Algorithme 2

Arbres couvrants de poids minimum

Louis-Claude Canon
louis-claude.canon@univ-fcomte.fr

Licence 2 Informatique – Semestre 4

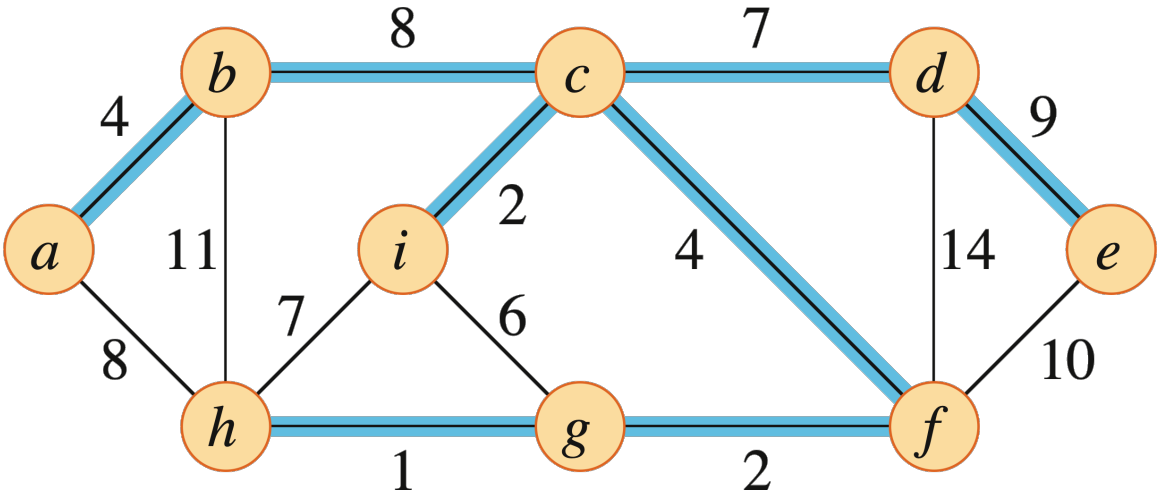
Plan

Plan

Définitions

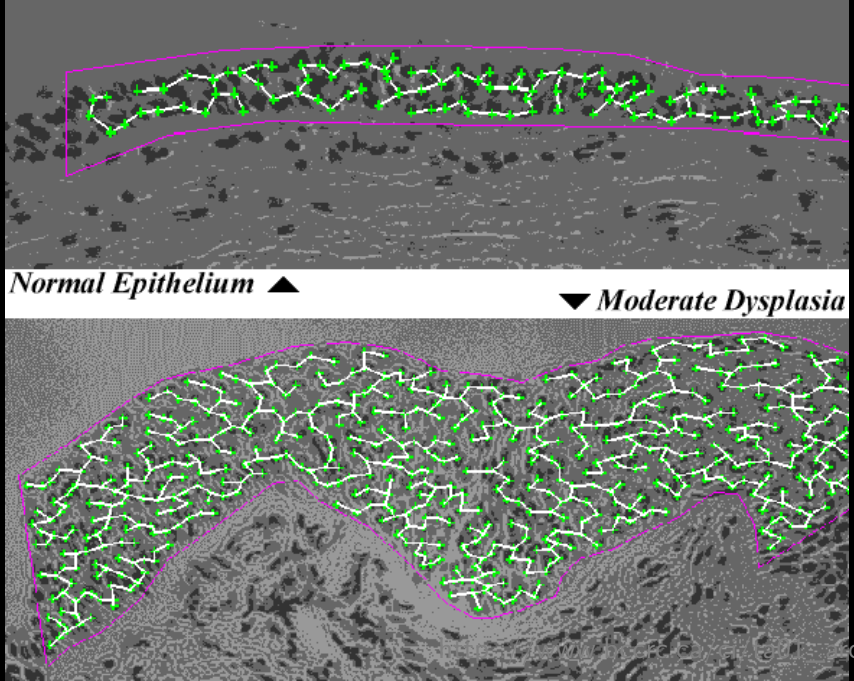
- Un *arbre couvrant* est un sous-ensemble d'arêtes d'un graphe non orienté $G = (V, E)$ qui forme un arbre (connecté et sans cycle) et qui couvre tous les sommets.
- Le *problème de l'arbre couvrant de poids minimum* consiste à déterminer l'arbre couvrant qui minimise la somme des poids $w(u, v)$ associée à chaque arête (u, v) conservée.
- On supposera que le graphe G est connexe (il existe un chemin entre chaque paire de sommets) et donc $O(V + E) = O(E)$.
- Les algorithmes pour ce problème s'appuient sur des structures de données avancées et qui servent à résoudre de nombreux autres problèmes.

Exemple



Contextes d'utilisation concrets

- Protocoles réseaux : STP (Spanning Tree Protocol) qui élimine les boucles dans les réseaux Ethernet.
- Réseaux de télécommunication.
- Réseaux de transport.
- Réseaux d'approvisionnement en eau.
- Réseaux électriques.
- Les arbres couvrants de poids minimum peuvent décrire l'arrangement des noyaux dans les cellules de la peau (recherche sur le cancer).



Utilisation en tant que sous-procédure

- Voyageur de commerce : algorithme de Christofides.
- Arbre de Steiner.
- Coupe minimum multi-terminal.
- Couplage parfait de coût minimum.

Plan

Algorithme glouton

Notion algorithmique

Un *algorithme glouton* est un algorithme qui fait le meilleur choix à un instant donné : il réalise, étape par étape, un choix optimum local afin d'obtenir un résultat optimum global.

Pour le problème de l'arbre couvrant de poids minimum, un algorithme glouton sélectionne les arêtes successivement en considérant la meilleure à chaque fois.

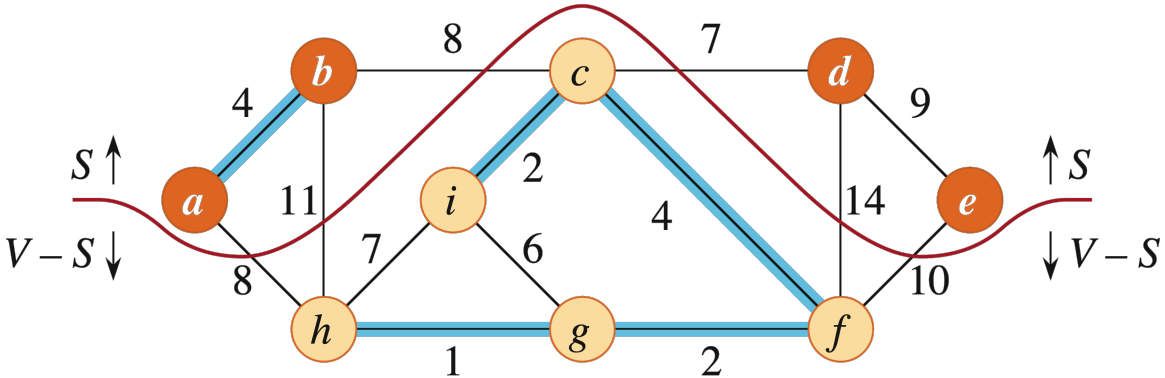
Pseudo-code de ACM-Générique

ACM-Générique(G, w)

$A \leftarrow \emptyset$
tant que A ne forme pas un arbre couvrant **alors**
  trouver une arête minimale (u, v) qui traverse une coupe
  $A \leftarrow A \cup \{(u, v)\}$
retourner A

- Une *coupe* $(S, V \setminus S)$ d'un graphe $G = (V, E)$ est une partition de V.
- Une arête (u, v) *traverse* la coupe $(S, V \setminus S)$ si $u \in S$ et $v \in V \setminus S$ (ou l'inverse).
- Une arête est *minimale* si elle est de poids minimum parmi toutes les arêtes qui traverse une coupe.

Coupe



Quelle est l'arête minimale qui traverse la coupe ?

Preuve du principe glouton

Théorème

Toute arête minimale (u, v) qui traverse une coupe $(S, V \setminus S)$ est dans un arbre couvrant de poids minimum.

Démonstration.

- Supposons que (u, v) ne soit dans aucun arbre couvrant de poids minimum.
- Ajouter (u, v) dans un arbre couvrant de poids minimum créerait un cycle.
- Il existe une arête (x, y) de cet arbre qui traverse la coupe $(S, V \setminus S)$.
- Remplacer (x, y) par (u, v) n'augmente pas le poids et reste donc optimal. Contradiction.

□

Preuve du principe glouton

Théorème

Toute arête minimale (u, v) qui traverse une coupe $(S, V \setminus S)$ est dans un arbre couvrant de poids minimum.

Démonstration.

- Supposons que (u, v) ne soit dans aucun arbre couvrant de poids minimum.
- Ajouter (u, v) dans un arbre couvrant de poids minimum créerait un cycle.
- Il existe une arête (x, y) de cet arbre qui traverse la coupe $(S, V \setminus S)$.
- Remplacer (x, y) par (u, v) n'augmente pas le poids et reste donc optimal. Contradiction.

□

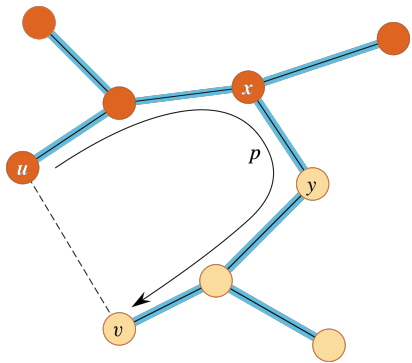
Preuve du principe glouton

Théorème

Toute arête minimale (u, v) qui traverse une coupe $(S, V \setminus S)$ est dans un arbre couvrant de poids minimum.

Démonstration.

- Supposons que (u, v) ne soit dans aucun arbre couvrant de poids minimum.
- Ajouter (u, v) dans un arbre couvrant de poids minimum créerait un cycle.
- Il existe une arête (x, y) de cet arbre qui traverse la coupe $(S, V \setminus S)$.
- Remplacer (x, y) par (u, v) n'augmente pas le poids et reste donc optimal. Contradiction.



□

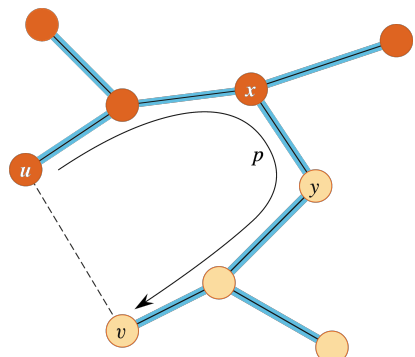
Preuve du principe glouton

Théorème

Toute arête minimale (u, v) qui traverse une coupe $(S, V \setminus S)$ est dans un arbre couvrant de poids minimum.

Démonstration.

- Supposons que (u, v) ne soit dans aucun arbre couvrant de poids minimum.
- Ajouter (u, v) dans un arbre couvrant de poids minimum créerait un cycle.
- Il existe une arête (x, y) de cet arbre qui traverse la coupe $(S, V \setminus S)$.
- Remplacer (x, y) par (u, v) n'augmente pas le poids et reste donc optimal. Contradiction.



□

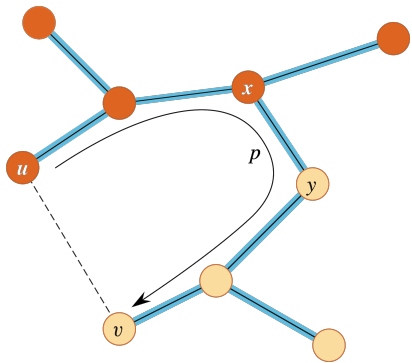
Preuve du principe glouton

Théorème

Toute arête minimale (u, v) qui traverse une coupe $(S, V \setminus S)$ est dans un arbre couvrant de poids minimum.

Démonstration.

- Supposons que (u, v) ne soit dans aucun arbre couvrant de poids minimum.
- Ajouter (u, v) dans un arbre couvrant de poids minimum créerait un cycle.
- Il existe une arête (x, y) de cet arbre qui traverse la coupe $(S, V \setminus S)$.
- Remplacer (x, y) par (u, v) n'augmente pas le poids et reste donc optimal. Contradiction.



□

Preuve de validité

Invariant de boucle Au début de chaque itération, A est un sous-ensemble d'un arbre couvrant de poids minimum.

Initialisation Avant la première itération, A est vide et satisfait donc l'invariant.

Conservation L'invariant est conservé à chaque itération car l'arête ajoutée est minimale pour une coupe.

Terminaison Toutes les arêtes ajoutées étant dans un arbre couvrant de poids minimal, A est forcément un arbre couvrant de poids minimal.

Preuve de validité

Invariant de boucle Au début de chaque itération, A est un sous-ensemble d'un arbre couvrant de poids minimum.

Initialisation Avant la première itération, A est vide et satisfait donc l'invariant.

Conservation L'invariant est conservé à chaque itération car l'arête ajoutée est minimale pour une coupe.

Terminaison Toutes les arêtes ajoutées étant dans un arbre couvrant de poids minimal, A est forcément un arbre couvrant de poids minimal.

Preuve de validité

Invariant de boucle Au début de chaque itération, A est un sous-ensemble d'un arbre couvrant de poids minimum.

Initialisation Avant la première itération, A est vide et satisfait donc l'invariant.

Conservation L'invariant est conservé à chaque itération car l'arête ajoutée est minimale pour une coupe.

Terminaison Toutes les arêtes ajoutées étant dans un arbre couvrant de poids minimal, A est forcément un arbre couvrant de poids minimal.

Preuve de validité

Invariant de boucle Au début de chaque itération, A est un sous-ensemble d'un arbre couvrant de poids minimum.

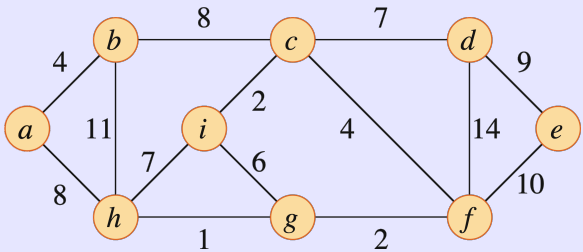
Initialisation Avant la première itération, A est vide et satisfait donc l'invariant.

Conservation L'invariant est conservé à chaque itération car l'arête ajoutée est minimale pour une coupe.

Terminaison Toutes les arêtes ajoutées étant dans un arbre couvrant de poids minimal, A est forcément un arbre couvrant de poids minimal.

Question

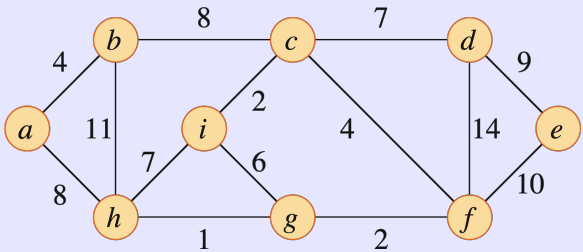
Quelle arête n'est minimale pour aucune coupe dans le graphe suivant ?



1. l'arête (i, g) de poids 6
2. l'arête (c, d) de poids 7
3. l'arête (b, c) de poids 8
4. l'arête (d, e) de poids 9

Question

Quelle arête n'est minimale pour aucune coupe dans le graphe suivant ?



1. l'arête (i, g) de poids 6 ✓
2. l'arête (c, d) de poids 7 $S = \{d\}$
3. l'arête (b, c) de poids 8 $S = \{a, b\}$
4. l'arête (d, e) de poids 9 $S = \{e\}$

Plan

Explication de ACM-Kruskal

- L'algorithme commence en considérant chaque sommet comme un sous-arbre et ajoute ensuite des arêtes pour fusionner/connecter ces sous-arbres.
- Les arêtes sont parcourues par ordre croissant de poids : on ajoute une arête (u, v) pour connecter le sous-arbre contenant u à celui contenant v, sauf si cela crée un cycle.
- Pour détecter un cycle, il faut déterminer si u et v sont dans 2 sous-arbres distincts.
- Il faut une structure de données pour ensembles disjoints.

Forêt d'ensembles disjoints / *union-find*

Notion algorithmique

- n éléments sont partitionnés en sous-ensembles disjoints : des sommets $V = V_1 \cup V_2 \cup \ldots \cup V_k$ où V_j est un sous-arbre par exemple.
- On cherche à déterminer si 2 éléments appartiennent au même ensemble : les sommets u et v par exemple.

Opération	description
CRÉER-ENSEMBLE(v)	indique que v est le représentant d'un singleton
UNION(u, v)	fusionne deux ensembles représentés par u et v
TROUVER-ENSEMBLE(v)	trouve le représentant de l'ensemble contenant v

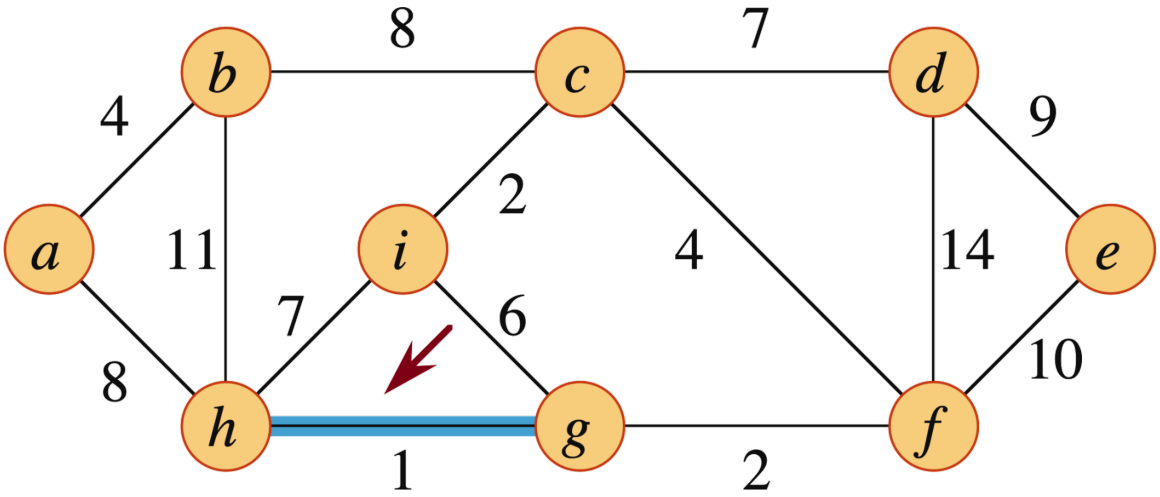
La coût amorti pour une requête `Trouver-Ensemble` est $O(\alpha(n))$ où $\alpha(n)$ est l'inverse de la fonction d'Ackermann ($\alpha(n) \leq 4$ dans l'univers).

Pseudo-code de ACM-Kruskal

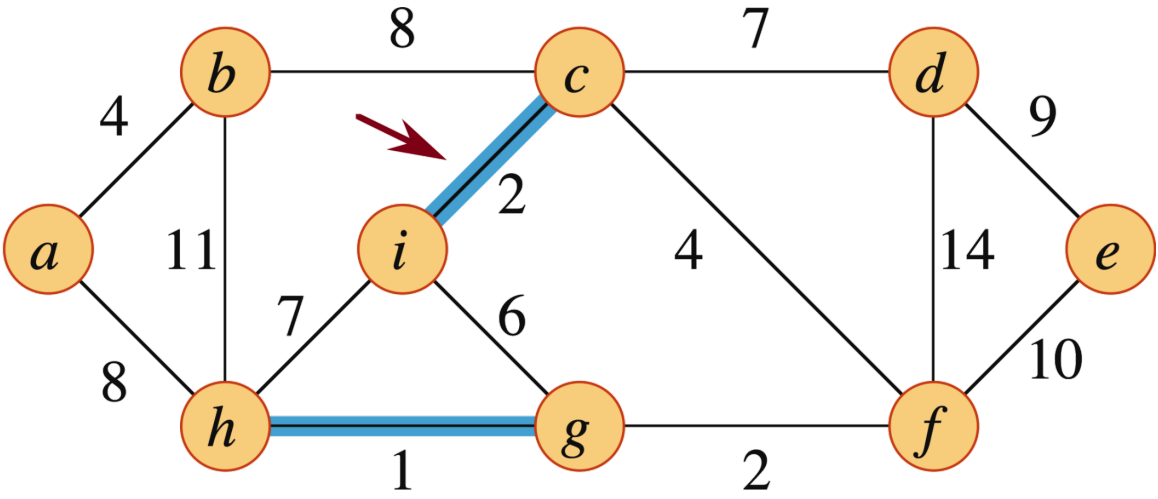
ACM-Kruskal(G, w)

$A \leftarrow \emptyset$
pour chaque sommet $v \in G.V$ **faire**
  Créer-Ensemble(v)
créer une liste unique d'arêtes de $G.E$
trier la liste d'arêtes par ordre croissant de poids w
pour chaque arête (u, v) pris dans la liste triée **faire**
  si Trouver-Ensemble$(u) \neq$ Trouver-Ensemble(v) **alors**
    $A \leftarrow A \cup \{(u, v)\}$
    Union(u, v)
retourner A

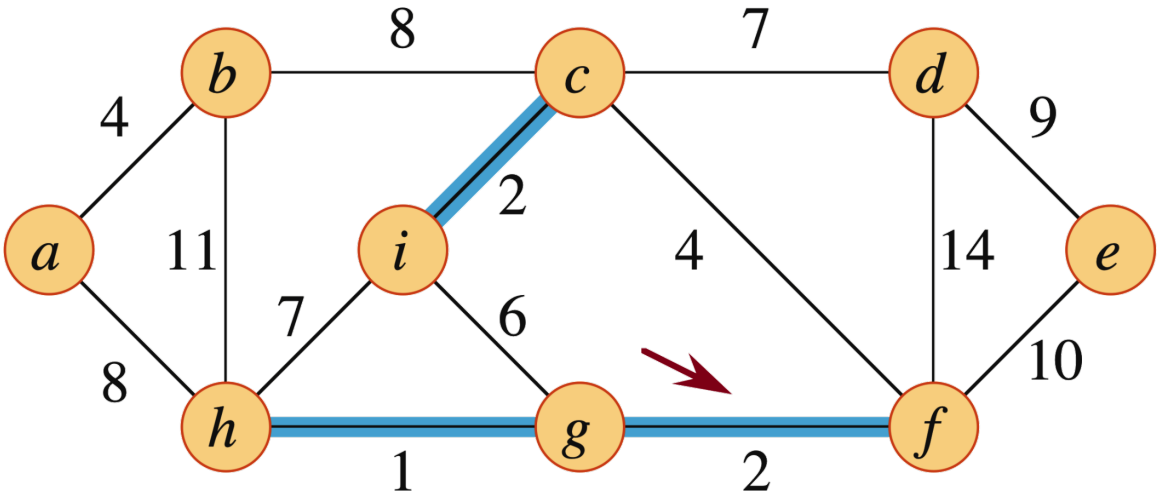
Exemple



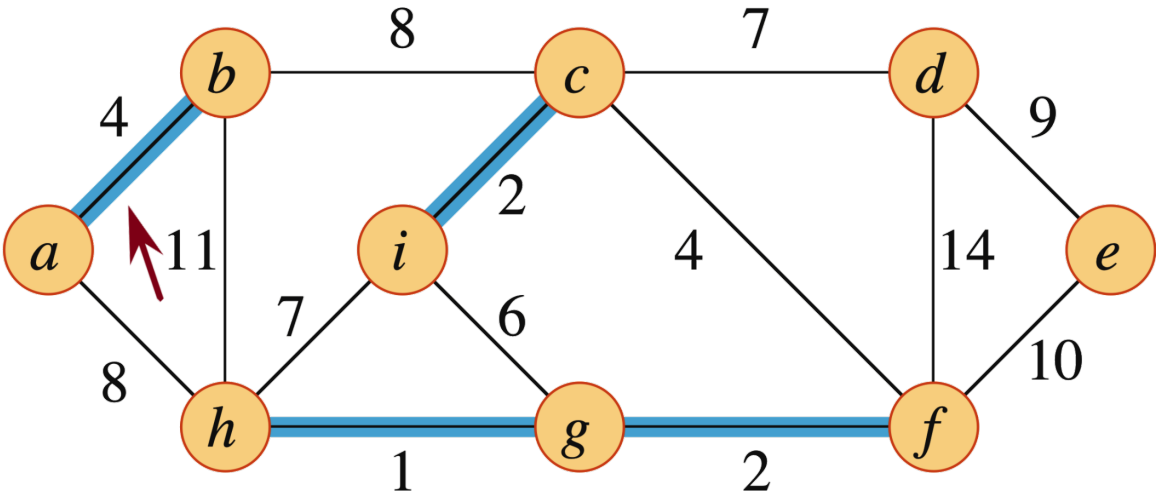
Exemple



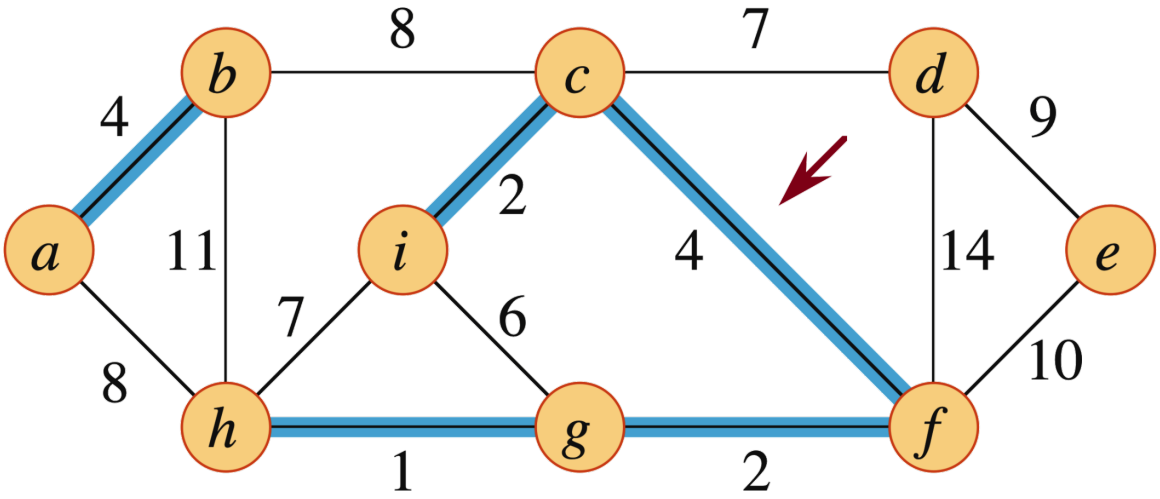
Exemple



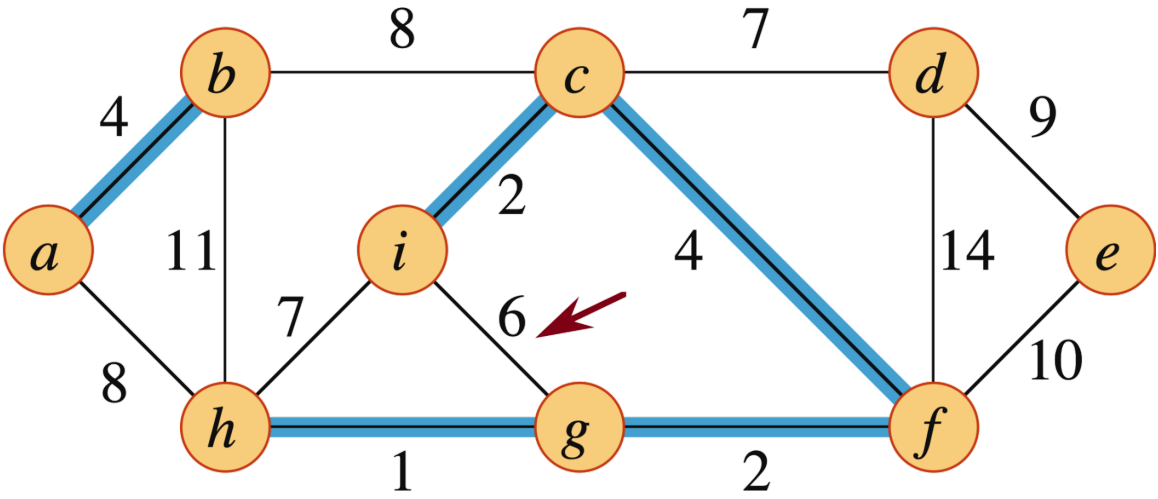
Exemple



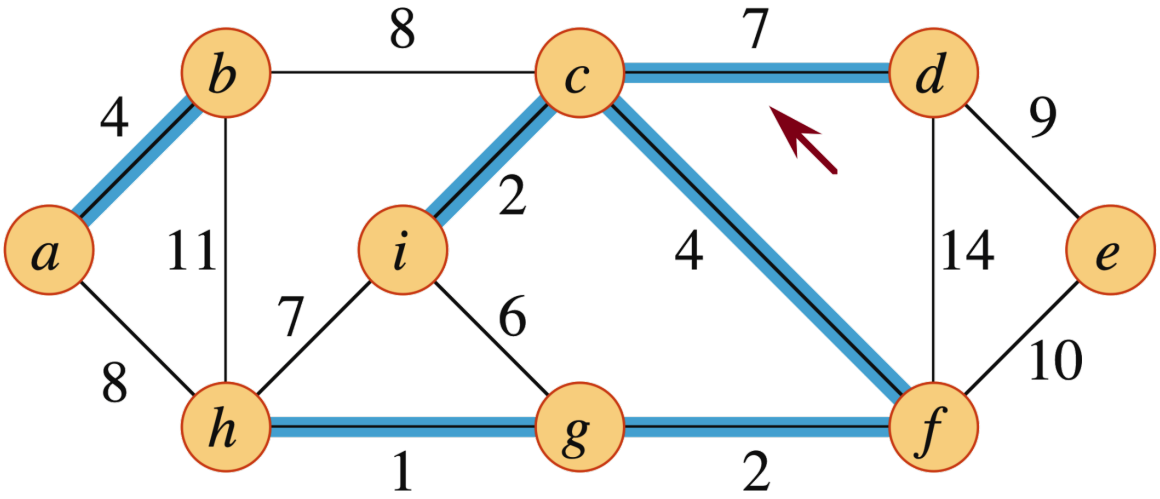
Exemple



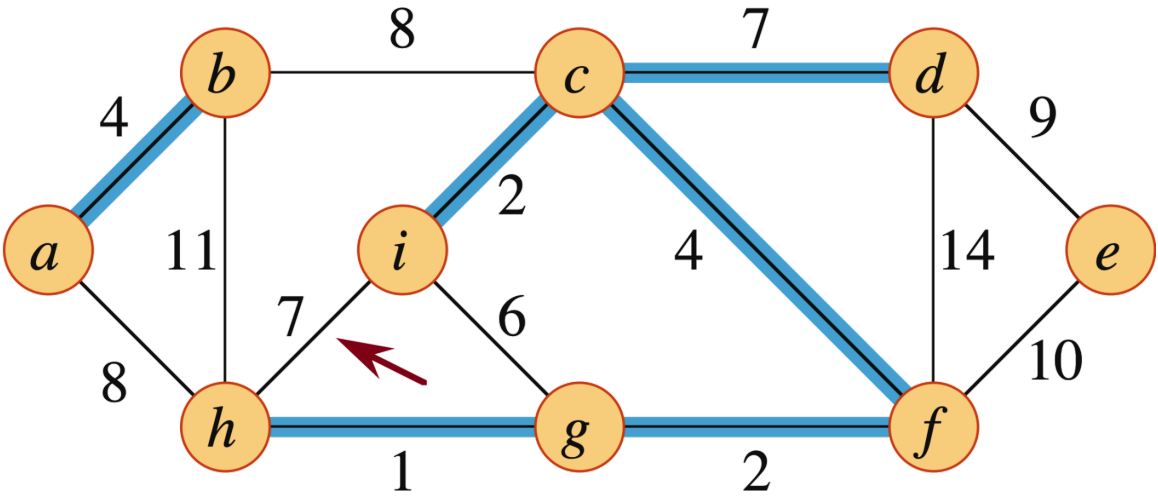
Exemple



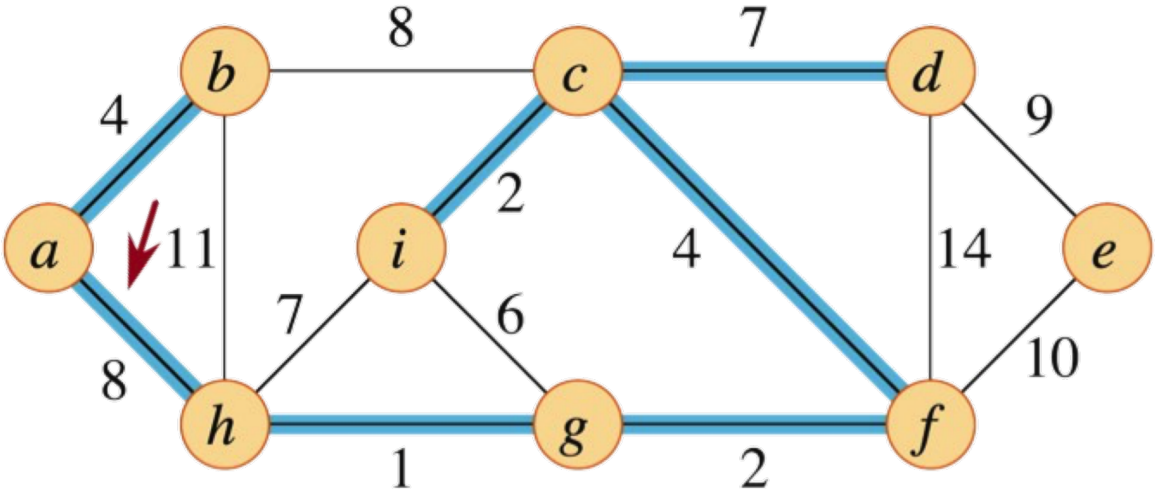
Exemple



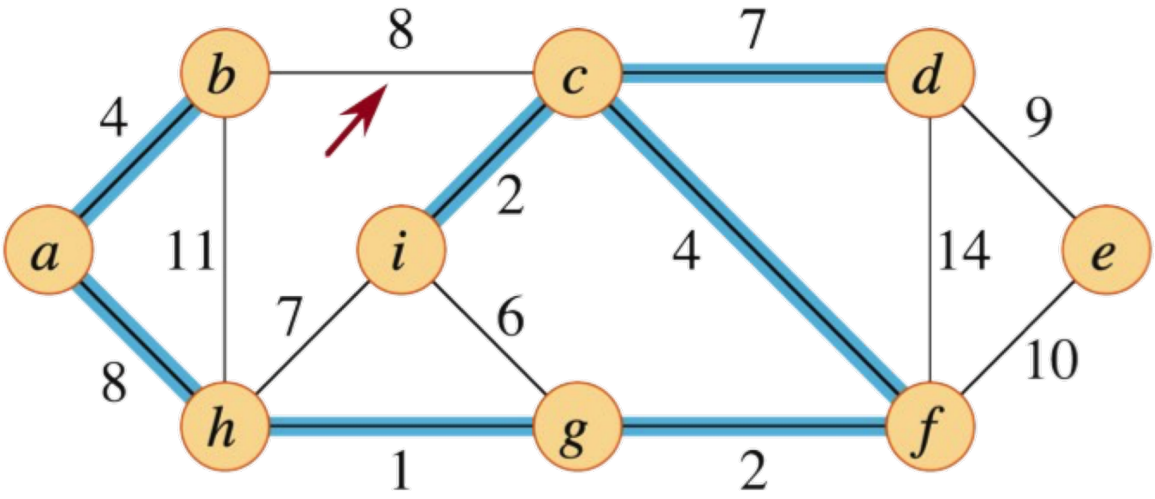
Exemple



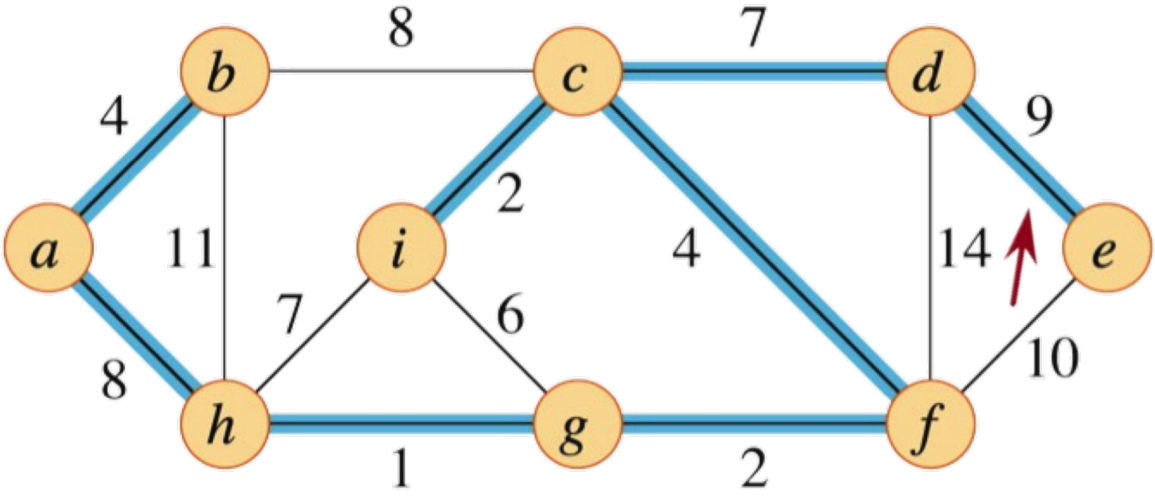
Exemple



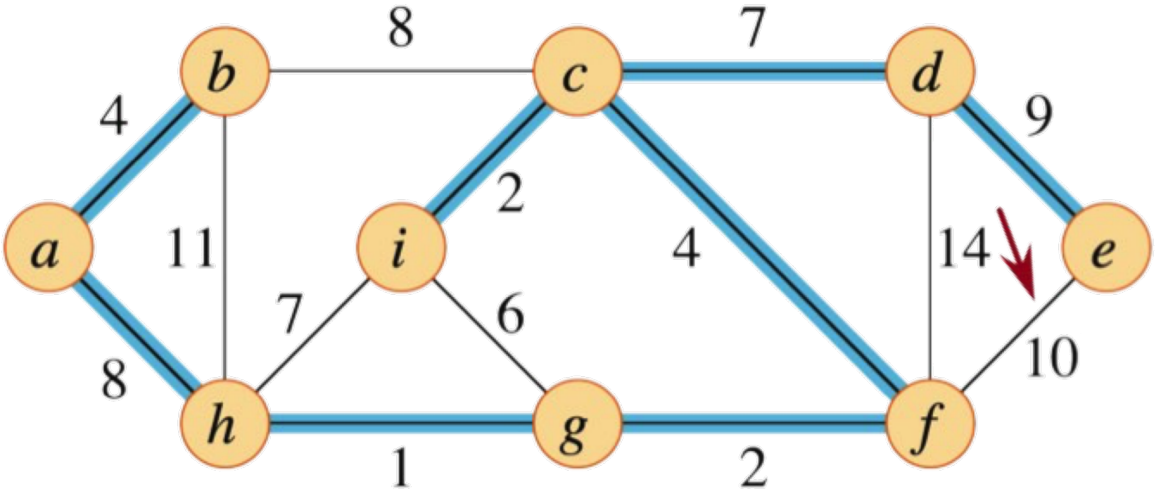
Exemple



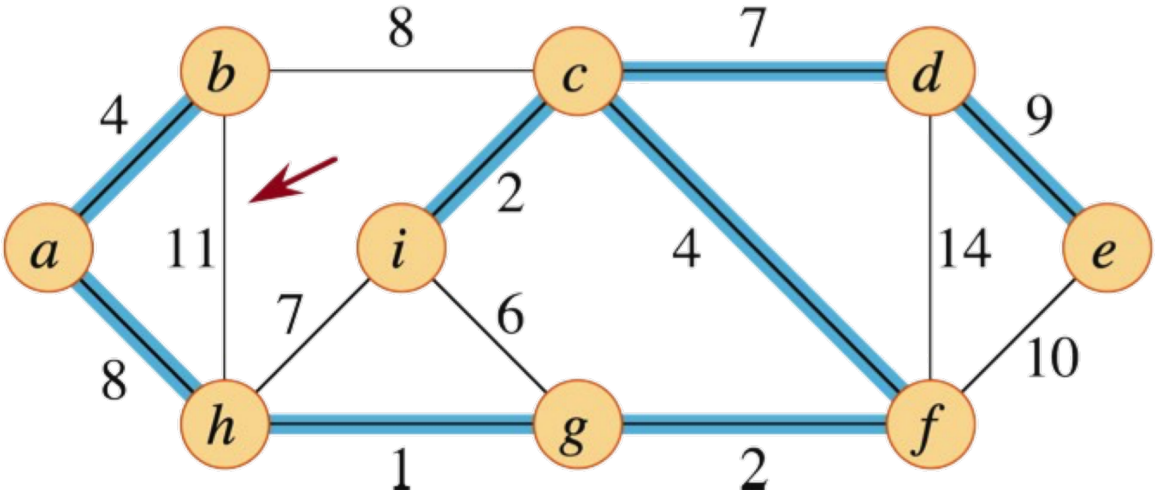
Exemple



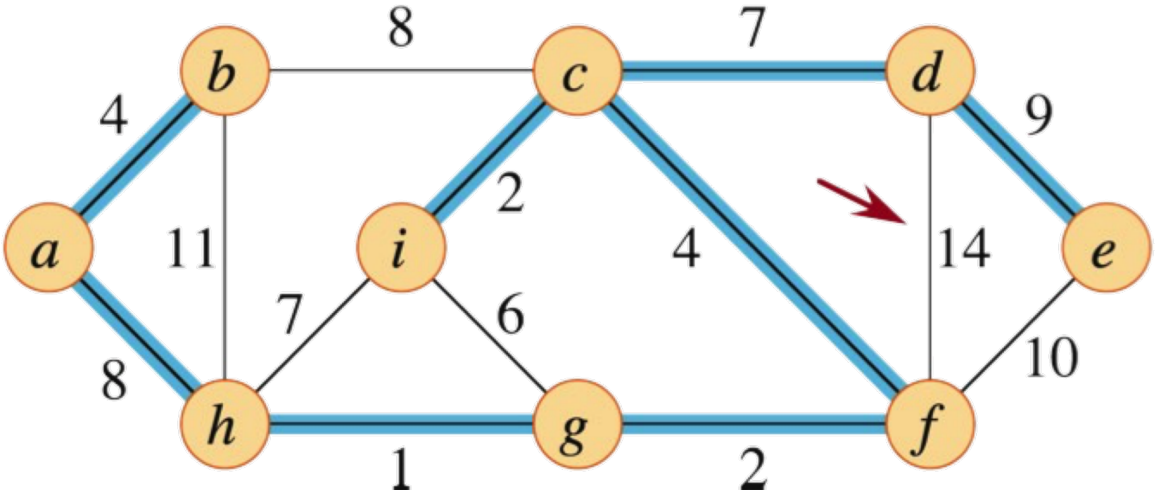
Exemple



Exemple



Exemple



Preuve de validité

On montre que l'algorithme est un cas de ACM-Générique :

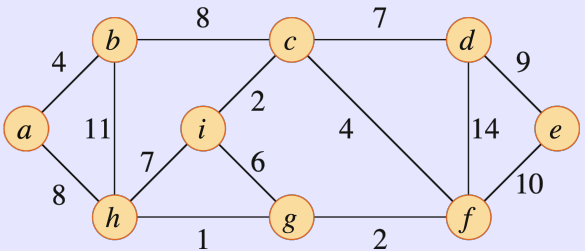
- Il faut prouver que chaque arête ajoutée (u, v) est minimale pour une coupe $(S, V \setminus S)$:
  - Comme on n'obtient pas de cycle en ajoutant (u, v), on peut séparer les sommets avec le sous-arbre contenant u d'un coté et celui contenant v de l'autre (et les autres sommets d'un coté ou de l'autre).
  - Il n'existe pas d'autre arête de poids inférieur qui traverse la coupe $(S, V \setminus S)$ car sinon, celle-ci aurait été ajoutée avant.
  - Chaque arête ajoutée est donc dans un arbre couvrant de poids minimum.
- Les arêtes retournées couvrent bien tous les sommets car sinon il existerait une coupe qu'une arête traverserait et qui aurait dû être ajoutée par ACM-Kruskal.

Analyse de complexité

- L'initialisation des ensembles prend un temps $O(V)$.
- Le tri des arêtes prend un temps $O(E \log E)$.
- Les opérations sur la structure de données pour ensembles disjoints prend un temps total $O(E\alpha(V))$.
- Comme G est connecté, $E \geq V - 1$.
- La complexité en temps est donc $O(E \log E)$.

Question

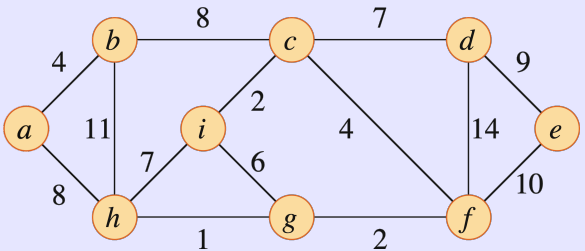
Combien y a t-il d'arêtes dans A juste après que ACM-KRUSKAL insère l'arête (c, d) pour le graphe suivant ? On supposera que les sommets et les arcs sont toujours considérés dans l'ordre alphabétique.



1. 4 arêtes
2. 5 arêtes
3. 6 arêtes
4. 7 arêtes

Question

Combien y a t-il d'arêtes dans A juste après que ACM-KRUSKAL insère l'arête (c, d) pour le graphe suivant ? On supposera que les sommets et les arcs sont toujours considérés dans l'ordre alphabétique.



1. 4 arêtes
2. 5 arêtes
3. 6 arêtes ✓((h, g), (c, i), (g, f), (a, b), (c, f), (c, d))
4. 7 arêtes

Plan

Explication de ACM-Prim

- L'algorithme construit itérativement un arbre en y connectant les sommets reliés par des arêtes de poids minimum.
- À chaque étape de la construction de l'arbre, il faut identifier quel sommet voisin y est connecté avec l'arête de poids minimum.
- On utilise une file de priorités contenant les sommets du graphe : pour un sommet donné, le poids correspond à celui de l'arête de plus petit poids qui y est connectée, ou l'infini si le sommet n'est pas un voisin.

Rappel sur les files de priorités

Notion algorithmique

- On associe une clé à chacun des n sommets : le plus petit poids d'une arête qui connecte un arbre à l'un de ses voisins par exemple.
- On souhaite extraire le sommet ayant la plus petite clé et réduire la clé associée à un élément dans la file.

Structure de données	Insérer	Extraire-Min	Diminuer-Clé
Tas	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(\log n)$
Tas de Fibonacci	$\Theta(1)$	$O(\log n)^{\text{am}}$	$\Theta(1)^{\text{am}}$

Pseudo-code de ACM-PRIM

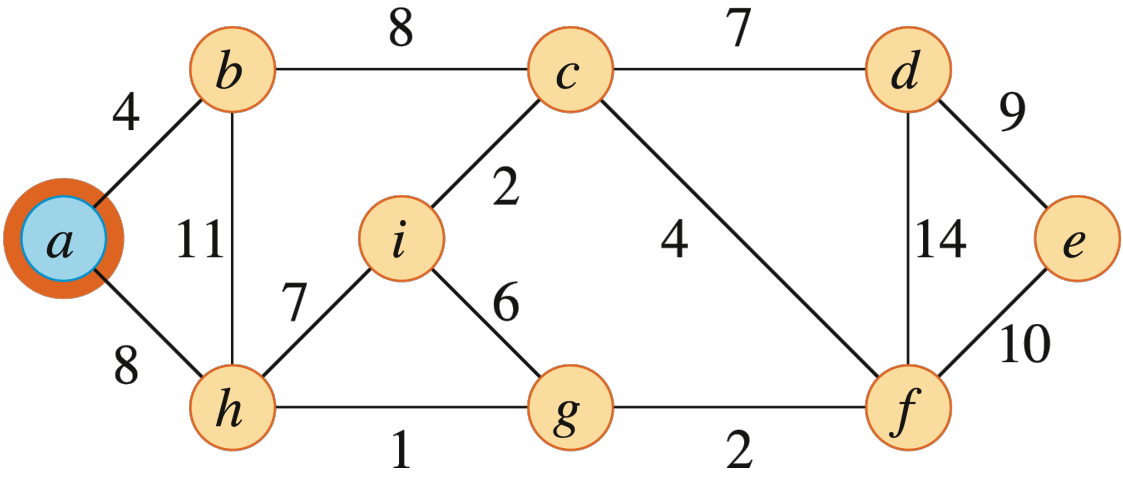
ACM-PRIM(G, w, r)

pour chaque sommet $v \in G.V$ **faire**
  $v.\text{clé} \leftarrow \infty$
  $v.\pi \leftarrow NIL$
$r.\text{clé} \leftarrow 0$
$Q \leftarrow \emptyset$
pour chaque sommet $v \in G.V$ **faire**
  INSÉRER(Q, v)
tant que $Q \neq \emptyset$ **faire**
  $u \leftarrow$ EXTRAIRE-MIN(Q)
  pour chaque sommet $v \in G.adj[u]$ **faire**
    si $v \in Q$ **et** $w(u, v) \leq v.\text{clé}$ **alors**
      $v.\pi \leftarrow u$
      $v.\text{clé} \leftarrow w(u, v)$
      DIMINUER-CLÉ$(Q, v, w(u, v))$

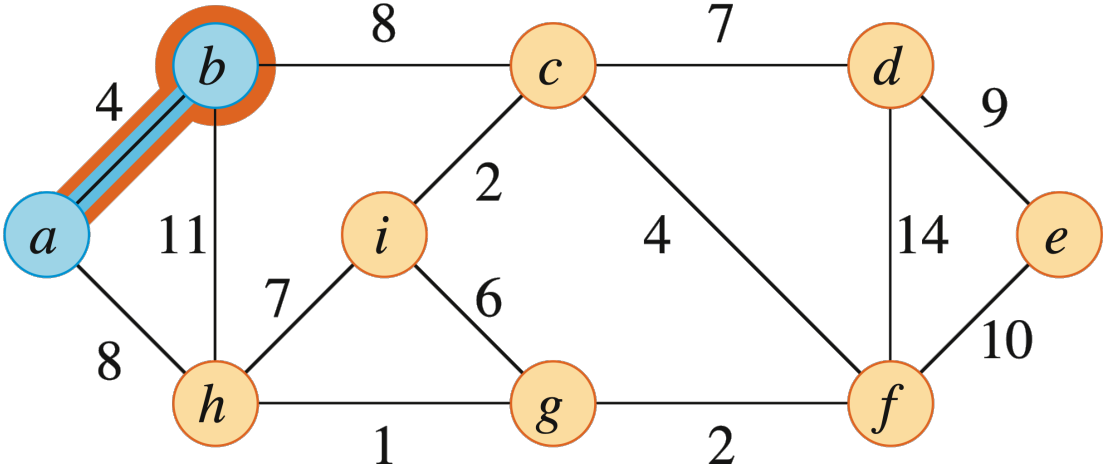
On construit un arbre en y connectant successivement une arête vers un nouveau sommet u :

- On initialise la distance de chaque sommet à l'arbre en construction à ∞ (sauf pour un sommet arbitraire r qui servira de base).
- À chaque itération, on ajoute le sommet voisin u pour lequel l'arête qui le connecte à l'arbre est de poids minimum.
- On modifie la file de priorités en mettant à jour la distance de l'arbre aux voisins du sommet ajouté u.

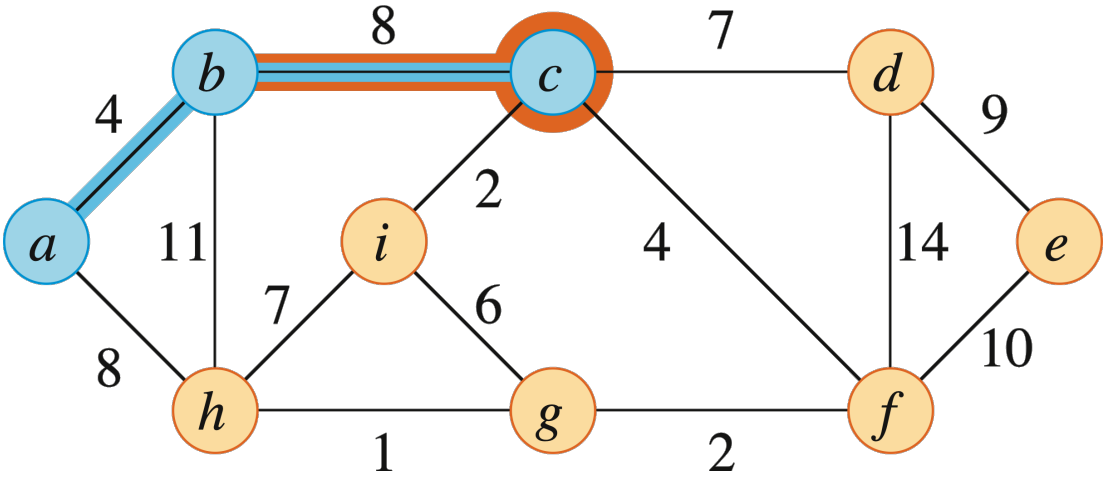
Exemple



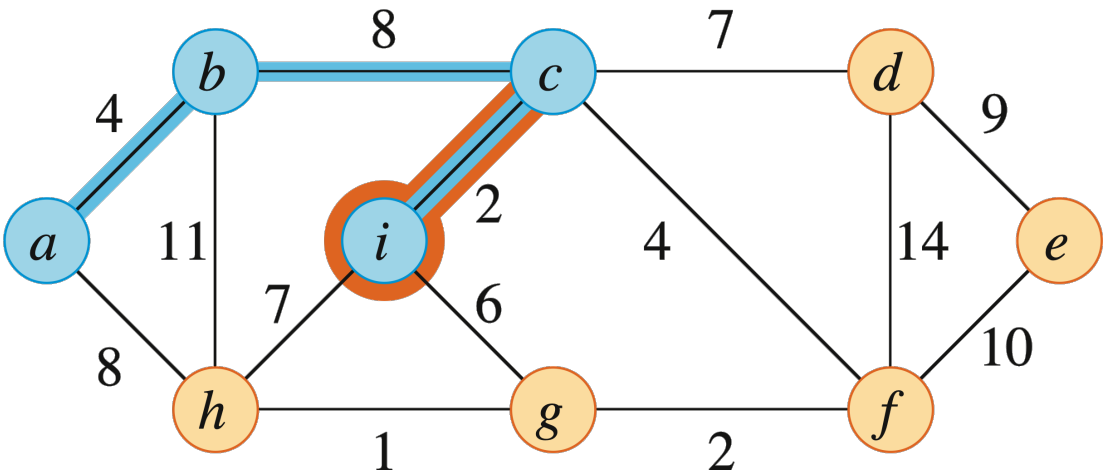
Exemple



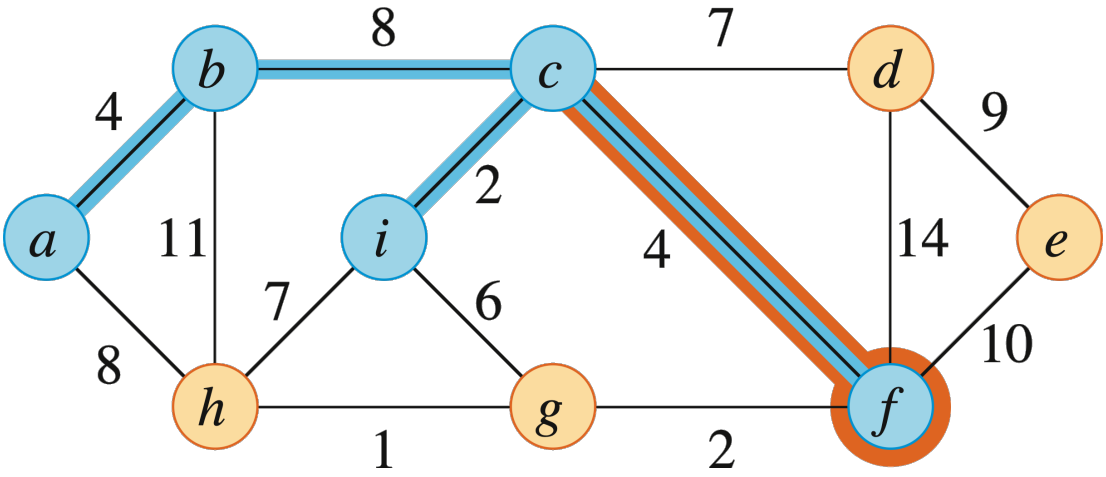
Exemple



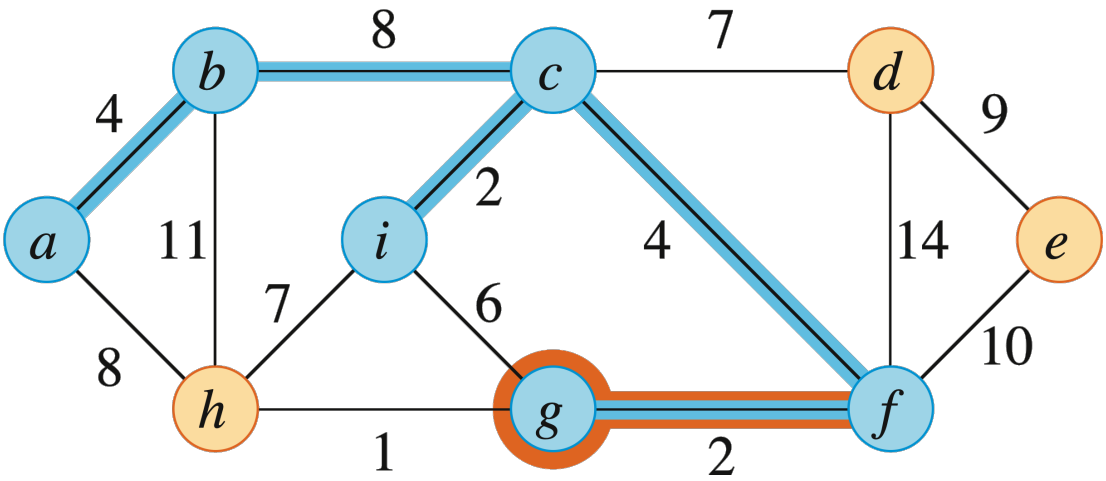
Exemple



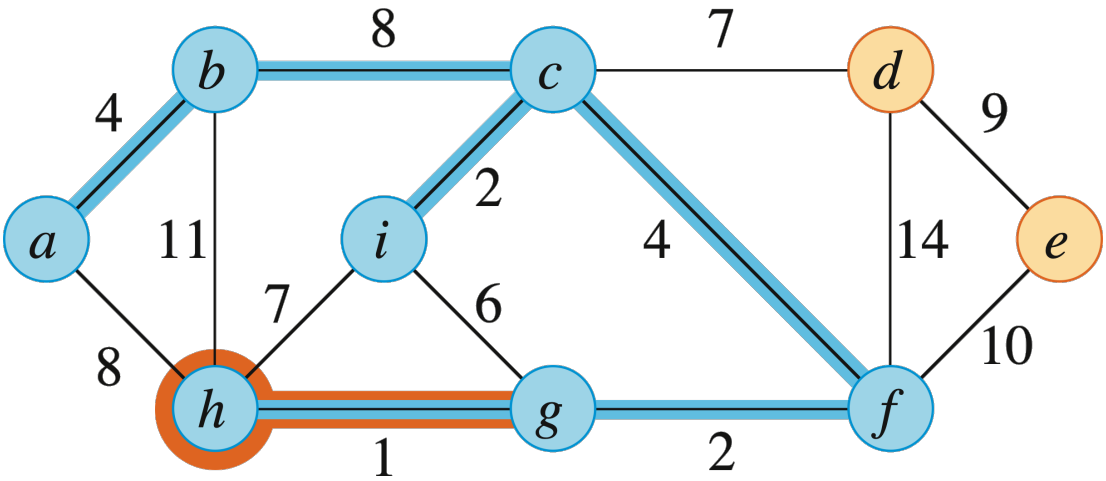
Exemple



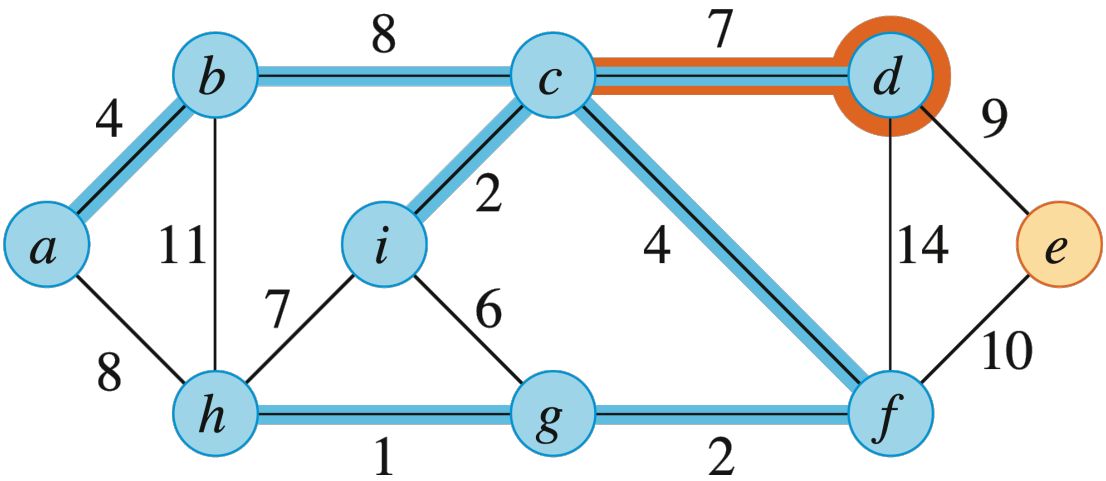
Exemple



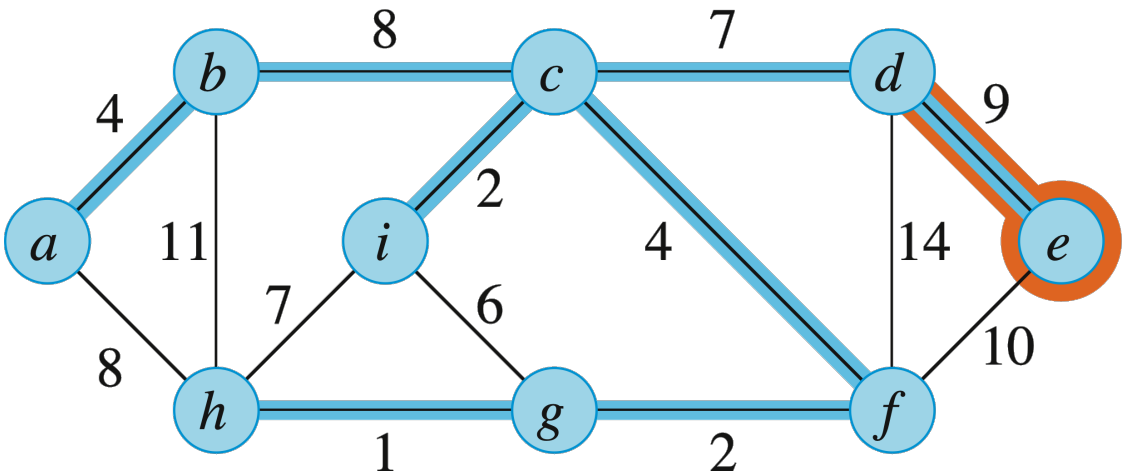
Exemple



Exemple



Exemple



Preuve de validité

On montre que l'algorithme est un cas de ACM-Générique :

- Il faut prouver que chaque arête ajoutée (u, v) est minimale pour une coupe $(S, V \setminus S)$. On considère $S = Q$ pour séparer l'arbre en construction du reste des sommets.
- Les arêtes $A = \{(v, v.\pi) : v \in V \setminus \{r\} \setminus Q\}$ couvrent bien tous les sommets car chaque sommet sauf r a une valeur pour le champ $v.\pi$.

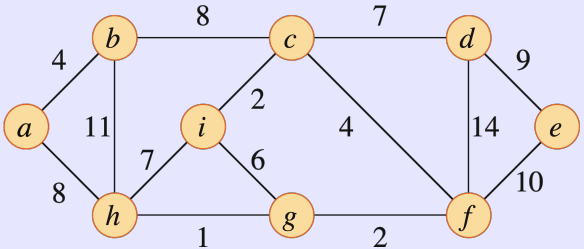
Pour une preuve plus formelle, on s'appuierait sur un invariant spécifiant les propriétés de Q par rapport à l'arbre en construction.

Analyse de complexité

- L'initialisation de la file de priorités avec un tas prend un temps $O(V \log V)$.
- La boucle **tant que** est exécutée $|V|$ fois et chaque appel à Extraire-Min prend un temps $O(\log V)$.
- La boucle **pour** totalise $|E|$ itérations car chaque arête est parcourue une fois.
- L'appartenance d'un sommet à la file peut se faire en $\Theta(1)$.
- L'appel à Diminuer-Clé prend un temps $O(\log V)$.
- On obtient donc une complexité en temps $O(E \log V)$ (ou $O(E + V \log V)$ avec un tas de Fibonacci).

Question

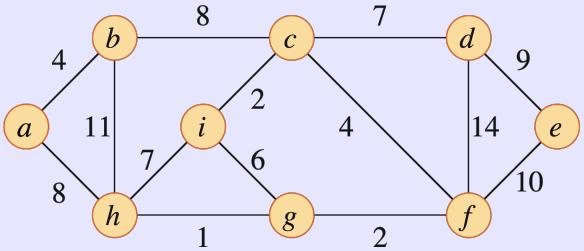
En commençant par le sommet $r = h$ avec ACM-PRIM pour le graphe suivant, combien de sommets sont accessibles par l'arbre couvrant lorsque le sommet a devient lui-même accessible ? On supposera que les sommets et les arcs sont toujours considérés dans l'ordre alphabétique.



1. 5 sommets
2. 6 sommets
3. 7 sommets
4. 8 sommets

Question

En commençant par le sommet $r = h$ avec ACM-PRIM pour le graphe suivant, combien de sommets sont accessibles par l'arbre couvrant lorsque le sommet a devient lui-même accessible ? On supposera que les sommets et les arcs sont toujours considérés dans l'ordre alphabétique.



1. 5 sommets
2. 6 sommets
3. 7 sommets
4. 8 sommets ✓(h, g, f, c, i, d, b, a)

Plan

Algorithme optimal

Curiosité

De meilleurs algorithmes existent pour ce problème (Karger, Klein et Tarjan, 1995 ; Chazelle 2000) : ils sont presque en $\Theta(E)$. En 2002, Pettie et Ramachandran proposent un algorithme optimal (sa complexité est la meilleure possible pour ce problème) mais sa complexité n'est pas connue.

Résumé

Contenu

- Un arbre couvrant de poids minimum est un sous-ensemble des arêtes d'un graphe qui forme un arbre.
- De nombreux algorithmes se basent sur un principe glouton : on ajoute successivement une arête minimale pour une coupe.
- L'algorithme de Kruskal traite les arêtes par ordre croissant de poids et repose sur une structure de données pour ensembles disjoints.
- L'algorithme de Prim construit itérativement un arbre et repose sur une file de priorités.

Résumé

Contenu

- Un arbre couvrant de poids minimum est un sous-ensemble des arêtes d'un graphe qui forme un arbre.
- De nombreux algorithmes se basent sur un principe glouton : on ajoute successivement une arête minimale pour une coupe.
- L'algorithme de Kruskal traite les arêtes par ordre croissant de poids et repose sur une structure de données pour ensembles disjoints.
- L'algorithme de Prim construit itérativement un arbre et repose sur une file de priorités.