ALGORITHMIQUE

SUR LES CHEMINS

Laboratoire Informatique Image Interaction (L3I)
Université de La Rochelle - Pôle Sciences et Technologie - Avenue Michel Crépeau - 17042 LA ROCHELLE CEDEX 1 France
Tél : +33 (0)5 46 45 82 62 – Fax : 05.46.45.82.42 – Site internet : http://l3i.univ-larochelle.fr/

PROGRAMME DU COURS

Algorithmes de graphes

- Calcul d'arbres couvrants minimum
- Calcul de plus courts chemins

Objectifs

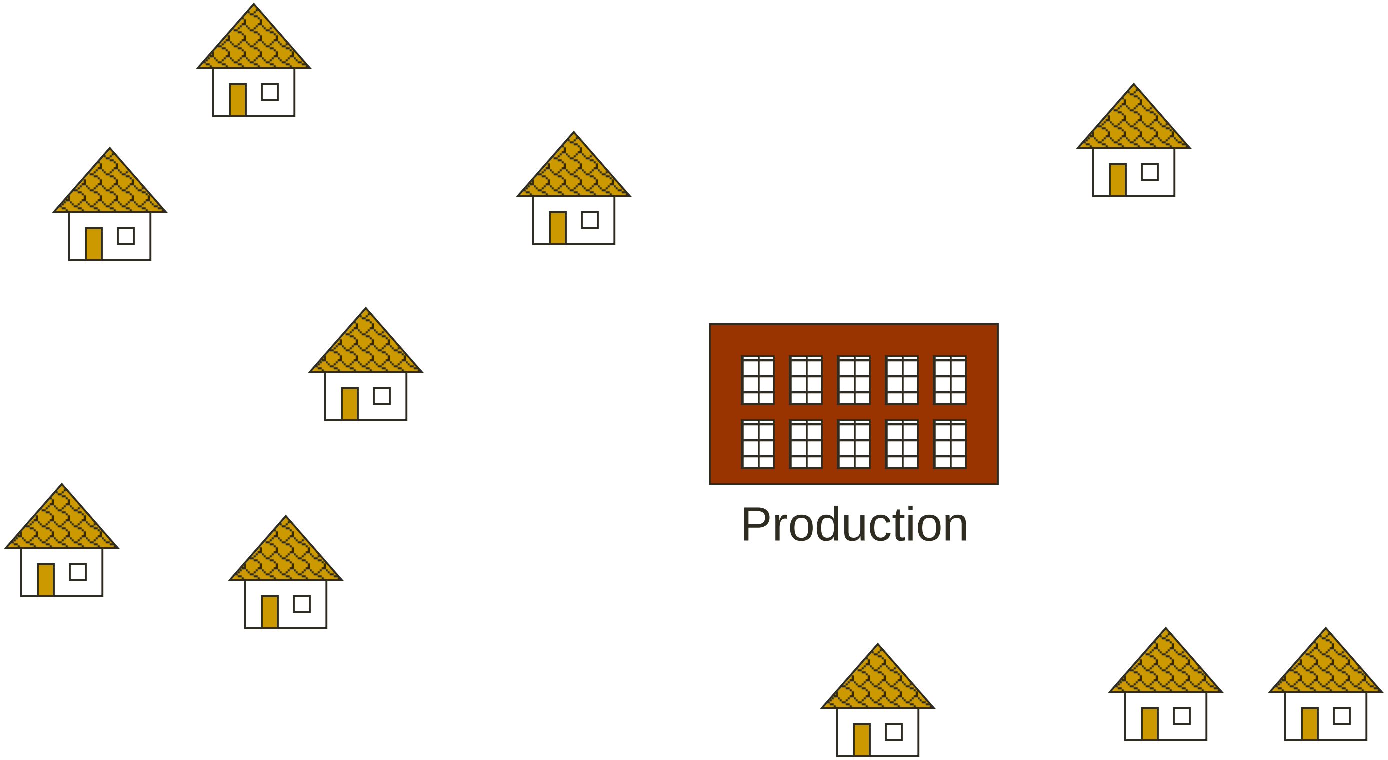
- Définition et compréhension des algorithmes
- « Preuve » des algorithmes
- Réflexion sur la complexité des algorithmes

Points à discuter, indiquées en violet dans le cours

ARBRE COUVRANT MINIMUM

PROBLÈME : CÂBLAGE

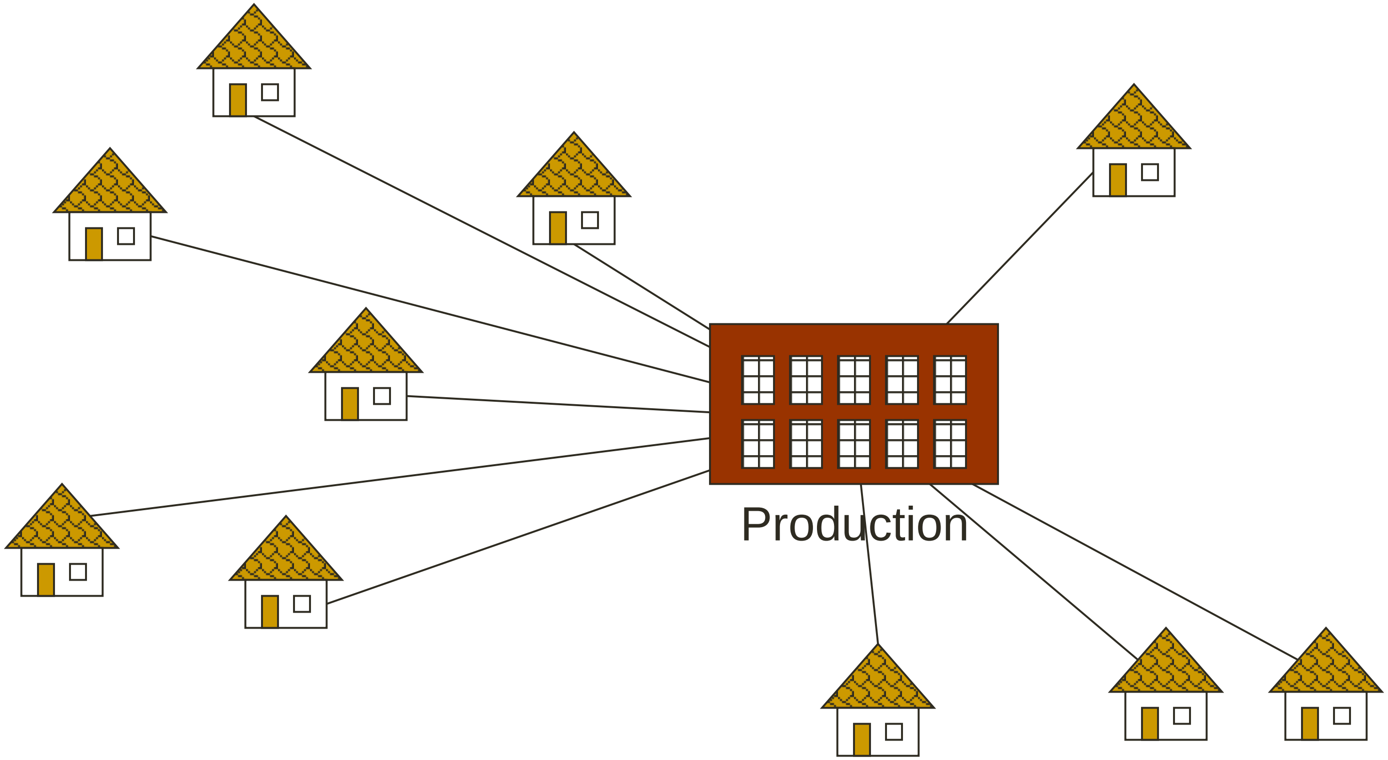
Objectif : raccorder le site de production à tous les sites de consommation (électricité, internet, etc.)



PROBLÈME : CÂBLAGE – APPROCHE NAÏVE

Cablage direct entre la source et les destinations

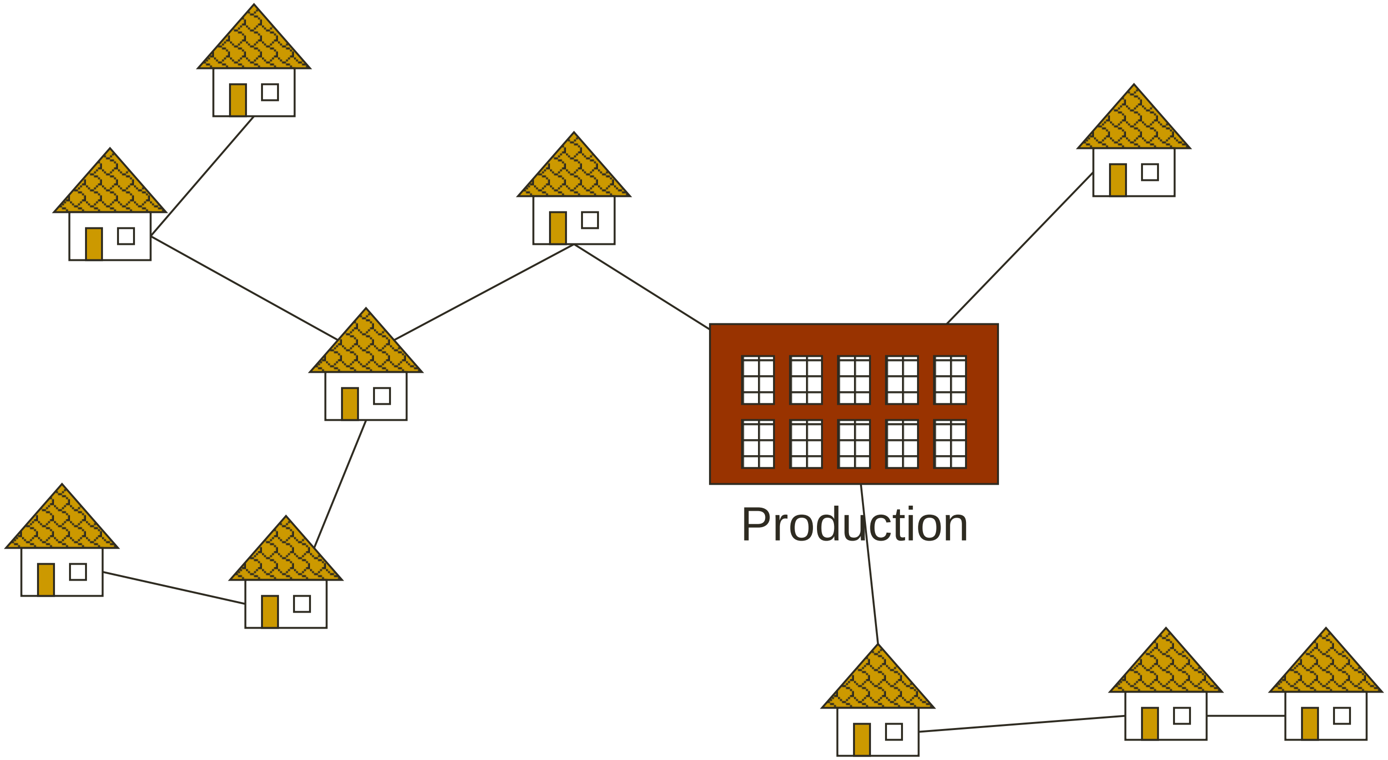
- Grande sécurité : chaque consommateur est indépendant
- Très couteux : somme des longueurs des cables



PROBLÈME : CÂBLAGE – APPROCHE MOINS NAÏVE

Cablage sous forme d'un arbre

- Peu couteux : potentiellement optimal en longueur de cables
- Risque de perte d'alimentation assez large si un cable casse quelque part



ARBRE COUVRANT MINIMUM

Câblage qui consomme le moins possible de ressources

- Conception de circuits électroniques
- Conception de réseaux d'écoulement d'eau
- Conception de réseau urbain
- Transmission de messages sur un réseau informatique (routage)
- Etc.

Pas toujours la meilleure idée car tout passe sur le même réseau

- Risque d'engorgement / pannes en cascade / etc.
- Le coût global est minimal mais les chemins peuvent être longs

ARBRE COUVRANT MINIMUM – MINIMUM SPANNING TREE

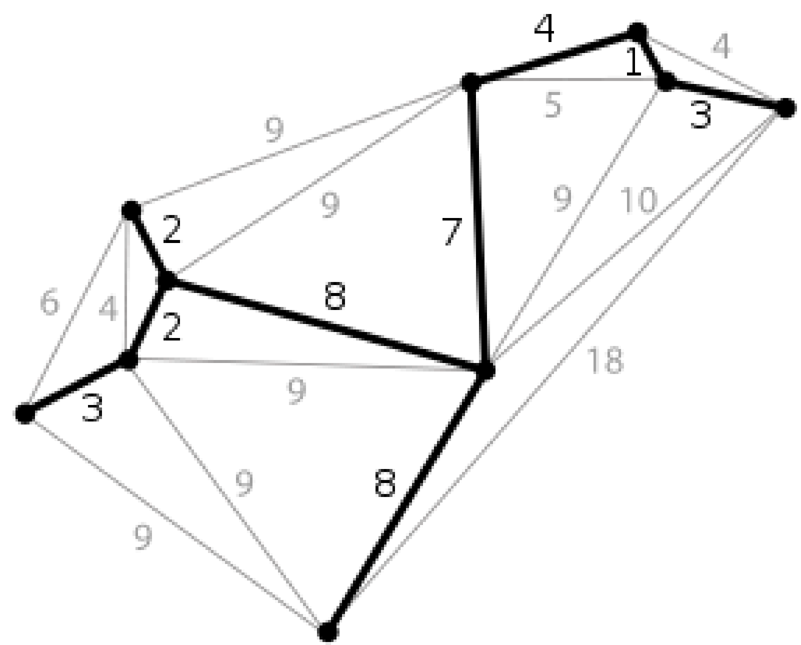
Un arbre couvrant de G=(V, E) est un graphe G'=(V', E') tel que

- G' est un sous-graphe de G
- G et G' ont les mêmes sommets
- G' est un arbre (donc sans cycle)

Un arbre couvrant est minimum si son poids total est minimum parmi tous les arbres couvrants

- Le poids total est la somme des poids des arêtes E'

N'est pertinent que pour les graphes pondérés



CONSTRUCTION D'UN ACM

Les algorithmes fonctionnent souvent de manière **gloutonne**

- Ajout d'arêtes une par une dans A jusqu'à former un arbre couvrant
- À chaque étape l'ensemble A est une partie d'un ACM donné
  - Une arête est dite **sûre** si cette propriété est vérifiée après son ajout

GENERIC-MST(G, w)

```
A = Ø
while A does not form a spanning tree
    find an edge (u, v) that is safe for A
    A = A ∪ {(u, v)}
return A
```

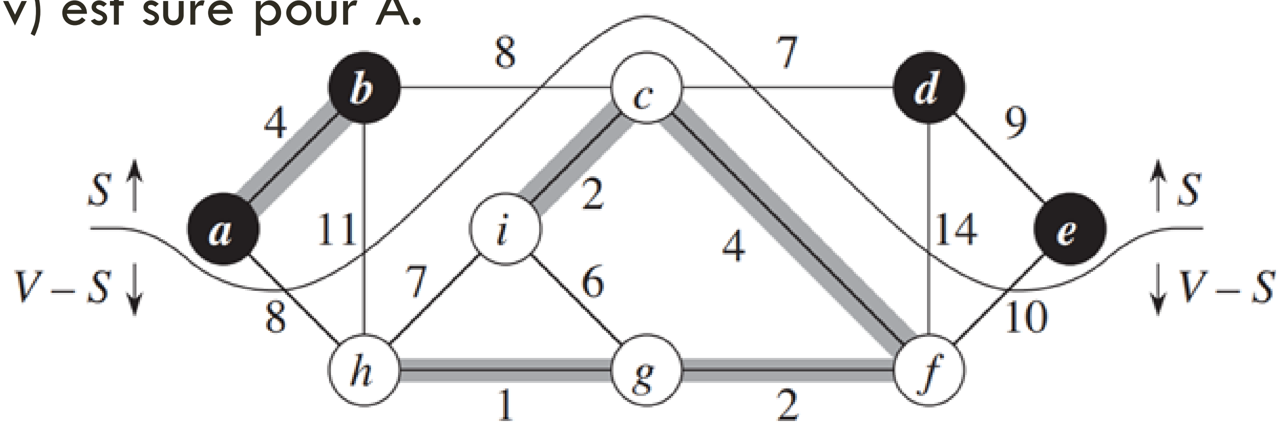
Les arêtes sûres existent, mais comment les trouver ?

CONSTRUCTION D'UN ACM - THÉORÈME

Théorème, soit

- G=(V, E) un graphe non orienté connexe pondéré
- A (arêtes grisées) un sous-ensemble d'arêtes de E contenu dans un arbre couvrant minimal
- (S, V-S) une coupe de G respectant A (aucune arête grisée n'est traversée par la coupe)
- (u, v) une arête de poids minimal traversant la coupe

Alors (u, v) est sûre pour A.

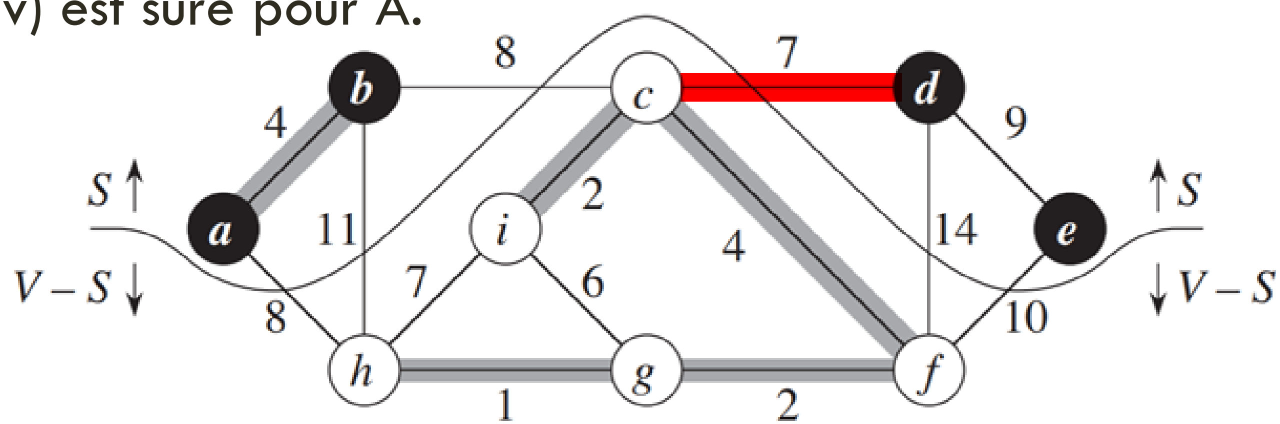


CONSTRUCTION D'UN ACM - THÉORÈME

Théorème, soit

- G=(V, E) un graphe non orienté connexe pondéré
- A (arêtes grisées) un sous-ensemble d'arêtes de E contenu dans un arbre couvrant minimal
- (S, V-S) une coupe de G respectant A (aucune arête grisée n'est traversée par la coupe)
- (u, v) une arête de poids minimal traversant la coupe

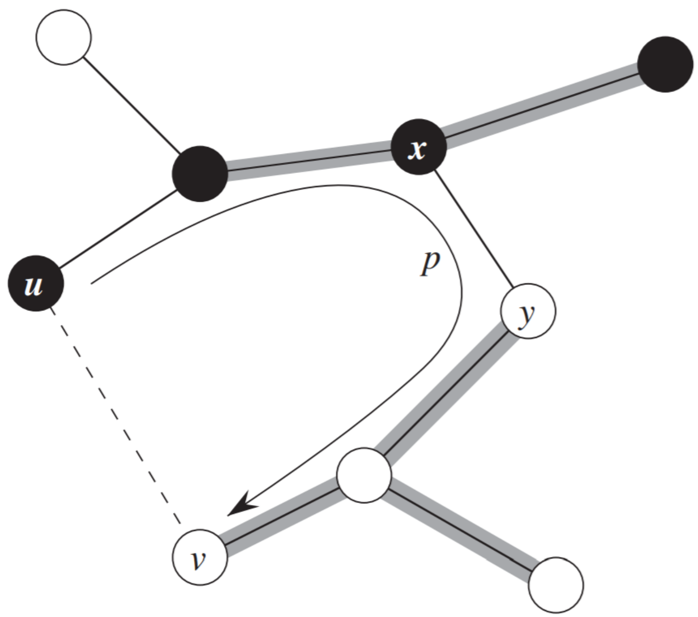
Alors (u, v) est sûre pour A.



CONSTRUCTION D'UN MST – IDÉE DE LA PREUVE

Preuve : Soit T un MST qui contient A mais pas (u, v)

- Soit p le chemin reliant u et v dans T. (u, v) + p forme donc un cycle.
- Si (u, v) traverse la coupe alors une autre arête (x, y) de T sur p aussi.
- (x, y) n'est pas dans A, car la coupe respecte A.
- Si on remplace (x, y) par (u, v) dans T, l'arbre T' obtenu est aussi un MST.
- (u, v) est donc sûre car elle appartient à un MST.

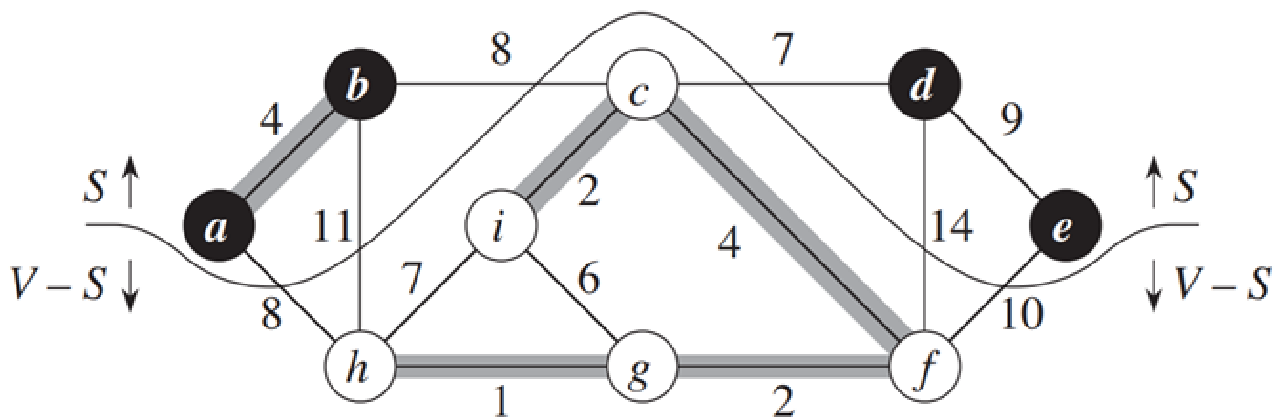


RETOUR SUR LE THÉORÈME

Qu’est-ce qu’une coupe qui respecte A ?

- Est-ce qu’un arbre de l’ensemble A peut avoir des sommets des deux côtés ?
  - Par ex. certains sommets de c-f-g-h-i sont dans V et d’autres dans S-V ?

Corollaire : Dans une coupe respectant A, chacun des arbres de l’ensemble A a tous ses sommets du même côté de la coupe.



RETOUR SUR LE THÉORÈME

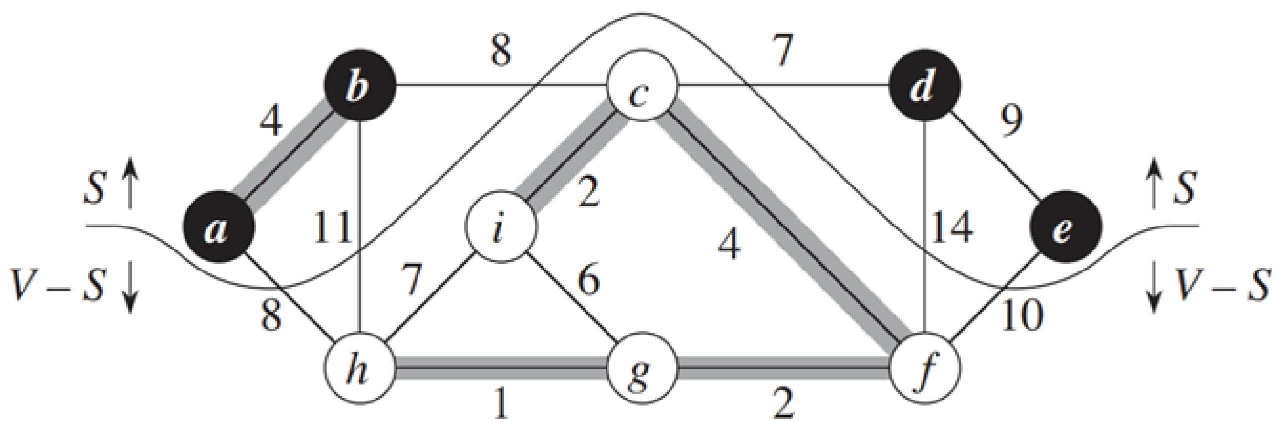
(S, V-S) une coupe de G respectant A. Quelle coupe choisir ?

- N'importe laquelle qui respecte A (Combien dans l'exemple plus bas ?)

(u, v) une arête de poids minimal traversant la coupe. Quelle arête ?

- N'importe laquelle si elle est de poids minimal (Combien dans l'exemple ?)

Corollaire : Une coupe respectant A est une répartition (quelconque) des arbres en deux groupes



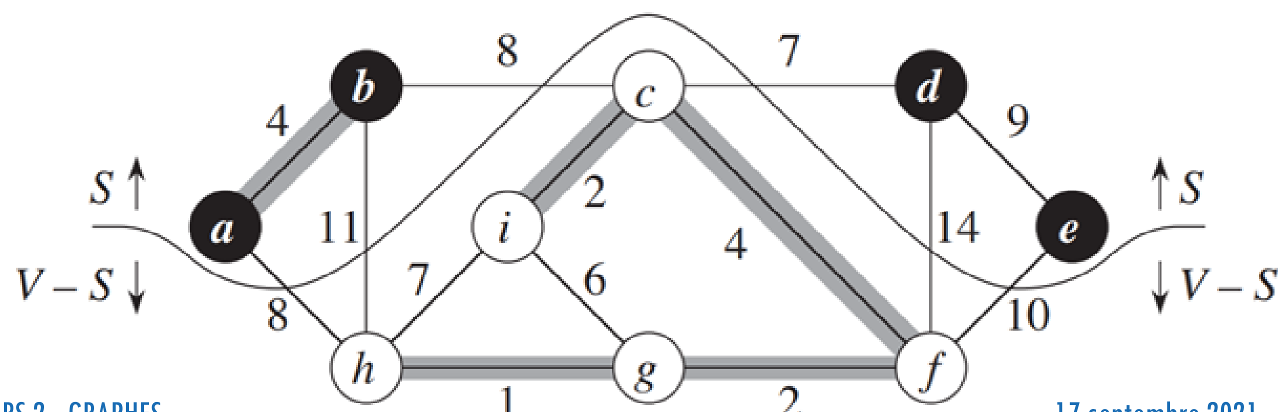
THÉORÈME + ALGORITHME

Algorithme simplifié : à chaque étape

- Prendre n’importe quel ensemble d’arbres comme coupe
  - En particulier n’importe quel arbre
- Prendre une des arêtes minimales (u, v) sortant de cet ensemble

Exemples :

- Coupe = (abde,cfghi) ⟶ arête = (c,d)
- Coupe = (ab,cdefghi) ⟶ arête = (b,c) ou (a,h)
- Coupe = (abcdfghi,e) ⟶ arête =(d,e)



COMPLEXITÉ DE L'ALGORITHME GÉNÉRIQUE

Boucle pour construire le MST

- Nombre d'itérations = n-1 (nombre d'arêtes dans un arbre couvrant)

Coût de chaque boucle

- Dépend de l'opération « trouver une arête sûre »
- Objectif = effectuer cette opération le plus efficacement possible
  - C'est-à-dire choisir des coupes qui simplifient la recherche de (u, v)

GENERIC-MST(G, w)

$A = \emptyset$
while A does not form a spanning tree
    find an edge (u, v) that is safe for A
    $A = A \cup \{(u, v)\}$
return A

DEUX ALGORITHMES

Principe de base

- Initialement on a une forêt avec autant d'arbres que de sommets

Kruskal (publié en 1956)

- On regroupe les arbres petit à petit
- Une arête sûre est une arête de poids minimum entre toute paire d'arbre

Prim (publié par Jarnik en 1930 puis redécouvert en 1959 par Prim)

- On fait grossir un seul arbre en lui ajoutant des sommets/arêtes
- La coupe est donc toujours (arbre, reste)
- Une arête sûre est une arête de poids minimum entre l'arbre et le reste

DEUX ALGORITHMES

Principe de base

- Initialement on a une forêt avec autant d'arbres que de sommets

Kruskal (publié en 1956)

- On regroupe les arbres petit à petit
- Une arête sûre est une arête de poids minimum entre toute paire d'arbre

Prim (publié par Jarnik en 1930 puis redécouvert en 1959 par Prim)

- On fait grossir un seul arbre en lui ajoutant des sommets/arêtes
- La coupe est donc toujours (arbre, reste)
- Une arête sûre est une arête de poids minimum entre l'arbre et le reste

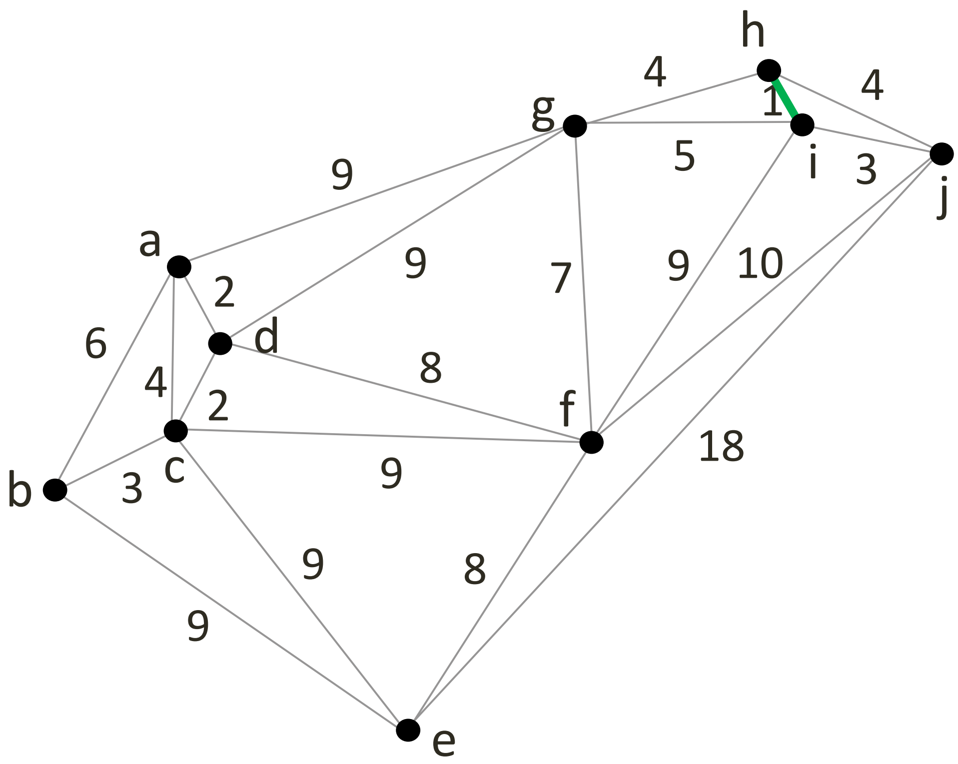
KRUSKAL – EXEMPLE D'EXÉCUTION

Arbres

- a
- b
- c
- d
- e
- f
- g
- h
- i
- j

Arête minimale :

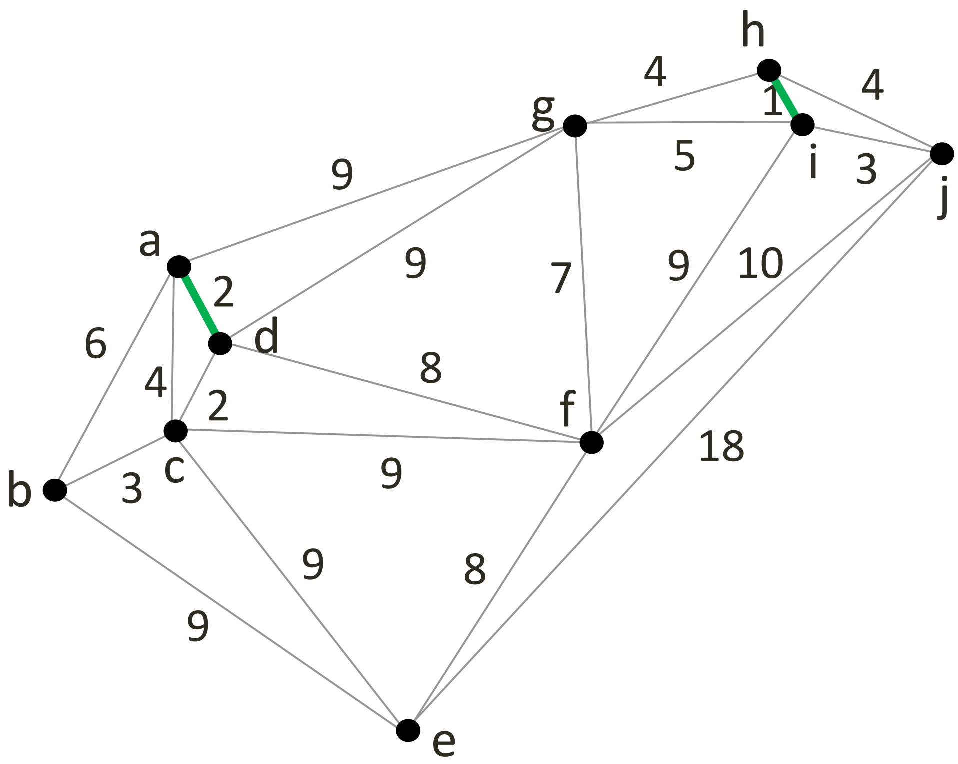
- (h,i)=1



KRUSKAL – EXEMPLE D'EXÉCUTION

Arbres

- a
- b
- c
- d
- e
- f
- g
- h,i
- j



Arêtes minimales :

- (a,d)=2
- (c,d)=2

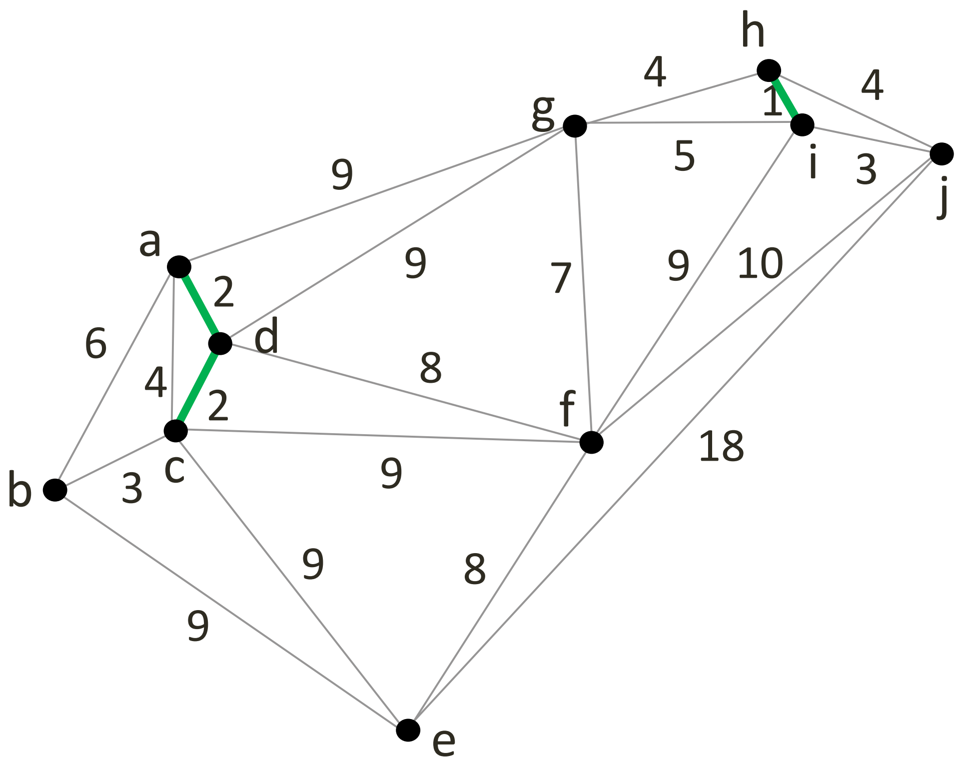
KRUSKAL – EXEMPLE D'EXÉCUTION

Arbres

- a,d
- b
- c
- e
- f
- g
- h,i
- j

Arête minimale :

- (c,d)=2



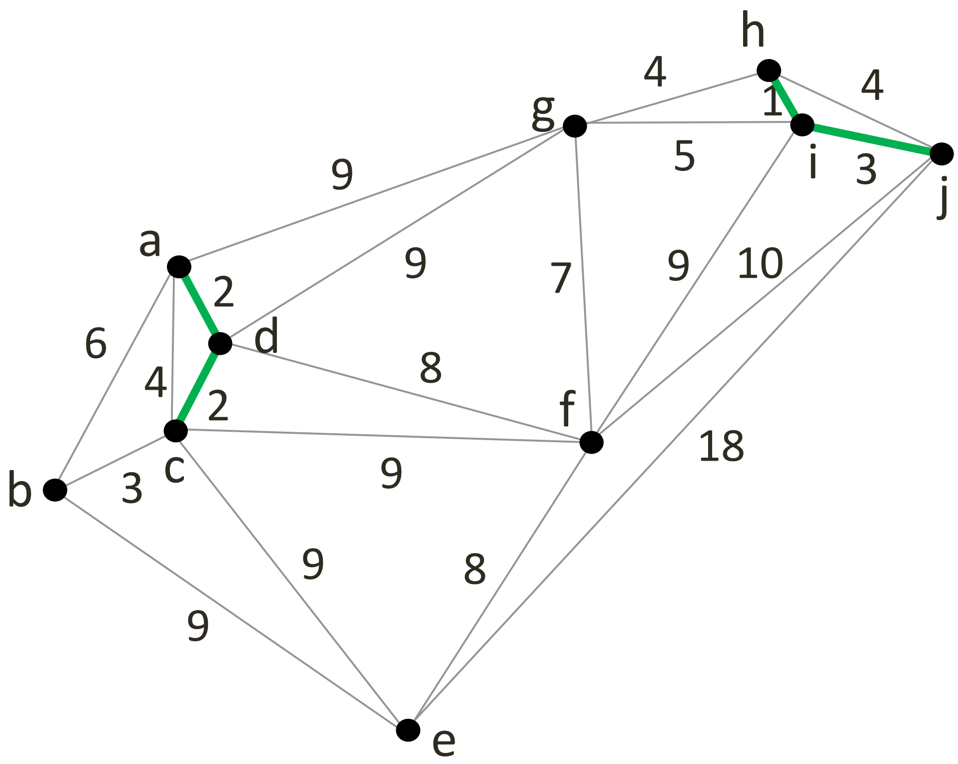
KRUSKAL – EXEMPLE D'EXÉCUTION

Arbres

- a,c,d
- b
- e
- f
- g
- h,i
- j

Arêtes minimales :

- (b,c)=3
- (i,j)=3



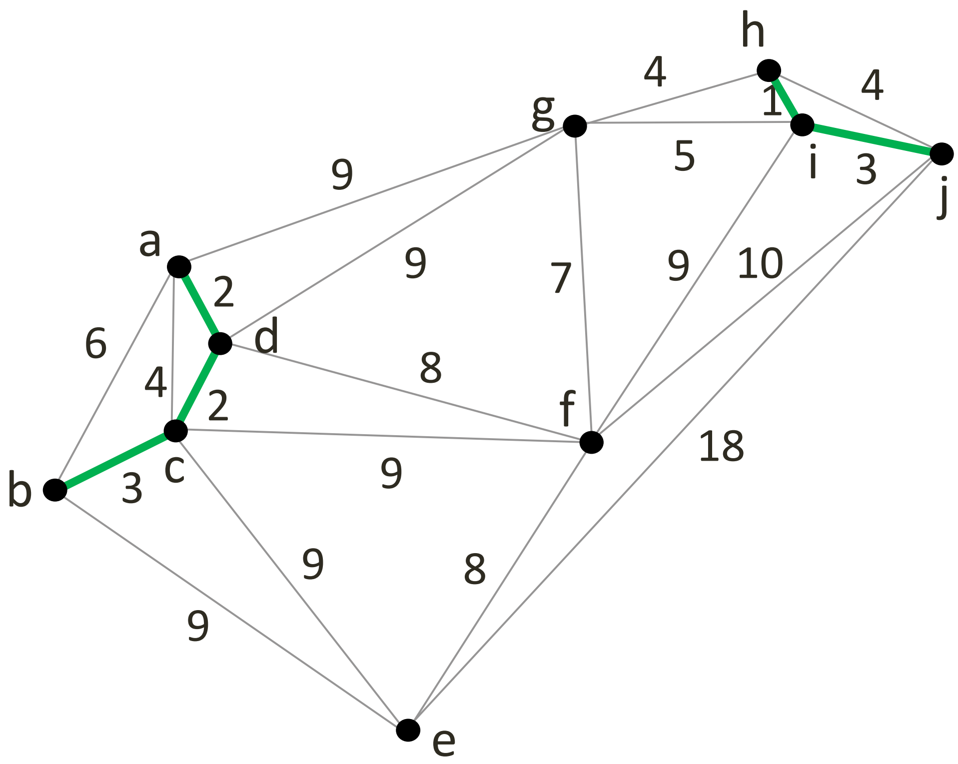
KRUSKAL – EXEMPLE D'EXÉCUTION

Arbres

- a,c,d
- b
- e
- f
- g
- h,i,j

Arête minimale :

- (b,c)=3



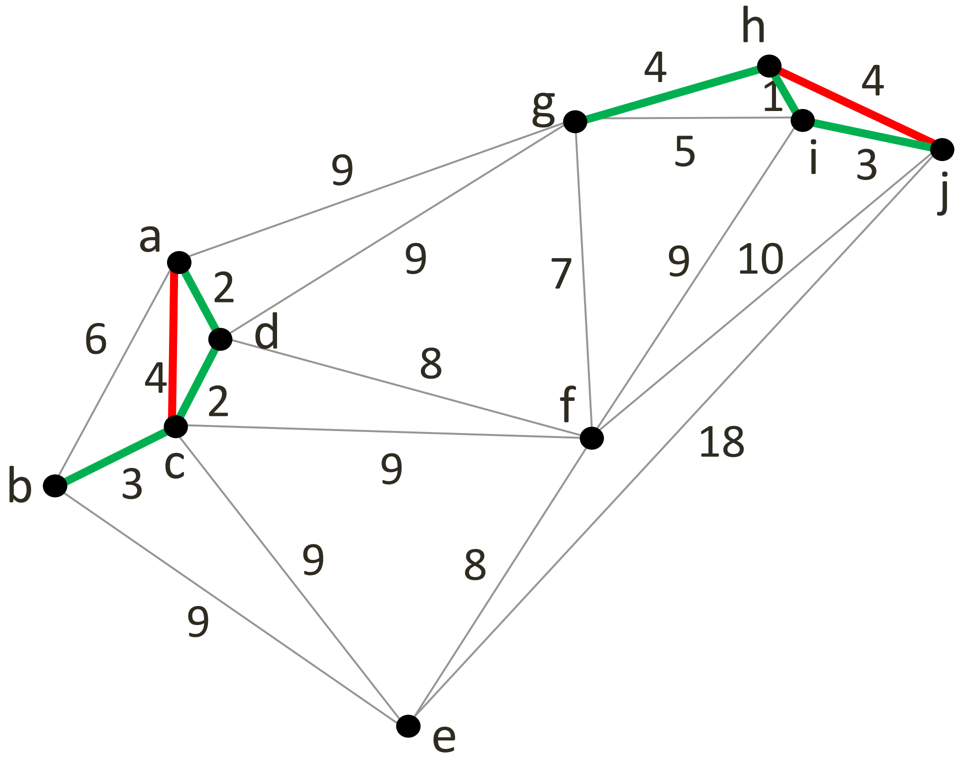
KRUSKAL – EXEMPLE D'EXÉCUTION

Arbres

- a,b,c,d
- e
- f
- g
- h,i,j

Arêtes minimales :

- (a,c)=4 non valable
- (h,j)=4 non valable
- (g,h)=4



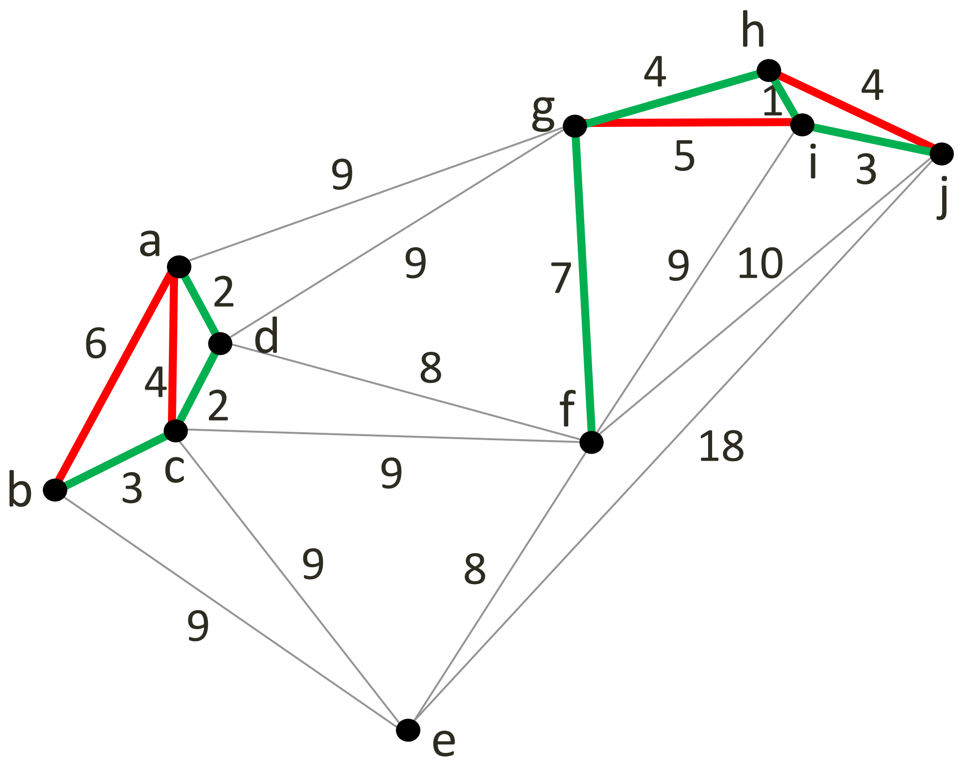
KRUSKAL – EXEMPLE D'EXÉCUTION

Arbres

- a,b,c,d
- e
- f
- g,h,i,j

Arêtes minimales :

- (g,i)=5 non valable
- (a,b)=6 non valable
- (g,f)=7



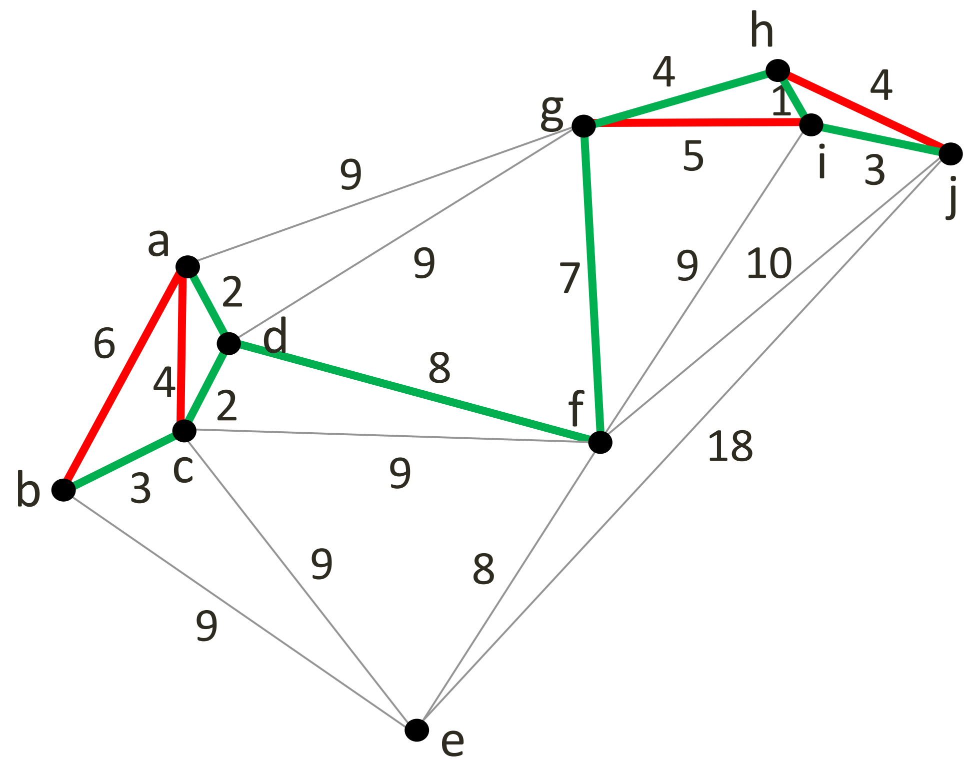
KRUSKAL – EXEMPLE D'EXÉCUTION

Arbres

- a,b,c,d
- e
- f,g,h,i,j

Arêtes minimales :

- (d,f)=8
- (e,f)=8



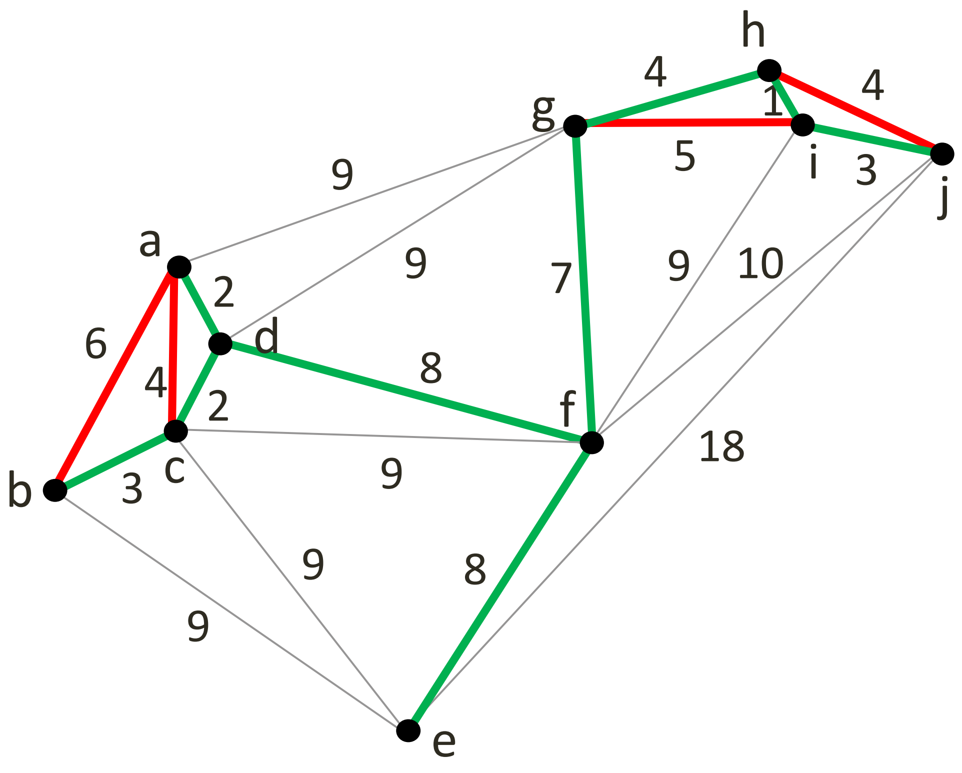
KRUSKAL – EXEMPLE D'EXÉCUTION

Arbres

- a,b,c,d,f,g,h,i,j
- e

Arête minimale :

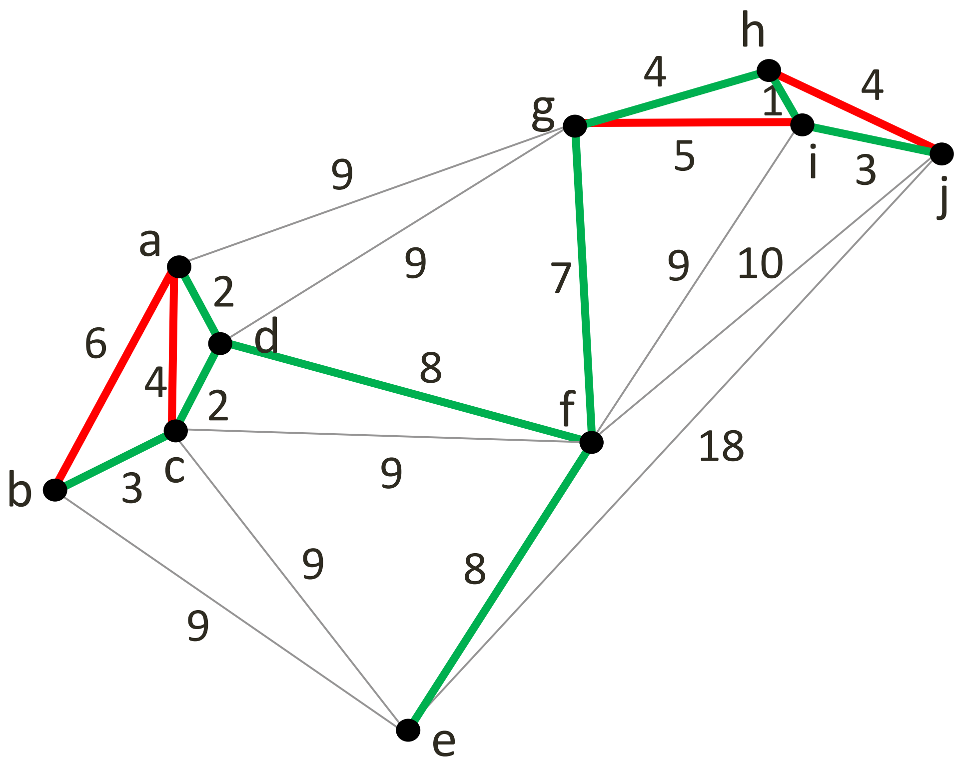
- (e,f)=8



KRUSKAL – EXEMPLE D'EXÉCUTION

Arbre

- a,b,c,d,e,f,g,h,i,j



KRUSKAL - ALGORITHME

À chaque étape on doit savoir quelle arête

- Est de poids minimum ET est entre deux arbres

Solution naïve :

- A chaque étape, on parcourt toutes les arêtes pour trouver la plus petite qui n'a pas ses deux extrémités dans le même arbre

Amélioration :

- On peut trier les arêtes par poids croissant avant l'algo
- Puis pour chaque arête par poids croissant on regarde
  - Si elle est entre deux arbres on l'ajoute dans le MST
  - Sinon on ne fait rien

KRUSKAL - ALGORITHME

Entrée : un graphe non orienté pondéré G=(S,A)

Sortie : un arbre couvrant minimum de G

Initialiser une forêt vide T

Soit A la liste des arêtes triée par poids croissant*

*Pour chaque arête (x,y) de A**

  Si l'ajout de (x,y) dans T ne crée pas de cycles

    Ajouter (x,y) dans T

Retourner T

KRUSKAL - ALGORITHME

Comment savoir si les extrémités d’une arête sont dans le même arbre ?

- Solution 1 : l’ajout de l’arête créé un cycle
  - Il suffit de faire un BFS ou DFS pour savoir
  - Complexité : O(m) pour chaque ajout d’arête
- Solution 2 : chaque arbre est identifié par un de ses élément
  - Il suffit de regarder si les extrémités ont le même identifiant
  - Complexité : ? (cf Union-Find en TD)

KRUSKAL – PREUVE DE L'ALGORITHME

À chaque étape

- On prend l'arête de poids minimum
- Elle se trouve entre deux arbres X et Y
- Donc la coupe (X, V-X) respecte l'arbre
- L'arête et minimale pour cette coupe (car elle est minimale pour tout le graphe) donc sûre

Donc Kruskal vérifie le théorème à chaque étape et l'arbre couvrant obtenu est donc minimal

KRUSKAL - COMPLEXITÉ

Implémentation par forêt d’arbres disjoints avec structure union-find

- Cf TD

Si tout est (très) bien implémenté, la complexité totale est O(E.log(V))

DEUX ALGORITHMES

Principe de base

- Initialement on a un forêt avec autant d'arbres que de sommets

Kruskal (publié en 1956)

- On regroupe les arbres petit à petit
- Une arête sûre est une arête de poids minimum entre toute paire d'arbre

Prim (publié par Jarnik en 1930 puis redécouvert en 1959 par Prim)

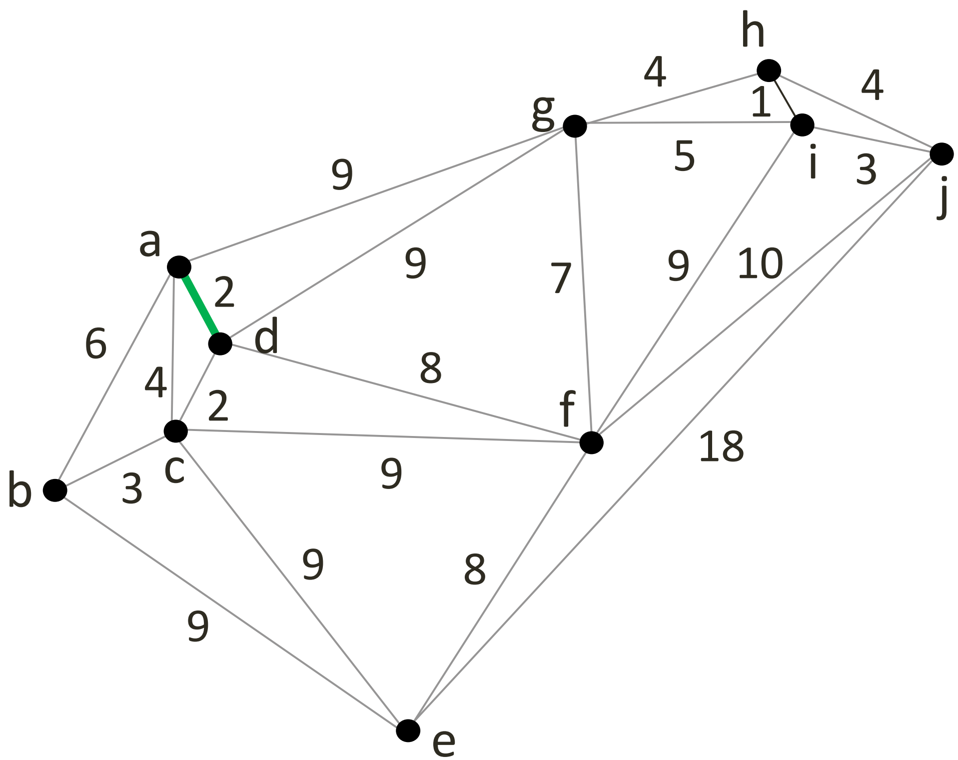
- On fait grossir un seul arbre en lui ajoutant des sommets/arêtes
- La coupe est donc toujours (arbre, reste du graphe)
- Une arête sûre est une arête de poids minimum entre l'arbre et le reste du graphe

PRIM – EXEMPLE D'EXÉCUTION

On part du sommet a

Arête minimale attachée à a :

- (a,d)=2

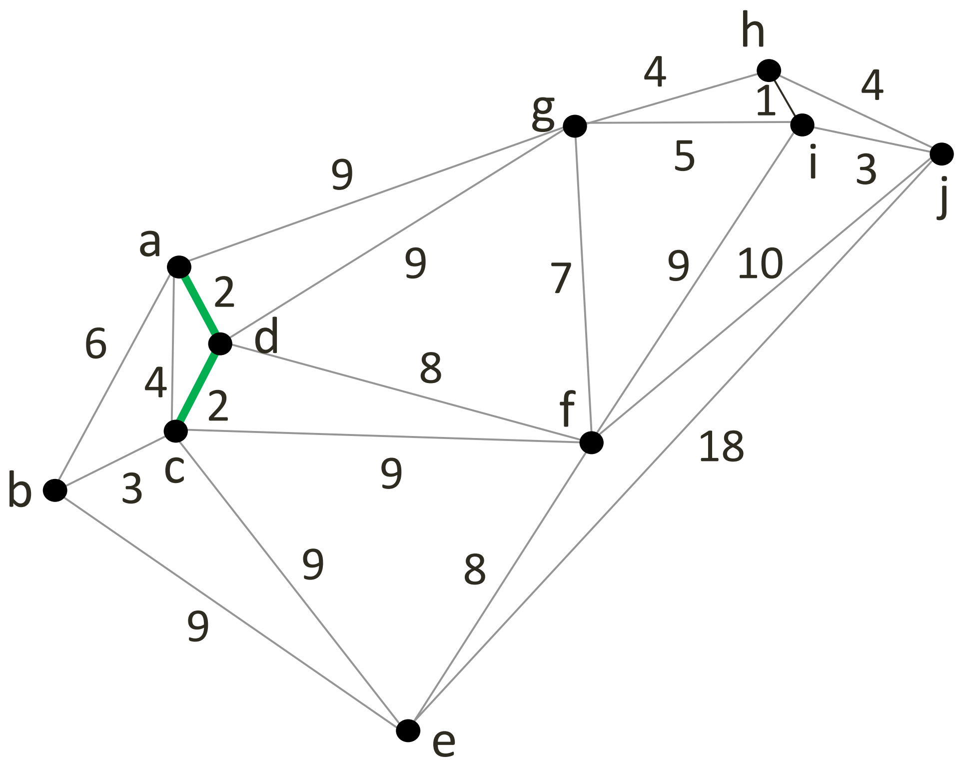


PRIM – EXEMPLE D'EXÉCUTION

On part de l'arbre ad

Arête minimale attachée à ad :

- (c,d)=2

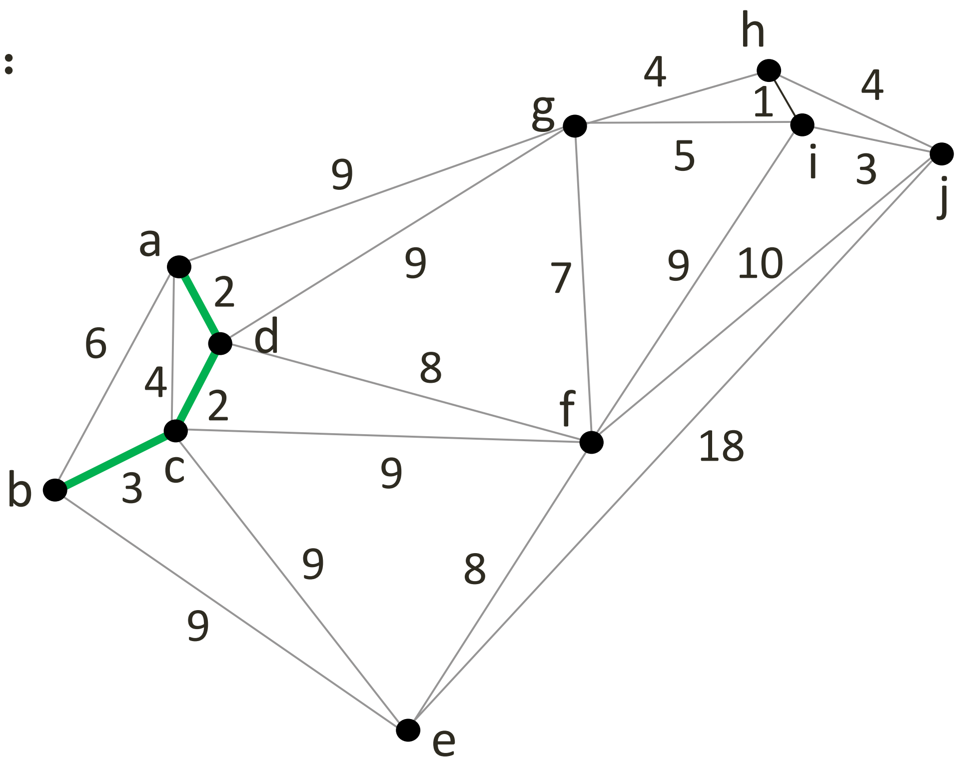


PRIM – EXEMPLE D'EXÉCUTION

On part de l'arbre acd

Arête minimale attachée à acd :

- (b,c)=3

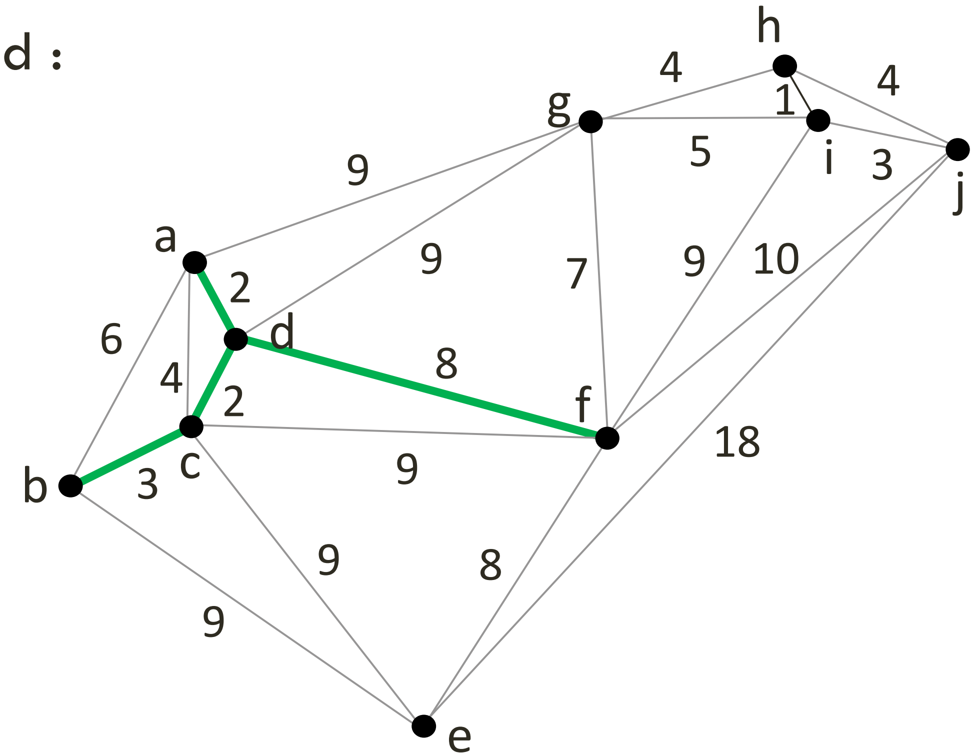


PRIM – EXEMPLE D'EXÉCUTION

On part de l'arbre abcd

Arête minimale attachée à abcd :

- (d,f)=8

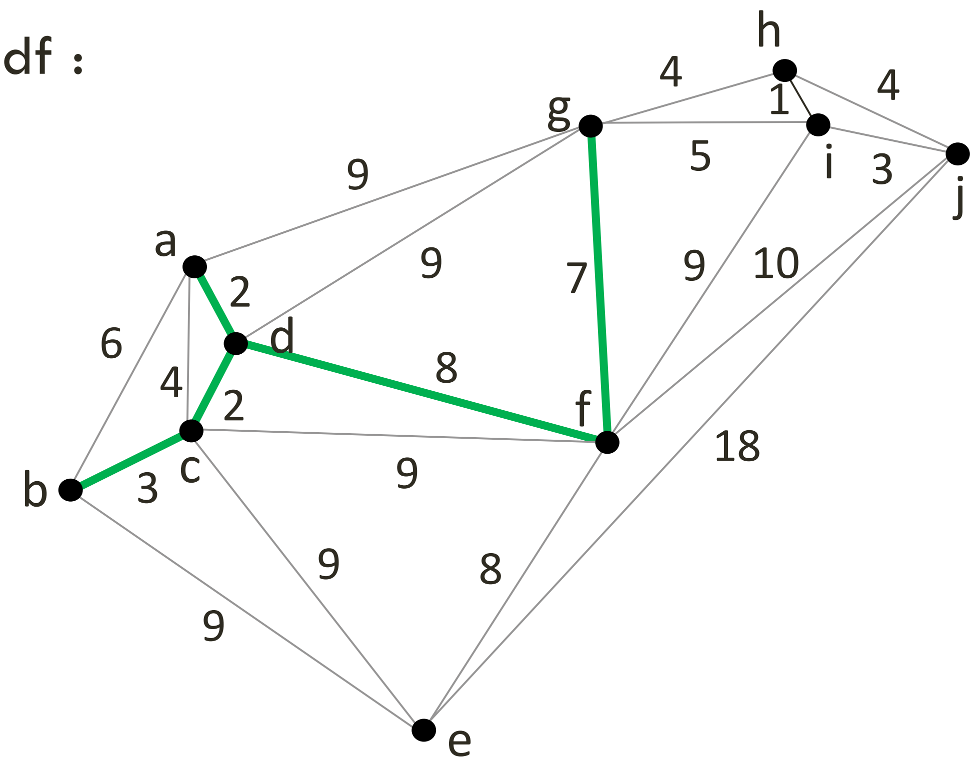


PRIM – EXEMPLE D'EXÉCUTION

On part de l'arbre abcdf

Arête minimale attachée à abcdf :

- (f,g)=7

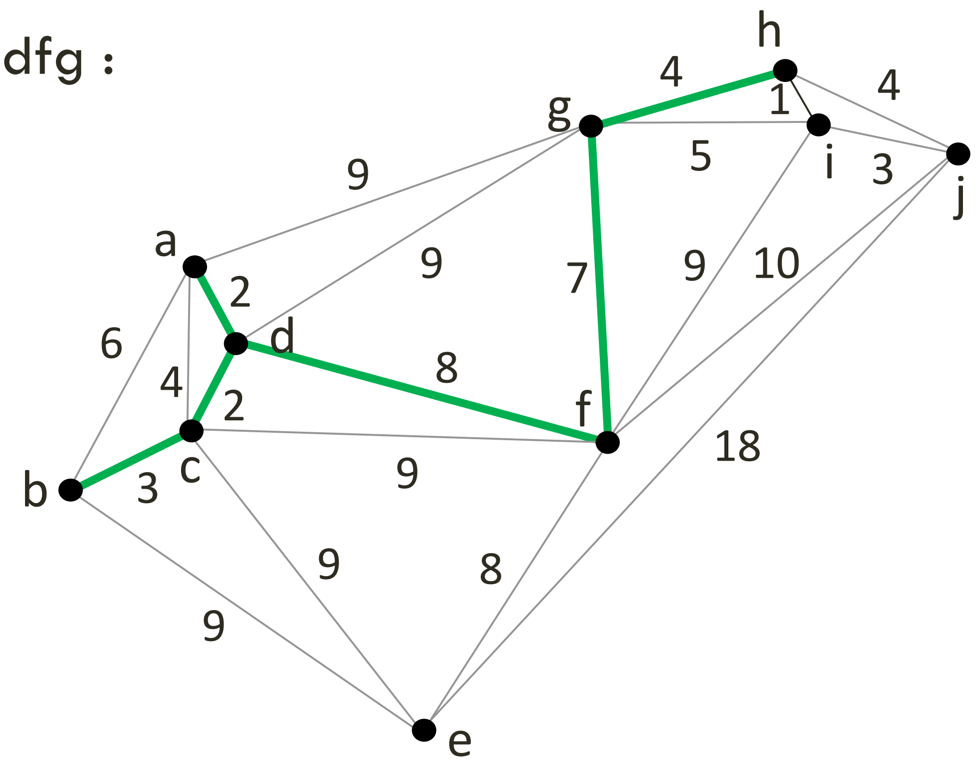


PRIM – EXEMPLE D'EXÉCUTION

On part de l'arbre abcdfg

Arête minimale attachée à abcdfg :

- (g,h)=4

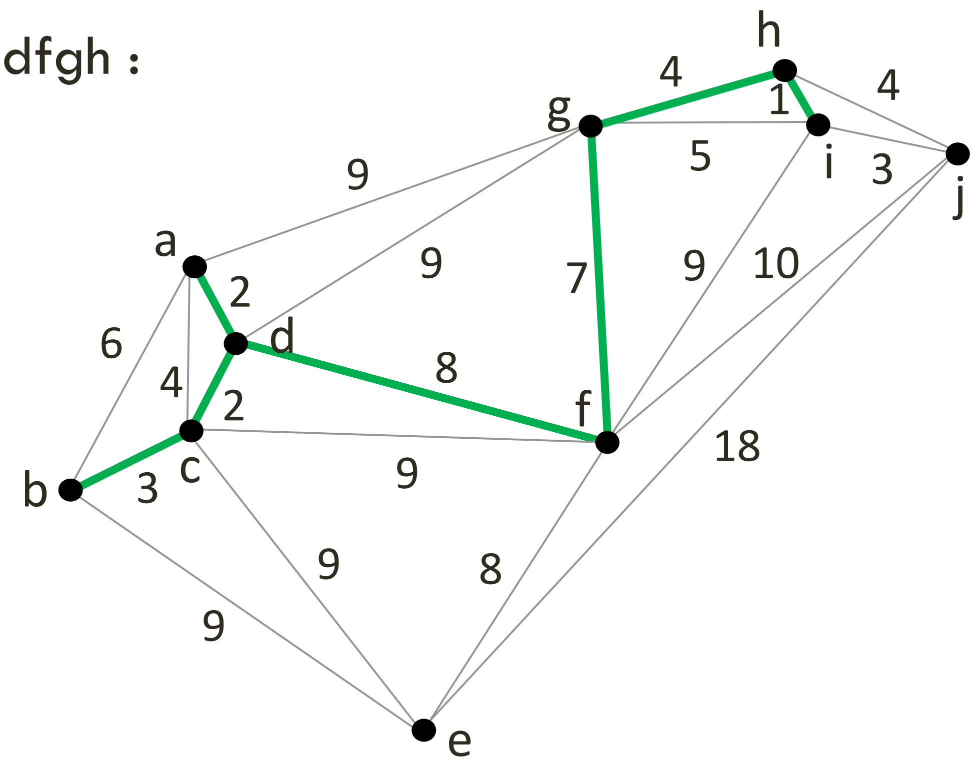


PRIM – EXEMPLE D'EXÉCUTION

On part de l'arbre abcdfgh

Arête minimale attachée à abcdfgh :

- (h,i)=1

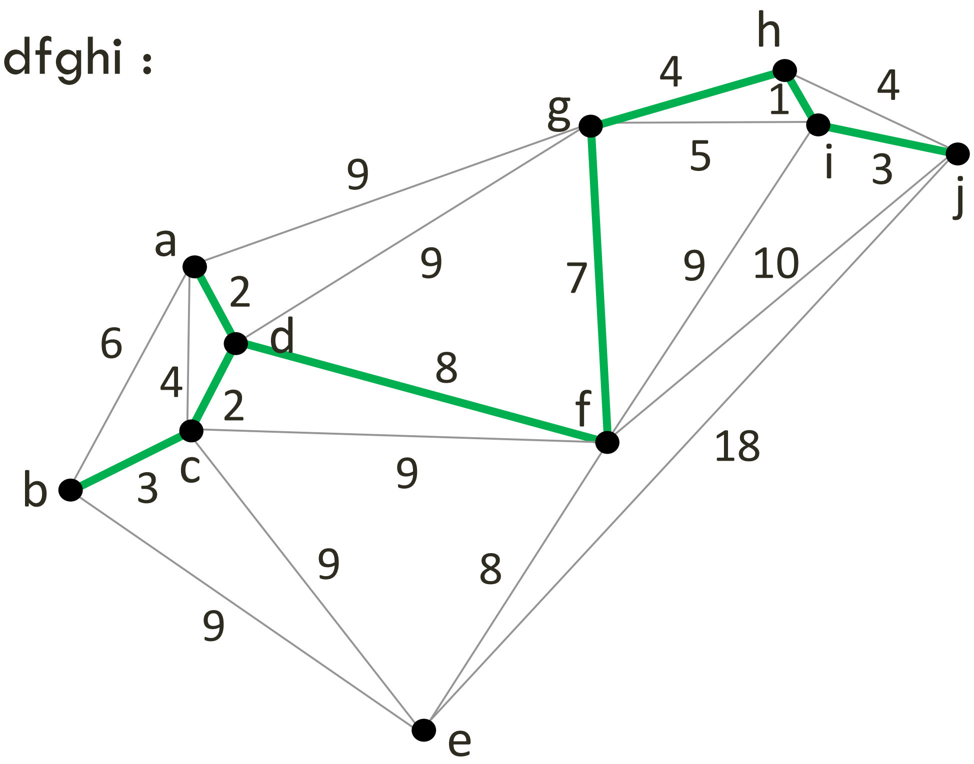


PRIM – EXEMPLE D'EXÉCUTION

On part de l'arbre abcdfghi

Arête minimale attachée à abcdfghi :

- (i,j)=3

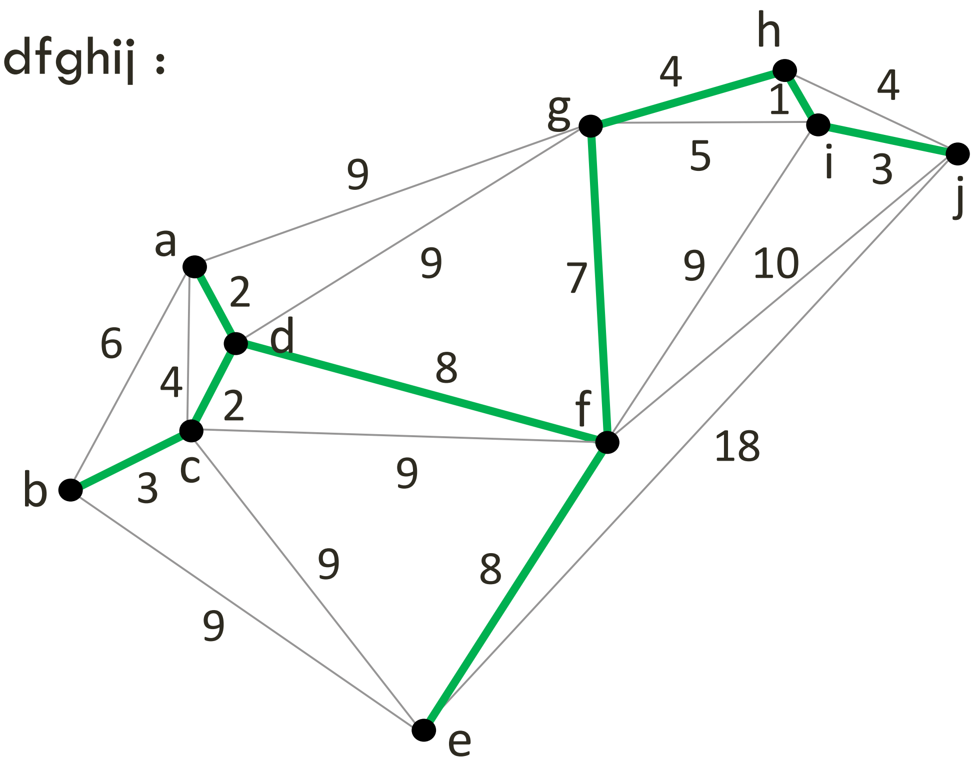


PRIM – EXEMPLE D'EXÉCUTION

On part de l'arbre abcdfghij

Arête minimale attachée à abcdfghij :

- (f,e)=8



PREUVE DE L'ALGORITHME

A nouveau, à chaque étape

- La coupe est entre l'arbre en construction et tout le reste, elle est valable car elle respecte notre arbre en construction
- On prend l'arête de poids minimum traversant la coupe
- Donc l'arête est sûre
- Et on s'arrête quand on a construit un arbre

Donc l'algorithme construit bien un MST

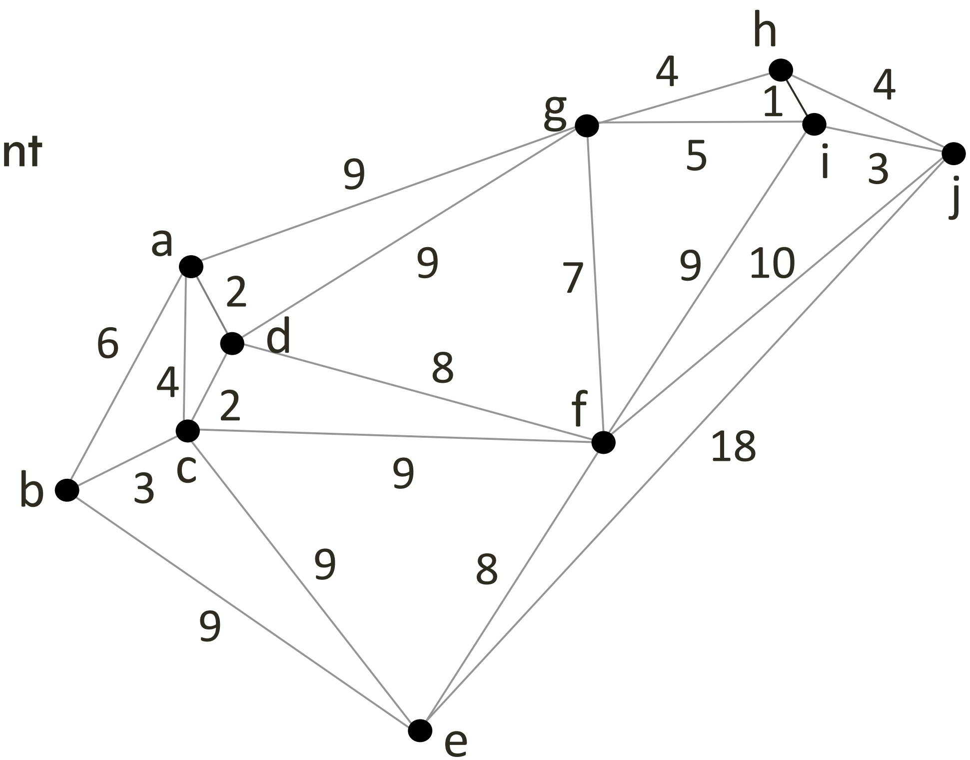
ALGORITHME DE PRIM - PRINCIPE

On stocke en permanence pour chaque sommet en dehors de l'arbre

- Le poids minimum qui le connecte à l'arbre (infini si pas d'arête)
- L'extrémité de l'arête

Si l'arbre est b, les distances sont

- (b,c,3), (b,a,6), (b,e,9)
- Et les autres sont infinies



ALGORITHME DE PRIM - PRINCIPE

On stocke en permanence pour chaque sommet en dehors de l'arbre

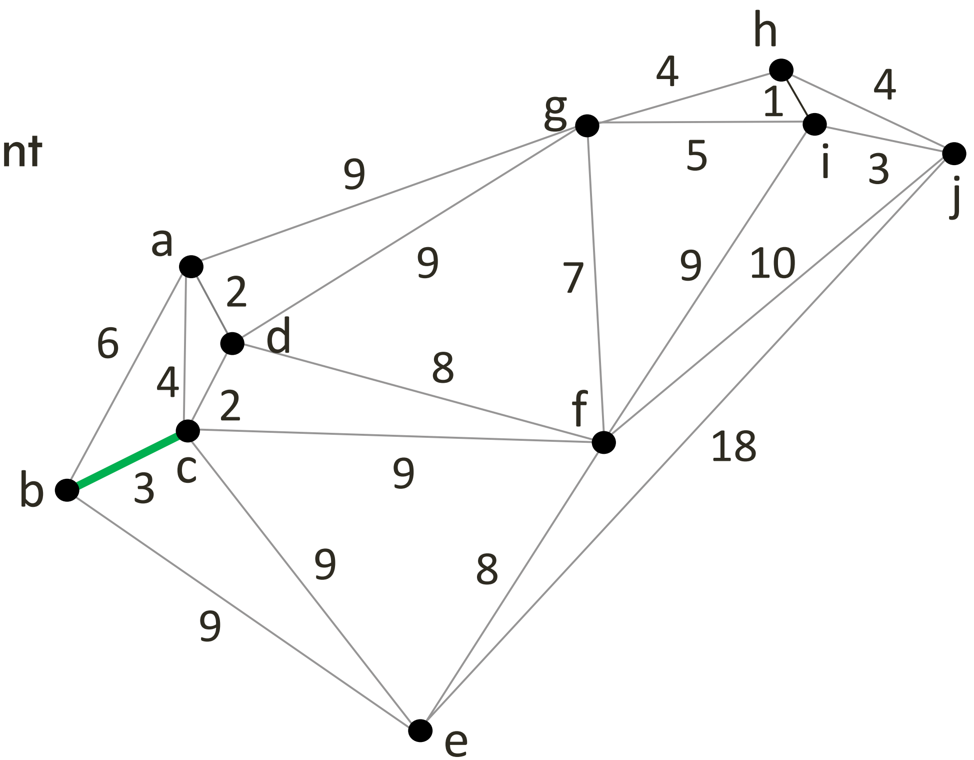
- Le poids minimum qui le connecte à l'arbre (infini si pas d'arête)
- L'extrémité de l'arête

Si l'arbre est b, les distances sont

- (b,c,3), (b,a,6), (b,e,9)
- Et les autres sont infinies

On ajoute l'arête (b,c)

- (c,d,2), (c,a,4), (b,e,9), (c,f,9)
- Et les autres sont infinies



ALGORITHME DE PRIM - PRINCIPE

On stocke en permanence pour chaque sommet en dehors de l'arbre

- Le poids minimum qui le connecte à l'arbre (infini si pas d'arête)
- L'extrémité de l'arête

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

ALGORITHME DE PRIM - COMPLEXITÉ

Utilisation d'une file de priorité (en ordre inverse)

- Création : toutes les distances sont infinies
- Recherche : retourne la plus petite clé (arête de poids minimum)
- Mise à jour : modifie une distance (uniquement en la diminuant)

Une file de priorité peut être implémentée de plusieurs façons

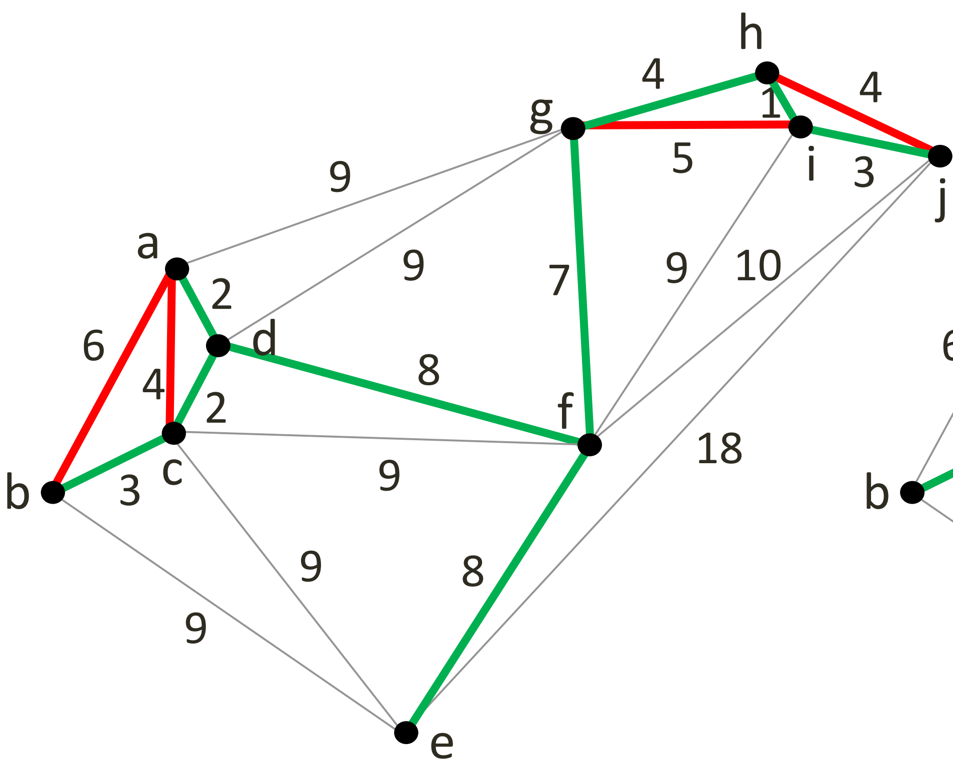
- Tableau (recherche pas efficace)
- Tableau trié (mise à jour pas efficace)
- Arbre binaire de recherche (mieux mais pas parfait)
- Tas de Fibonacci (recherche amortie O(log(V)), diminution amortie O(1), insertion O(1))

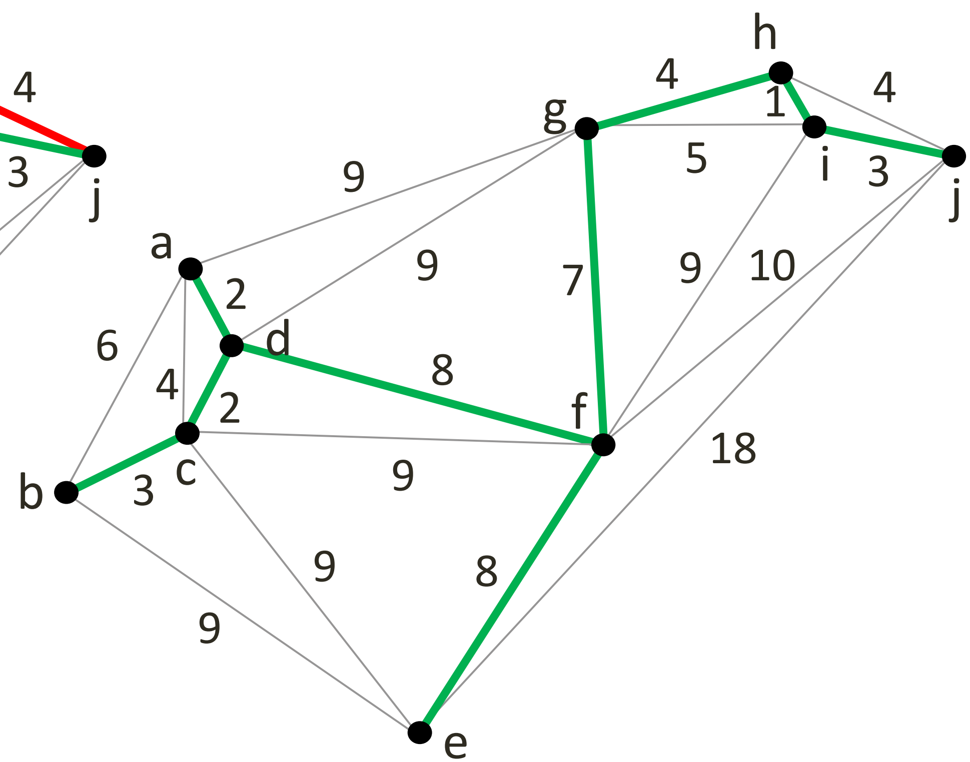
Si tout est (très) bien implémenté, la complexité est O(E +V.log(V))

COMPARAISON DU RÉSULTAT DES DEUX ALGORITHMES

Sur cet exemple ils sont identiques mais en général non

- Si plusieurs arêtes avec le même poids on doit souvent choisir





COMPARAISON DES COMPLEXITÉS

Kruskal : O(E.log(V))

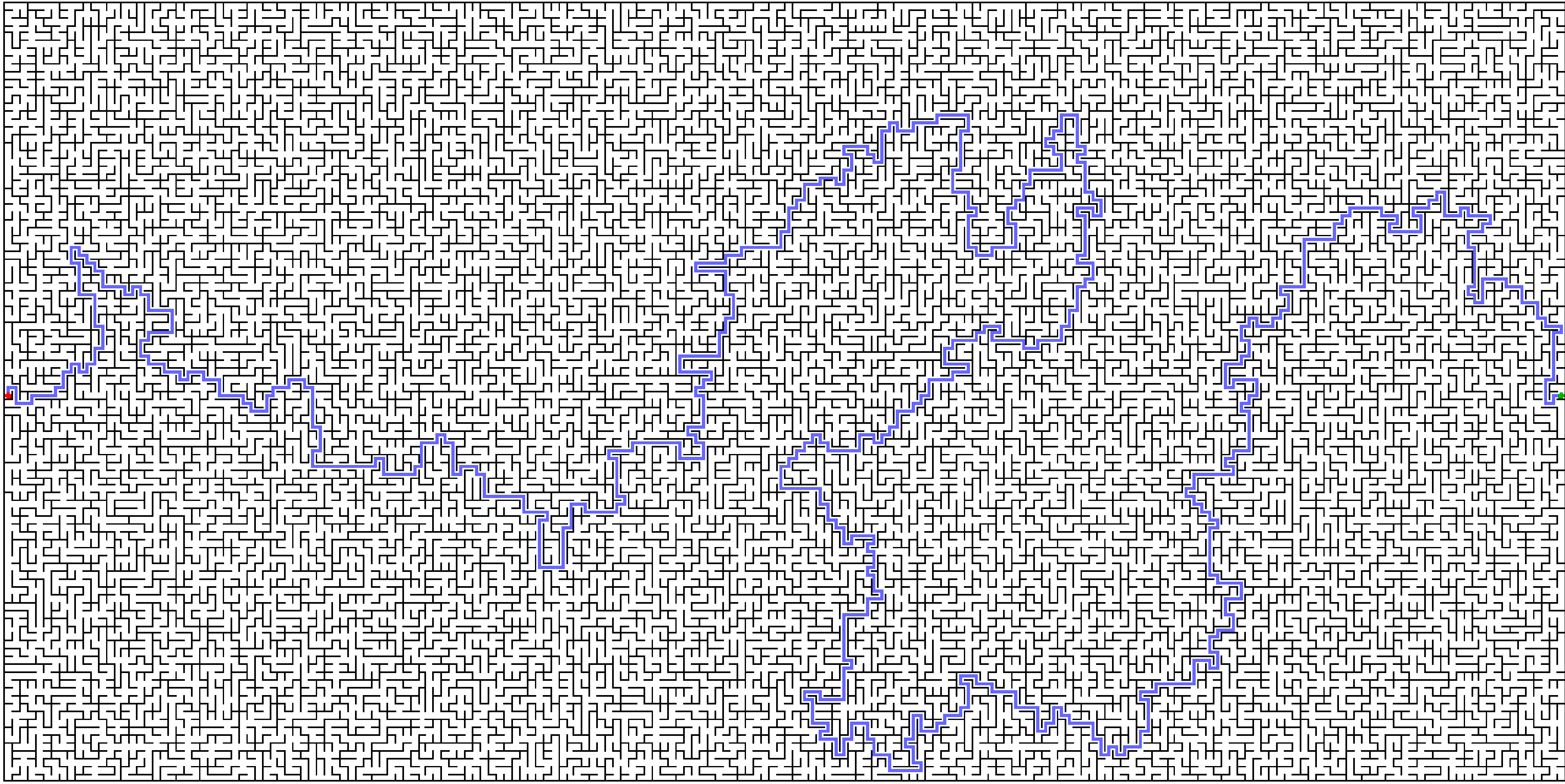
Prim : O(E +V.log(V))

Lequel choisir ?

- Kruskal est plus facile à implémenter
- Si E est élevé, plutôt Prim
- Si on peut trier les poids en temps linéaire alors Kruskal plus rapide

Et d'autres algorithmes

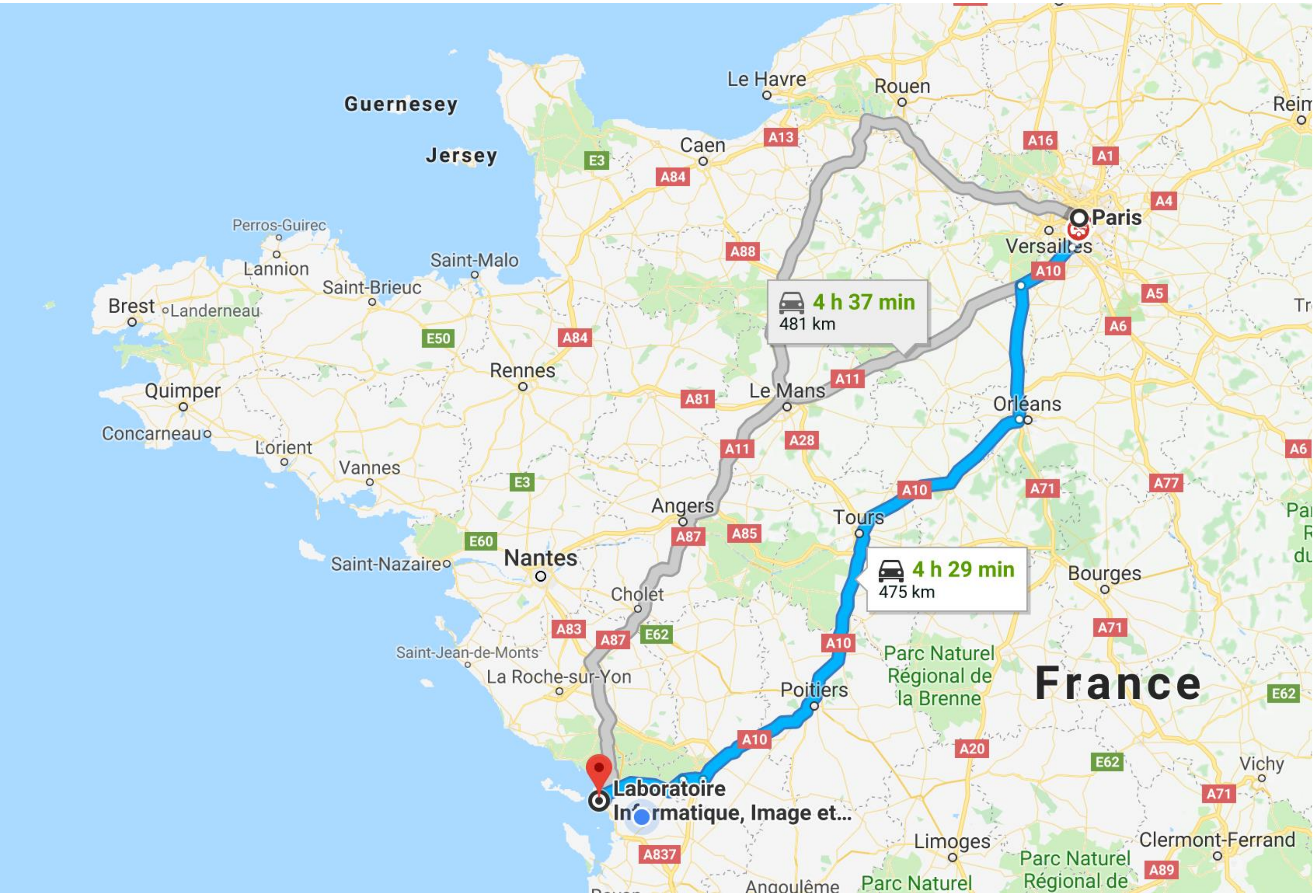
- Borůvka
- Chazelle (pas glouton)
- …



PLUS COURTS CHEMINS

Laboratoire Informatique Image Interaction (L3I)
sité de La Rochelle - Pôle Sciences et Technologie - Avenue Michel Crépeau - 17042 LA ROCHELLE CEDEX 1 Fra
Tél : +33 (0)5 46 45 82 62 – Fax : 05.46.45.82.42 – Site internet : http://l3i.univ-larochelle.fr/

EXEMPLE TYPIQUE : CALCUL DE TRAJET



QUELQUES PROBLÈMES LIÉS AUX PCC

Plus court chemin entre deux sommets

- Un plus court chemin entre une source unique et une destination unique

Problème plus général = **PCC à origine unique**

- Un plus court chemin depuis une source vers tous les autres sommets
- On peut résoudre ce problème aussi efficacement que le premier (!)

Plus courts chemins à destination unique

- Un plus court chemin depuis tous les sommets vers une unique destination
- Il suffit d'inverser les arêtes pour se ramener au problème précédent

Tous les plus courts chemins

- Un plus court chemin depuis tous les sommets vers tous les sommets
- On peut faire mieux que PCC à origine unique depuis toutes les sources

Presque plus courts chemins

- Deuxième plus court chemin, ou plus court chemin avec contraintes

SOUS-CHEMIN

Théorème : Toute partie d'un plus court chemin est un plus court chemin

Preuve (par contradiction)

- Supposons qu'un PCC entre v_1 et v_k soit
$$v_1 \xrightarrow{p_{1x}} v_x \xrightarrow{p_{xy}} v_y \xrightarrow{p_{yk}} v_k$$
- Si p_{xy} n'est pas le plus court chemin entre x et y alors il existe un chemin p'_{xy} plus court
- Mais dans ce cas $v_1 \xrightarrow{p_{1x}} v_x \xrightarrow{p'_{xy}} v_y \xrightarrow{p_{yk}} v_k$ est plus court que le chemin original qui n'était donc pas un PCC

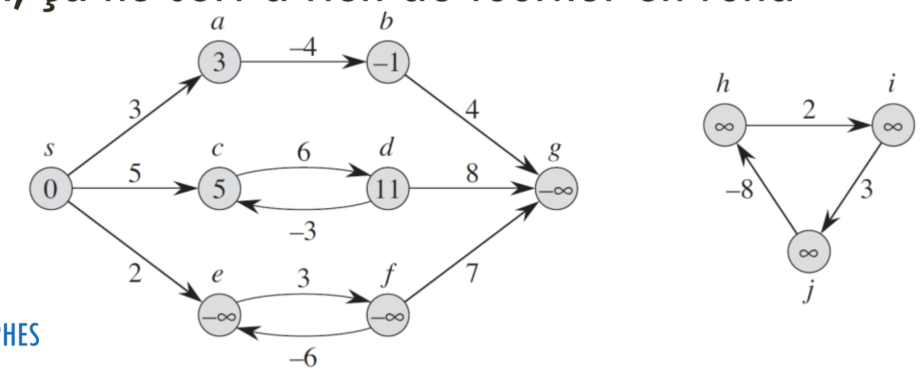
POIDS NÉGATIFS ET CYCLE

Les définitions restent valables s'il y a des arêtes de poids négatif

- Ex : trajet qui coute le moins cher
  - péage, essence : poids positif (ça coute de l'argent)
  - Co-voiturage : poids négatif (ça rapporte de l'argent)

On peut toujours supposer qu'un PCC ne contient pas de cycle

- Si poids positif on ne veut pas l'emprunter
- Si poids négatif ce n'est pas bien défini (on répète le cycle à l'infini, comme un taxi qui ferait exprès de tourner en rond sur un rond-point…)
- Si poids nul, ça ne sert à rien de tourner en rond



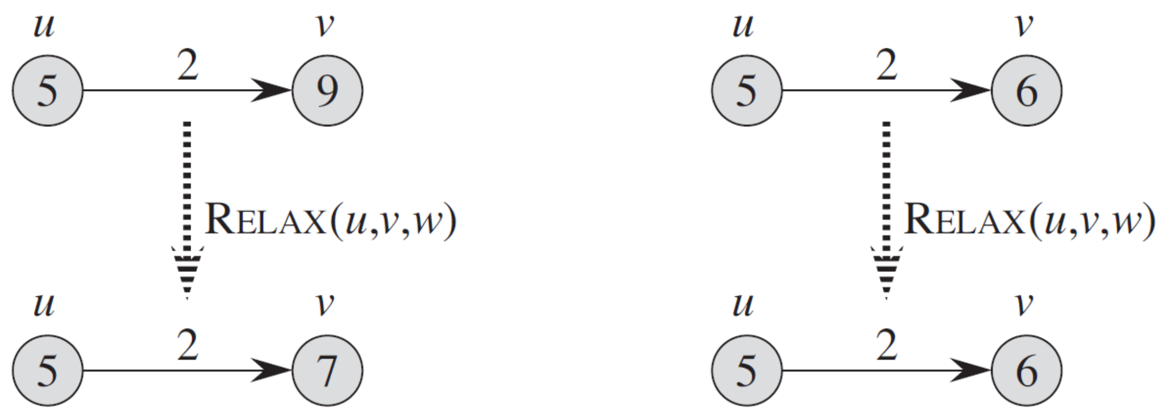
ALGORITHMES STANDARDS

Les algorithmes classiques gardent en permanence une borne supérieure de la longueur des plus courts chemins

- Initialement tous les sommets sont à distance ∞ de la source
- A chaque découverte d'un nouveau chemin plus court on remplace l'ancien
- Si on découvre un chemin plus long on ne fait rien

Cette opération est appelée relaxation

- Ex : si u est à distance au plus 5, v à distance au plus 9 or d(u,v)=2, alors ?
- Ex : si u est à distance au plus 5, v à distance au plus 6 or d(u,v)=2, alors ?



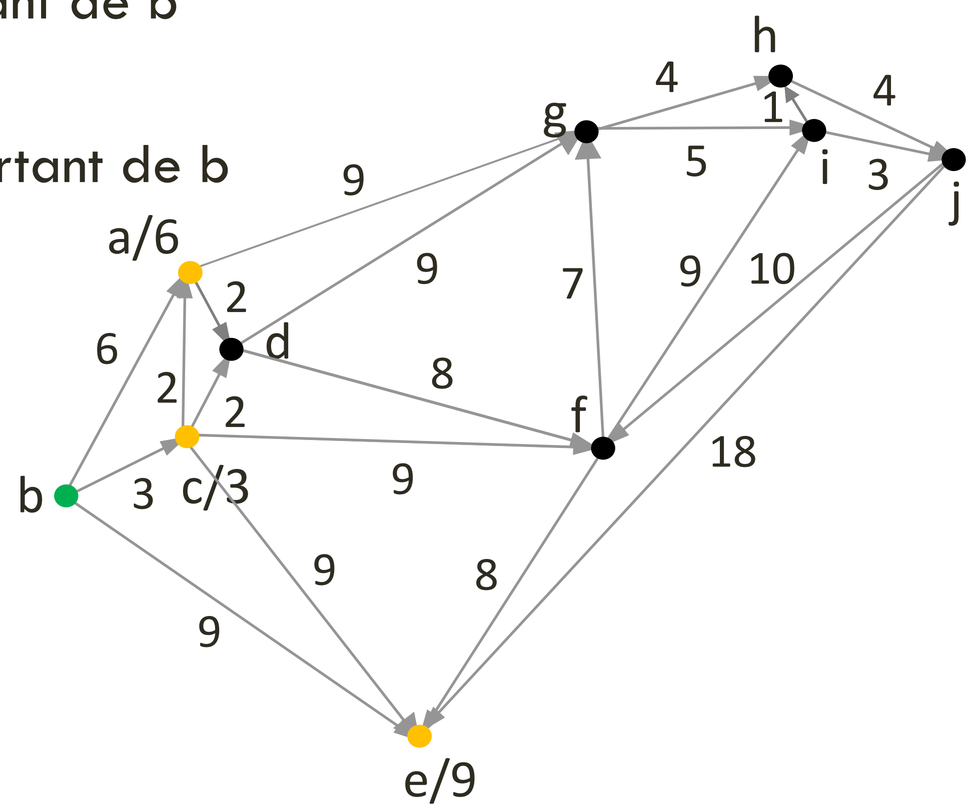
ALGORITHME DE DIJKSTRA

Distances certaines en partant de b

- b=0

Distances surestimées en partant de b

- a=6
- c=3 est sûre
- e=9
- Autres = ∞



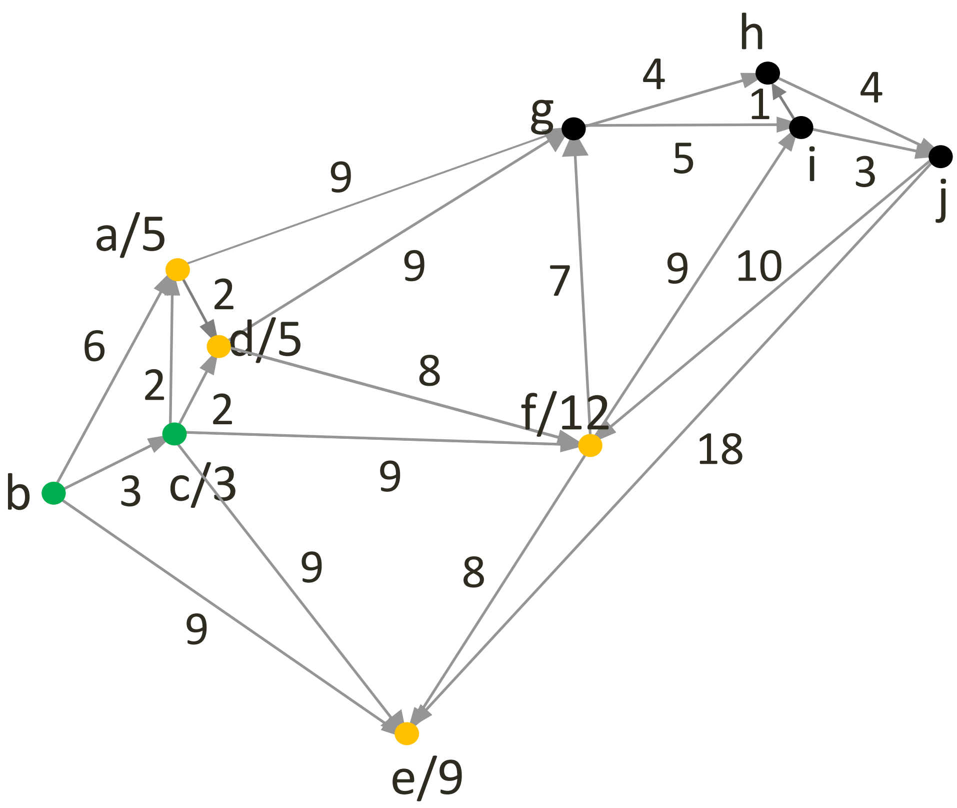
ALGORITHME DE DIJKSTRA

Distances certaines

- b=0
- c=3

Distances surestimées

- a=~~6~~ 5 par relaxation
- d=5
- e=9 ~~12~~ par relaxation
- f=12
- Autres = ∞



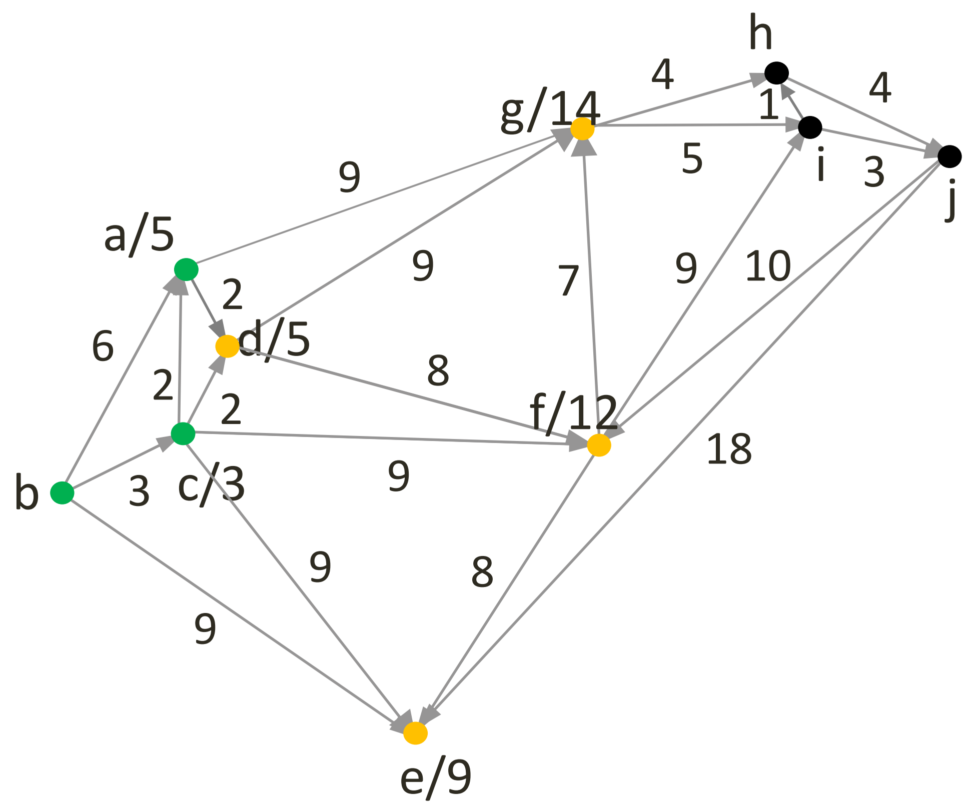
ALGORITHME DE DIJKSTRA

Distances certaines

- b=0
- c=3
- a=5

Distances surestimées

- d=5
- e=9
- f=12
- g=14
- Autres = ∞



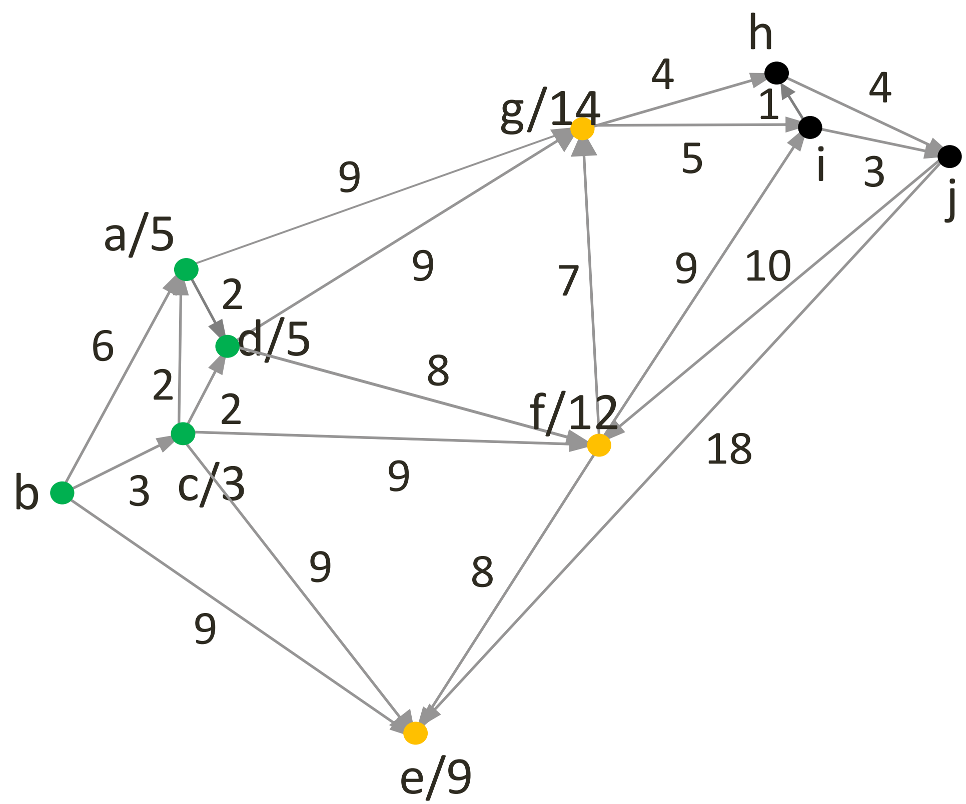
ALGORITHME DE DIJKSTRA

Distances certaines

- b=0
- c=3
- a=5
- d=5

Distances surestimées

- …



ALGORITHME DE DIJKSTRA

Parcours du graphe par distance croissante

```
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
S = ∅
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

```
RELAX(u, v, w)
if v.d > u.d + w(u, v)
    v.d = u.d + w(u, v)
    v.π = u
```

Principe : Toutes les distances pour les sommets de S sont correctes

- Preuve ?

COMPLEXITÉ

L'algorithme ressemble fortement à celui de Prim

- On ajoute les sommets un par un dans l'ensemble
- On met à jour les distances à la source (plutôt qu'à l'ensemble)

Utilisation d'une file de priorité (en ordre inverse)

- Création : toutes les distances sont infinies
- Recherche : retourne la plus petite clé (sommet le plus proche)
- Relaxation : modifie une distance (uniquement en la diminuant)

La complexité est $O(E + V.\log(V))$ avec une file de priorité

POIDS NÉGATIFS

Pourquoi Dijkstra ne fonctionne pas si poids négatifs ?