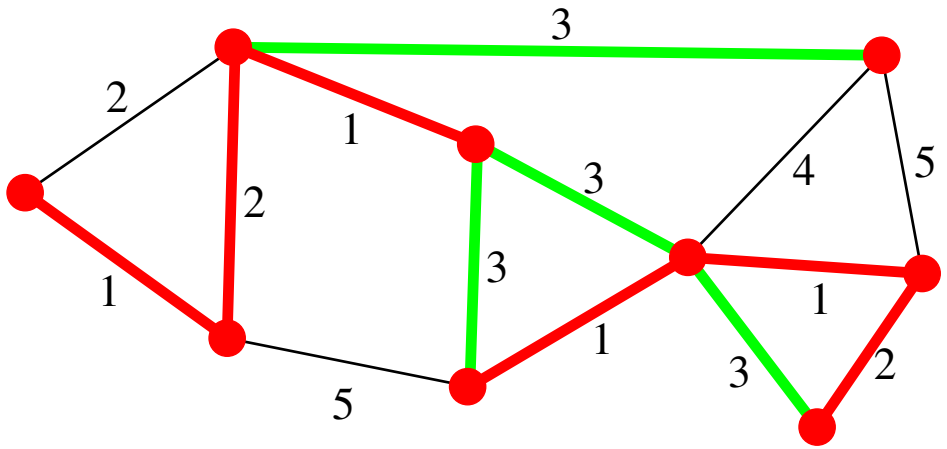
Exemple



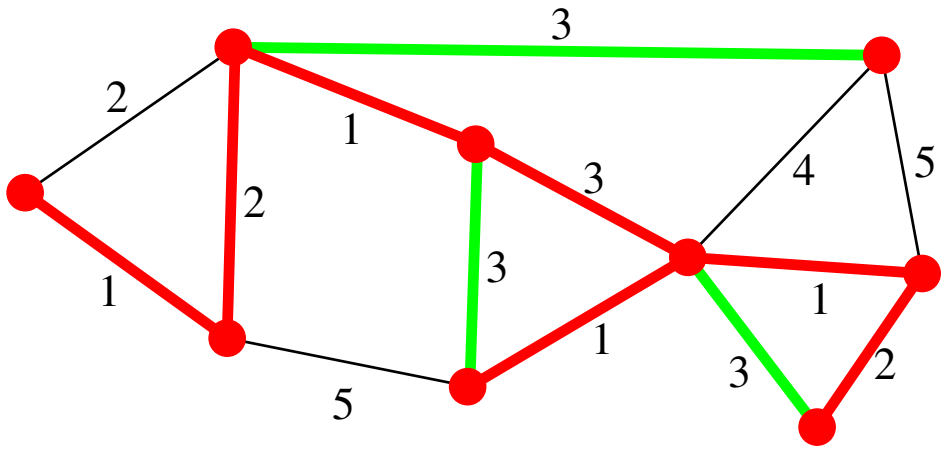
Exemple



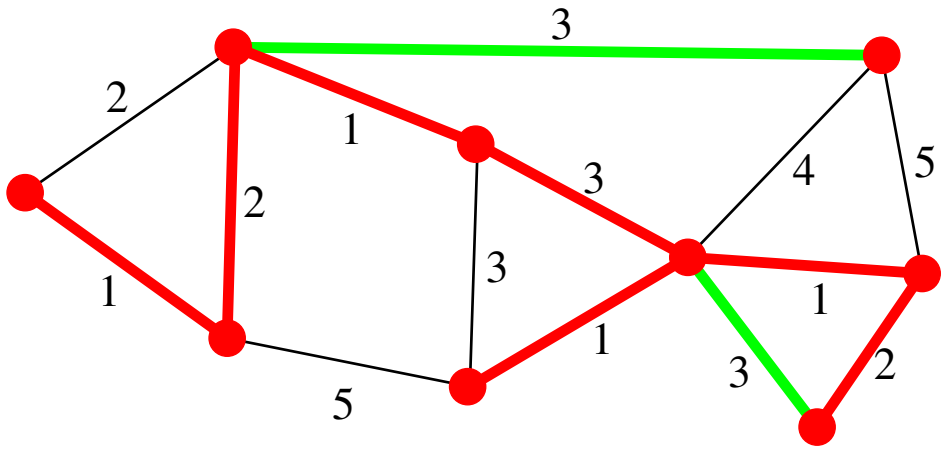
Exemple



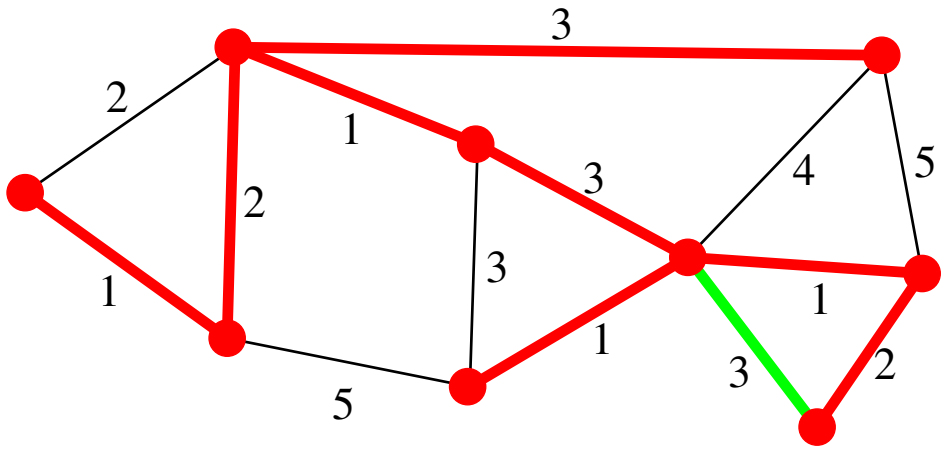
Exemple



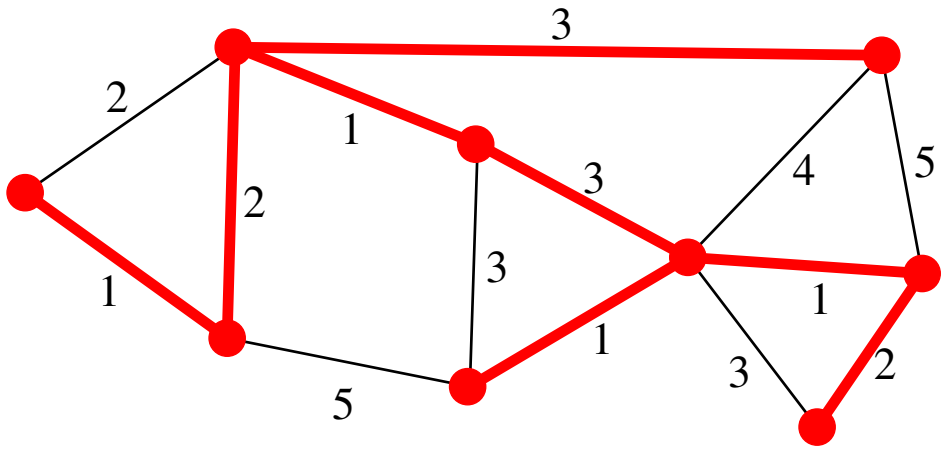
Exemple



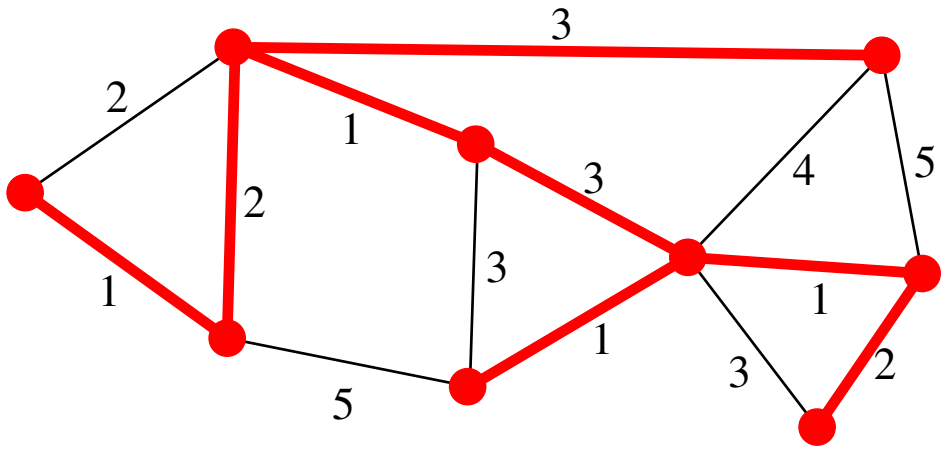
Exemple



Exemple



Exemple



$1 + 1 + 1 + 1 + 2 + 2 + 3 + 3 = 14$

On obtient un arbre différent, mais de même poids.

Rappel de l'algorithme de Kruskal

- Initialiser T avec
  $\begin{cases} \text{sommets : tous les sommets de } G \\ \text{arêtes : aucune} \end{cases}$
- Traiter les arêtes de G l'une après l'autre par poids croissant :
  - Si une arête permet de connecter deux composantes connexes de T,
    - alors l'ajouter à T
    - sinon ne rien faire
  - Passer à l'arête suivante
- S'arrêter dès que T est connexe
- Retourner T

Pourquoi l'algorithme de Kruskal construit-il un arbre couvrant de poids minimal ?

Il faut une preuve !

Pourquoi l'algorithme de Kruskal construit-il un arbre couvrant de poids minimal ?

- Kruskal s'arrête toujours
- Kruskal construit un arbre couvrant
- l'arbre couvrant construit par Kruskal est de poids minimal

Pourquoi l'algorithme de Kruskal construit-il un arbre couvrant de poids minimal ?

- Kruskal s'arrête toujours
- Kruskal construit un arbre couvrant
- l'arbre couvrant construit par Kruskal est de poids minimal

T finit par être connexe car G est connexe, donc en parcourant (au pire) toutes les arêtes, on arrive à connecter tous les sommets

Pourquoi l'algorithme de Kruskal construit-il un arbre couvrant de poids minimal ?

- Kruskal s'arrête toujours
- Kruskal construit un arbre couvrant
- l'arbre couvrant construit par Kruskal est de poids minimal

Propriétés du graphe T retourné par Kruskal :

- Sommets de T = Sommets de G
- Arêtes de $T \subseteq$ Arêtes de G
- T connexe
- T sans cycle car on n'ajoute une arête que pour connecter deux composantes connexes, ce qui ne peut pas créer de cycle, et initialement T n'a pas d'arêtes, donc pas de cycles.

Pourquoi l'algorithme de Kruskal construit-il un arbre couvrant de poids minimal ?

- Kruskal s'arrête toujours
- Kruskal construit un arbre couvrant
- l'arbre couvrant construit par Kruskal est de poids minimal

Supposons que ce ne soit pas le cas.

- Soit T' un arbre couvrant de poids minimal de G, qui différe le moins possible de T.
- Soit e la première arête de T qui n'est pas une arête de T'.
- On ajoute e à T', et on crée un cycle.
- Ce cycle ne contient pas que des arêtes de T (car T n'a pas de cycle).

Pourquoi l'algorithme de Kruskal construit-il un arbre couvrant de poids minimal ?

- Kruskal s'arrête toujours
- Kruskal construit un arbre couvrant
- l'arbre couvrant construit par Kruskal est de poids minimal

Soit e' une arête du cycle de $T' \cup \{e\}$ qui n'est pas une arête de T.

- On note $T'' = T' \cup \{e\} - \{e'\}$.
- T'' est un arbre couvrant de G.
- On a $Poids(T'') \geq Poids(T')$ (car T' est de poids minimal) donc $Poids(e) \geq Poids(e')$.

Pourquoi l'algorithme de Kruskal construit-il un arbre couvrant de poids minimal ?

- Kruskal s'arrête toujours
- Kruskal construit un arbre couvrant
- l'arbre couvrant construit par Kruskal est de poids minimal

Avant l'étape d'ajout de e, T , T' et T'' sont identiques.

- Or e est choisie de poids minimal en construisant T.
- Comme on n'a pas choisi e' à la place de e, c'est que en fait $Poids(e) = Poids(e')$.
- Donc on a aussi $Poids(T'') = Poids(T')$, càd T'' est de poids minimum.

Pourquoi l'algorithme de Kruskal construit-il un arbre couvrant de poids minimal ?

- Kruskal s'arrête toujours
- Kruskal construit un arbre couvrant
- l'arbre couvrant construit par Kruskal est de poids minimal

Mais T'' a plus d'arêtes communes avec T que T', ce qui est contraire à l'hypothèse de départ : "T' est un arbre couvrant de poids minimal de G, différant le moins possible de T".

Conclusion : T' n'existe pas et par conséquent T est un arbre couvrant de poids minimal de G.