Algorithmique avancée

L3 Informatique

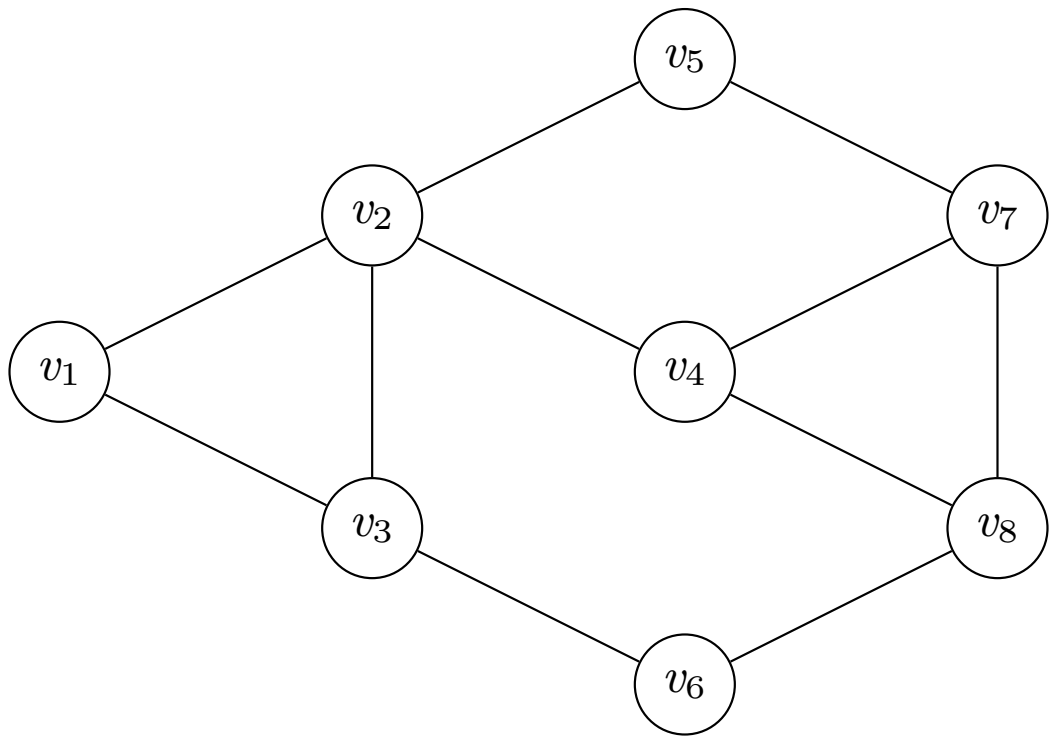
Etienne Birmelé

I. Introduction : Notion de graphe

Notion de graphe

Graphe

Un graphe G est un couple (V, E) où V est l'ensemble des sommets et $E \subset V \times V$ est l'ensemble des arêtes.
Il est représenté graphiquement par un ensemble de points (les sommets) reliés par des segments (les arêtes).



Notion de graphe

- Le graphe est l'objet mathématique permettant de décrire des interactions entre sujets.
- Applications aussi diverses que l'informatique, les télécommunications, la sociologie ou la biologie.
- Principe : traduire le problème posé en un problème d'algorithmique à résoudre sur des graphes.

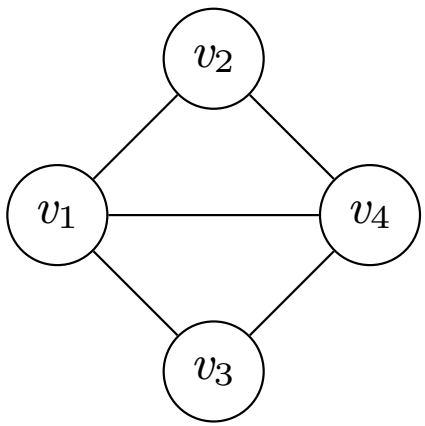
Exemple : Attribution de fréquences à des émetteurs.

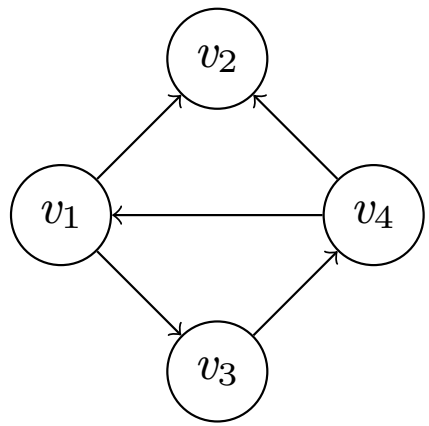
Deux émetteurs trop proches et non séparés par un obstacle naturel ne peuvent recevoir le même fréquence. Le problème devient un problème de coloration de graphes.

Types de graphes

Suivant le type de problème que l'on cherche à résoudre, on peut considérer des graphes

- non-orienté ou orienté.

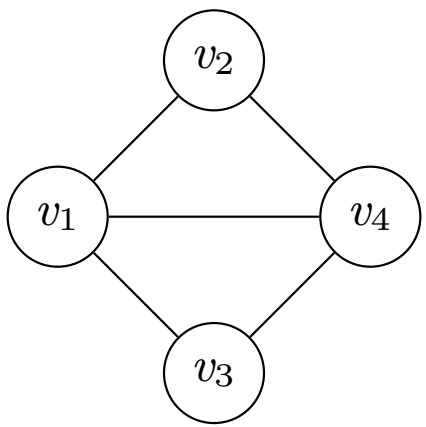


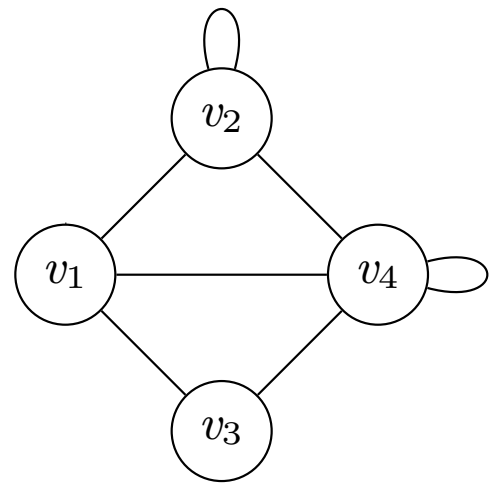


Types de graphes

Suivant le type de problème que l'on cherche à résoudre, on peut considérer des graphes

- non-orienté ou orienté.
- avec ou sans boucle

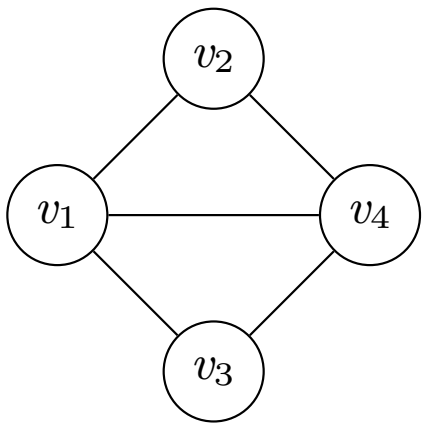


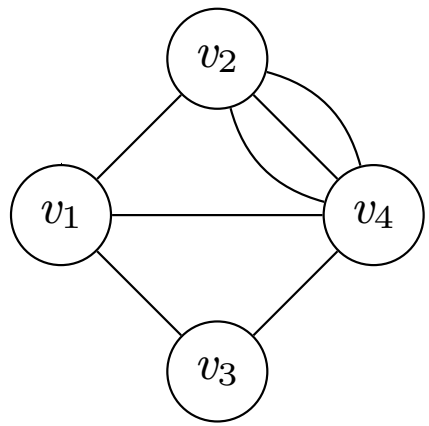


Types de graphes

Suivant le type de problème que l'on cherche à résoudre, on peut considérer des graphes

- non-orienté ou orienté.
- avec ou sans boucle
- simple ou avec des arêtes multiples

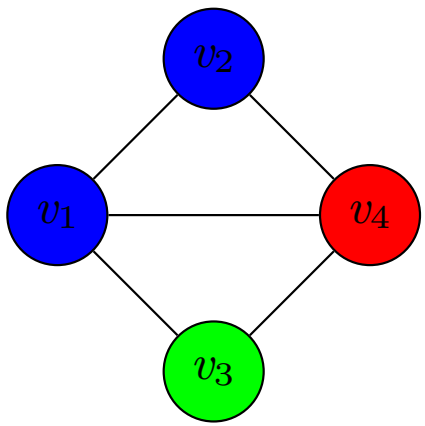


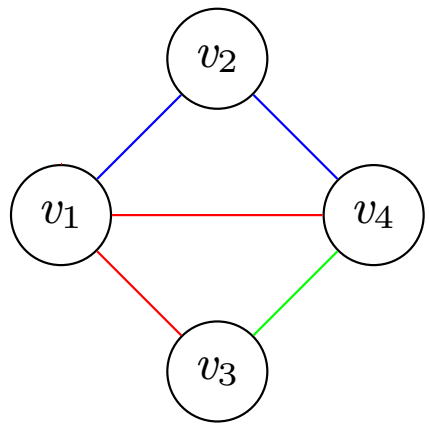


Types de graphes

Suivant le type de problème que l'on cherche à résoudre, on peut considérer des graphes

- non-orienté ou orienté.
- avec ou sans boucle
- simple ou avec des arêtes multiples
- colorés (sommets ou arêtes)

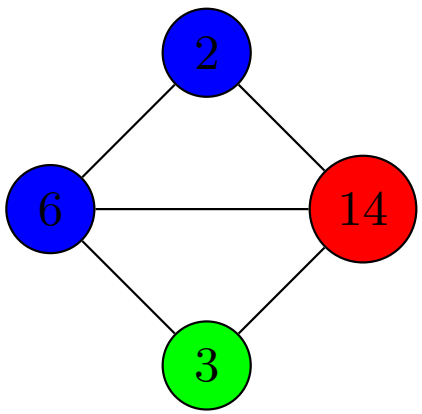


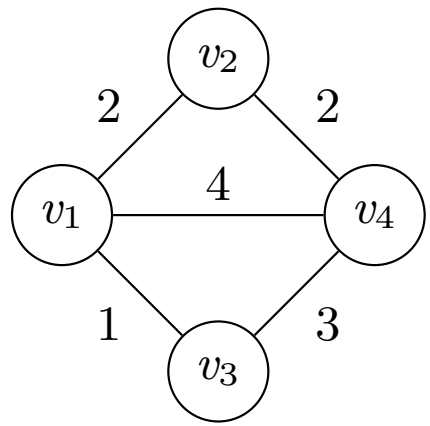


Types de graphes

Suivant le type de problème que l'on cherche à résoudre, on peut considérer des graphes

- non-orienté ou orienté.
- avec ou sans boucle
- simple ou avec des arêtes multiples
- colorés (sommets ou arêtes)
- valués (sommets ou arêtes)





Degrés, densité

Notation

On note $n(G) = |V|$ et $m(G) = |E|$, ou n et m quand il n'y a pas d'ambiguité.

Densité d'un graphe

La densité d'un graphe simple G est le nombre d'arêtes présentes divisé par le nombre d'arêtes possibles :

$$dens(G) = \frac{2m}{n(n-1)}$$

- La densité est toujours comprise entre 0 et 1. Une densité de 0 correspond à un graphe sans arête, une densité de 1 correspond à un graphe complet, également appelé clique.
- Cette notion peut être élargie à d'autres familles de graphes, il suffit de savoir calculer le nombre d'arêtes possibles. Si on considère les graphes orientés autorisant deux arêtes en sens contraire et des boucles, on obtient n^2 arêtes au maximum donc $d(G) = \frac{m}{n^2}$.

La densité est une caractéristique globale du graphe mais ne reflète pas sa structure locale.

Degrés, densité

Definition

Le degré d'un sommet v, noté $d(v)$, désigne son nombre de voisins.
Dans le cas des graphes dirigés, on distingue le degré sortant $d^+(v)$ et le degré entrant $d^-(v)$.
Dans le cas de graphe valué, le degré considéré est parfois la somme des poids des arêtes incidentes au sommet v.

Proposition

Un graphe non orienté vérifie

$$\sum_{v \in V(G)} d(v) = 2m.$$

Un graphe orienté vérifie

$$\sum_{v \in V(G)} d^+(v) = \sum_{v \in V(G)} d^-(v) = m.$$

Codage d'un graphe

Il y a principalement trois types de codages pour un graphe :

Matrice d'adjacence matrice carrée M de taille $n \times n$ tel que M_{uv} représente l'interaction entre u et v : 0 s'il n'y a pas d'arête, 1 si il y a une arête de u vers v.
Dans le cas non-dirigé, cette matrice est symétrique.
Dans le cas arête-valué, on remplace le coefficient 1 par le poids de l'arête concernée.

Liste d'arêtes matrice de taille $m \times 2$, chaque ligne représentant les deux sommets reliés par une arête.
Dans le cas orienté, l'ordre d'apparition des sommets sur la ligne indique le sens de l'arête.
Dans le cas arête-valué, on ajoute une troisième colonne contenant les poids.

Liste de voisinages Liste ayant un élément par sommet. Cet élément contient l'identité d'un sommet puis la liste de ses voisins (seulement les voisins externes dans les cas d'un graphe orienté).

Codage d'un graphe

	Stockage	u et v sont-ils voisins ?	degré d'un sommet v
Matrice d'adjacence	$O(n^2)$	$O(1)$	$O(n)$
Liste d'arête	$O(m)$	$O(m)$	$O(m)$
Liste d'adjacence	$O(m)$	$O(n)$	$O(n)$

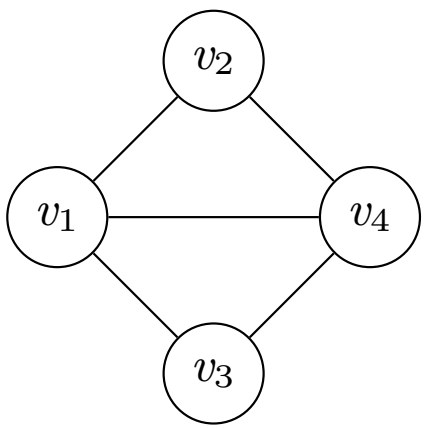
- Les deux dernières facons de stocker un graphe sont beaucoup plus efficaces, surtout dans le cas de graphes creuses ($m << n^2$).

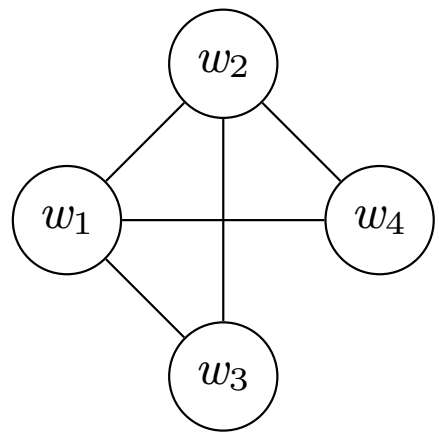
Isomorphisme de graphes

La représentation graphique d'un graphe n'est pas unique.

Isomorphisme de graphes

Deux graphes $G = (V(G), E(G))$ et $H = (V(H), E(H))$ sont isomorphes s'il existe une bijection $\phi : E(G) \to E(H)$ telle que $(u, v) \in E(G)$ si et seulement si $(\phi(u), \phi(v)) \in E(H)$.



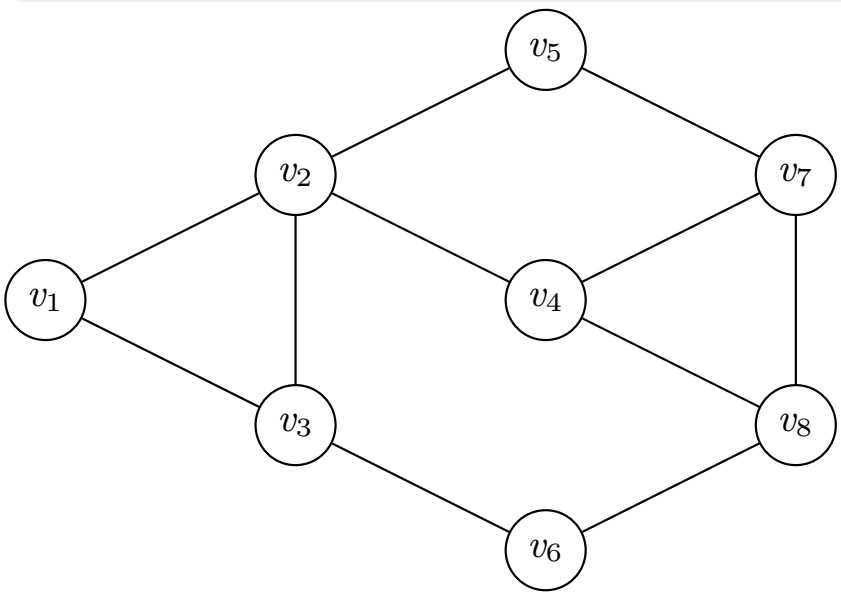


- Dans le cas des graphes valués, il faut que la bijection préserve aussi les poids.

Chemins et marches

Chemin et marche

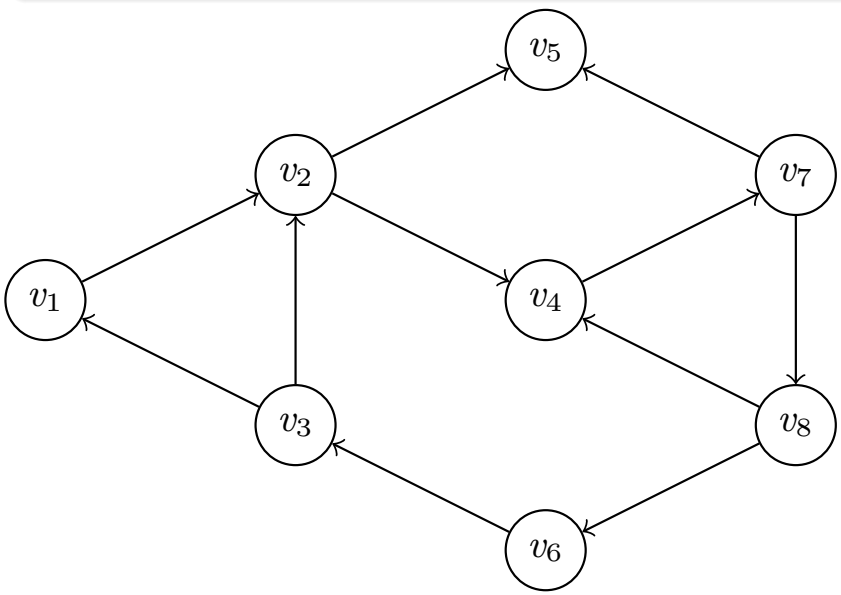
Un chemin de longueur k entre deux sommets u et v est une suite de k arêtes (u_i, u_{i+1}) tels que $u_0 = u$, $u_k = v$ et tous les u_i sont disjoints.
Dans le cas de graphes orientés, un chemin orienté nécessite l'orientation des arêtes dans le sens $\overrightarrow{u_i u_{i+1}}$.
Si les arêtes et les sommets ne sont pas tous disjoints, on parle de marche entre u et v.



- v_6, v_3, v_2, v_4 est un chemin de longueur 3.
- v_7, v_4, v_8, v_7, v_5 est une marche.

Chemin et marche

Un chemin de longueur k entre deux sommets u et v est une suite de k arêtes (u_i, u_{i+1}) tels que $u_0 = u$, $u_k = v$ et tous les u_i sont disjoints.
Dans le cas de graphes orientés, un chemin orienté nécessite l'orientation des arêtes dans le sens $\overrightarrow{u_i u_{i+1}}$.
Si les arêtes et les sommets ne sont pas tous disjoints, on parle de marche entre u et v.

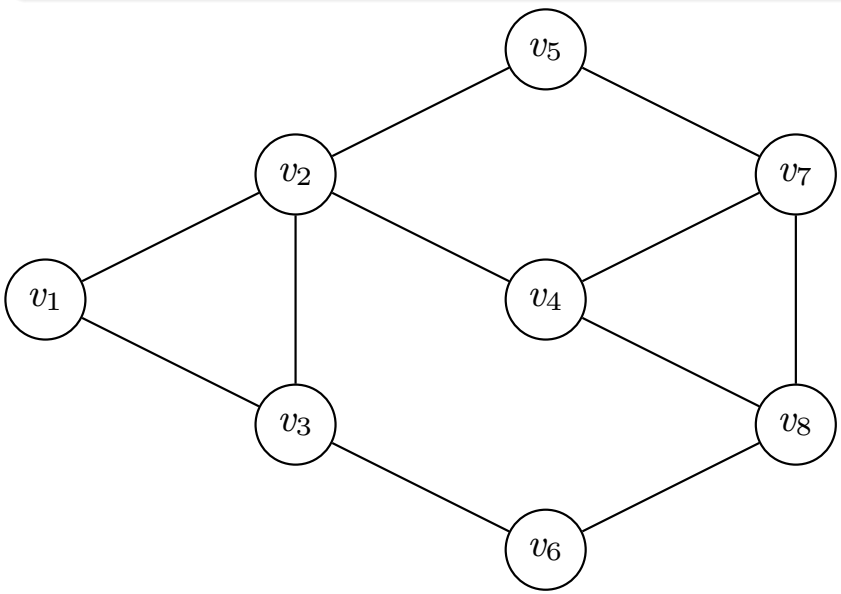


- v_6, v_3, v_2, v_4 est un chemin orienté.
- v_4, v_8, v_7, v_5 n'en est pas un.

Cycles

Definition

Un cycle de longueur k est une suite de k arêtes (u_i, u_{i+1}) tels que $u_0 = u_k$ et tous les u_i sont disjoints.
Dans le cas de graphes orientés, un cycle orienté nécessite l'orientation des arêtes dans le sens $\overrightarrow{u_i\, u_{i+1}}$.

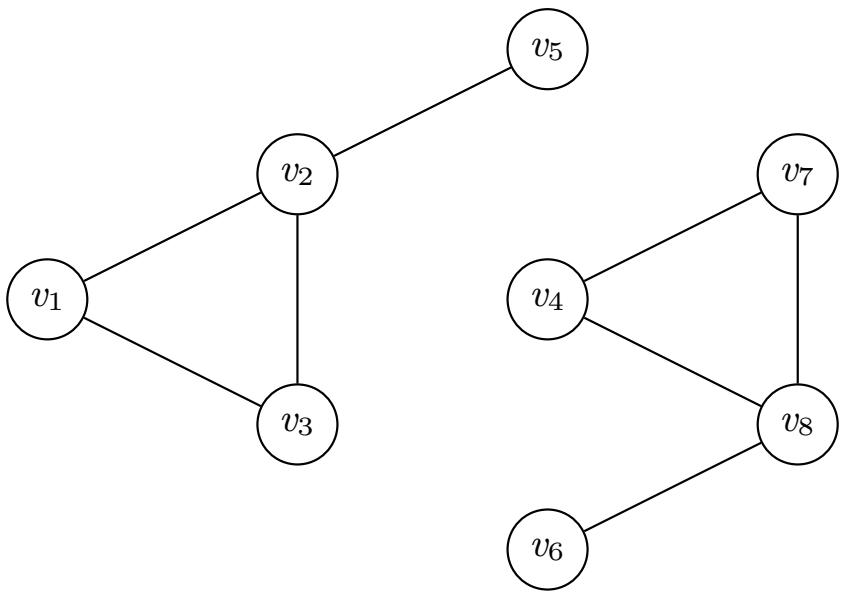


- $v_6, v_3, v_2, v_5, v_7, v_8$ est un cycle de longueur 6.

Graphe connexe

Connexité

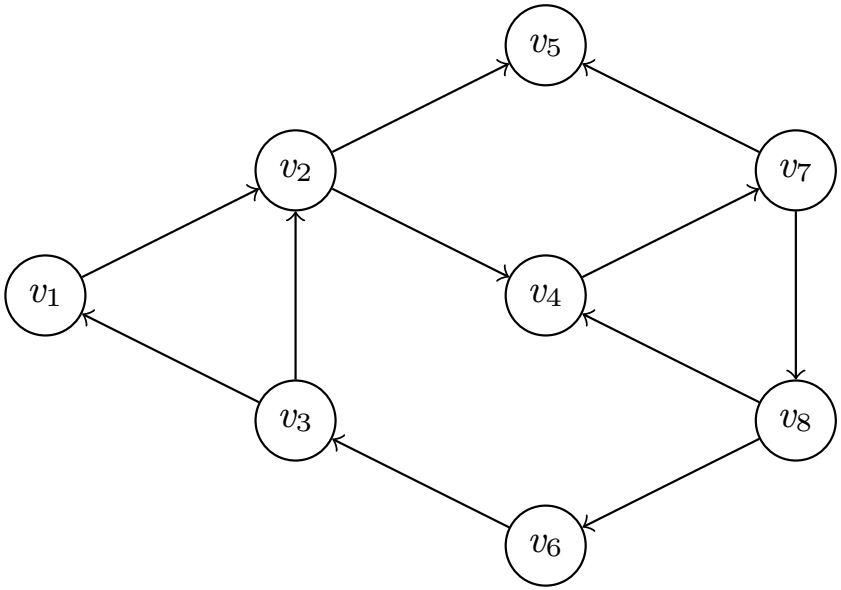
Un graphe non orienté est connexe si toute paire de sommets est reliée par un chemin.
Si le graphe n'est pas connexe, il peut être décomposé de façon unique en composantes connexes, qui sont les ensembles maximaux de sommets induisant des sous-graphes connexes.

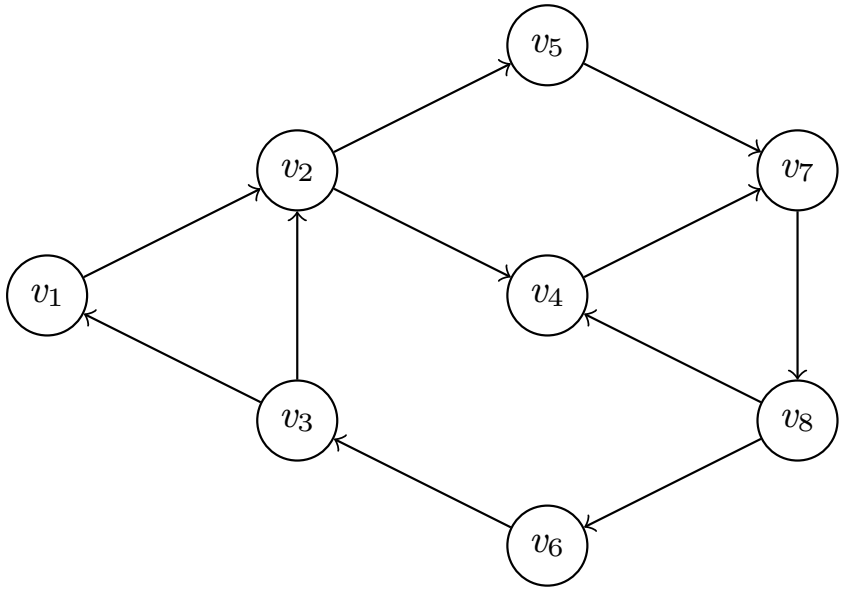


Graphe orienté fortement connexe

Definition

Un graphe orienté est fortement connexe si pour toute paire (u, v) de sommets, il existe un chemin orienté de u vers v et un chemin orienté de v vers u.
Il est dit simplement connexe si le graphe non-orienté sous-jacent est connexe.





Distance et diamètre

Definition

La distance entre deux sommets u et v appartenant à la même composante connexe d'un graphe arête-valué est le poids minimum d'un chemin entre u et v.

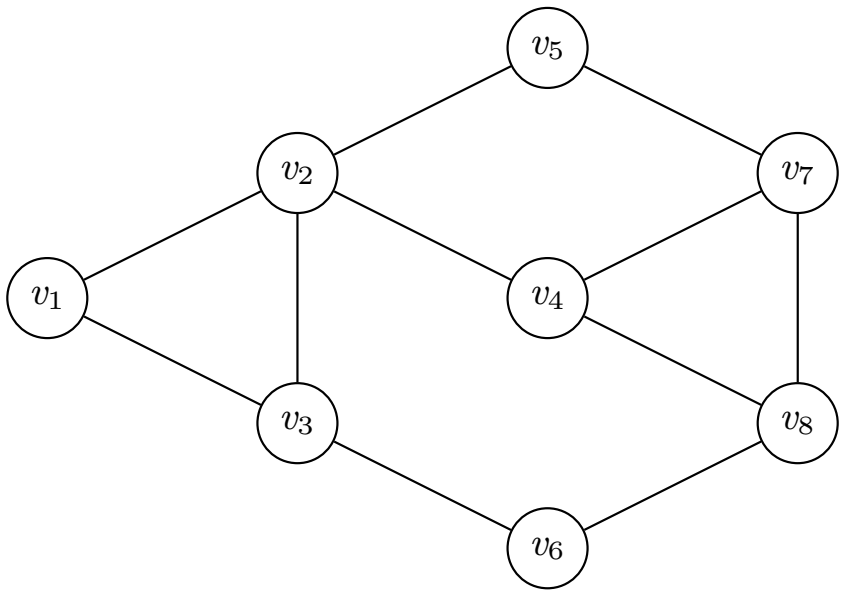
- Dans le cas de graphe non-valué, cela revient à considérer la longueur du plus court chemin.
- Dans le cas des graphes dirigés, il faut préciser si on se restreint aux chemins dirigés (auquel cas la distance n'est plus symétrique et donc plus une distance au sens mathématique !) ou si on travaille avec le graphe non-dirigé sous-jacent.
- Dans le cas de graphes non connexes, les sommets appartenant à des composantes différentes sont considérés comme ayant une distance infinie.

Distance et diamètre

Definition

Le diamètre d'un graphe est la plus grande distance existant entre deux sommets d'un graphe.

$$diam(G) = \max(d(u,v)|u,v \in V) = \min(d|\forall u,v \in V, d(u,v) \leq d)$$



- La distance de v_3 à v_5 est de 2.
- Quel est le diamètre de G ?

Pourquoi ce module

Problème

Un GPS doit trouver le chemin le plus court entre deux sommets dans un graphe valué à quelques millions de sommets (distance kilométrique, prix, ...).

Pourquoi ce module

Problème

Un GPS doit trouver le chemin le plus court entre deux sommets dans un graphe valué à quelques millions de sommets (distance kilométrique, prix, ...).

- Essayer tous les chemins est beaucoup trop long !
- Utiliser un algorithme glouton est faux (choisir à chaque carrefour la route la plus courte ne vous amènera pas au bon endroit de façon optimale).

Pourquoi ce module

Problème

Un GPS doit trouver le chemin le plus court entre deux sommets dans un graphe valué à quelques millions de sommets (distance kilométrique, prix, ...).

- Essayer tous les chemins est beaucoup trop long !
- Utiliser un algorithme glouton est faux (choisir à chaque carrefour la route la plus courte ne vous amènera pas au bon endroit de façon optimale).

Objectifs

- Trouver un algorithme intelligent.
- Déterminer sa complexité pour estimer à quel taille de graphe is sera applicable.
- Prouver qu'il est juste.

La preuve théorique est une partie essentielle. Elle ne saurait se résumer à un dessin ou un exemple !

II. Parcours de graphes : arbres couvrants

II.1 Arbres

Définition

Arbre

Un graphe non-orienté connexe et acyclique est appelé un arbre.

Proposition

Un graphe connexe est un arbre si et seulement si toute suppression d'arête le rend non-connexe.

En d'autres termes, un arbre est un graphe connexe minimal.

Arbre enraciné

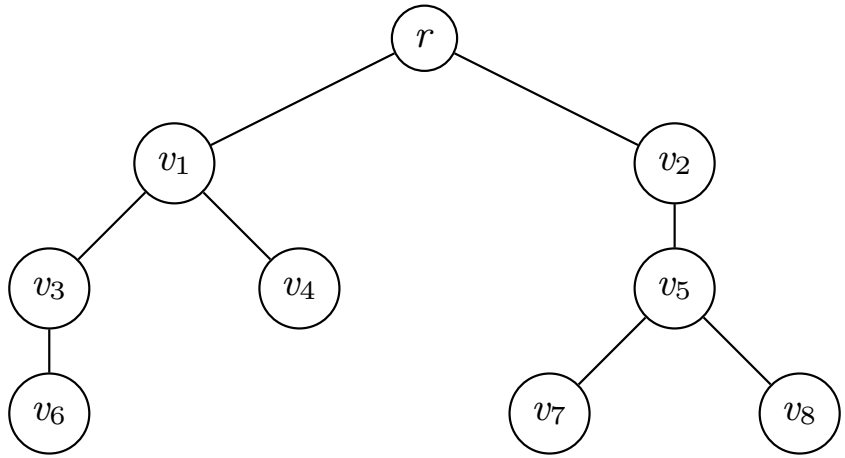
Définition

Considérons un entier n et un graphe G dont les sommets ont une étiquette appelée $niveau$ construit par le processus suivant :

- on initialise $V(G)$ à un sommet r de niveau 0, $E(G)$ à l'ensemble vide.
- pour i entre 1 et n,
  pour tout sommet v de niveau i,
  on ajoute à $V(G)$ un nombre fini de sommets $w_1, \ldots, w_{k_v}$ dont le niveau est $i+1$, et on ajoute à $E(G)$ les arêtes $\{(v, w_i), 1 \leq i \leq w_i\}$.

Le graphe G est un arbre enraciné. r est appelé racine de G et les sommets de niveau n sont appelés les feuilles de G.
Les sommets $(w_1, \ldots, w_{k_v}$ de niveau $i+1$ liés à un sommet v de degré i sont appelés les fils de v. v est le père de ses sommets. On en déduit la notion de descendants et d'ancêtres d'un sommet.

Arbre enraciné



Proposition

Tout arbre enraciné est un arbre. De plus, pour tout arbre T et tout sommet de $r \in V(T)$, T peut être construit comme un arbre enraciné de racine r.

Conséquence : Tout arbre a $n - 1$ arêtes.

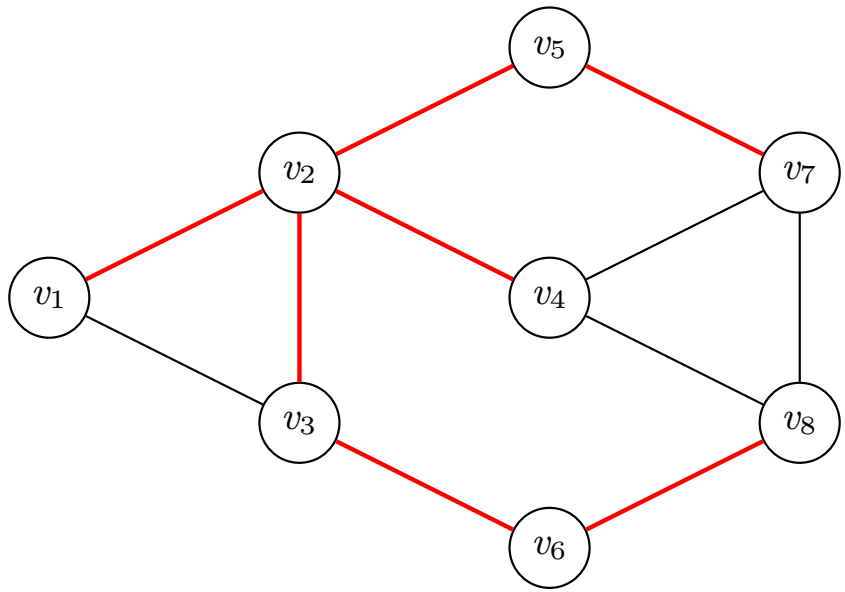
Intérêt des arbres

1. Structure adéquate pour certaines modélisations (évolution, pedigrees...). De nombreux problèmes ne pouvant pas être résolus en temps polynomial sur les graphes en général peuvent l'être en temps polynomial sur les arbres.
2. Structure naturelle pour résoudre des problèmes d'énumération.
   - Enumérer l'ensemble des mots de quatre lettres qu'on peut écrire avec ABC
   - Enumérer l'ensemble des mots de quatre lettres qu'on peut écrire avec ABC et contenant au moins 2 lettres A
3. Structure la moins lourde en termes d'arêtes pour encoder la connexité : n sommets reliés par un arbre forment un ensemble connexe, et on ne peut pas utiliser moins d'arêtes pour y arriver.

Arbre couvrant

Definition

Soit G un graphe. Un arbre couvrant de G est un sous-graphe T de G tel que T est un arbre et $V(T) = V(G)$.



Arbre couvrant

Théorème

Un graphe est connexe si et seulement si il admet un arbre couvrant.

Conséquences :

1. Un graphe connexe est un arbre si et seulement si il a $n - 1$ arêtes.
2. Décider si un graphe est connexe est équivalent à décider qu'il admet un arbre couvrant.

II.2 Parcours en largeur

Problème

Soit G un graphe.

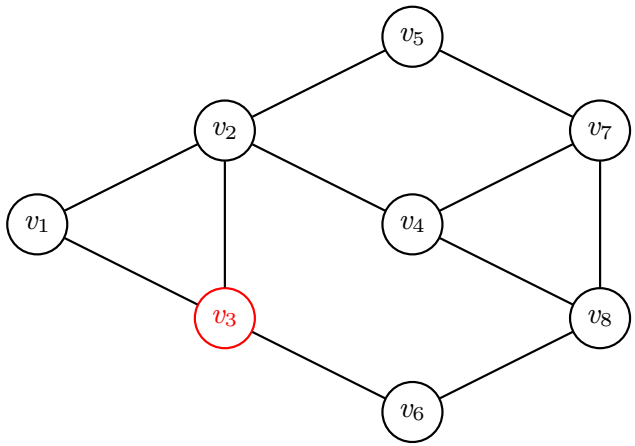
- Est-t il connexe ?
- Combien a-t-il de composantes connexes ?
- Lister tous les sommets dans la même composante connexe qu'un sommet d'intérêt.

Approche : Rechercher un (ou des) arbres couvrants, possiblement enracinés sur le sommet d'intérêt.

Arbre de parcours en largeur

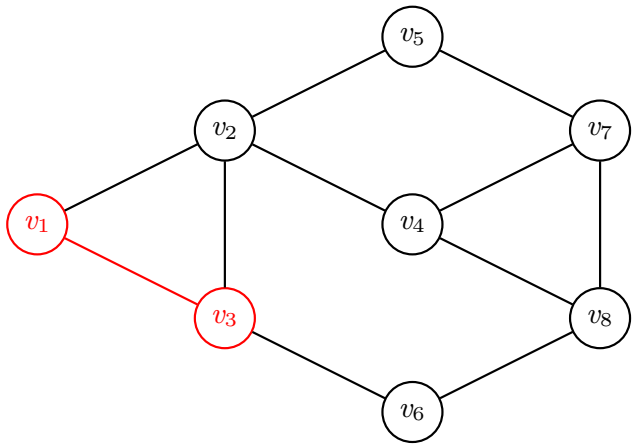
- Appelé BFS pour *Breadth-First Search*
- L'idée est de prendre un à un les sommets déjà visités et de nettoyer leur voisinage, en ajoutant tous leur voisins non-visités.
- D'un point de vue pratique, cela revient à utiliser une file, ou FIFO (First In, First Out), pour stocker les sommets visités.
- Le premier sommet de la file est le sommet courant, ses voisins sont ajoutés un à un en bout de file. Quand tous ses voisins sont visités, le sommet est supprimé de la file et ne la réintégrera plus.

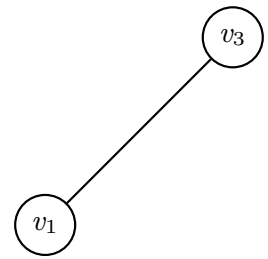
BFS : illustration



$$F = \{v_3\}$$

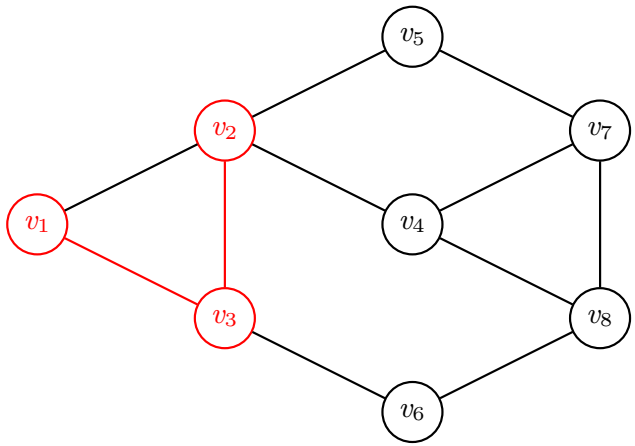
BFS : illustration

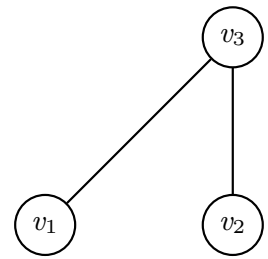




$F = \{v_3, v_1\}$

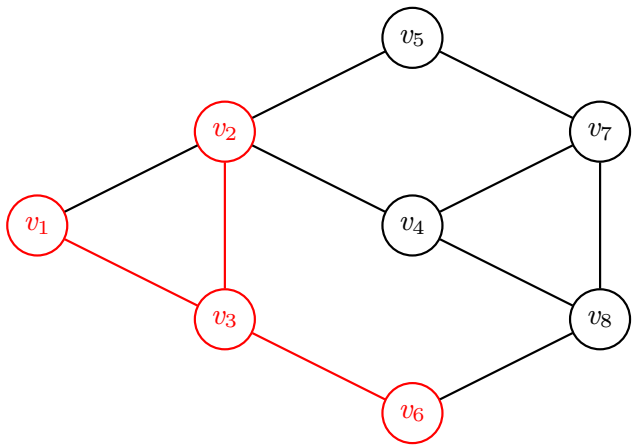
BFS : illustration

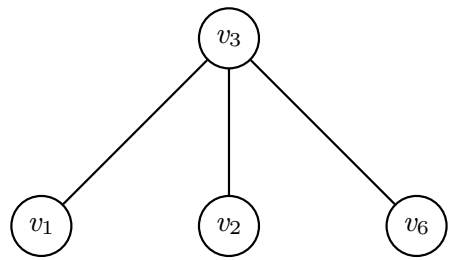




$$F = \{v_3, v_1, v_2\}$$

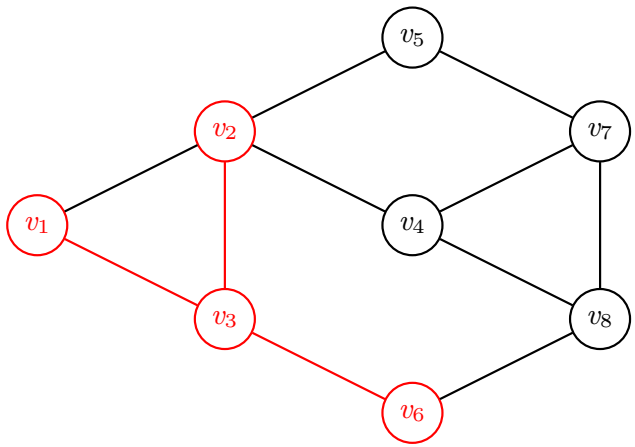
BFS : illustration

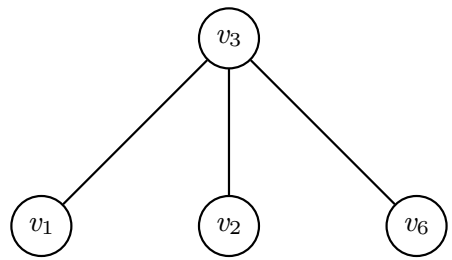




$F = \{v_3, v_1, v_2, v_6\}$
puis $F = \{v_1, v_2, v_6\}$

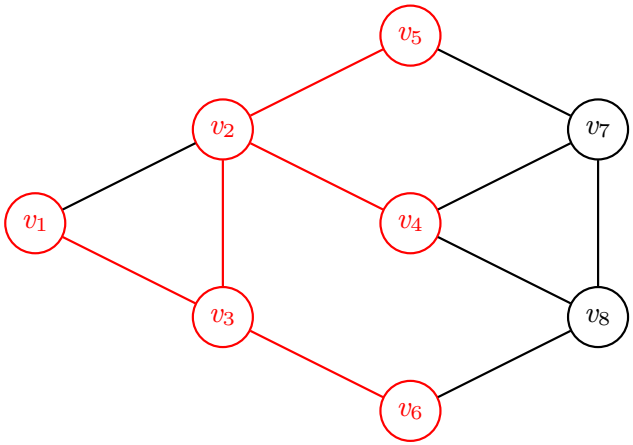
BFS : illustration

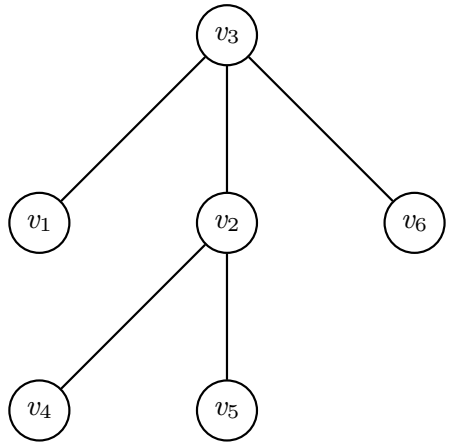




$F = \{v_1, v_2, v_6\}$
puis $F = \{v_2, v_6\}$.
Aucun sommet n'est ajouté à la file.

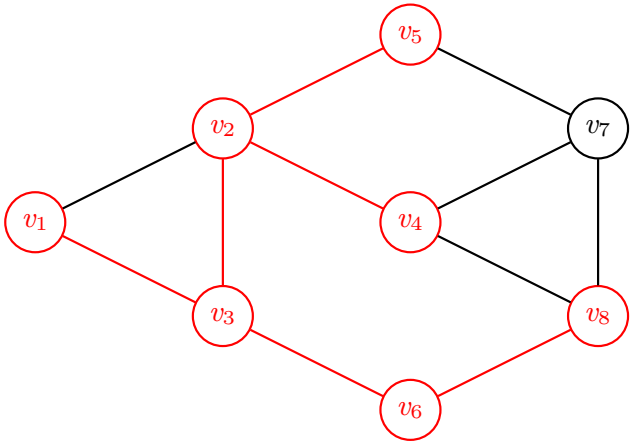
BFS : illustration

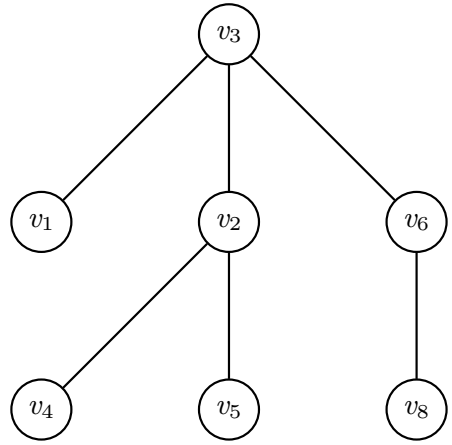




$F = \{v_2, v_6, v_4\}$
puis $F = \{v_2, v_6, v_4, v_5\}$
puis $F = \{v_6, v_4, v_5\}$

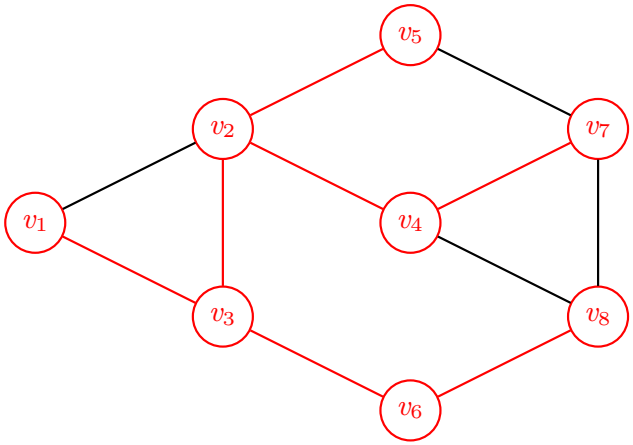
BFS : illustration

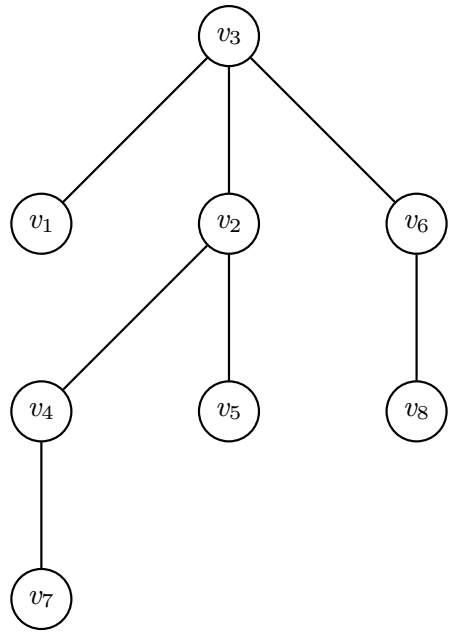




$F = \{v_6, v_4, v_5, v_8\}$
puis $F = \{v_4, v_5, v_8\}$

BFS : illustration





$F = \{v_4, v_5, v_8, v_7\}$
puis F se vide et l'algorithme s'arrête.

BFS : pseudo-code

Data: Un graphe G et un sommet v
Result: Un arbre T enraciné en v

$F = \emptyset$; $T = v$
for u *sommet de* G **do**
  u.visité=FALSE
end

while $F \neq \emptyset$ **do**
  $u =$ premier élément de F
  for w *voisin de* u **do**
    if w.*visité* =*FALSE* **then**
      w.visité=TRUE
      Ajouter w à F
      Ajouter w et (u, w) à T
    end
  end
  Supprimer u de F
end

Algorithm 1: Algorithme BFS

BFS

Proposition

BFS construit un arbre enraciné en v contenant tous les sommets de la composante connexe de v.

Proposition

BFS est de complexité $\mathcal{O}(m)$.

- Non-unicité du parcours suivant l'ordre dans lequel les voisins sont parcourus.
- En ajoutant une boucle choisissant un nouveau point de départ tant qu'il reste des sommets non-visités, on obtient un algortihme qui couvre tout graphe avec une forêt contenant autant d'arbres que le graphe a de composantes connexes.

BFS et distance

On considère un graphe G non valué.
Soit T un arbre BFS enraciné en v. Pour tout sommet u, on note $niv(u)$ le niveau de u dans T.

Proposition

Pour tout $(u, v) \in E(G)$, $|niv(u) - niv(v)| \leq 1$.

Proposition

Pour tout $u \in V(G)$, $niv(u) = d(u, v)$.

BFS orienté

On considère un graphe G orienté, et on applique BFS en n'ajoutant à chaque étape que les voisins extérieurs du sommet courant.

- Les sommets visités sont ceux qui peuvent être atteints depuis la racine par un chemin orienté.
- G ne contient aucune arête orientée d'un sommet u vers un sommet de niveau $\geq niv(u) + 2$.
- Le chemin du BFS de la racine vers tout sommet visité est un plus court chemin orienté.

II.3 Parcours en profondeur

Problème

Soit G un graphe.

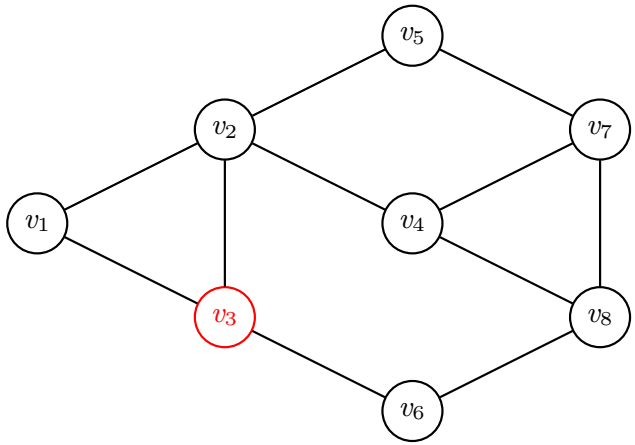
- Est-t il connexe ?
- Combien a-t-il de composantes connexes ?
- Lister tous les sommets dans la même composante connexe qu'un sommet d'intérêt.

Approche : Rechercher un (ou des) arbres couvrants, possiblement enracinés sur le sommet d'intérêt.

Arbre de parcours en profondeur

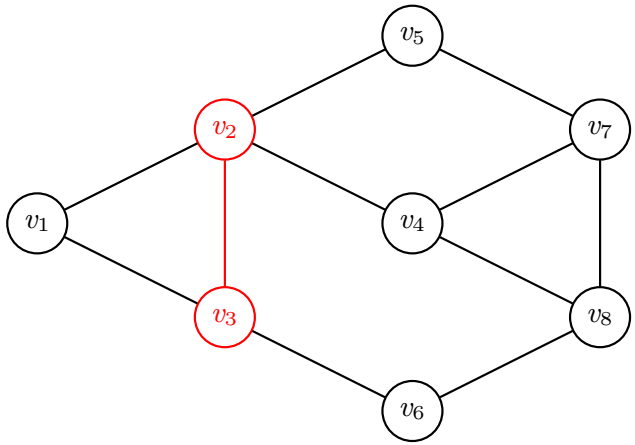
- Appelé DFS pour *Depth-First Search*
- L'idée est d'aller aussi loin que possible dans le graphe, et de rebrousser chemin quand on ne peut plus avancer.
- D'un point de vue pratique, cela revient à utiliser une pile, ou LIFO (Last In, First Out), pour stocker les sommets visités.
- Le sommet du haut de la pile est le sommet courant. S'il a un voisin non visité, celui-ci est ajouté sur la pile. S'il n'en a pas, le sommet courant est supprimé.
  Un sommet peut être le sommet courant à des moments distincts de l'algorithme mais lorsqu'il est supprimé de la pile, il ne la réintégrera plus.

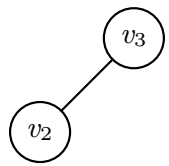
DFS : illustration



$$P = \{v_3\}$$

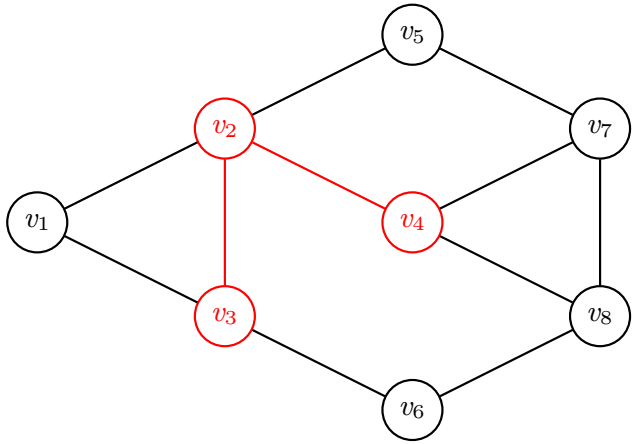
DFS : illustration

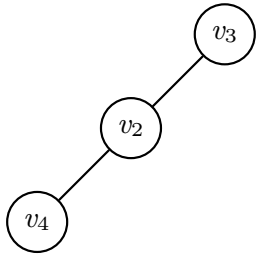




$$P = \{v_3, v_2\}$$

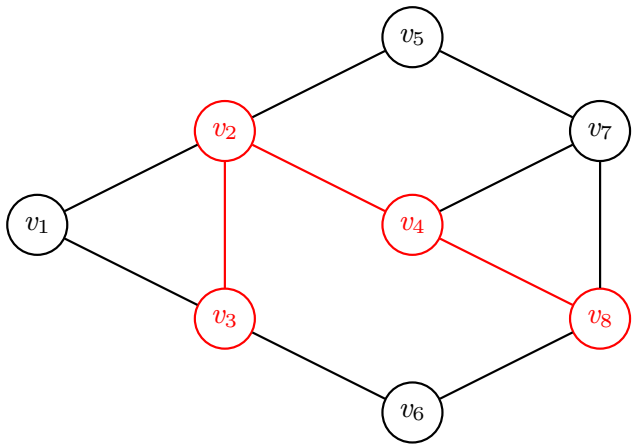
DFS : illustration

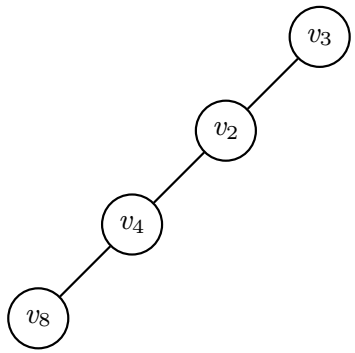




$$P = \{v_3, v_2, v_4\}$$

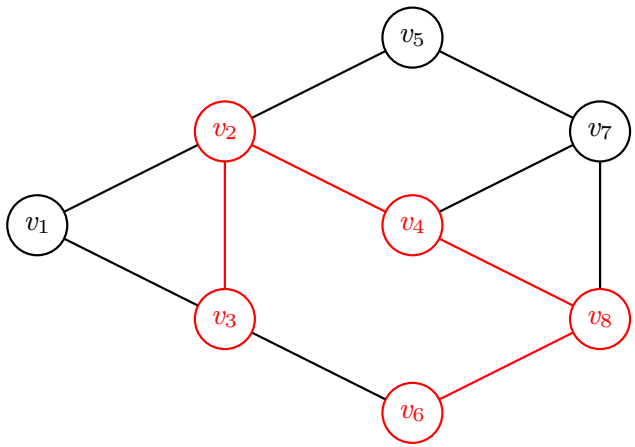
DFS : illustration

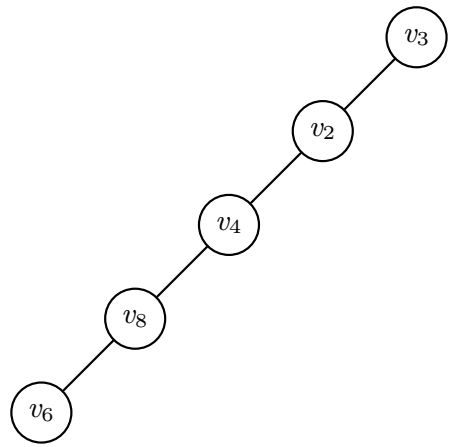




$$P = \{v_3, v_2, v_4, v_8\}$$

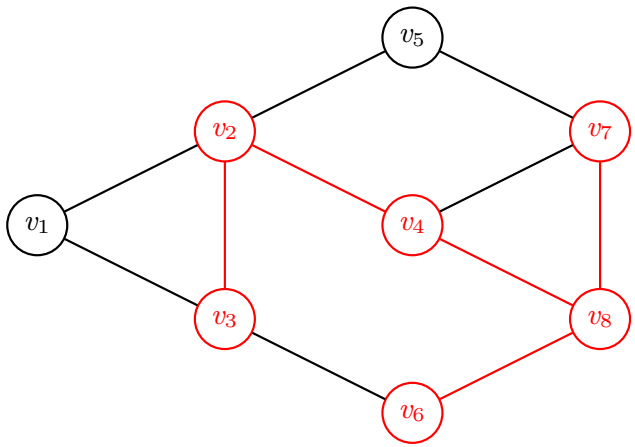
DFS : illustration

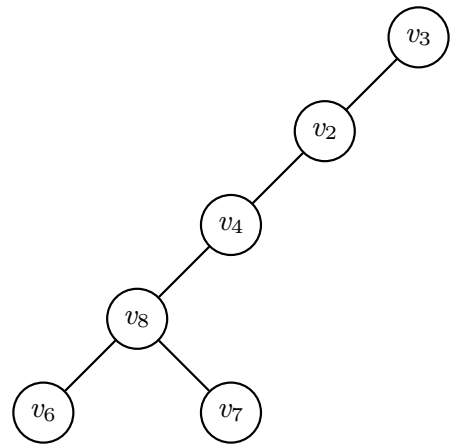




$P = \{v_3, v_2, v_4, v_8, v_6\}$
puis $P = \{v_3, v_2, v_4, v_8\}$

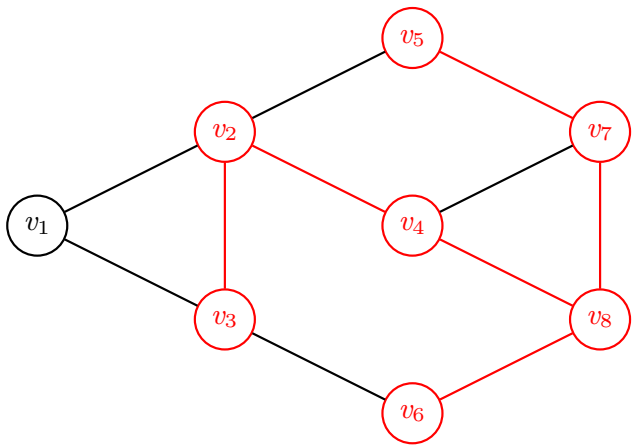
DFS : illustration

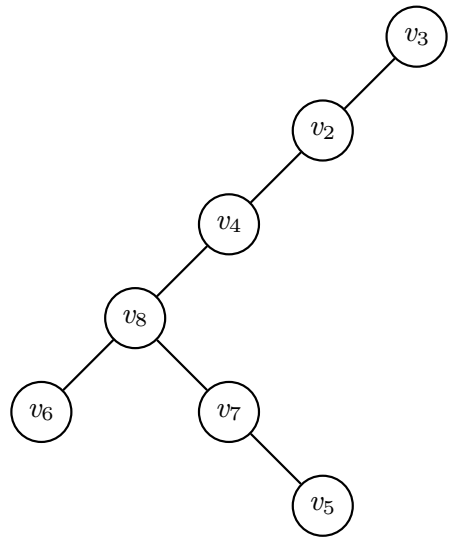




$$P = \{v_3, v_2, v_4, v_8, v_7\}$$

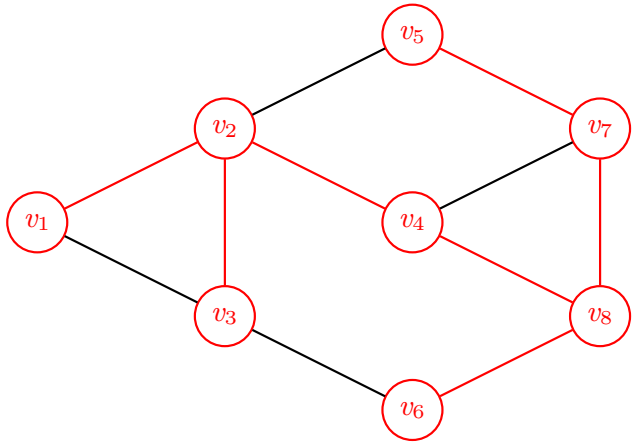
DFS : illustration

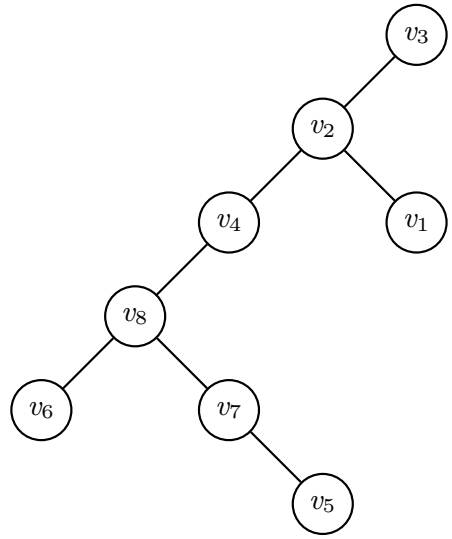




$P = \{v_3, v_2, v_4, v_8, v_7, v_5\}$
puis se dépile jusqu'à
$P = \{v_3, v_2\}$

DFS : illustration





$$P = \{v_3, v_2, v_1\}$$

puis se dépile jusqu'à être vide.

DFS : pseudo-code

Data: Un graphe G et un sommet v
Result: Un arbre T enraciné en v

$P = \emptyset$; $T = v$;
for u *sommet de* G **do**
  u.visité=FALSE
end
v.visité=TRUE

while $P \neq \emptyset$ **do**
  $u =$ dernier élément de P
  if *il existe* w *voisin de* u *avec* w*.visité=FALSE* **then**
    w.visité=TRUE
    Ajouter w à P
    Ajouter w et (u, w) à T
  end
  else
    Supprimer u de P ;
  end
end

Algorithm 2: Algorithme DFS

DFS : pseudo-code récursif

Data: Un graphe G et un sommet v
Result: Un arbre T enraciné en v

$P = \emptyset$; $T = v$
for u *sommet de* G **do**
  u.visité=FALSE
end

Function *DFS*(G,T,u)
  u.visité = TRUE
  for w *voisin de* u **do**
    if w.*visité=FALSE* **then**
      Ajouter w et (u, w) à T
      DFS(G,T,w)
    end
  end

DFS(G,T,v)

Algorithm 3: Algorithme DFS récursif

DFS

Proposition

DFS construit un arbre enraciné en v contenant tous les sommets de la composante connexe de v.

Proposition

DFS est de complexité $\mathcal{O}(m)$.

- Non-unicité du parcours suivant l'ordre dans lequel les voisins sont parcourus.
- En ajoutant une boucle choisissant un nouveau point de départ tant qu'il reste des sommets non-visités, on obtient un algorithme qui couvre tout graphe avec une forêt contenant autant d'arbres que le graphe a de composantes connexes.
- Toutes ces propriétés sont identiques avec BFS.

Topologie des DFS

Soit T un DFS sur un graphe G.

- Aucun rapport entre la distance à la racine et la distance dans G (ex : graphe réduit à un cycle).
- Par contre, aucune arête non découverte ne correspond à une arête transversale entre deux branches :

Proposition

Soit (u, v) une arête de G, u étant le premier sommet découvert par le DFS. Alors u est un ancêtre de v.

DFS orienté

On considère un graphe G orienté, et on applique DFS en n'ajoutant à chaque étape que les voisins extérieurs du sommet courant.

- Les sommets visités sont ceux qui peuvent être atteints depuis la racine par un chemin orienté.
- Si on trace les branches de T de la gauche vers la droite, il n'y a aucune arête dans G qui ajouterait à T une arête transversale de la gauche vers la droite.

II.4 Arbre couvrant de poids minimal

Problème

Soit G un graphe valué.
Trouver un arbre couvrant de poids minimal.

Applications

- directes : construire des réseaux (électriques, informatiques, ...) les moins chers possibles.
- indirectes : brique de base pour de nombreux autres algorithmes (par ex. approximation du voyageur de commerce).

Algorithme de Kruskal

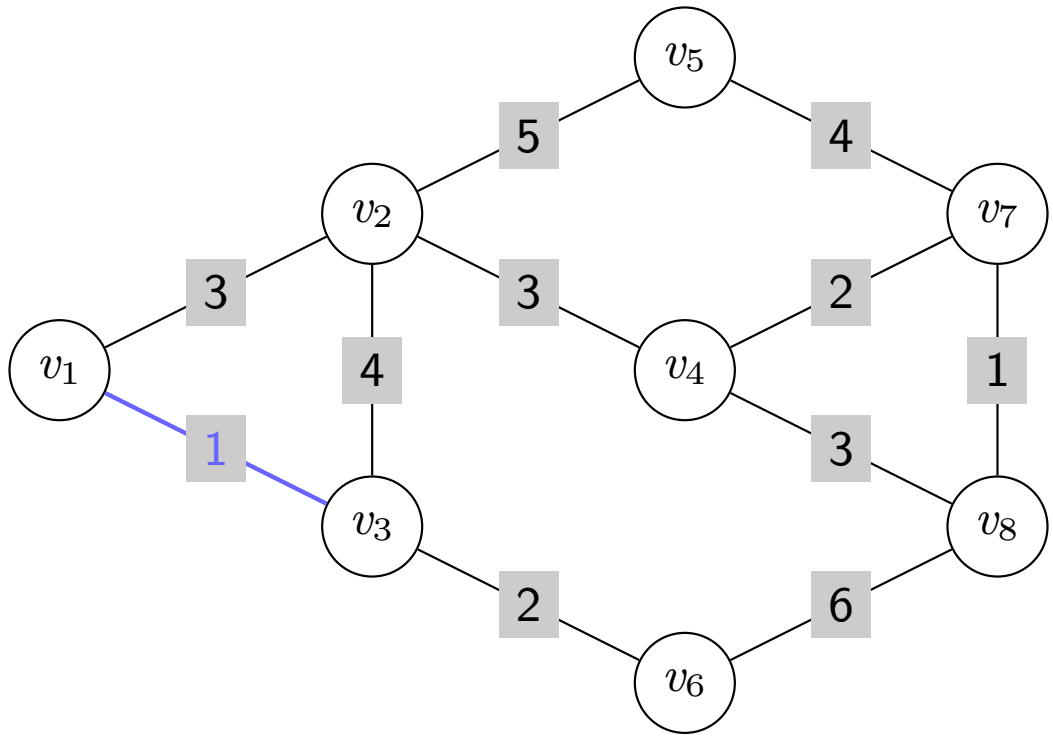
Data: Un graphe connexe valué G
Result: Un arbre couvrant T de poids minimal

Ranger les arêtes de G par poids croissant dans une liste L ;
F une forêt couvrante sans arêtes

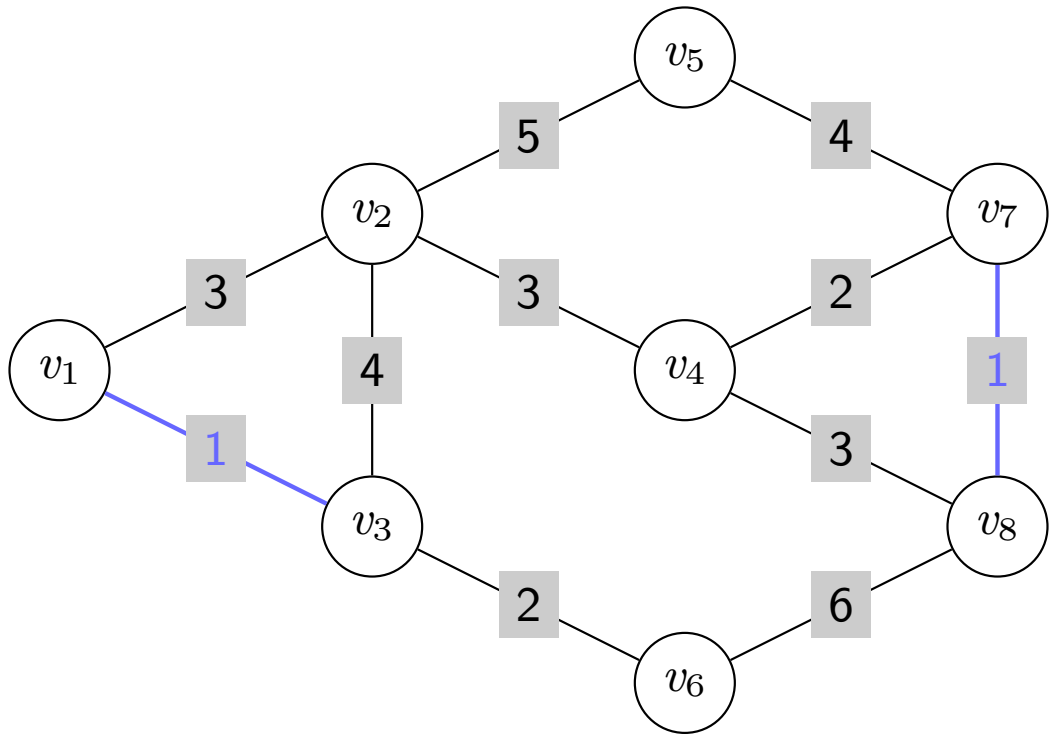
while *F n'est pas un arbre* **do**
    $e =$ première arête de L
    if *les extremités de e ne sont pas dans le même arbre de F* **then**
        $F = F + e$
    end
    Supprimer e de L
end

Algorithm 4: Algorithme de Kruskal

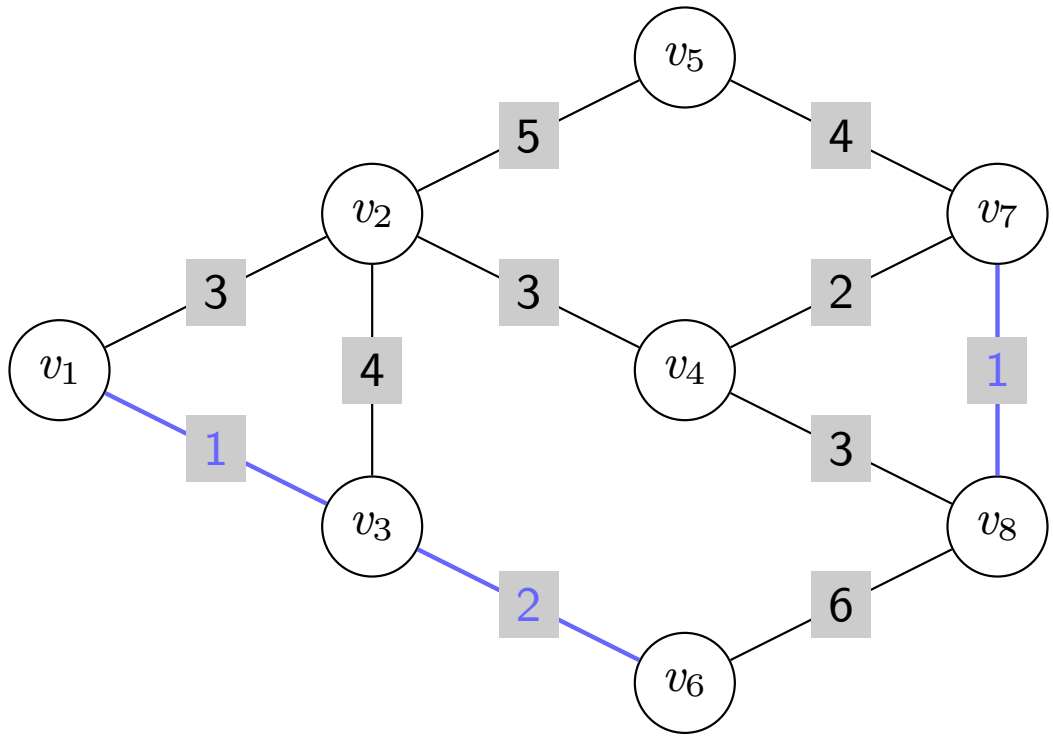
Algorithme de Kruskal : exemple



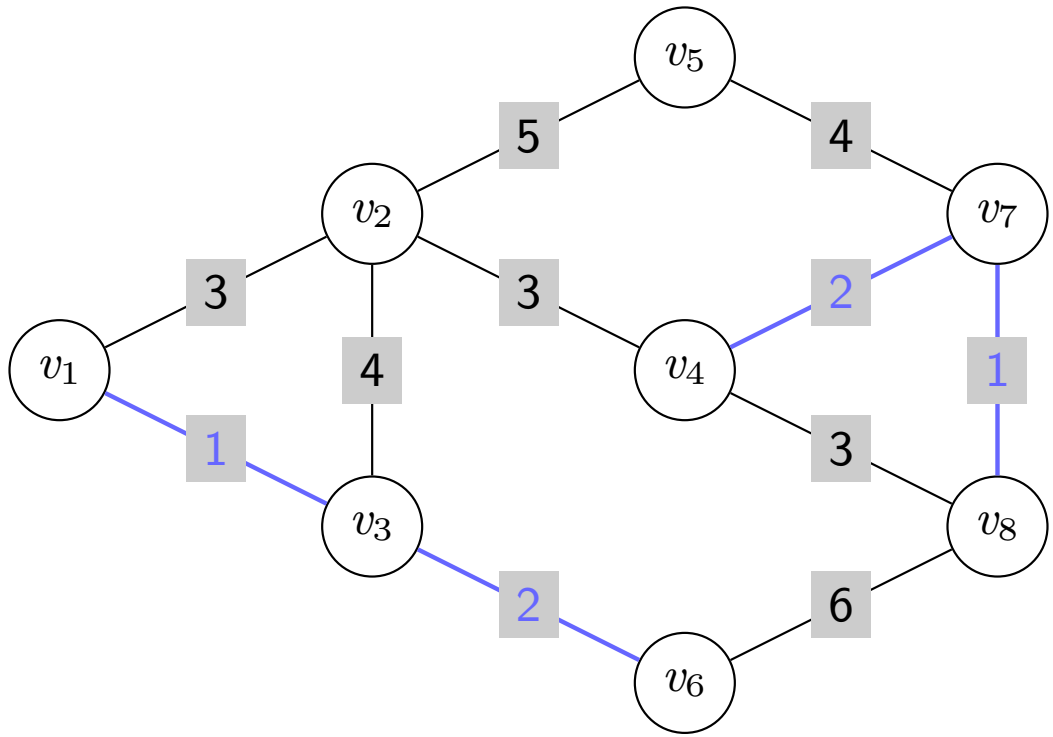
Algorithme de Kruskal : exemple



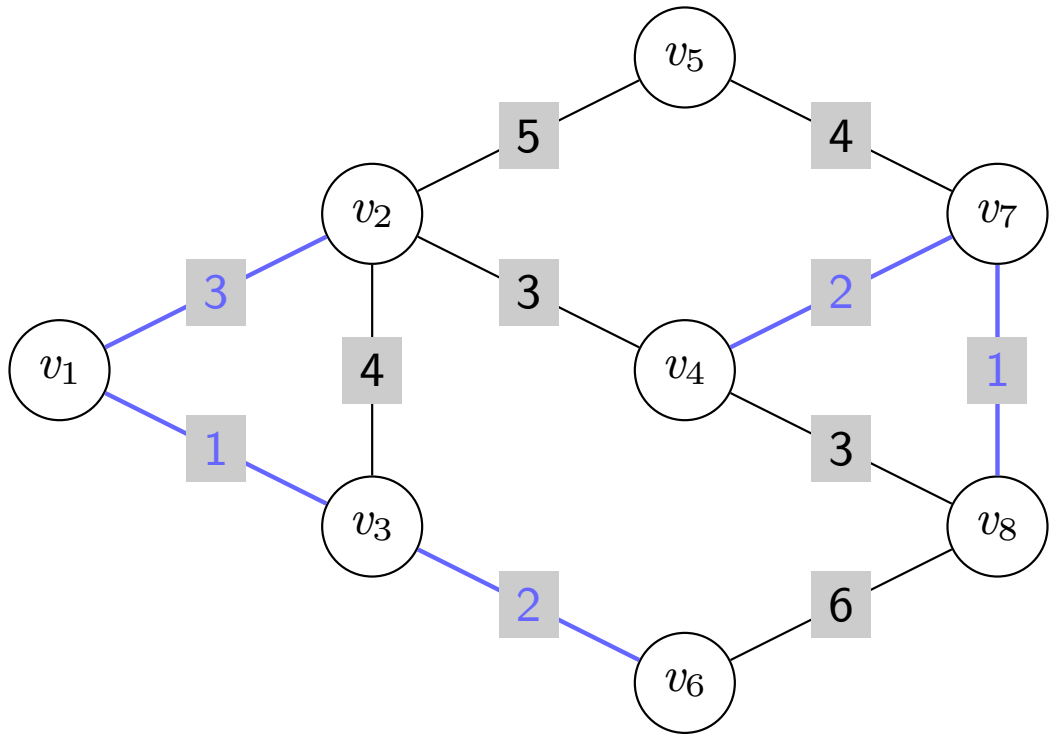
Algorithme de Kruskal : exemple



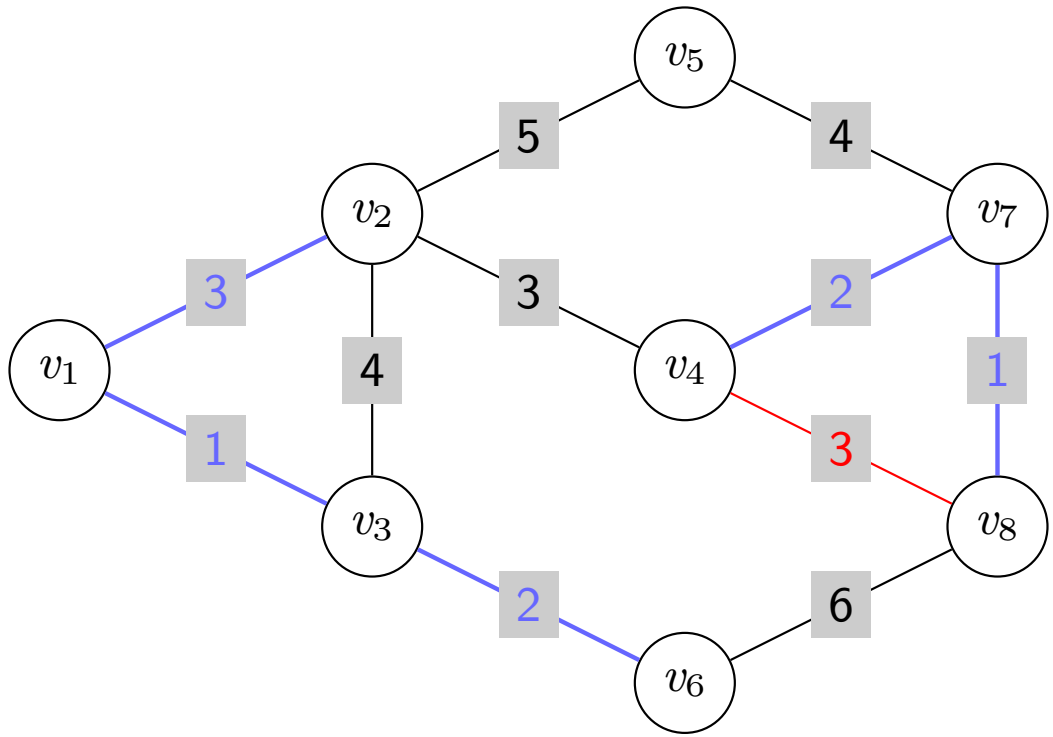
Algorithme de Kruskal : exemple



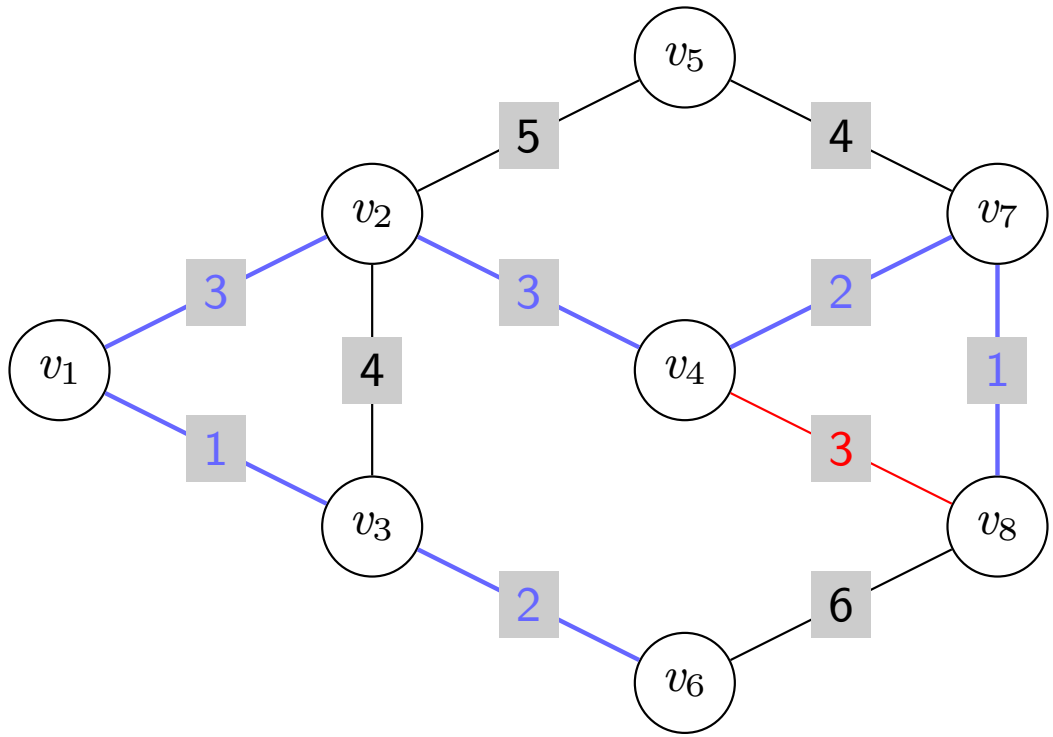
Algorithme de Kruskal : exemple



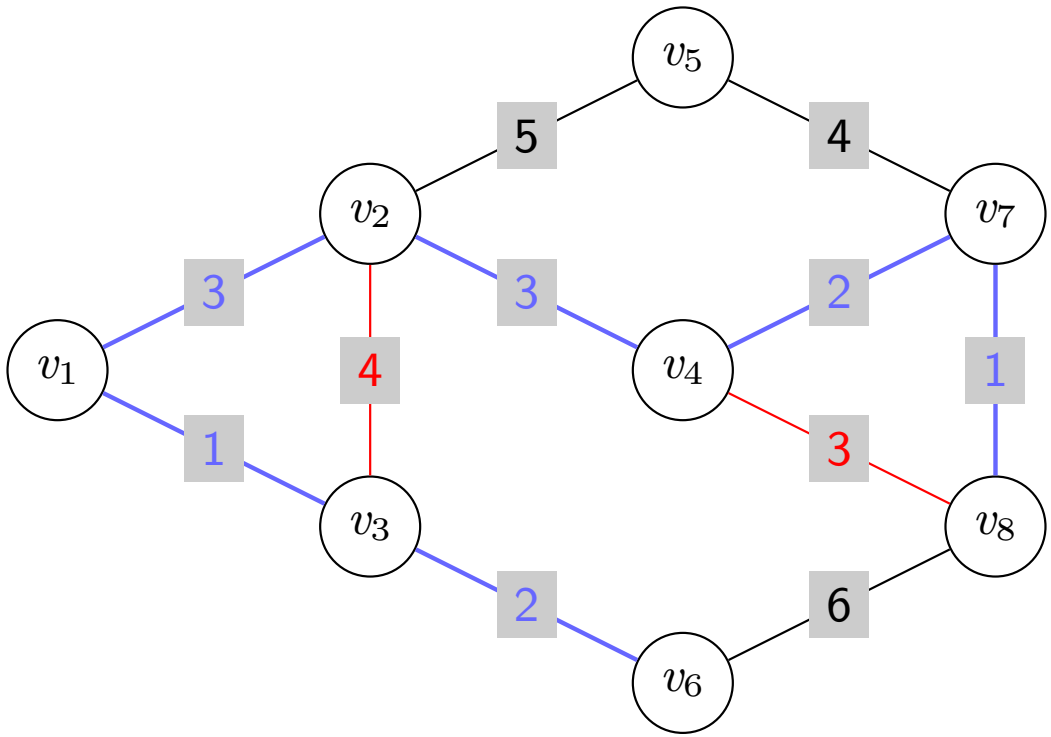
Algorithme de Kruskal : exemple



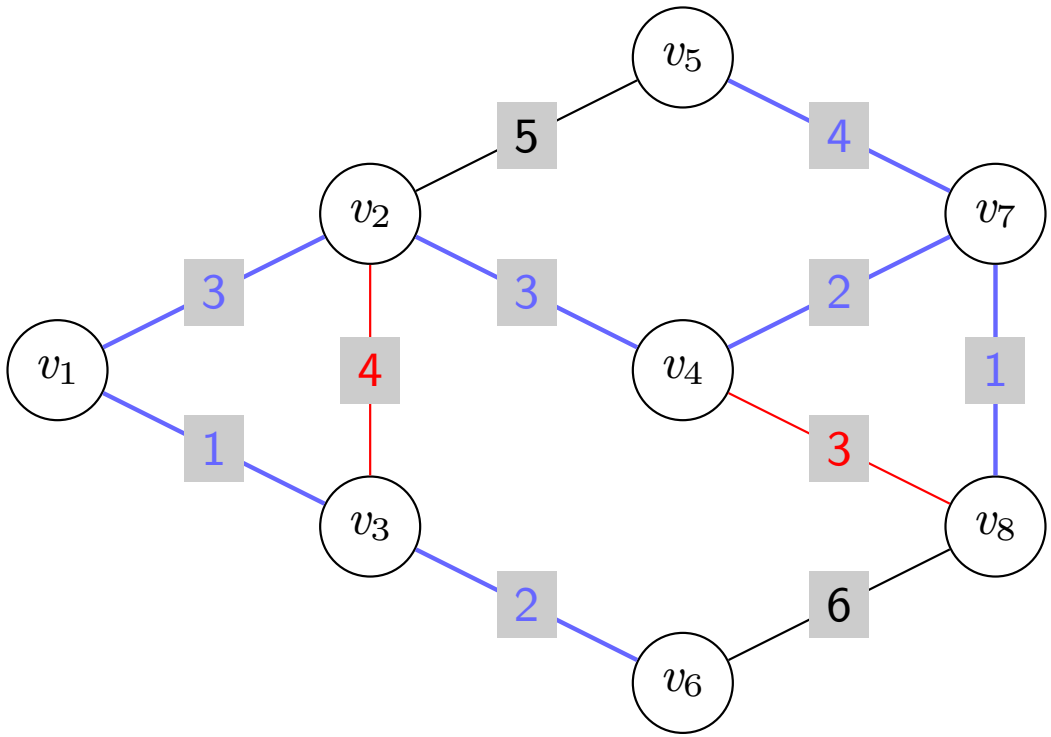
Algorithme de Kruskal : exemple



Algorithme de Kruskal : exemple



Algorithme de Kruskal : exemple



Algorithme de Kruskal

Proposition

L'algorithme de Kruskal renvoie bien, pour un graphe connexe, un arbre couvrant de poids minimal.

Proposition

Sa complexité est de $\mathcal{O}(m \log m)$.

- Si G n'est pas connexe, le résultat est une forêt dont chaque arbre est un arbre couvrant de poids minimal de l'une des composantes connexes de G.
- En cas d'existence d'arêtes de poids égal, la solution peut ne pas être unique.
- La manière de coder les composantes connexes est primordiale pour obtenir une complexité optimale.
- L'opération limitante du point de vue de la complexité est le tri.

Complexité de Kruskal : Union-Find

- Trier m arêtes se fait en $\mathcal{O}(m \log m)$
- Pour chaque arête (x, y), il faut
  - trouver la composante connexe de x et y : fonction $FIND$
  - comparer $FIND(x)$ et $FIND(y)$
  - si ils sont différents, fusionner les deux composantes connexes : opération $UNION$

Soit $\alpha(n, m)$ la somme des complexités de $FIND$ et $UNION$. La complexité de l'algorithme de Kruskal est alors $\mathcal{O}(m \log m + m\alpha(n, m))$.

Complexité de Kruskal : Union-Find

Une solution simple mais non-optimale

On stocke pour chaque sommet un nombre correspondant à sa composante connexe.

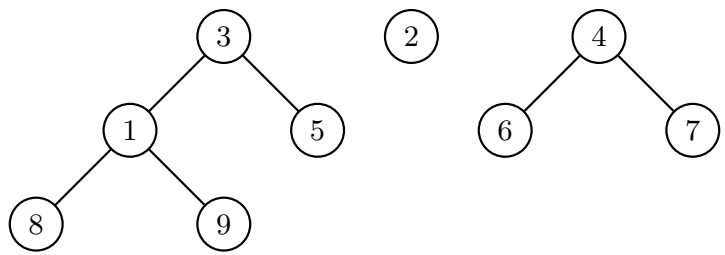
sommet	1	2	3	4	5	6	7	8	9
composante	1	2	1	3	1	3	3	1	1

$FIND$ est en temps constant, $UNION$ en $\mathcal{O}(n)$: $\alpha = \mathcal{O}(n)$.

Complexité de Kruskal : Union-Find

Autre solution : chaque composante connexe est représentée par un arbre enraciné

- il suffit de stocker son père pour chaque sommet, les racines étant leur propre père. Les racines servent de représentant de la classe.

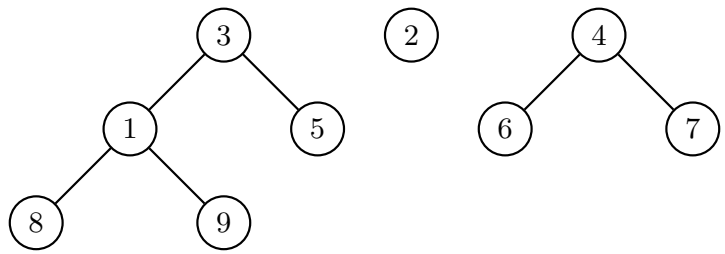


sommet	1	2	3	4	5	6	7	8	9
père	3	2	3	4	3	4	4	1	1

Complexité de Kruskal : Union-Find

Autre solution : chaque composante connexe est représentée par un arbre enraciné

- $FIND(x)$ revient à remonter l'arbre de père en père jusqu'à faire du sur-place. On est alors à la racine, et on renvoit la valeur de celle-ci : complexité $\mathcal{O}(h)$, où h est la hauteur de l'arbre.



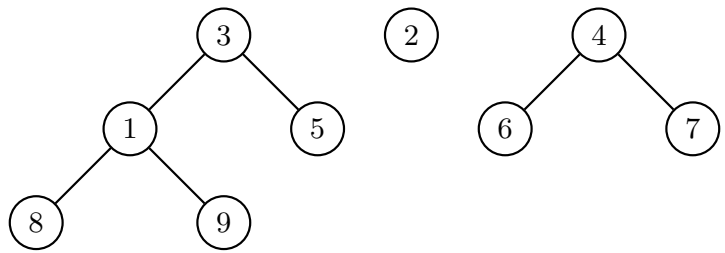
sommet	1	2	3	4	5	6	7	8	9
père	3	2	3	4	3	4	4	1	1

$FIND(8) = 3$

Complexité de Kruskal : Union-Find

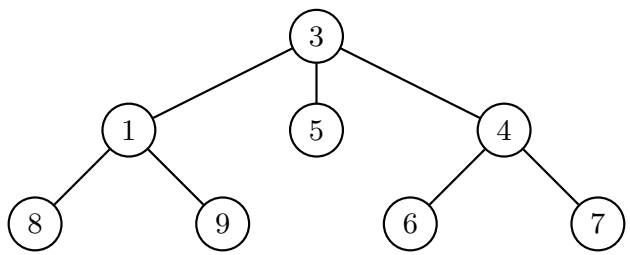
Autre solution : chaque composante connexe est représentée par un arbre enraciné

- $UNION$ consiste à rassembler les deux arbres en un seul. Pour cela, on déclare comme père de la racine du plus petit arbre la racine du plus grand : complexité en temps constant.



$UNION(1, 7)$

sommet	1	2	3	4	5	6	7	8	9
père	3	2	3	3	3	4	4	1	1



Complexité de Kruskal : Union-Find

Proposition

La hauteur d'un arbre avec s sommets, construit par une suite d'applications de $UNION$, est majoré par $1 + \log_2 s$.

On en déduit que $\alpha = \mathcal{O}(\log m)$ et donc que l'algorithme de Kruskal est de complexité $\mathcal{O}(m \log m)$.

Complexité de Kruskal : Union-Find

Proposition

La hauteur d'un arbre avec s sommets, construit par une suite d'applications de $UNION$, est majoré par $1 + \log_2 s$.

On en déduit que $\alpha = \mathcal{O}(\log m)$ et donc que l'algorithme de Kruskal est de complexité $\mathcal{O}(m \log m)$.

Complexité amortie

- Quand on remonte dans l'arbre vers la racine r, on fait de chaque sommet parcouru un fils de r.
- Opérer r $FIND$ et s $UNION$ se fait en $\prime(r + sf(r, s))$ où f est une fonction qui croît très lentement ($f(n, m) \leq 4$ pour $n, m \leq 2^{2048}$).
- La sélection d'arêtes peut donc être considérée comme de complexité $\mathcal{O}(m)$, ce qui permet d'obtenir une meilleure complexité si les poids des arêtes sont telles que le tri peut se faire plus rapidement.

Algorithme de Prim

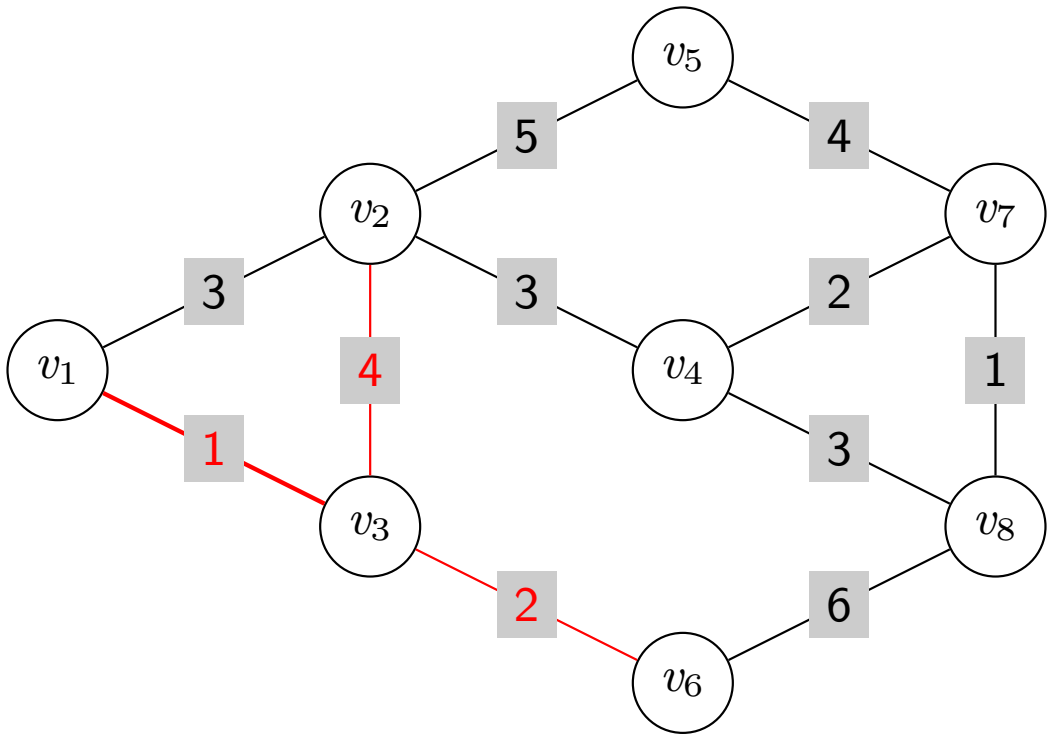
Data: Un graphe connexe valué G et un sommet v
Result: Un arbre couvrant T de poids minimal

$T = \{v\}$

while *il existe des arêtes de* T *vers* $G \setminus T$ **do**
    Trouver l'arête e de poids minimal de T vers $G \setminus T$
    $T = T + e$
end

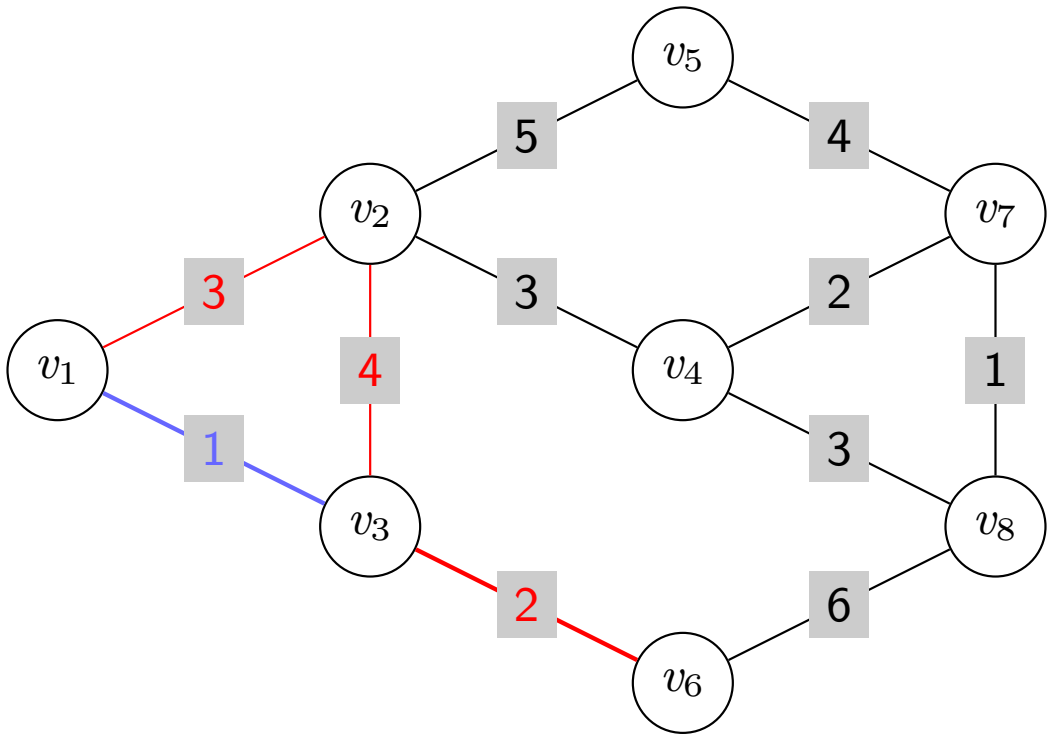
Algorithm 5: Algorithme de Prim

Algorithme de Prim : exemple



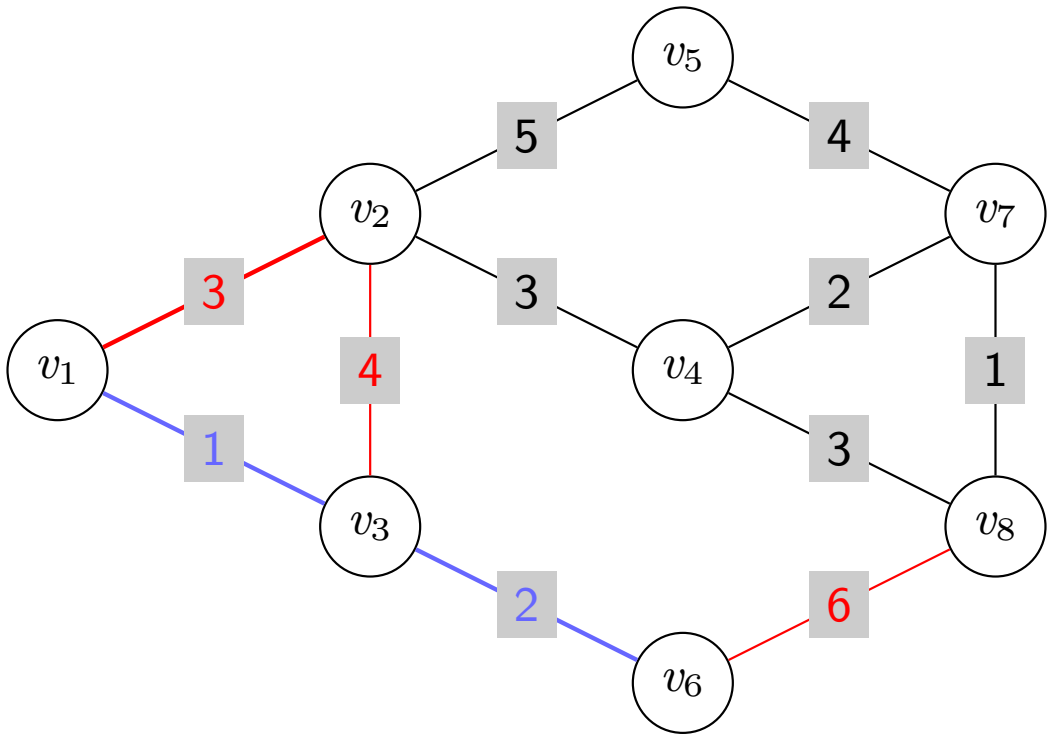
$V(T) = \{v_3\}$

Algorithme de Prim : exemple



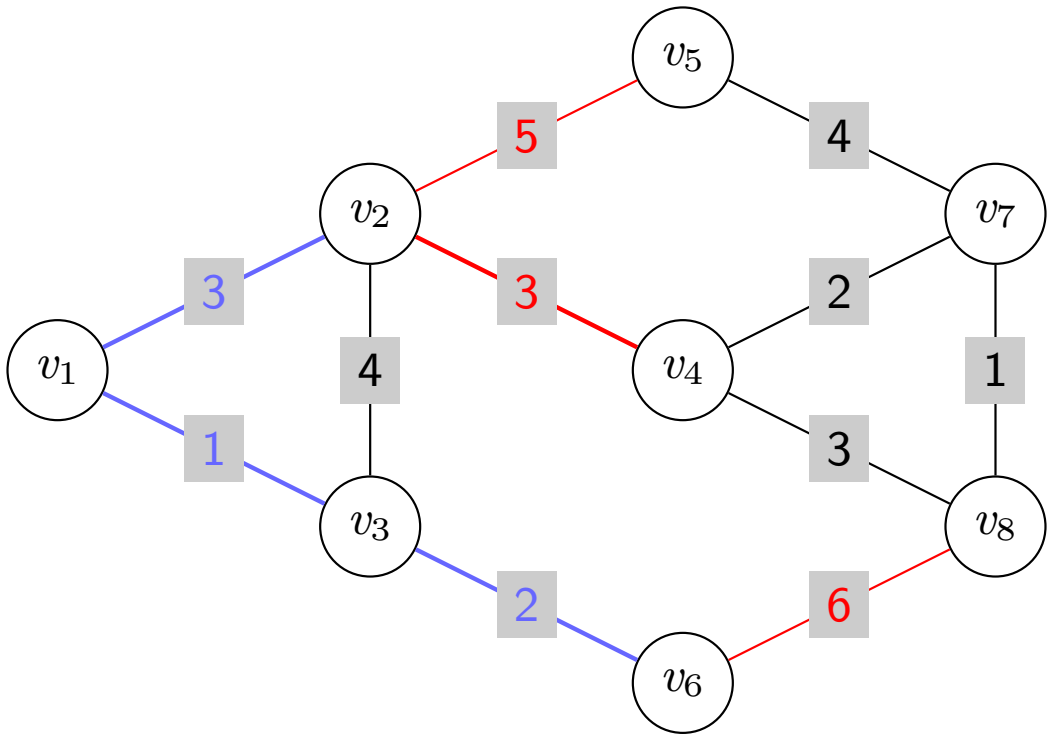
$V(T) = \{v_3, v_1\}$

Algorithme de Prim : exemple



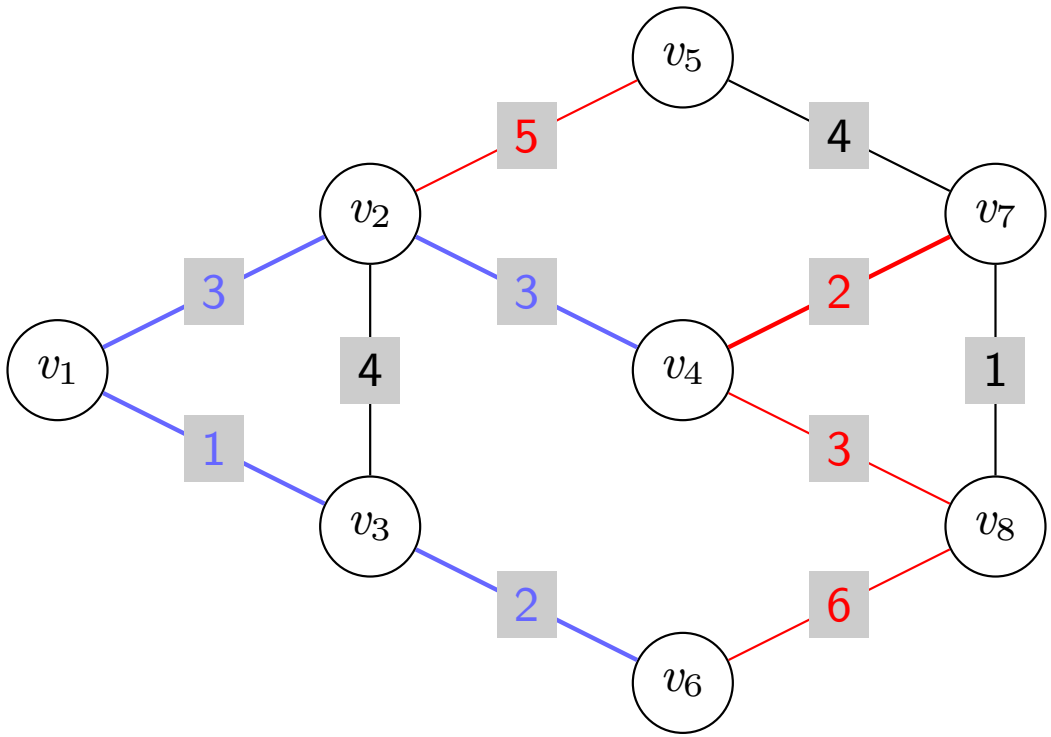
$V(T) = \{v_3, v_1, v_6\}$

Algorithme de Prim : exemple



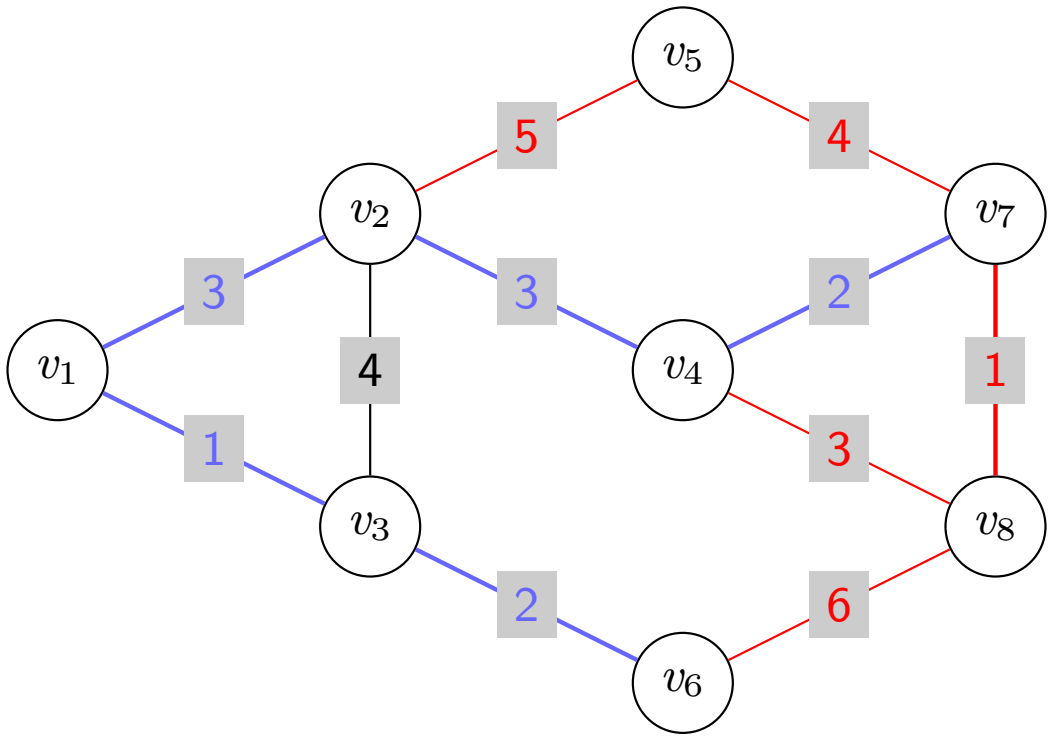
$V(T) = \{v_3, v_1, v_6, v_2\}$

Algorithme de Prim : exemple



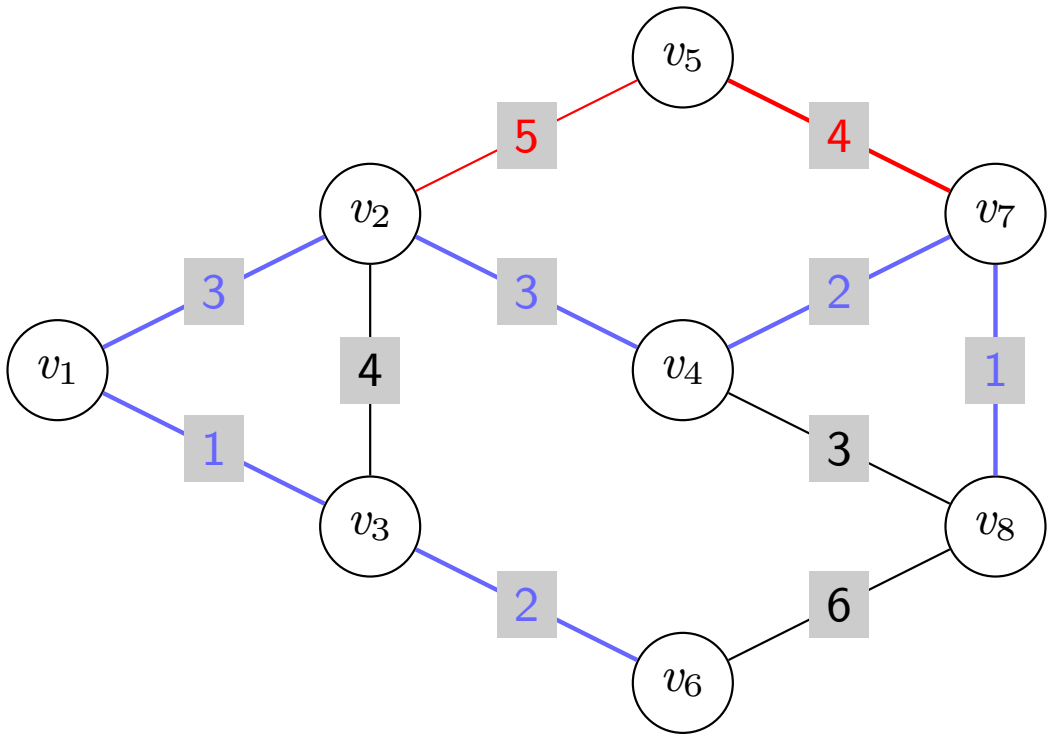
$V(T) = \{v_3, v_1, v_6, v_2, v_4\}$

Algorithme de Prim : exemple



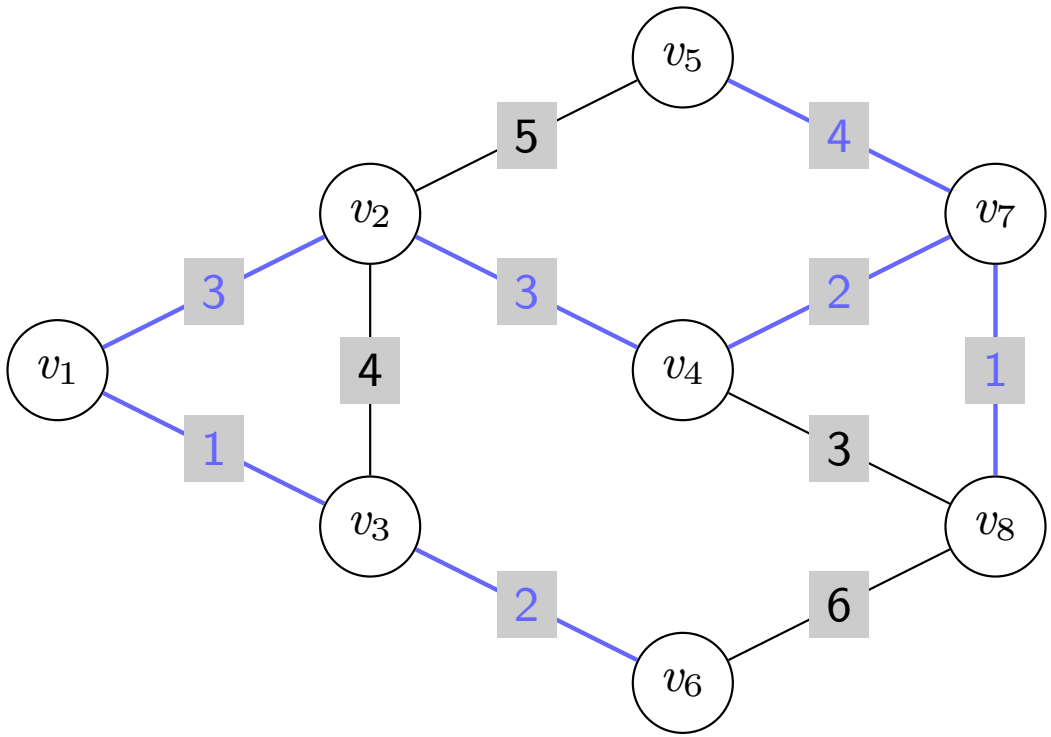
$V(T) = \{v_3, v_1, v_6, v_2, v_4, v_7\}$

Algorithme de Prim : exemple



$V(T) = \{v_3, v_1, v_6, v_2, v_4, v_7, v_8\}$

Algorithme de Prim : exemple



$V(T) = \{v_3, v_1, v_6, v_2, v_4, v_7, v_8, v_5\}$

Algorithme de Prim

Proposition

L'algorithme de Prim renvoie bien, pour un graphe connexe, un arbre couvrant de poids minimal.

Proposition

Sa complexité est de $\mathcal{O}(mn)$ pour un code naïf.
Elle peut être abaissée en $\mathcal{O}(m \log n)$ en codant à l'aide de tas binaires et en $\mathcal{O}(m + n \log n)$ à l'aide de tas de Fibonacci.

- Si G n'est pas connexe, le résultat est un arbre couvrant de poids minimal de la composante connexe de G. Le relancer tant qu'il existe des sommets non couverts permet de trouver une forêt couvrante de poids minimal.
- En cas d'existence d'arêtes de poids égal, la solution peut ne pas être unique.
- La complexité théorique (pire cas) est légèrement meilleure que celle de Kruskal au prix d'une implémentation minutieuse.
- Il ne nécessite pas l'exploration à priori de tout le graphe.

II.5 Algorithme de Dijkstra

Problème

On considère un graphe G valué et deux sommets u et v de G. Trouver le plus court chemin (au sens de la somme des poids) entre u et v.

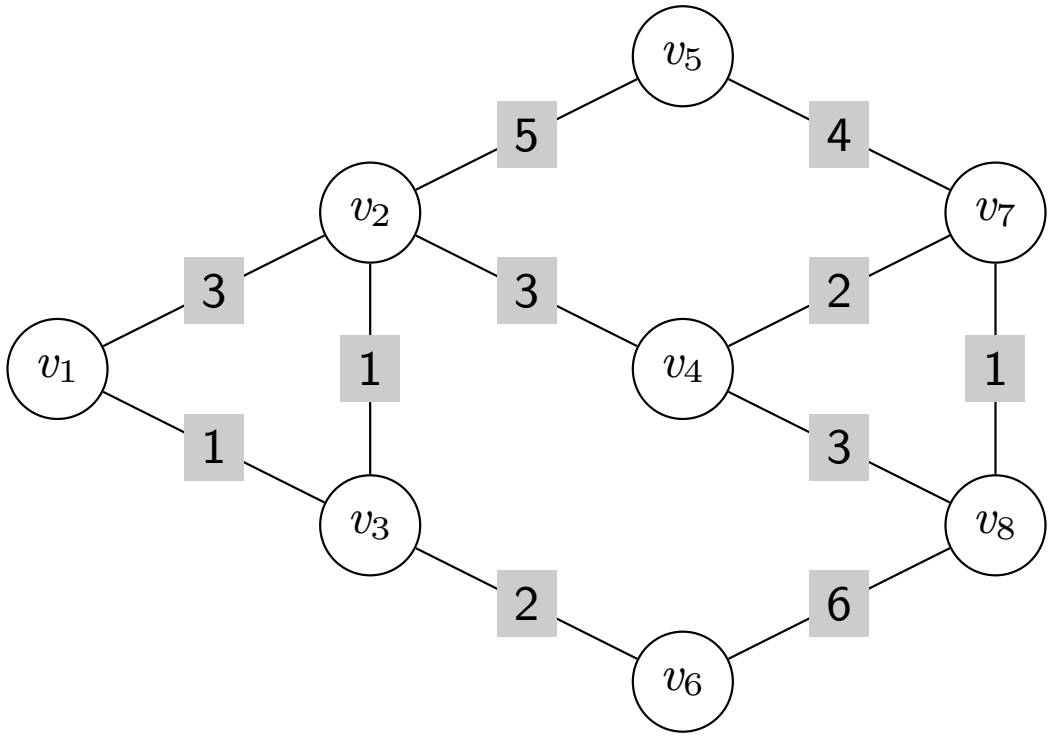
Problème

On considère un graphe G valué et deux sommets u et v de G. Trouver le plus court chemin (au sens de la somme des poids) entre u et v.

- l'approche gloutonne échoue ici.
- le problème est trivial à résoudre dans un arbre.
- l'algorithme de Dijkstra résout un problème plus général

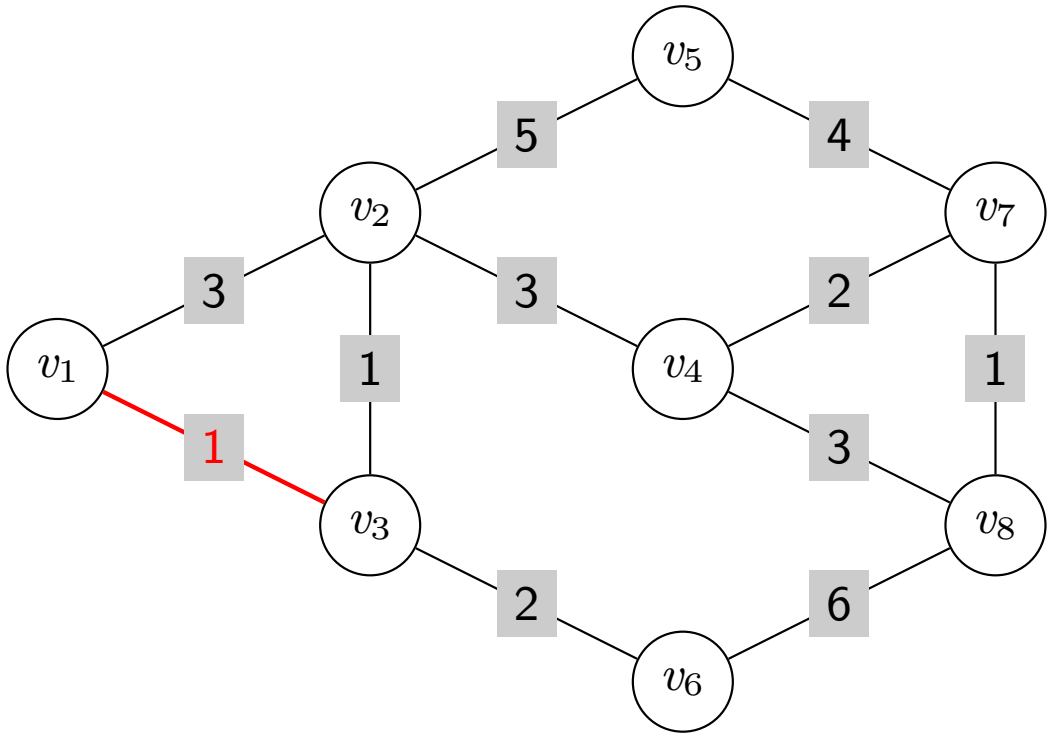
On considère un graphe valué G et un sommet u de G. Construire un arbre couvrant T_u tel que, pour tout sommet v, la distance de u à v est la même dans G et dans T_u.

Illustration



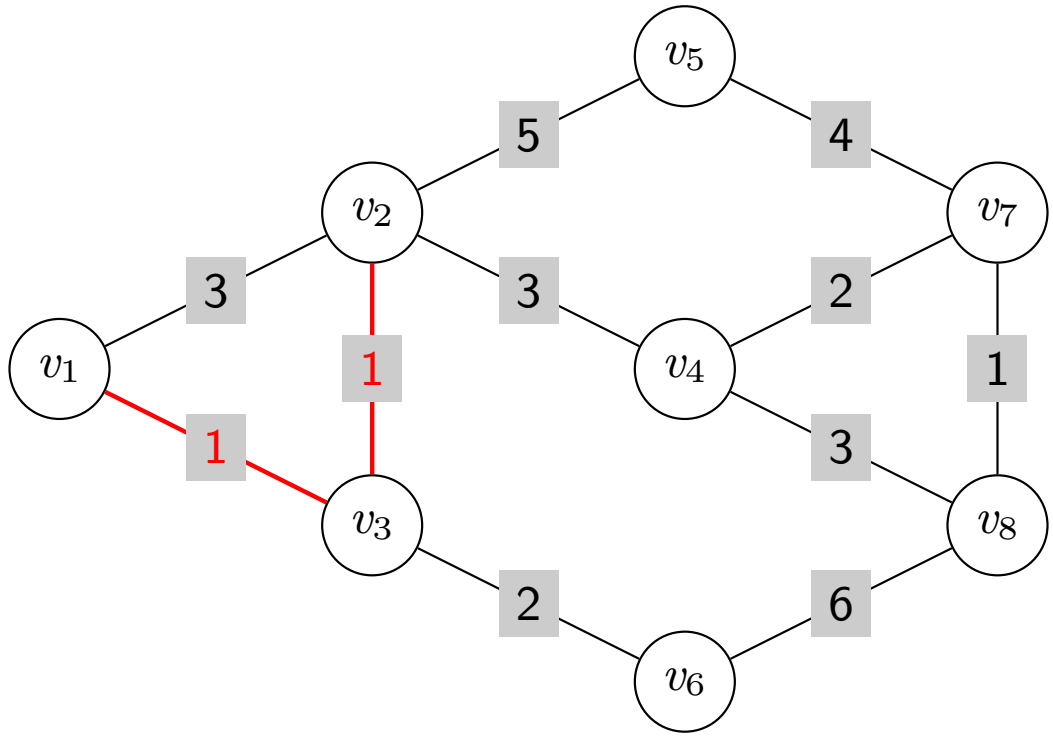
Sommet	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Distance (rouge si finale)	0	3	1	∞	∞	∞	∞	∞

Illustration



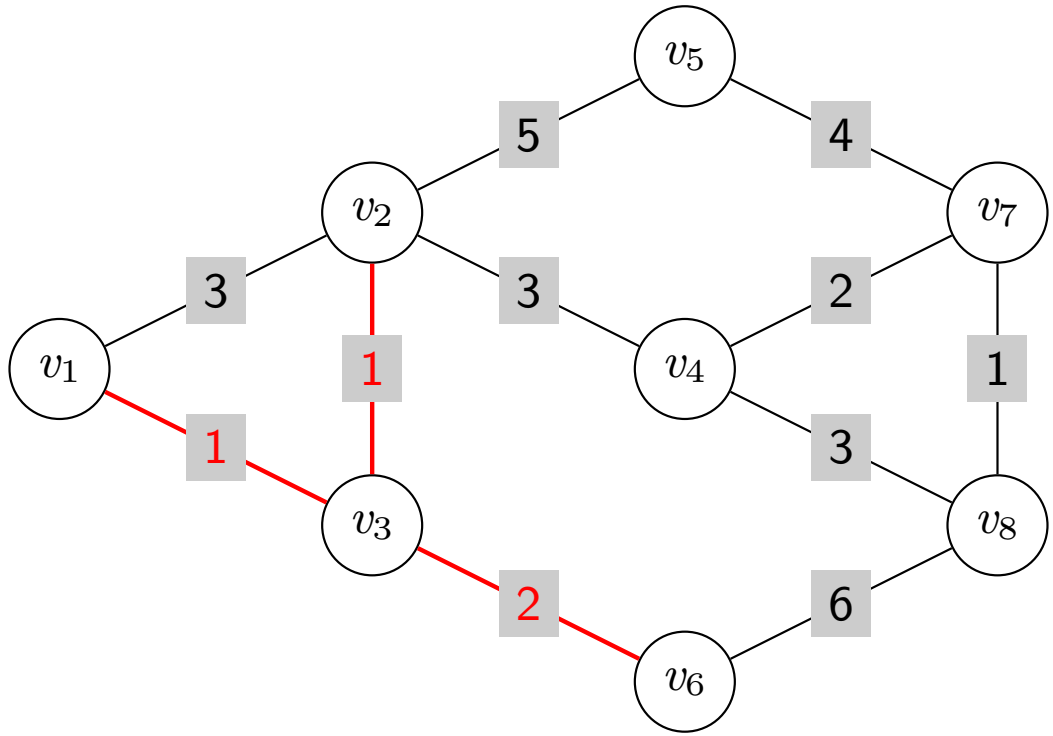
Sommet	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Distance (rouge si finale)	0	2	1	∞	∞	3	∞	∞

Illustration



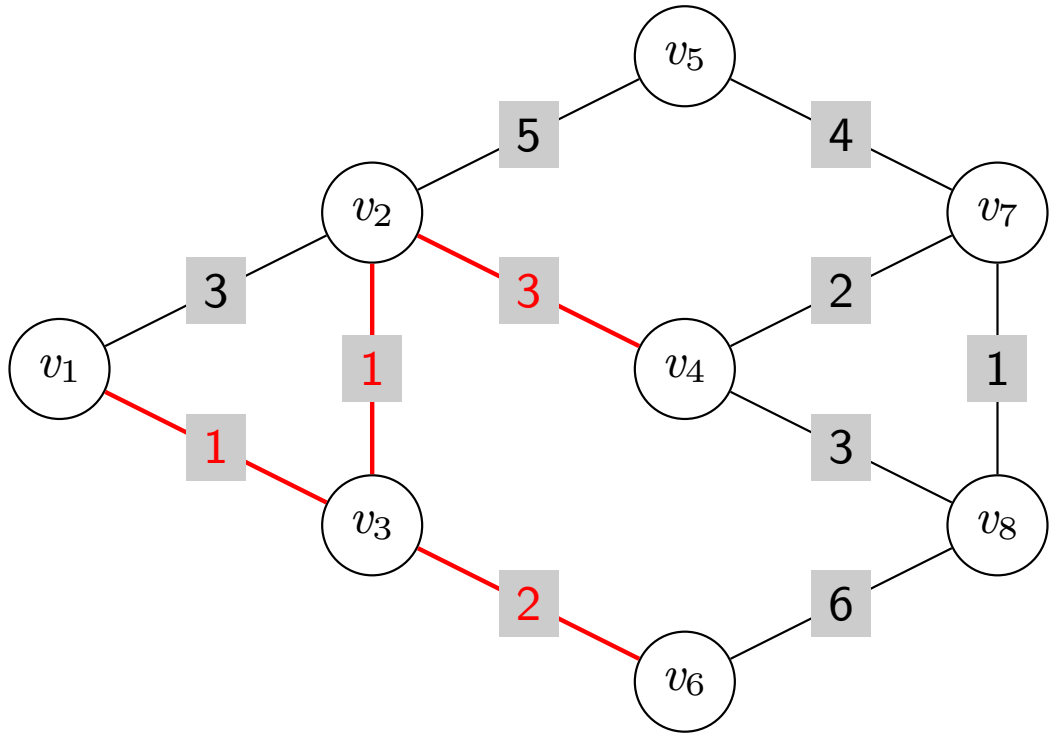
Sommet	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Distance (rouge si finale)	0	2	1	5	7	3	∞	∞

Illustration



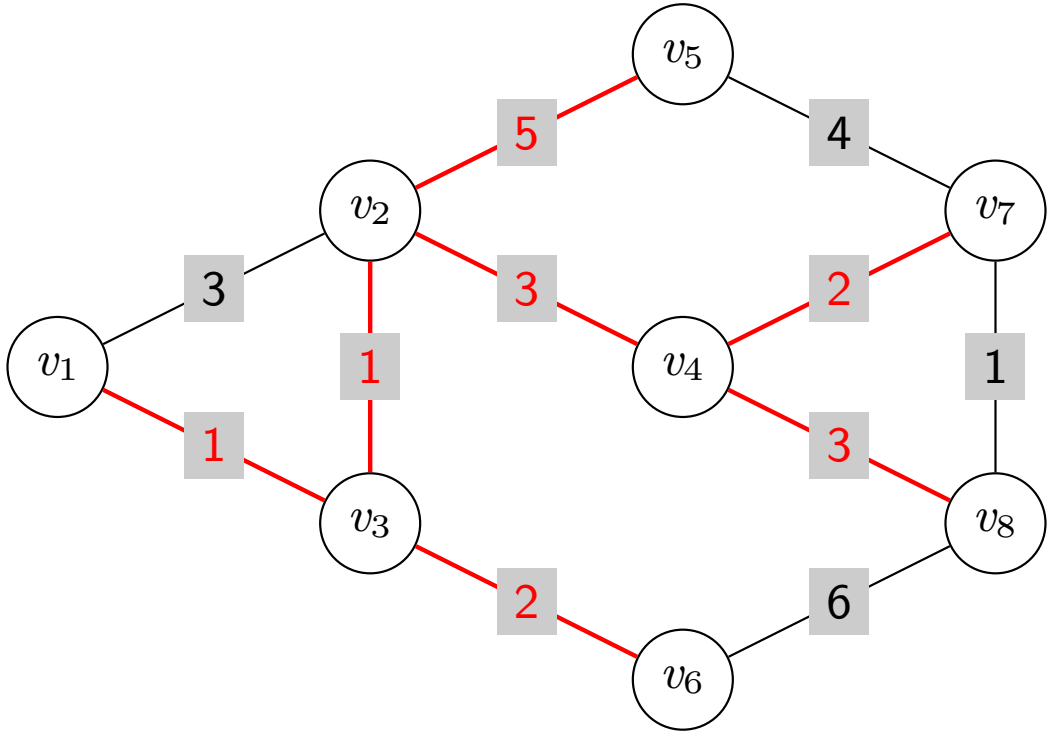
Sommet	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Distance (rouge si finale)	0	2	1	5	7	3	∞	9

Illustration



Sommet	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Distance (rouge si finale)	0	2	1	5	7	3	7	8

Illustration



Sommet	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Distance (rouge si finale)	0	2	1	5	7	3	7	8

Data: Un graphe connexe valué G de fonction de poids ω et un sommet u
Result: Un arbre couvrant T et la distance $D(v) = d_G(u, v)$ pour tout v

$T = \{u\}$
for $v \in V(G)$ **do**
  $dist\_prov(v) := \infty$; $pere(v) := \emptyset$
end
$dist\_finale(u) := 0$; $dernier\_ajout := u$

while $V(T) \neq V(G)$ **do**
  for v *voisin de* $dernier\_ajout$ **do**
    if $dist\_finale(dernier\_ajout) + \omega(dernier\_ajout, v) < dist\_prov(v)$ **then**
      $dist\_prov(v) := dist\_finale(dernier\_ajout) + \omega(dernier\_ajout, v)$
      $pere(v) := dernier\_ajout$
    end
  end
  Selectionner $v \notin V(T)$ tel que $dist\_prov$ est minimum
  Ajouter $(pere(v), v)$ a T
  $dist\_finale(v) := dist\_prov(v)$
  $dernier\_ajout := v$
end

Algorithm 6: Algorithme de Dijkstra

Algorithme de Dijkstra

Proposition

L'algorithme de Dijkstra renvoie bien la distance de tout sommet au sommet d'origine.

Proposition

La complexité est polynomiale. Elle est en $\mathcal{O}(n^2)$ pour un code naïf, et peut être réduite à $\mathcal{O}(m + n \log n)$ à l'aide de tas binaire et même $\mathcal{O}(m + n \log n)$ à l'aide de tas de Fibonacci.

Algorithme de Floyd-Warshall

- L'algorithme de Dijkstra ne s'applique pas si le graphe contient des arêtes de poids négatif.

Floyd-Warshall

L'algorithme de Floyd-Warshall permet de calculer les distances entre toutes les paires de sommets, même si certaines arêtes sont de poids négatif, à condition qu'il n'y ait aucun cycle de poids strictement négatif.

Algorithme de Floyd-Warshall : principe

On numérote les sommets de 1 à n.
On calcule pour tout k la matrice $\mathcal{W}^k$ telle que $\mathcal{W}^k_{uv}$ contient le poids du chemin de poids minimal entre u et v dont tous les sommets internes appartiennent à $\{1, \ldots, k\}$.

- $\mathcal{W}^k_{ij} = \min(\mathcal{W}^{k-1}_{ij}, \mathcal{W}^{k-1}_{ik} + \mathcal{W}^k_{kj})$
- $\mathcal{W}^n_{ij}$, $1 \leq k \leq n$ est le poids du plus court chemin.
- La complexité est de $\mathcal{O}(n^3)$.

III. Cycles couvrants - Introduction à la complexité

Problèmes considérés

Chinese Postman problem : Un postier cherche l'itinéraire le plus court pour accomplir sa tournée, sachant qu'il doit parcourir toutes les rues dont il a la charge au moins une fois, et revenir à son point de départ.

Travelling salesman problem (problème du voyageur de commerce) : Un agent commercial doit faire le tour des clients dont il a la charge et revenir à son point de départ, tout en minimisant un certain coût (euros, temps, kiomètres...).

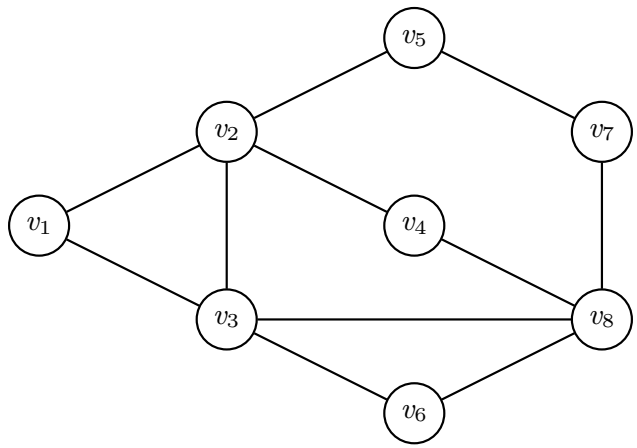
- Recherche de cycles couvrant de poids minimum dans un graphe arête-valué : le postier doit couvrir toutes les arêtes, alors que le voyageur doit 'seulement' couvrir tous les sommets.
- Cette différence rend ces deux problèmes très différents : celui du postier peut être résolu de façon exacte, alors que celui de l'agent ne peut faire l'objet que d'une résolution approchée (heuristique).

III.1 Graphes Eulériens

Graphes eulériens

Definition

Un *circuit eulérien* dans un graphe est une marche empruntant chaque arête exactement une fois et terminant en son point de départ.
Un graphe eulérien est un graphe admettant un circuit eulérien.



Chemin eulérien : $\{v_1, v_3, v_6, v_8, v_7, v_5, v_2, v_4, v_8, v_3, v_2, v_1\}$

Graphes eulériens

Definition

Un *circuit eulérien* dans un graphe est une marche empruntant chaque arête exactement une fois et terminant en son point de départ.
Un graphe eulérien est un graphe admettant un circuit eulérien.

- Leonhard Euler est considéré comme le fondateur de la théorie des graphes. Il avait posé en 1738 la question de savoir s'il existait un moyen de traverser les septs ponts reliant les deux îles de la ville de Königsberg entre elles et au continent et de revenir au point de départ sans emprunter deux fois le même pont. Ce problème revient à la recherche d'un circuit eulérien.
- Dans le cas du postier, un circuit eulérien est la solution optimale. Reste à savoir s'il est possible de déterminer polynomialement l'existence d'un tel circuit.

Proposition

Un graphe connexe est eulérien si et seulement si tous ses sommets sont de degré pair.

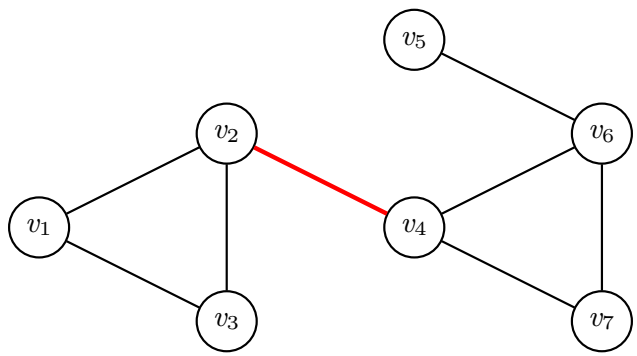
Proposition

Un graphe connexe est eulérien si et seulement si tous ses sommets sont de degré pair.

Avant de démontrer le théorème, il faut définir la notion de pont.

Definition

Un *pont* est une arête dont la suppression augmente le nombre de composantes connexes.



Proposition

De chaque côté d'un pont se trouve au moins un sommet de degré impair.

Algorithme de Fleury

1. Choisir un sommet v_0 et poser $W_0 = \emptyset$.
2. Supposons que le parcours (liste d'arête) $W_i = e_1 \dots e_m$ a été contruit. Soit v_m le dernier sommet de ce parcours.
3. Si il existe des arêtes adjacentes à v_m non-encore utilisées, on en choisit une appelée e_{m+1}. Sauf s'il n'y a pas le choix, on ne choisit pas e_{m+1} comme étant un pont dans $G_m = G - \{e_1, \dots, e_m\}$. On pose $W_{m+1} = W_m \cup e_{m+1}$.
4. Sinon, on arrête et on retourne W_m

Chemin eulérien

Un parcours eulérien est une marche couvrant toutes les arêtes en les empruntant de façon unique (on supprime simplement la condition de retour au point de départ).

Proposition

Un graphe contient un parcours eulérien si et seulement si au plus deux de ses sommets ont un degré impair.

III.2 Chinese Postman Problem

Résolution du Chinese Postman Problem

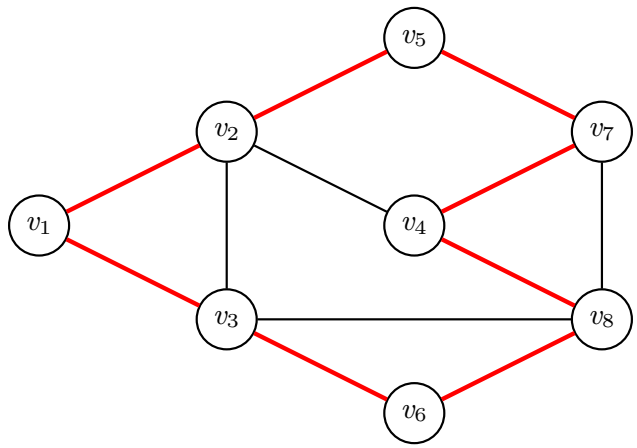
1. Enumérer l'ensemble S des sommets de degrés impair.
2. Pour toute paire de sommets de S, chercher un chemin de longueur minimale entre ces deux sommets (Algorithme de Dijkstra)
3. Créer un graphe complet H avec $V(H) = S$ dans lequel les arêtes sont valués par le poids du plus court chemin précédent.
4. Chercher un couplage parfait de poids minimal dans ce graphe. Un couplage est un ensemble d'arêtes tel que tout sommet soit incident à exactement une arête. Ce problème peut être résolu en temps polynomial par un algorithme de programmation linéeaire. (hors cadre de ce cours)
5. Doubler les arêtes le long des chemins correspondant à ce couplage minimal
6. Appliquer l'algorithme de Fleury sur le graphe résultant.

III.3 Graphes hamiltoniens

Graphe hamiltonien

Definition

Un *cycle hamiltonien* est un cycle qui passe par tous les sommets d'un graphe.
Un graphe est dit *hamiltonien* s'il admet un circuit hamiltonien.



- Dans le cas où toutes les arêtes sont de poids égaux et où il existe un circuit hamiltonien, ce dernier est la solution optimale pour le problème du voyageur de commerce.

III.4 Un peu de complexité

P et NP

- Un problème de décision est un problème dont la réponse est oui ou non.
- Un problème est dans la classe P si il peut être résolu en en temps polynomial en la taille de l'instance (ex : existence d'un chemin eulérien).
- Un problème est dans la classe NP si la validité d'une solution peut être vérifiée en un temps polynomial (ex : chemin hamiltonien).
- Un problème est NP-complet si son appartenance à P implique que tout porblème de NP est dans P.

P=NP ?

0n a facilement que P$\subset$NP. Il en résulte que :

- soit il existe un problème NP-complet qui est dans P et P=NP
- soit P et NP sont disjoints et un problème NP complet ne peut pas être résolu en un temps polynomial.

Savoir laquelle de ces assertions est vraie reste une conjecture ouverte mais la seconde est plus que vraisemblablement la bonne.

Problèmes NP-complets

- On démontre qu'un problème P_1 est NP complet en montrant que si on peut le résoudre en temps polynomial, alors on peut résoudre en temps polynomial un autre problème P_2 déjà connu comme étant NP-complet. On parle de réduction du problème P_2 au problème P_1.
- Karp (1972) a publié une liste de 21 problèmes NP-complets qui servent de référence. Le principal étant le problème 3-SAT.

Definition

Problème 3-SAT On considère n variables booléennes $x_1, \ldots, x_n$, et m clauses $C_1, \ldots, C_m$ de type *ou* incluant chacune 3 variables ou leur négation (ex : $C_1 = x_1 \vee \overline{x_2} \vee x_4$).
Décider s'il existe une affectation des variables telle que $C = C_1 \wedge \ldots \wedge C_m$ est satisfaite.

Problèmes NP-complets

- On démontre qu'un problème P_1 est NP complet en montrant que si on peut le résoudre en temps polynomial, alors on peut résoudre en temps polynomial un autre problème P_2 déjà connu comme étant NP-complet. On parle de réduction du problème P_2 au problème P_1.
- Karp (1972) a publié une liste de 21 problèmes NP-complets qui servent de référence. Le principal étant le problème 3-SAT.
- Ces notions s'étendent aux problèmes qui ne sont pas des problèmes de décision mais peuvent se décomposer en un nombre polynomial de problèmes de décision.
  Ex : trouver la longueur maximale d'un chemin se décompose en le problème de décision pour l'existence d'un chemin de longueur k pour tout k entre 1 et n.

III.5 Retour au voyageur de commerce

NP-complétude

Proposition

Le problème de décision du cycle hamiltonien est NP-complet.

Proposition

Le problème de décision du voyageur de commerce est NP-complet.

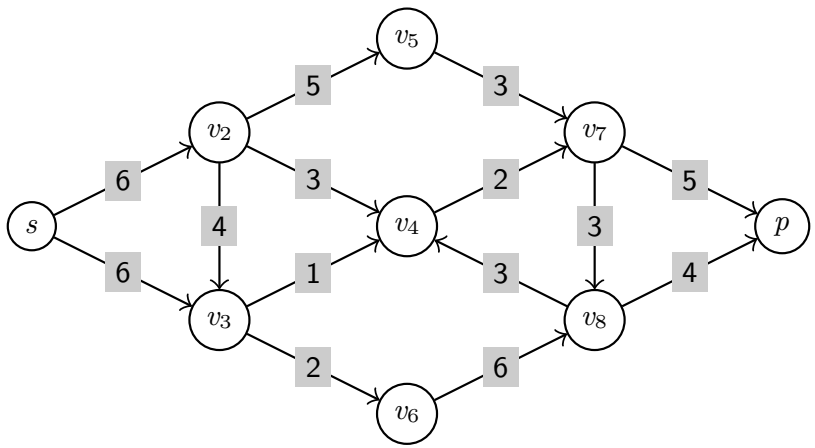
Il faut donc se contenter d'heuristiques, c'est-à-dire d'algorithmes donnant une solution approchée.

IV. Flots dans les graphes

Problème

On considère un graphe dirigé G tel que :

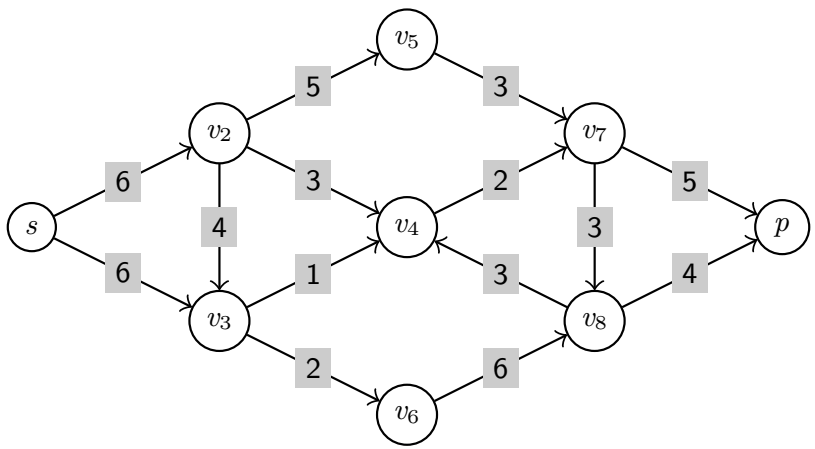
1. il existe deux ensembles disjoints S et P de sommets, respectivement appelés sources et puits ;
2. chaque arête e est valué par une capacité $c(e)$ correspondant au flux maximum pouvant transiter par cette arête.



Problème

Quel est le flux maximal pouvant aller des sources vers les puits sans perte intermédiaire ?

- Dans le cas d'un graphe non-dirigé, on remplace chaque arête par deux arêtes dirigées opposées de même capacité que l'arête initiale.



Flot

Notations

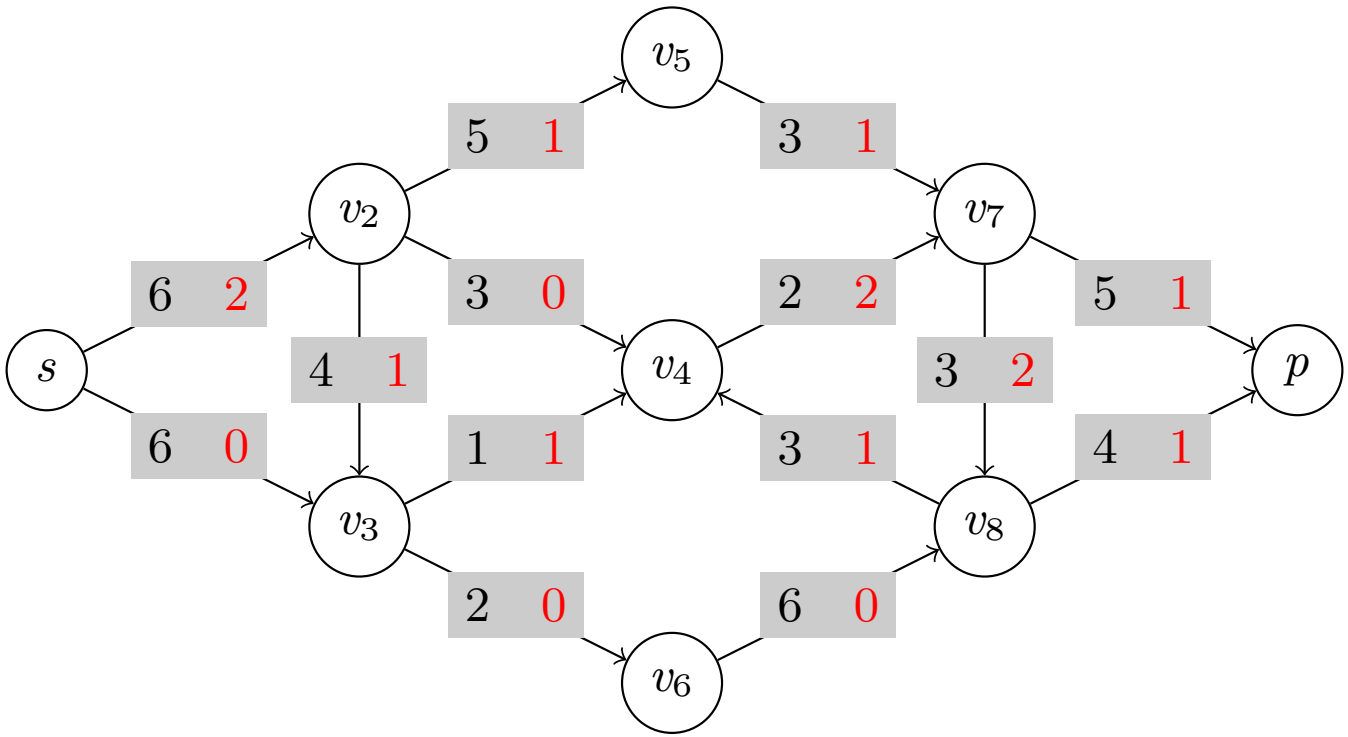
- Pour tout ensemble A de sommets, on note $\overline{A} = V(G) \setminus A$ et $(A, \overline{A})$ l'ensemble des arêtes de A vers $\overline{A}$.
- Pour toute fonction $f : E(G) \to \mathbb{R}$, on note $f^+(A) = \sum_{c \in (A, \overline{A})} f(c)$ et $f^-(S) = \sum_{c \in (\overline{A}), A} f(c)$.

Definition

Un *flot entier* est une fonction $f : E(G) \to \mathbb{N}$ telle que :

1. pour tout e, $0 \leq f(e) \leq c(e)$;
2. pour tout sommet $v \notin S \cup P$, $f^+(v) = f^-(v)$.

Flot



Valeur d'un flot

Proposition

Pour tout flot f dans G, $f^+(S) - f^-(S) = f^-(P) - f^+(P)$. Cette valeur est notée $val(f)$.

Le flot de l'exemple précédent est de de valeur 2.

Definition

Un flot f est maximal s'il n'existe pas de flot f' avec $val(f') > val(f)$.

Combien de sources et de puits ?

Proposition

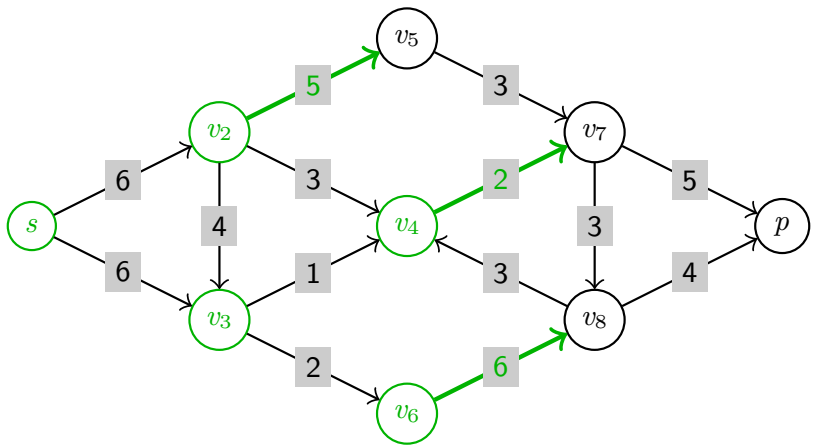
Problème 1 : *Soit (G, c) un graphe dirigé valué, $S \subset V(G)$, $P \subset V(G)$.*
Déterminer un flot maximal de S vers P.
Problème 2 : *Soit (H, c) un graphe dirigé valué, $s \in V(H)$, $p \in V(H)$.*
Déterminer un flot maximal de s vers p.
Les problèmes 1 et 2 sont équivalents.

- On se limite dorénavant au cas où il y a une source et un puits

Coupe

Definition

Une *coupe* est un couple $K = (A, \overline{A})$ tel que $s \in A$ et $p \in \overline{A}$. La *capacité d'une coupe* $cap(K)$ est la somme des capacités des arêtes allant de A à $\overline{A}$.



La coupe pour l'ensemble A correspondant aux sommets verts est de capacité 13.

Coupes et flots

Proposition

Pour toute coupe $K = (A, \overline{A})$ *et tout flot* f, $val(f) = f^+(A) - f^-(A)$.

Par conséquent, pour toute coupe K *et tout flot* f, $val(f) \leq cap(K)$.
En particulier, $\max_f val(f) \leq \min_K cap(K)$.

Théorème max-flot min-cut

Théorème

Dans tout réseau, le flot maximal a pour valeur la capacité de la coupe minimale.

- La preuve de ce théorème est constructive, à savoir qu'on va montrer l'existence de ce flot en présentant une manière de le construire (et donc de résoudre le Problème 2).

Saturation d'un chemin

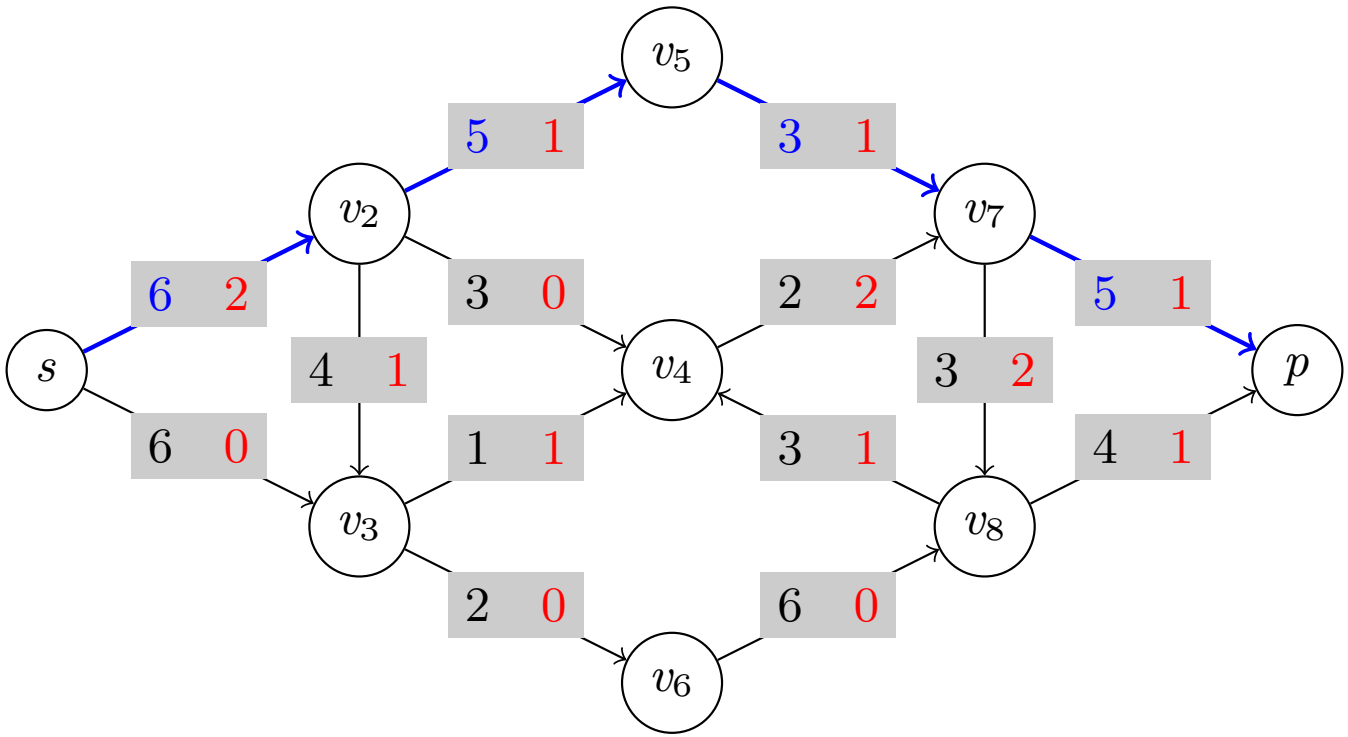
Definition

Soit f un flot et P un chemin entre s et p, les arêtes pouvant être parcourues dans les deux sens. Soit $e(P)^-$ les arêtes parcourues à contre-sens par P et $e(P)^+$ celle parcourues dans le bon sens. La saturation du chemin P est alors définie par

$$sat(P) = \min\Big(\min_{e \in e(P)^+} (c(e) - f(e)), \min_{e \in e(P)^-} f(e) \Big)$$

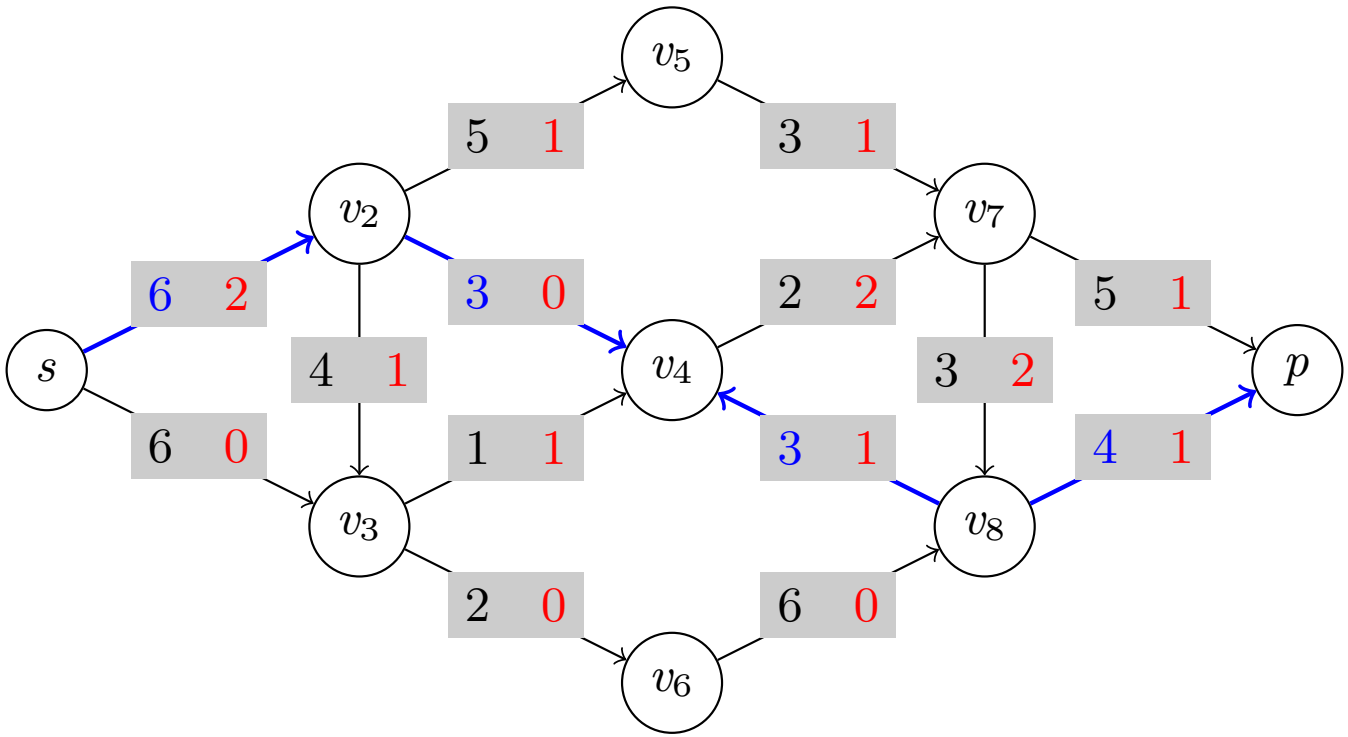
Si $sat(P) = 0$, le chemin est dit *saturé*.

Saturation d'un chemin



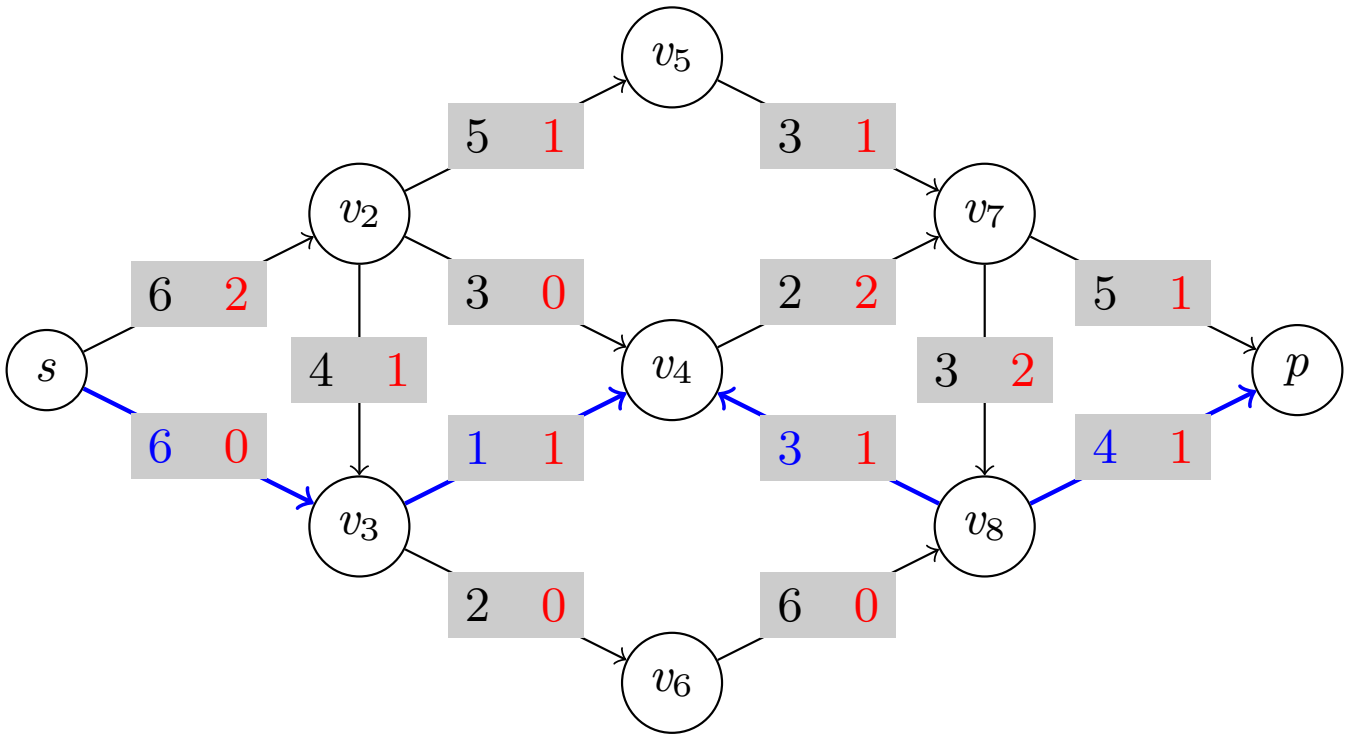
La saturation du chemin bleu est de 2.

Saturation d'un chemin



La saturation du chemin bleu est de 1.

Saturation d'un chemin



Le chemin bleu est saturé.

Saturation d'un chemin

- L'existence d'un chemin non saturé permet d'augmenter la valeur de f. En effet, en ajoutant $sat(P)$ sur les arêtes de $e(P)^+$ et en enlevant $sat(P)$ sur les arêtes de $e(P)^-$, on obtient un nouveau flot dont la valeur a augmenté de $sat(P)$.
- En fait, il s'agit d'une condition nécessaire et suffisante pour pouvoir augmenter un flot :

Proposition

Un flot f est maximal si et seulement si il n'y a pas de chemin non-saturé entre s et p.

Construction d'un flot maximal

Recherche d'un chemin saturé

On utilise une variante de l'arbre de parcours en profondeur enraciné en s :

- on parcourt une arête dans le bon sens uniquement si elle n'est pas saturée ;
- on parcourt une arête à contresens uniquement si son flot est non-nul.

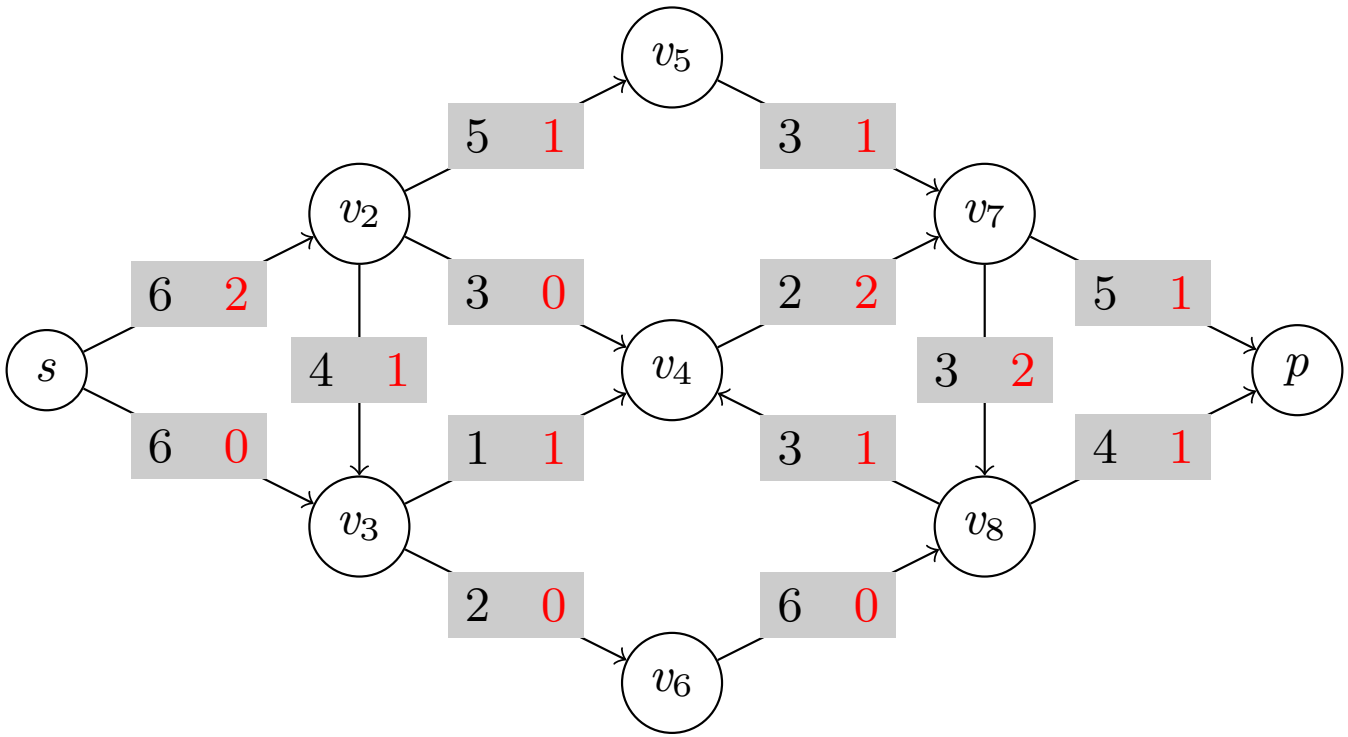
Si p est atteint lors de ce parcours, le chemin P allant de s à p dans l'arbre de parcours est un chemin non-saturé. Sa saturation vaut

$$\alpha = \min\Big(\min_{e \in e(P)^+} (c(e) - f(e)), \min_{e \in e(P)^-} f(e) \Big)$$

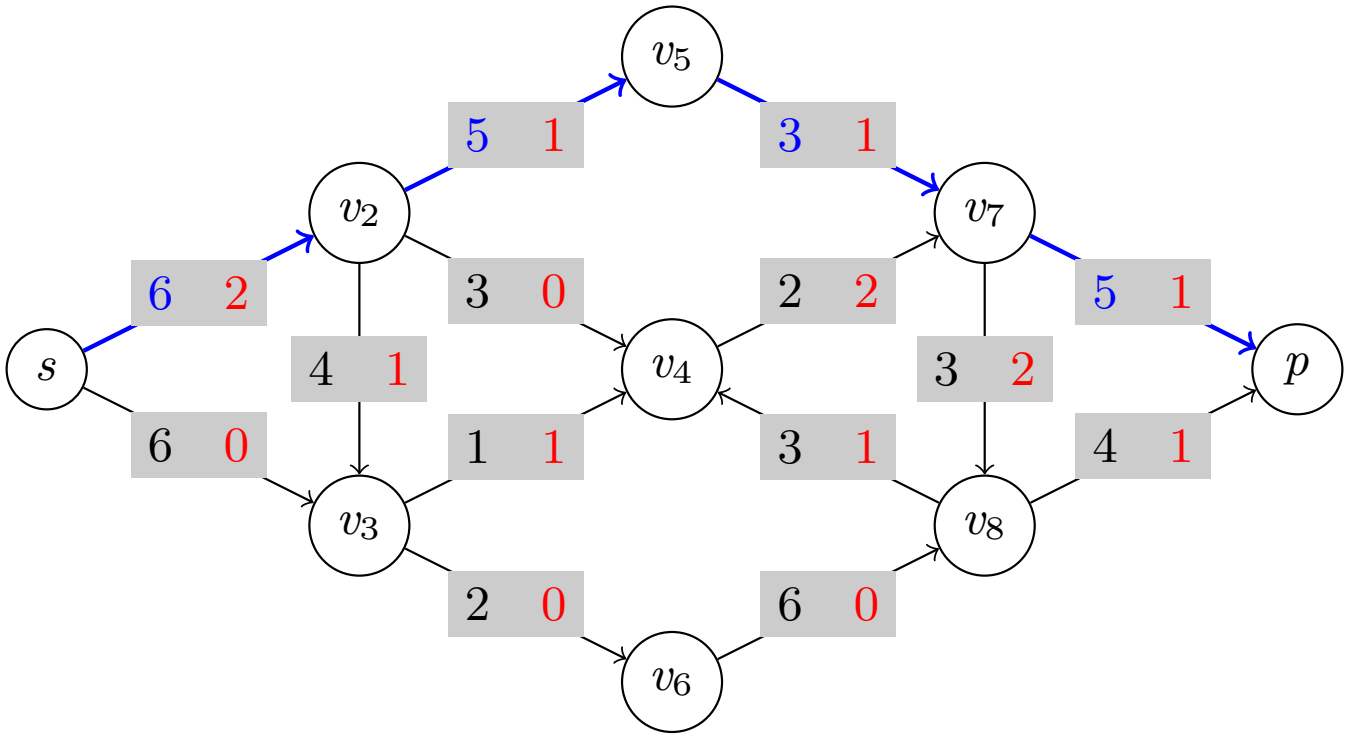
Construction d'un flot maximal

- on part d'un flot connu, par exemple le flot nul.
- tant que l'on trouve un chemin non-saturé entre s et p (parfois appelé chemin augmentant), on remet le flot à jour. Quand cela n'est plus possible, l'algorithme stoppe.

Exemple

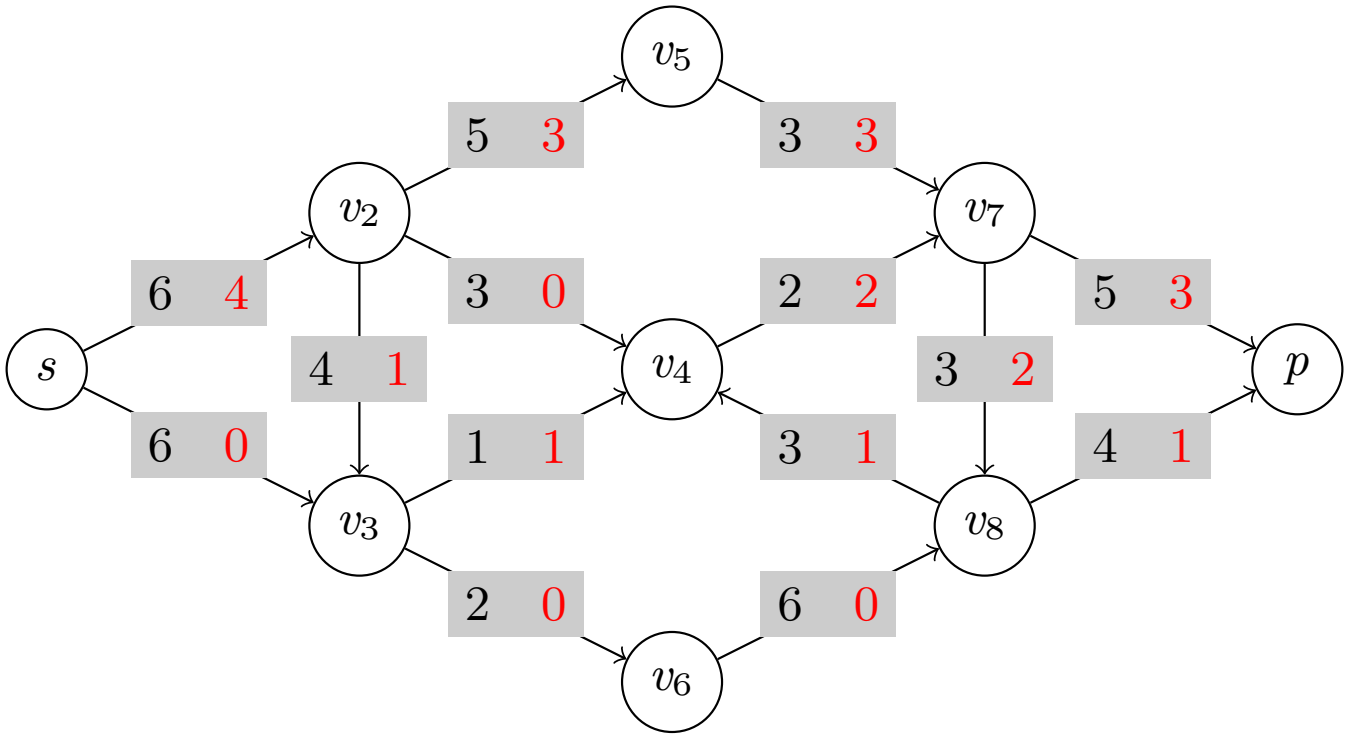


Exemple



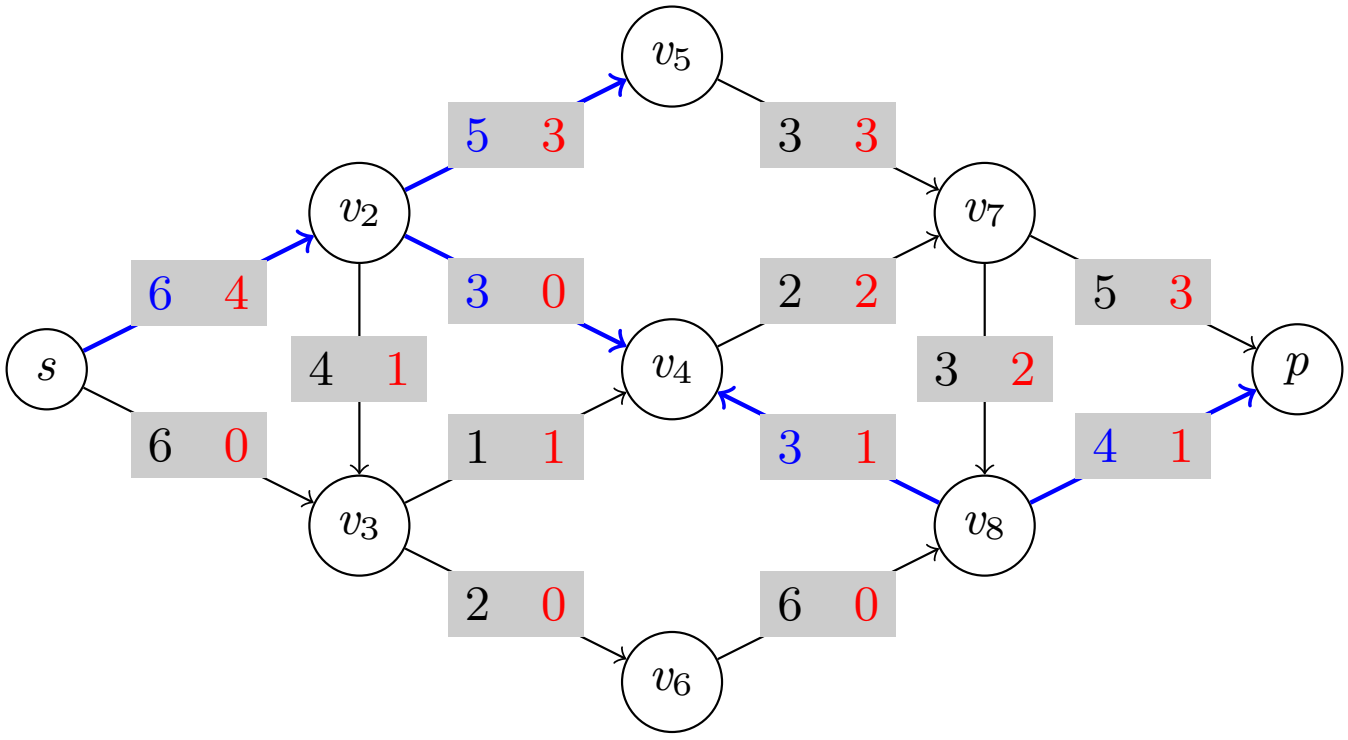
Le parcours en profondeur donne un chemin augmentant de saturation 2.

Exemple



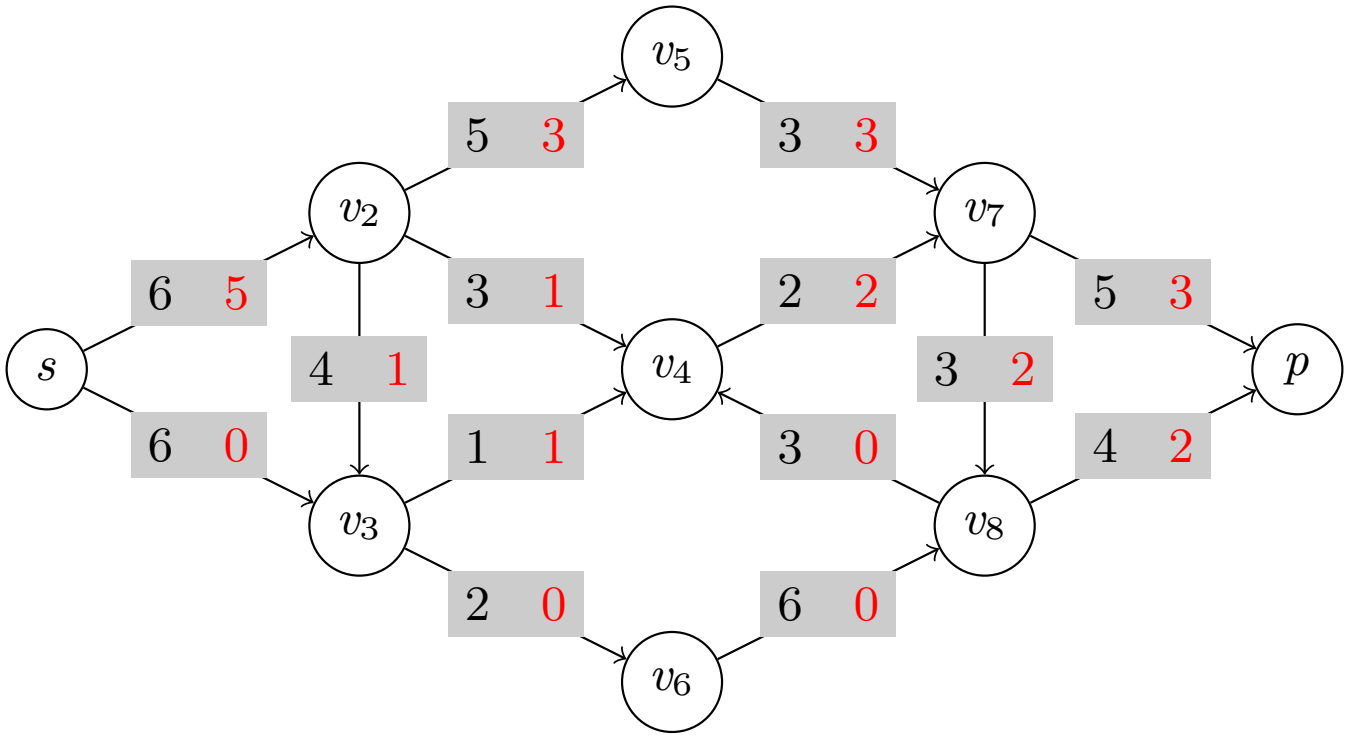
On augmente le flot le long de ce chemin.

Exemple



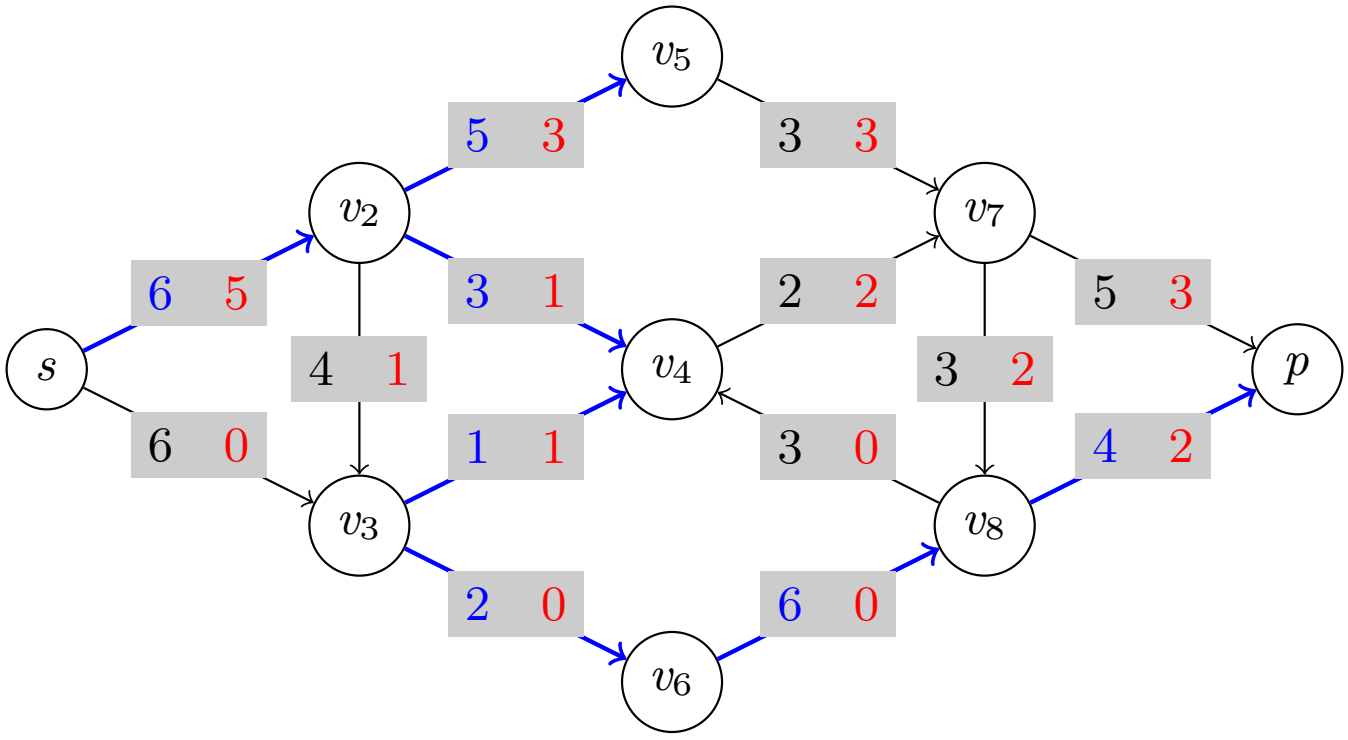
Le parcours en profondeur donne un chemin de saturation 1.

Exemple



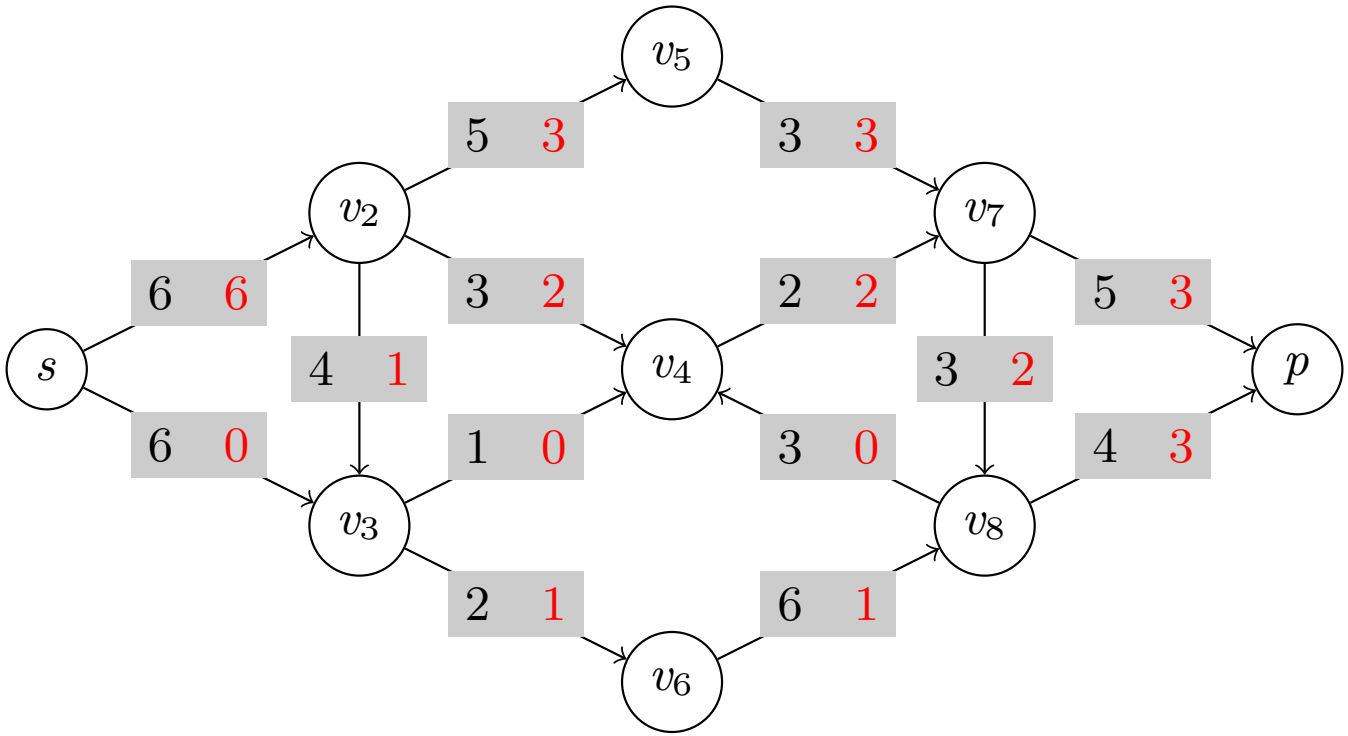
On augmente le flot le long de ce chemin.

Exemple



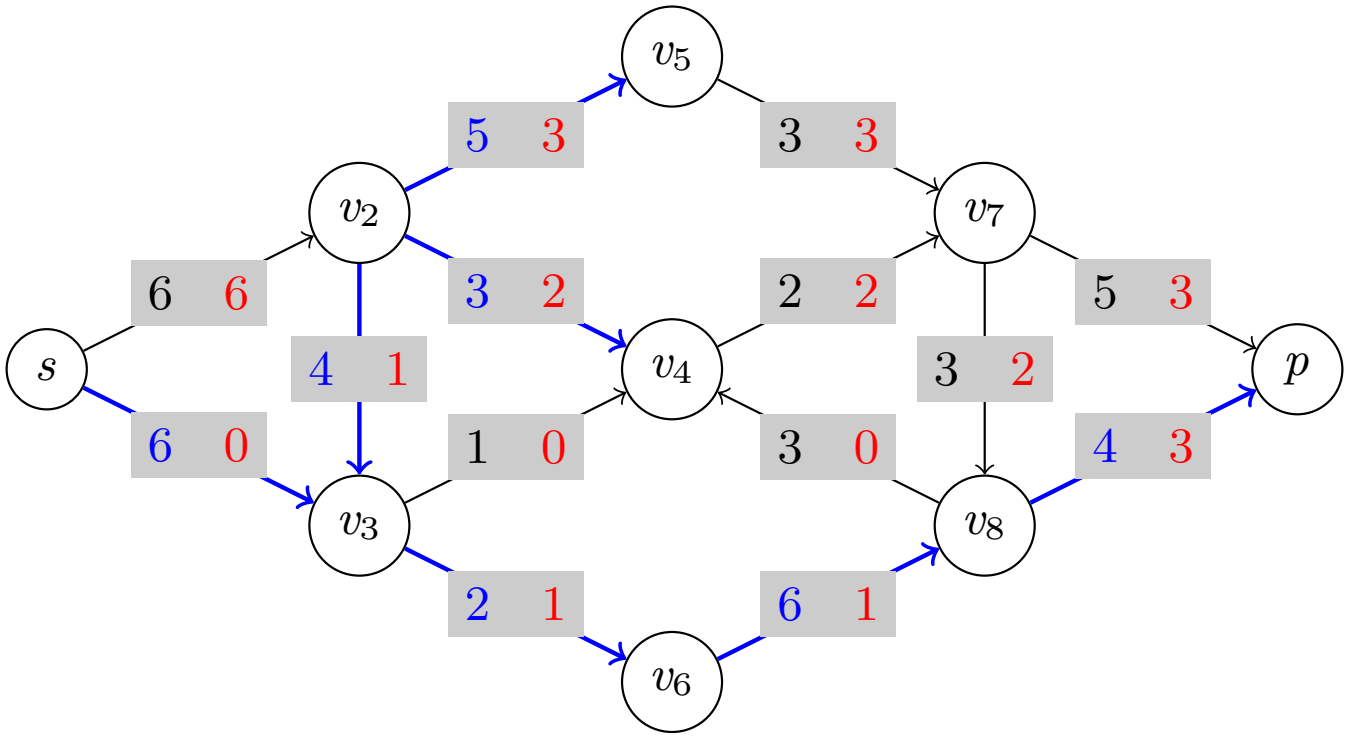
Le parcours en profondeur donne un chemin de saturation 1.

Exemple



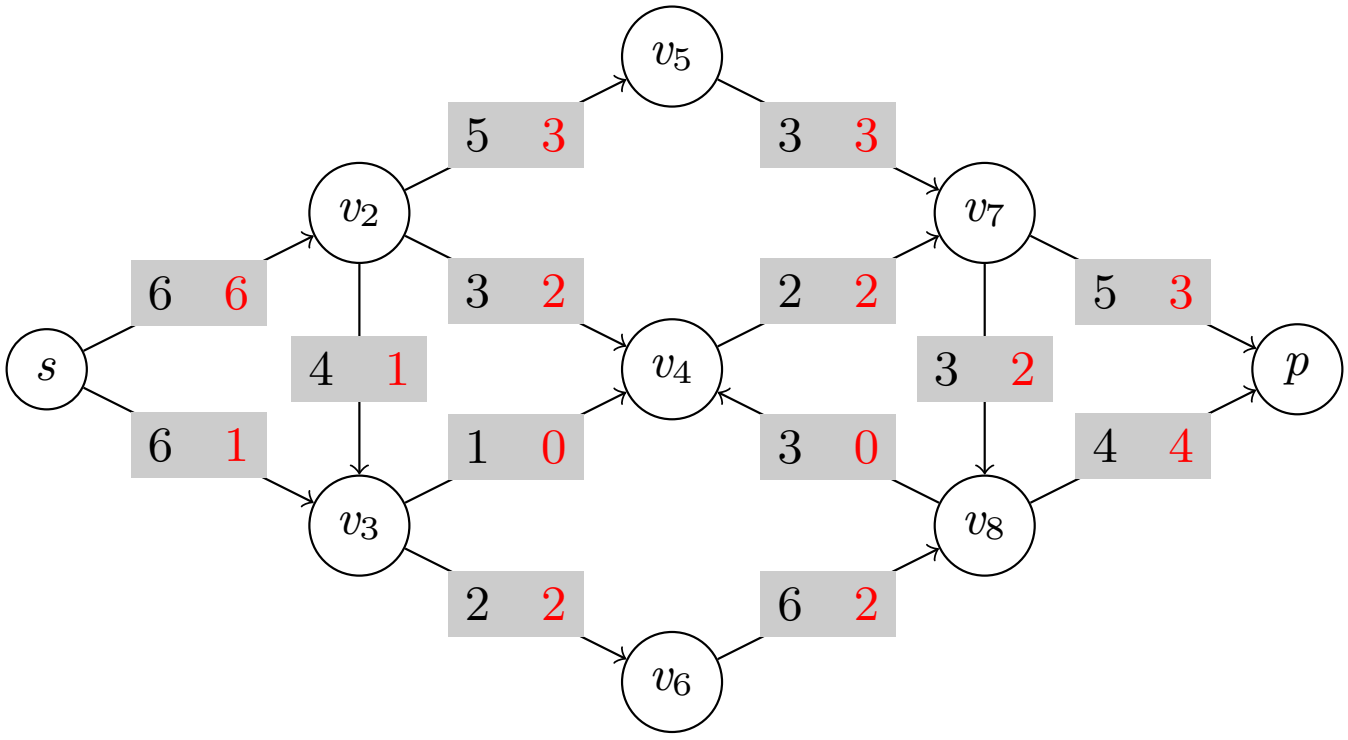
On augmente le flot le long de ce chemin.

Exemple



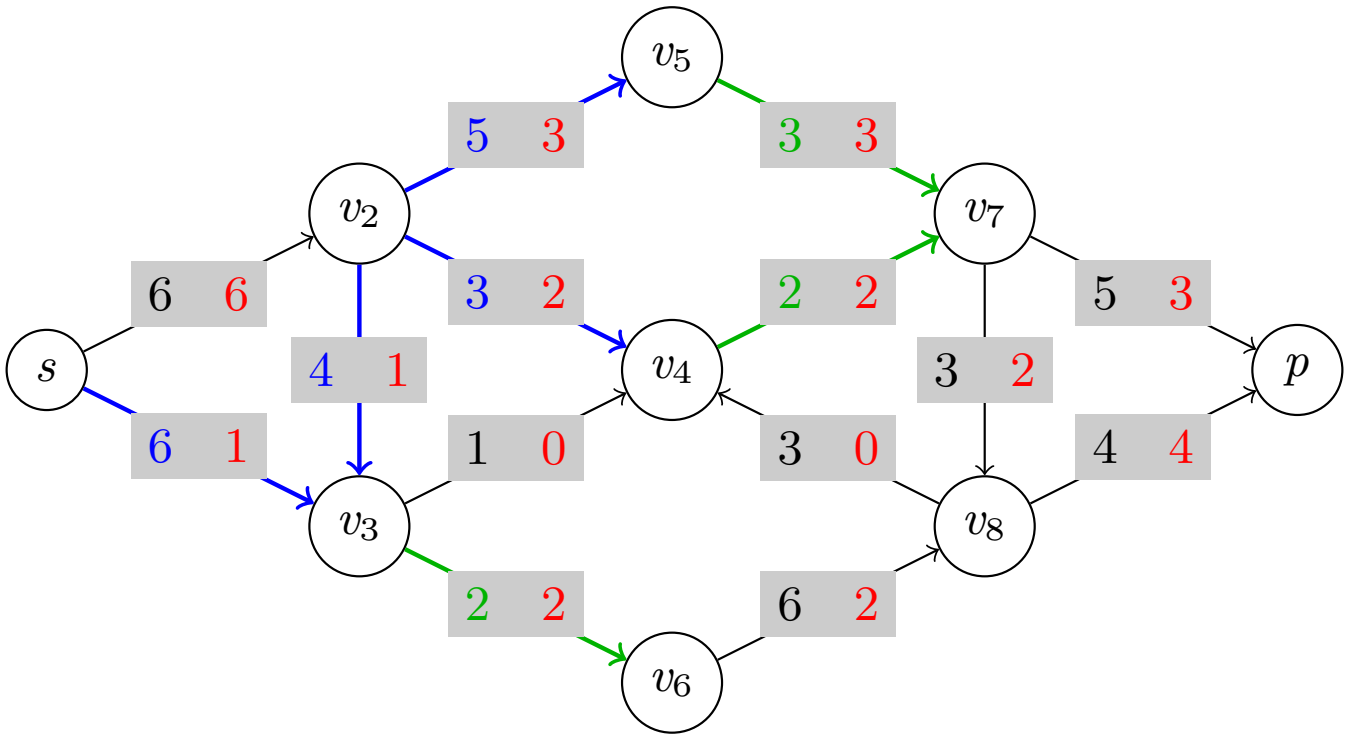
Le parcours en profondeur donne un chemin augmentatn de saturation 1.

Exemple



On augmente le flot le long de ce chemin.

Exemple



Le parcours en profondeur ne donne plus de chamin augmentant, l'algorithme s'arrête.
Le flot maximal (pas forcément unique) est de valeur 7, une coupe minimale est $(A, \overline{(A)})$, où A désigne les sommets atteints par le dernier parcours en largeur.

Une conséquence : le(s) théorème(s) de Menger

Théorème

Soit S et P deux ensembles de sommets d'un graphe G. Pour tout entier k, soit il existe $k+1$ chemins arêtes-disjoints reliant S à P, soit il existe au plus k arêtes dont la suppression déconnecte S de P.

Definition

Un séparateur d'un graphe est un ensemble de sommets X tel que $G \setminus X$ a un nombre de composantes connexes supérieur à celui de G.
Soit S et P deux ensembles de sommets. Un (S, P)-séparateur X est un séparateur tel qu'aucun chemin reliant un sommet de S à un sommet de P n'existe dans $G \setminus X$.

Théorème

Un graphe contient k chemins disjoints entre deux ensembles de sommets S et P si et seulement si G ne contient pas de (S, P)-séparateur de taille au plus $k-1$.

Graphes k-connexes

Definition

Un graphe est k-connexe si, pour toute paire de sommets u et v, il existe k chemins disjoints, c'est-à-dire n'ayant aucun sommet interne en commun.

Théorème

Un graphe est k-connexe si et seulement si G ne contient pas de séparateur de taille au plus $k-1$.

IV. Chaînes de Markov

I.1. Définition et classification

Marche aléatoire sur un graphe

Definition

Soit G un graphe orienté à n sommets dénotés $v_1, \ldots, v_n$ et dont les arêtes sont valuées par des poids p_{uv} tels que

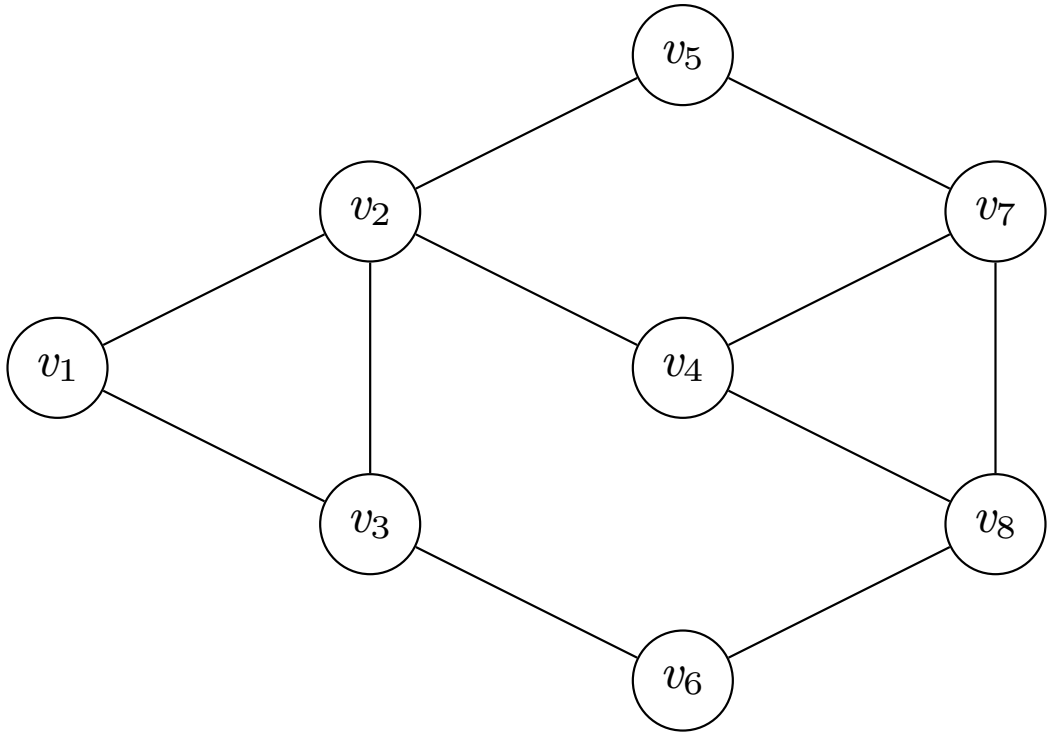
$$\forall u \in V(G), \quad \sum_{v \in N^+(u)} p_{uv} = 1$$

Une *marche aléatoire sans mémoire* sur G est une marche aléatoire définie de la façon suivante :

- on note x_0 la position initiale ;
- à l'instant i, la marche est en x_i et, pour tout $v \in N^+(x_i)$, $\mathbb{P}(x_{i+1} = v) = p_{x_i v}$.

En d'autres termes, p_{uv} est la probabilité d'emprunter l'arête (u, v) quand la marche est en u. Elle est dite sans mémoire car seule sa position à l'instant i détermine la position à l'instant $i + 1$.

Exemple



On considère une marche uniforme, c'est-à-dire que la marche choisit une des arêtes possibles avec équiprobabilité.

$$p_{v_2 v_4} = \frac{1}{4} \qquad \text{et} \qquad p_{v_4 v_2} = \frac{1}{3}$$

Marche aléatoire sur un graphe

- les points potentiellement visités sont ceux tels qu'il existe un chemin de probabilités positives depuis la source.
- on peut définir une notion de *popularité* d'un sommet x liée à la fréquence du passage de la marche en x.

Questions

- Quelle est la position 'moyenne' de la marche après un grand nombre de pas si on lance un grand nombre de marcheurs ? C'est la notion de loi de probabilité limite.
- Combien de temps faudra-t-il faire en moyenne avant de visiter un sommet donné ?
- Quelle est l'influence du choix du sommet de départ ?

Chaîne de Markov

De nombreuses questions peuvent s'écrire dans ce contexte. Il suffit pour cela que l'objet d'interêt soit une suite $(s_n)_{n\in\mathbb{N}}$ d'états tels que :

- le nombre d'états possibles $(v_1, \ldots, v_N)$ est fini ;
- il existe des probabilités de transition entre états $p_{ij} = \mathbb{P}(S_{n+1} = v_i | S_n = v_j)$ invariables dans le temps ;
- la chaîne est sans mémoire : soit S_n la variable aléatoire indiquant l'état dans lequel se trouve la chaîne après n étapes. Alors,

$$\mathbb{P}(S_n = s_n | S_{n-1} = s_{n-1}, \ldots, S_0 = s_0) = \mathbb{P}(S_n = s_n | S_{n-1} = s_{n-1})$$

Entre d'autres termes, la prochaine transition ne dépend que de l'état courant et pas du reste de l'histoire de la trajectoire suivie.

Definition

Un processus satisfaisant les contraintes ci-dessus est appelé *chaîne de Markov*.

Classification : état récurrent/transient

Soit v un état et p la probabilité, étant donné que le point de départ de la chaîne est v, de revenir en v.

- Si $p = 1$, v est un état récurrent.
- En particulier, si $p_{vv} = 1$, l'état est dit absorbant.
- Si $p < 1$, l'état est dit transient ou transitoire.
  Dans ce cas, le nombre de passages en v de la marche est presque sûrement fini et suit une loi géométrique de paramètre p.

Classification : état récurrent/état transient

- Une chaîne de Markov peut etre représentée par un graphe orienté G : (u, v) est une arête ssi $p_{uv} \neq 0$.
- Soit H un graphe ayant un sommet pour chaque composante fortement connexe de G et tel que $(u, v) \in E(H)$ s'il existe une arête de G allant de la composante correspondant à u à la composante correspondant à v. Alors le graphe H est acyclique. De plus, les états récurrents sont les états situés dans les composantes connexes dont le degré sortant dans H est nul.

Definition

Une chaîne de Markov est *irréductible* si le graphe associé est fortement connexe, ou autrement dit s'il existe un chemin entre toute paire d'états.

Classification : chaîne périodique

Définition

Un état x est un état *périodique* si le PGCD de l'ensemble $\{n|P^n(x,x)\}$ est supérieur à 1. Une chaîne est périodique si tous ses états sont périodiques. Sinon, elle est apériodique.

Remarque : En pratique, on construit toujours les chaînes de façon à ce qu'elles soient non périodiques.

I.2. Rappels d'algèbre linéaire

Valeurs et vecteurs propres

Definition

Soit A une matrice carrée de taille N. Soit X un vecteur de taille N et $\lambda \in \mathbb{R}$ tels que

$$AX = \lambda X$$

λ est une valeur propre de A et X un vecteur propre à droite associé.
Si

$$^tXA = \lambda ^tX$$

λ est une valeur propre de A et X un vecteur propre à gauche associé.

- Les valeurs propres sont les racines du polynôme $det(A - \lambda I)$.
- A et ^tA ont le mêmes valeurs propres.
- les vecteurs propres à gauche de A sont les vecteurs propres à droite de ^tA.

Matrices stochastiques

Definition

Une matrice P est une *matrice stochastique* si

1. $0 \leq p_{ij} \leq 1$ pour tout (i, j).
2. $\sum_j p_{ij} = 1$, pour tout i.

En particulier, la matrice des probabilités de transition d'une chaîne de Markov est une matrice stochastique.

Matrices stochastiques

Proposition

Soit P une matrice stochastique. Le vecteur $\begin{pmatrix} 1 \\ \cdots \\ 1 \end{pmatrix}$ est un vecteur propre à droite associé à la valeur propre 1 pour P. De plus, toute autre valeur propre λ de P vérifie $|\lambda| \leq 1$.

I.3. Convergence des chaînes de Markov

Distribution

- On note X_0 le vecteur de probabilités de la position initiale et X_n celui au bout de n pas (c'est-à-dire que X_n contient la distribution de la variable aléatoire S_n).
- X_n peut être vu comme la distribution de la position d'une trajectoire au bout de n pas ou comme la répartition d'un très grand nombre de trajectoires lancées an parallèle au bout de n pas.

Proposition

Soit P la matrice de transition de la chaîne de Markov. Pour tout n,

$$ {}^t X_n = {}^t X_0 P^n $$

Distribution invariante

En passant à la limite dans l'égalité ${}^tX_{k+1} = {}^tX_k P$, on pressent que si la marche a une distribution limite quand le nombre de pas tend vers l'infini, cette distribution devra vérifier ${}^t\mu = {}^t\mu P$, c'est-à-dire être un vecteur propre à gauche associé à la valeur propre 1.

Definition

Une mesure invariante ${}^t\mu$ pour une chaîne de Markov de matrice de transition P est un vecteur vérifiant ${}^t\mu = {}^t\mu P$.

Questions

- une distribution limite existe-t-elle toujours ?
- est-elle unique ?
- la suite des distributions $(X_n)_n$ convergent-elle vers une telle mesure ?

Existence et unicité de la mesure invariante : Théorème de Perron-Frobenius

Théorème

Perron-Frobenius Soit P la matrice d'une chaîne de Markov **irréductible**. *Alors :*

1. 1 *est une valeur propre simple.*
2. *tout vecteur propre à gauche associé à* 1 *a toutes ses coordonnées de même signe. En particulier, celui de somme* 1 *correspond bien à une distribution de probabilités.*
3. *si la chaîne est apériodique, toute autre valeur propre* λ *vérifie* $|\lambda| < 1$.

En d'autres termes, toute chaîne de Markov irréductible admet une unique mesure invariante.

- Si la chaîne n'est pas irréductible, la partie concernant la monotonie du signe du vecteur propre est encore valable. Par contre, l'espace propre peut être de dimension supérieure : il n'y a plus unicité de la mesure invariante.

Convergence vers la mesure invariante

Théorème

Soit P la matrice d'une chaîne de Markov **irréductible et apériodique** *et μ l'unique mesure invariante associée. Alors, pour tout X_0, $\lim_{n\to+\infty}^{t} X_0 P^n =^t \mu$. De plus, la vitesse de convergence est en $|\lambda_2|^n$, où λ_2 est la valeur propre de valeur absolue maximale parmi les valeurs propres différentes de 1.*

En résumé

Cas possibles

- Si la chaîne est irréductible apériodique, la distribution de S_n tend vers l'unique mesure invariante, et ce quelle que soit la distribution de départ.
- Si la chaîne est irréductible mais périodique, la mesure invariante est unique mais suivant la distribution de départ, il peut ne pas y avoir convergence
- Si la chaîne n'est pas irréductible, il y a plusieurs mesures invariantes. Si elle est apériodique, il y aura bien convergence de la distribution, mais la limit dépend de la distribution de départ.

En pratique

Quand c'est possible, on considère des chaînes irréductibles apériodiques.
Exemple : PageRank et les algorithmes d'optimisation MCMC.

I.4. Un exemple : PageRank

Graphes de états

- On considère le graphe du web, les arêtes étant les pages référencées (dont un attribut est le texte qu'elles contiennent)
- On ajoute une arête orientée de i vers j si la page i pointe vers la page j.
- On considéère une marche aléatoire uniforme, c'est-à-dire que $p_{ij} = \frac{1}{d(i)}$ si l'arête (i, j) est présente.
- La popularité d'une page est définie par la mesure limite de la chaîne de Markov.

Graphes de états

- On considère le graphe du web, les arêtes étant les pages référencées (dont un attribut est le texte qu'elles contiennent)
- On ajoute une arête orientée de i vers j si la page i pointe vers la page j.
- On considéère une marche aléatoire uniforme, c'est-à-dire que $p_{ij} = \frac{1}{d(i)}$ si l'arête (i, j) est présente.
- La popularité d'une page est définie par la mesure limite de la chaîne de Markov.

Problème

Le graphe n'est pas fortement connexe : la chaîne n'est pas irréductible.

Chaîne de Markov modifiée

- n le nombre de sommets, P la matrice de transition définie précédemment, E la matrice dont tous les coefficients valent 1. On définit

$$A = \frac{1-d}{n} E + \frac{d}{n} P$$

- La chaîne définie par A a une probabilité non nulle d'aller de tout état à tout autre état : elle est irréductible apériodique.
- La deuxième valeur propre est de valeur absolue $\leq d$: la covergence se fait à la vitesse d^n.

En pratique

- d est choisi aux environs de 0.85.
- Simuler des marches est aisé car la multiplication par E est une somme et P est creuse
- La convergence des marches est rapide ($0.85^{100} < 1e-7$).
- Il suffit de classer les pages (à refaire de temps en temps pour se mettre à jour) puis de trier celle contenant la bonne chaîne de caractères.
- En réalité, seules les K premières du classement contenant la bonne chaîne sont gardées, puis d'autres sont recrutées par similarité + beaucoup d'autres subtilités publiques ou non !

III. Applications des chaînes de Markov

III.1 Partie publique de PageRank

Graphe considéré

- les noeuds sont les URLs
- une arête relie deux URLs si la première comporte un hyperlien vers la seconde

Principe

La distribution limite peut être considérée comme un classement des URLs.

Application du théorème principal des chaînes de Markov

La chaîne est-elle apériodique ? Oui

La chaîne est-elle irréductible ? Non.

Solution

Soit P la matrice d'adjacence du graphe, N le nombre d'URLs et Q la matrice de même taille dont tous les coefficients valent $\frac{1}{N}$.
Soit

$$P_\beta = (1-\beta)P + \beta Q$$

Pour tout $\beta > 0$, la chaîne est irréductible et apériodique.

Application du théorème principal des chaînes de Markov

- Il suffit donc de choisir β faible (en pratique 0.15) et de déterminer la distribution limite.

  Problème Déterminer un vecteur propre sur un matrice de millions de noeuds est trop long.

  Solution Utiliser ${}^tX_n = {}^tX_0 P_\beta$ et que la convergence est exponentielle en $|\lambda_2|$ pour approximer la distribution limite par X_{50}.

- Il faut cependant calculer des produits $matrice \times vecteur$ pour des tailles de plusieurs millions. Ceci est possible car, si on pose X_0 le vecteur dont toutes les coordonnées valent $\frac{1}{N}$,

$$ {}^tX_n P_\beta = (1-\beta)\,{}^tX_n P + \beta\,{}^tX_0 $$

  Le premier produit peut être effectué efficacement car la matrice P est creuse.

III.2 Algorithme MCMC : Metropolis-Hastings

Chaîne de Markov sur un espace continu

Une suite de variables aléatoires $(X_i)_{i\geq 0}$ définies sur un ensemble $\mathcal{X}$ est une chaîne de Markov si $X_{i+1}|X_0,\ldots,X_i$ suit la même loi que $X_{i+1}|X_i$.

La fonction K telle que

$$X_{i+1}|X_0,\ldots,X_i \sim K(X_i, X_{i+1})$$

est appelé noyau markovien. Si f_i désigne la densité de X_i, on a alors

$$f_{i+1}(y) = \int_{\mathcal{X}} K(x,y) f_i(x)\, dx$$

Exemple : La marche aléatoire sur $\mathbb{R}$ définie par $X_{i+1} = X_i + \epsilon_i$ avec $\epsilon_i \sim \mathcal{N}(0,1)$ est une chaîne de Markov dont le noyau $K(X_i, X_{i+1})$ correspond à la densité de $\mathcal{N}(X_i, 1)$.

Loi limite

Si la chaîne est irréductible, il existe une unique loi stationnaire f qui est presque surement la loi limite de la chaîne de Markov.

Théorème ergodique

On considère une chaîne de Markov de distribution limite f. Pour toute fonction intégrable h,

$$\lim_{n \to +\infty} \frac{1}{n} \sum_{i=0}^{n-1} h(X_i) = \int_{\mathcal{X}} h(x) f(x) \, dx$$

En particulier, en prenant pour h la fonction indicatrice d'un sous-ensemble A de $\mathcal{X}$, on obtient que la mesure de A suivant f est égale à la proportion des éléments de la chaîne appartenant à A quand la chaîne devient infinie.
En d'autres termes, **générer une chaîne suffisamment longue de noyau K revient à simuler suivant f.**

Principe

- On considère une distribution f sur $\mathbb{R}$ suivant laquelle on souhaite échantillonner.

Idée

Si on peut construire une chaîne de Markov irréductible dont la distribution invariante est f, il suffit de générer une trajectoire très longue pour récupérer un échantillon distribué suivant f.

Algorithme de Metropolis-Hastings symétrique

On considère une distribution symétrique g suivant laquelle on sait simuler (une loi uniforme sur $[-\delta, \delta]$, une loi normale centrée ...)
Etant donné x_n,

1. Générer ϵ_n suivant g et $y_n = x_n + \epsilon_n$
2. Choisir

$$x_{n+1} = \begin{cases} y_n & \text{avec probabilité} \quad \rho(x_n, y_n) \\ x_n & \text{avec probabilité} \quad 1 - \rho(x_n, y_n) \end{cases}$$

où

$$\rho(x, y) = \min\left\{\frac{f(y)}{f(x)}, 1\right\}$$

Algorithme de Metropolis-Hastings

Théorème

Les (x_n) forment une chaîne de Markov. Si q est tel que cette chaîne est irréductible, sa distribution limite est f.

- Le choix de q n'influe pas sur le fait qu'il y a convergence, mais il influe sur la vitesse de celle-ci. Certains choix sont privilégiés.
- Il est possible de généraliser la démarche pour des distributions non définies sur $\mathbb{R}$.

Exemple

- jeu de données `esoph` sous `R` : nombre de cancer de l'oesophage et de patients sains dans un échantillon stratifié suivant l'âge, la consommation d'alcool et la consommation de tabac.
- Y_i la variable aléatoire correspondant à l'indicatrice du fait que l'individu i développe un cancer de l'oesophage.
- modèle de régression logistique :

$$\log\left(\frac{\mathbb{P}(Y_i = 1)}{1 - \mathbb{P}(Y_i = 1)}\right) = \alpha + \beta Age_i + \gamma Tab_i + \delta Alc_i$$

Question

Trouver un intervalle de confiance de niveau 95% pour la probabilité de développer un cancer pour un individu dont les variables Age_i, Tab_i et Alc_i sont connues.

Exemple

- $\boldsymbol{\theta} = (\alpha, \beta, \gamma, \delta)$ le vecteur des paramètres, $\mathbf{X}_i = (1, Age_i, Tab_i, Alc_i)$ le vecteur des données de l'individu i. La vraisemblance est

$$\log \mathcal{L}(\boldsymbol{\theta}) = \sum_i \log \frac{exp({}^t\boldsymbol{\theta}\, X_i)}{1 + exp({}^t\boldsymbol{\theta}\, X_i)}$$

- On se place un cadre bayésien, avec une loi à priori pour laquelle les quatre coefficients sont indépendants, de loi normale centrée réduite
- ϕ la densité de la gaussienne centrée réduite

$$\mathbb{P}(\boldsymbol{\theta} \mid \mathbf{X}) \propto \prod_i \frac{exp({}^t\boldsymbol{\theta}\, X_i)}{1 + exp({}^t\boldsymbol{\theta}\, X_i)} \times \prod_{k=1}^{4} \log \phi(\theta_k)$$

On cherche à simuler suivant cette dernière distribution pour obtenir des intervalles de confiance.

Exemple

- L'estimateur du maximum de vraisemblance $\hat{\theta}$ peut être déterminé.

```
> model <- glm(cbind(ncases,ncontrols) ~ unclass(agegp)+unclass(alcgp)+
> EMV <-  model$coefficients
> EMV
```

```
  (Intercept) unclass(agegp) unclass(alcgp) unclass(tobgp)
   -5.5959444      0.5286674      0.6938248      0.2744565
```

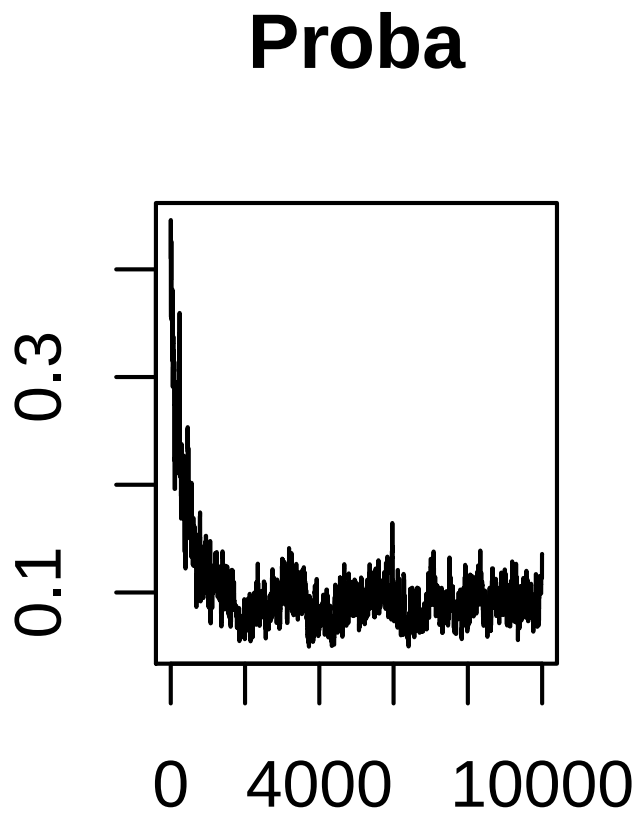
Exemple

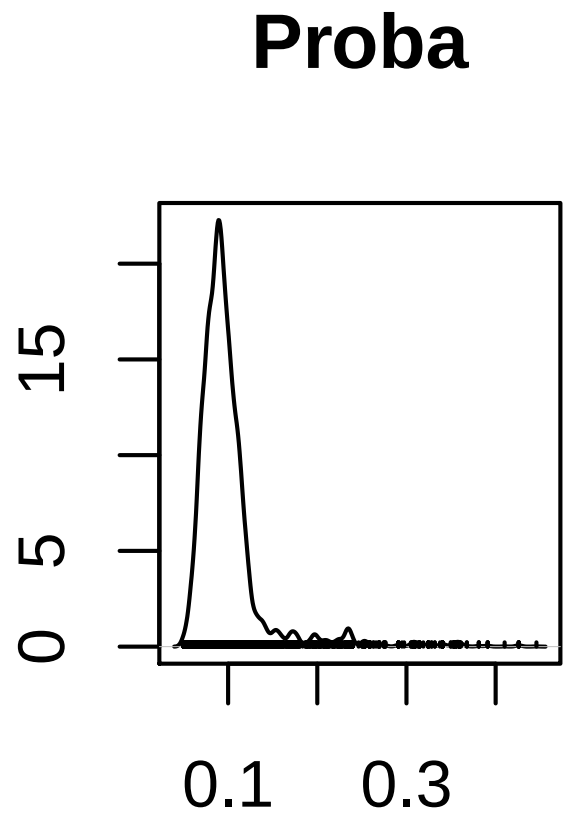
```
> #Calcul de la vraisemblance à une constante près pour une valeur de Theta
> logit <- function(x){
+   return(exp(x)/(1+exp(x)))
+ }
> LogLikelihood <- function(Theta, data){
+   logL <- 0
+   coeffmatrix <- cbind(1,data$agegp,data$alcgp,data$tobgp) #matrice des coefficients correspondant à chaque possibilité
+   for (i in 1:dim(data)[1]){
+     proba <- logit(t(Theta)%*%coeffmatrix[i,])
+     logL <- logL+log(proba)*data$ncases[i]+log(1-proba)*data$ncontrols[i]
+   }
+   logL <- logL + sum(log(dnorm(Theta)))   # ajouter la loi à priori où chacune prise comme loi normale central réduite
+   return(logL)
+ }
> trajectoryRW <- function(Nsim,data,width,X0){
+
+   X <- matrix(X0,1,4)
+   proba <- c()
+   for (n in 2:Nsim){
+     Y <- runif(4,-width,width)
+     rho <- exp(LogLikelihood(X[n-1,]+Y,data) - LogLikelihood(X[n-1,],data))
+     X <- rbind(X, X[n-1,] + Y * (runif(1)<rho))
+     if (floor(n/100)==(n/100)) { print(n)}
+     s <- t(X[n,])%*%c(1,1,3,1)
+     proba <- c(proba,exp(s)/(1+exp(s)))
+   }
+   return(list(X=X,proba=proba))
+ }
> data <- esoph
> data$tobgp <- unclass(data$tobgp)
> data$alcgp <- unclass(data$alcgp)
> data$agegp <- unclass(data$agegp)
> trajectory <- trajectoryRW(10000,data,.1,c(0,0,0,0))
[1] 100
[1] 200
[1] 300
[1] 400
[1] 500
[1] 600
[1] 700
[1] 800
[1] 900
```

Exemple

On considère un non-fumeur ($F_i = 1$), consommateur moyen d'alcool ($A_i = 3$) de 30 ans. On peut déterminer à chaque étape de la simulation la probabilité de développer un cancer de l'oesophage.
On récupère alors une simulation de la distribution de cette probabilité

```
> plot(prRW,main='Proba')
```





Calibration de la proposition

- des propositions trop proches du point courant vont favoriser des marches qui restent toujours dans la même région : risque d'avoir raté des pans entiers de l'espace à explorer au moment où on arrête la simulation.
  De plus, si la distribution est multimodale, la marche risque de rester enfermée dans un mode car il faudrait accepter successivement un grand nombre de sauts défavorables pour en sortir (ce qui arrive avec probabilité non nulle mais tellement faible qu'on ne le voit jamais).
- des propositions à trop longue distance ne sont pas forcément faciles à formuler. De plus, lorsqu'on est proche d'un maximum local de f, on risque d'y rester très longtemps avant d'accepter un mouvement, ce qui entraîne une chaîne très fortement corrélée.

III.3 Recuit simulé

Mesure de Gibbs

- On considère une fonction f que l'on souhaite minimiser
- La mesure de Gibbs associée à f et à la température T est définie par

$$\mu_T(x) = \frac{1}{Z_T} e^{-f(x)/T} \text{ avec } Z_T = \int_x e^{-f(x)/T} dx$$

- La mesure μ_T est maximale clairement en les minima de f. Cependant, si T est très faible, elle est beaucoup plus piquée que f. En effet,

$$\frac{\mu_T(x)}{\mu_T(x^*)} = exp(\frac{f(x^*) - f(x)}{T})$$

- A la limite, $\lim_{T \to 0} \mu_T(x) = 0$ si x n'est pas un minimum de f et $\lim_{T \to 0} \mu_T(x) = 1/k$ si f admet k minimum et que x est l'un d'eux.

Mesure de Gibbs

Plutôt que de simuler suivant f afin de trouver son minimum, il est par conséquent intéressant de simuler suivant μ_T avec un T petit. En effet, les cuvettes de f correspondant aux minima globaux ont alors une plus grande probabilité d'apparition. La difficulté apparente est le calcul de Z_T mais on peut s'en passer en simulant suivant un algorithme de Metropolis-Hastings.

Etant donné x_n,

1. Générer $\xi_n \sim g(\xi)$, g symétrique
2. Choisir

$$x_{n+1} = \begin{cases} x_n + \xi_n & \text{avec probabilité} \quad \rho(x_n, x_n + \xi_n) \\ x_n & \text{avec probabilité} \quad 1 - \rho(x_n, y_n) \end{cases}$$

où

$$\rho(x, y) = \min \left\{ \exp\left(\frac{f(x_n + \xi_n) - f(x_n)}{T}\right), 1 \right\}$$

Recuit simulé

- On reprend l'idée précédente en cherchant à simuler non pas toute une suite suivant μ_T, mais une suite (x_n) telle que (x_n) soit distribuée suivant μ_{T_n}, avec T_n tendant vers 0. On s'attend en effet à ce que dans ce cas, x_n tende vers un minimum global.

Etant donné x_n,

1. Générer $\xi_n \sim g(\xi)$, g symétrique
2. Choisir

$$x_{n+1} = \begin{cases} x_n + \xi_n & \text{avec probabilité} \quad \rho(x_n, x_n + \xi_n) \\ x_n & \text{avec probabilité} \quad 1 - \rho(x_n, y_n) \end{cases}$$

où

$$\rho(x, y) = \min\left\{ \exp\left(\frac{f(x_n + \xi_n) - f(x_n)}{T_n}\right), 1\right\}$$

Recuit simulé

- Algorithme très semblable à celui de Metropolis-Hastings par marche aléatoire : si le mouvement proposé représente un gain, il est systématiquement accepté, s'il représente une perte, il est accepté avec un probabilité d'autant plus petite que la perte est importante.
- La probabilité d'acceptation pour un perte donnée diminue avec l'allongement de la chaîne. En d'autres termes, la chaîne va accepter avec assez grande probabilité des mouvements non croissants au début de son mouvement, et les acceptera de plus en plus difficilement par la suite.

Théorème

Pour toute fonction f, il existe une constante C_f telle que $T_n \leq \frac{C_f}{\log n}$ entraîne que x_n tend vers un minimum global de f avec probabilité 1.

- Si la chaîne finit théoriquement toujours par converger, on ne sait pas quand elle l'a effectivement fait. Elle peut passer un temps très long dans un minimum local avant de finalement découvrir une nouvelle région plus intéressante.
- Un choix de forme logarithmique $T_n = \frac{C}{\log(n)}$ converge lentement mais a de meilleures chances de ne pas rester enfermée dans un minimum local, alors qu'un choix géométrique de la forme $T_n = \alpha^n T$, α proche de 1, donne plus rapidement une impression de convergence.

Exemple : le voyageur de commerce

Problème

- Un voyageur doit passer par n villes numérotées de 1 à n et revenir à son point de départ, la distance entre les villes étant donnée par la fonction d.
- Quelle est le meilleur choix d'itinéraire, c'est-à-dire la permutation σ minimisant $f(\sigma) = \sum_{i=1}^{n-1} d(\sigma(i), \sigma(i+1))$?

- Le problème est NP-complet : une heuristique est nécessaire.
- L'approche MCMC donne de bons résultats.

Exemple : le voyageur de commerce

```
> M <- matrix(runif(10000,.5,1.5),100,100)
> M <- M+t(M)                    #M est une matrice symétrique dont tout coe
> diag(M) <- 0
> for (i in 1:99){
+   M[i,i+1] <- 1
+   M[i+1,i] <- 1
+ }
> M[100,1] <- 1
> M[1,100] <- 1
```

Exemple : le voyageur de commerce

```
> cost <- function(M,sigma){ #cout du chemin correspondant au chemin si
+   cost <- M[sigma[100],sigma[1]]
+   for (i in 1:99){
+     cost <- cost + M[sigma[i],sigma[i+1]]
+   }
+   return(cost)
+ }
> shuffle <- function(sigma){
+   newsigma <- sigma
+   exchange <- sample(c(1:100),2,replace=FALSE)
+   newsigma[exchange[1]] <- sigma[exchange[2]]
+   newsigma[exchange[2]] <- sigma[exchange[1]]
+   return(newsigma)
+ }
```

Exemple : le voyageur de commerce

```
> simulatedAnnealing <- function(M,Temp){ #recuit simulé pour la matric
+
+   sigma <- sample(c(1:100),100,replace=FALSE)
+   cost <- cost(M,sigma)
+   costvector <- c(cost)
+
+   for (n in 1:length(Temp)){
+      newsigma <- shuffle(sigma)
+      newcost <- cost(M,newsigma)
+      costvector <- c(costvector,newcost)
+
+      rho <- exp((-cost(M,newsigma) + cost(M,sigma))/Temp[n])  #car on
+
+      if (runif(1)<rho){
+        sigma <- newsigma
+        cost <- newcost
+      }
+   }
+
+   return(costvector)
+ }
```

Exemple : le voyageur de commerce

```
> Temp <- 1 / log(1:20000)
> traj <- simulatedAnnealing(M,Temp)
> codatraj <- as.mcmc(traj)
> plot(codatraj)
```