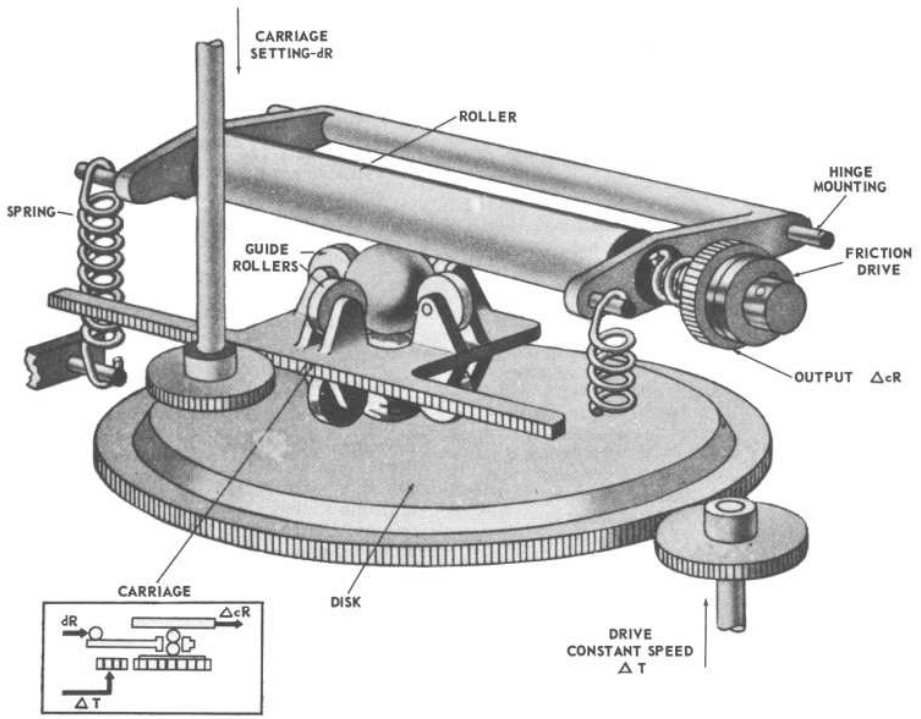
Cours 9: NP-complétude. Suite.



Olivier Bournez
bournez@lix.polytechnique.fr

Ecole Polytechnique
INF412

Au menu

Exemples de preuve de NP-complétude

Quelques autres problèmes NP-complets célèbres

Compléments sur la NP-complétude

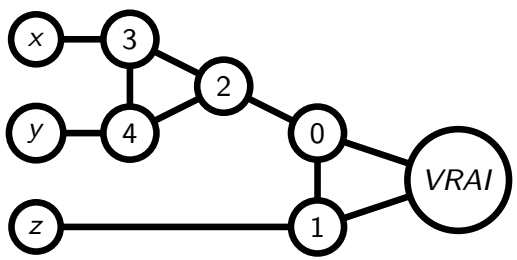
Gérer la NP-complétude

Théorème

3-COLORABILITE *est* NP-*complet.*

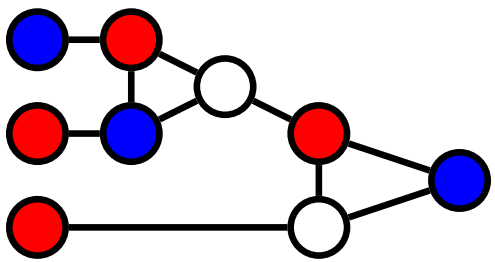
Preuve

- 3-COLORABILITE est dans NP : déjà vu.
- 3-SAT $\leq$ 3-COLORABILITE :
  - On se donne donc une conjonction F de clauses à 3 littéraux, et il nous faut à partir de là construire un graphe.
    - Il faut parvenir à traduire deux contraintes : une variable peut prendre la valeur 0 ou 1 d'une part, et les règles d'évaluation d'une clause d'autre part.
  - On construit un graphe ayant $3+2n+5m$ sommets, où n est le nombre de variables, m de clauses.
    1. Les trois premiers, notés *VRAI*, *FAUX*, *NSP* sont reliés deux à deux en triangle.
    2. On associe pour chaque variable x_i un sommet x_i et un sommet $\neg x_i$: on relie deux à deux en triangle les sommets x_i, $\neg x_i$, et *NSP*.
    3. On ajoute 5 sommets pour chaque clause $x \vee y \vee z$ selon le dessin suivant :



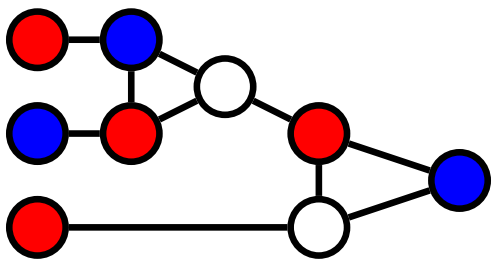
Le gadget

- Si l'on impose que les trois sommets à gauche sont soit bleus soit rouges,
  - alors, on peut colorer le sommet le plus à droite en bleu si et seulement si au moins un sommet à gauche est en bleu.



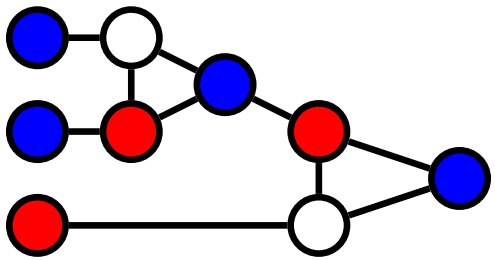
Le gadget

- Si l'on impose que les trois sommets à gauche sont soit bleus soit rouges,
  - alors, on peut colorer le sommet le plus à droite en bleu si et seulement si au moins un sommet à gauche est en bleu.



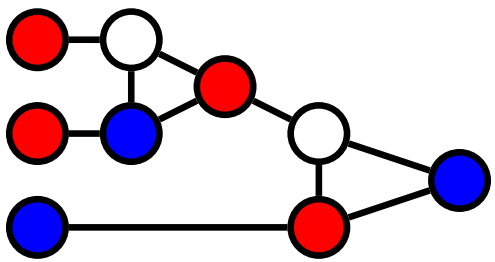
Le gadget

- Si l'on impose que les trois sommets à gauche sont soit bleus soit rouges,
  - alors, on peut colorer le sommet le plus à droite en bleu si et seulement si au moins un sommet à gauche est en bleu.



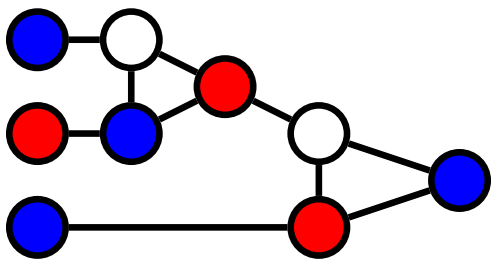
Le gadget

- Si l'on impose que les trois sommets à gauche sont soit bleus soit rouges,
  - alors, on peut colorer le sommet le plus à droite en bleu si et seulement si au moins un sommet à gauche est en bleu.



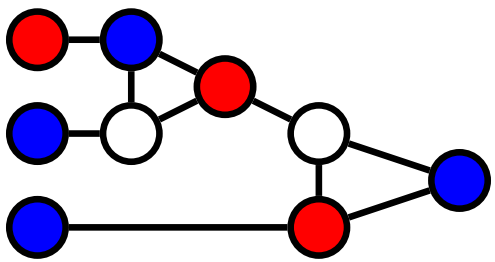
Le gadget

- Si l'on impose que les trois sommets à gauche sont soit bleus soit rouges,
  - alors, on peut colorer le sommet le plus à droite en bleu si et seulement si au moins un sommet à gauche est en bleu.



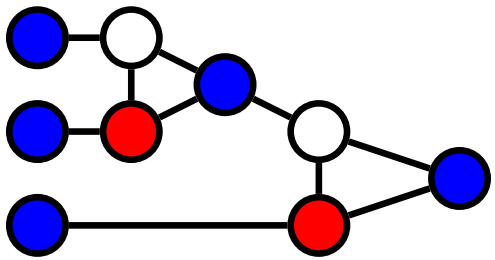
Le gadget

- Si l'on impose que les trois sommets à gauche sont soit bleus soit rouges,
  - alors, on peut colorer le sommet le plus à droite en bleu si et seulement si au moins un sommet à gauche est en bleu.



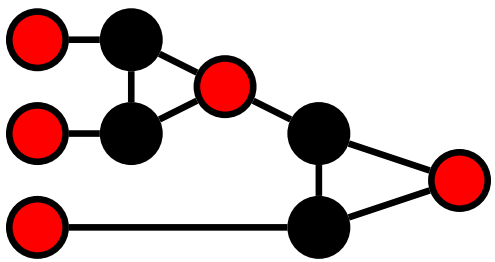
Le gadget

- Si l'on impose que les trois sommets à gauche sont soit bleus soit rouges,
  - alors, on peut colorer le sommet le plus à droite en bleu si et seulement si au moins un sommet à gauche est en bleu.



Le gadget

- Si l'on impose que les trois sommets à gauche sont soit bleus soit rouges,
  - alors, on peut colorer le sommet le plus à droite en bleu si et seulement si au moins un sommet à gauche est en bleu.



- Si F est satisfiable, on peut colorier le graphe $G(F)$ avec 3-couleurs :
  - mettre les variables vraies avec la couleur du sommet *VRAI*, les variables fausses avec la couleur du sommet *FAUX*, et compléter.
- Si le graphe $G(F)$ est coloriable avec trois couleurs :
  - Le triangle 1. assure que *VRAI* et *FAUX* ne sont pas de la même couleur.
  - Le triangle 2. assure que x_i et $\neg x_i$ ne sont pas de la même couleur pour chaque variable x_i, et soit de la couleur de *VRAI* ou de la couleur de *FAUX*.
  - Le gadget 3. assure que pour chaque clause C au moins x ou y ou z est de la couleur de *VRAI*.
  - En prenant pour vraies toutes les variables de la couleur de *VRAI*, on obtient une affectation des variables qui satisfait la formule F.
- Bref : $F \in$ 3-SAT ssi $G(F) \in$ 3-COLORABILITE.
- La fonction qui à F associe $G(F)$ se calcule bien en temps polynomial.

Au menu

Exemples de preuve de NP-complétude

Quelques autres problèmes NP-complets célèbres

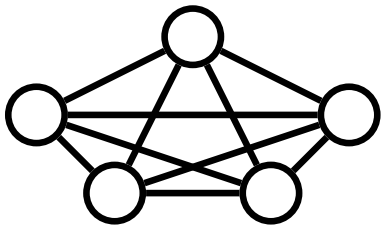
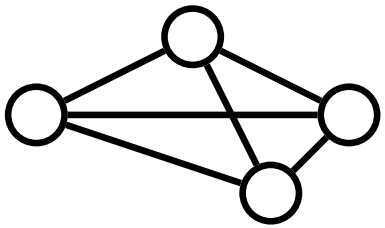
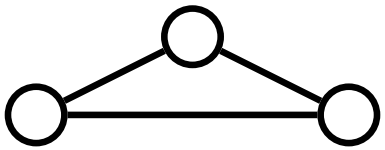
Compléments sur la NP-complétude

Gérer la NP-complétude

Sur les graphes

- *CLIQUE*

Donnée: Un graphe $G = (V, E)$ non-orienté et un entier k.
Réponse: Décider s'il existe $V' \subset V$, avec $|V'| = k$, tel que $u, v \in V' \Rightarrow (u, v) \in E$.

K_5 :  K_4 :  K_3 : 

En passant au complémentaire sur les arêtes :

- *STABLE*

Donnée: Un graphe $G = (V, E)$ non-orienté et un entier k.
Réponse: Décider s'il existe $V' \subset V$, avec $|V'| = k$, tel que $u, v \in V' \Rightarrow (u, v) \notin E$.

En passant au complémentaire sur les sommets :

- *RECOUVREMENTDESOMMETS*

Donnée: Un graphe $G = (V, E)$ non-orienté et un entier k.
Réponse: Décider s'il existe $V' \subset V$, avec $|V'| = k$, tel que toute arête de G ait au moins une extrémité dans V'.

Sur les graphes

- *CIRCUITHAMILTONIEN*

Donnée: Un graphe $G = (V, E)$ (non-orienté).

Réponse: Décider s'il existe un **circuit hamiltonien**, c'est-à-dire un chemin de G passant une fois et une seule par chacun des sommets et revenant à son point de départ.

- *VOYAGEURDECOMMERCE*

Donnée: Un couple (n, M), où M est une matrice $n \times n$ d'entiers et un entier k.

Réponse: Décider s'il existe une permutation π de $[1, 2, \cdots, n]$ telle que

$$\sum_{1 \leq i \leq n} M_{\pi(i)\pi(i+1)} \leq k.$$

- Ce problème porte ce nom, car on peut voir cela comme l'établissement de la tournée d'un voyageur de commerce devant visiter n villes, dont les distances sont données par la matrice M de façon à faire moins de k kilomètres.

Sur les entiers

- *SOMMEDESOUS – ENSEMBLE*

Donnée: Une suite finie d'entiers $x_1, x_2, \ldots, x_n$ et un entier t.
Réponse: Décider s'il existe $E \subset \{1, 2, \ldots, n\}$ tel que $\sum_{i \in E} x_i = t$.

- *PARTITION*

Donnée: Une suite finie d'entiers $x_1, x_2, \ldots, x_n$.
Réponse: Décider s'il existe $E \subset \{1, 2, \ldots, n\}$ tel que $\sum_{i \in E} x_i = \sum_{i \notin E} x_i$.

- *SACADOS*

Donnée: Un ensemble de poids $a_1, \cdots, a_n$, un ensemble de valeurs $v_1, \cdots, v_n$, un poids limite A, et un entier V.
Réponse: Décider s'il existe $E' \subset \{1, 2, \cdots, n\}$ tel que $\sum_{i \in E'} a_i \leq A$ et $\sum_{i \in E'} v_i \geq V$.

Au menu

Exemples de preuve de NP-complétude

Quelques autres problèmes NP-complets célèbres

Compléments sur la NP-complétude

Gérer la NP-complétude

Plus précisément

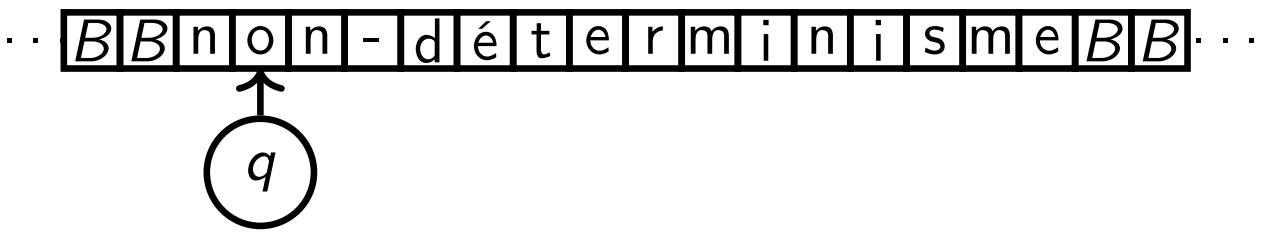
Compléments sur la NP-complétude

- Non-déterminisme
- Théorème de Cook-Levin
- Digression
- Preuve du théorème de Cook-Levin
- Décision vs Construction

Machines de Turing non-déterministes

- Le concept de machine de Turing non-déterministe est une variante de la notion de machine de Turing :
  - la définition d'une machine de Turing non-déterministe est exactement comme celle de la notion de machine de Turing (déterministe) sauf sur un point :
    - δ n'est plus une fonction (possiblement partielle) de $Q \times \Gamma$ dans $Q \times \Gamma \times \{\leftarrow, \rightarrow\}$, mais une relation de la forme

$$\delta \subset (Q \times \Gamma) \times (Q \times \Gamma \times \{\leftarrow, \rightarrow\}).$$

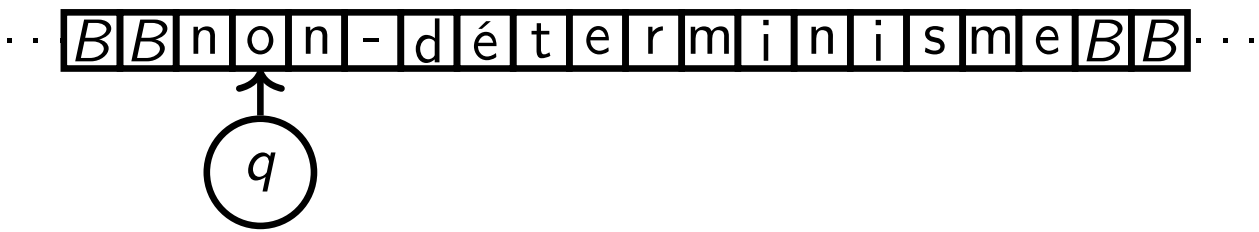


- En d'autres termes, pour un état et une lettre lue en face de la tête de lecture donnée, δ ne définit pas un seul triplet de $Q \times \Gamma \times \{\leftarrow, \rightarrow\}$, mais un ensemble de triplets.

Machines de Turing non-déterministes

- Le concept de machine de Turing non-déterministe est une variante de la notion de machine de Turing :
  - la définition d'une machine de Turing non-déterministe est exactement comme celle de la notion de machine de Turing (déterministe) sauf sur un point :
    - δ n'est plus une fonction (possiblement partielle) de $Q \times \Gamma$ dans $Q \times \Gamma \times \{\leftarrow, \rightarrow\}$, mais une relation de la forme

$$\delta \subset (Q \times \Gamma) \times (Q \times \Gamma \times \{\leftarrow, \rightarrow\}).$$



- En d'autres termes, pour un état et une lettre lue en face de la tête de lecture donnée, δ ne définit pas un seul triplet de $Q \times \Gamma \times \{\leftarrow, \rightarrow\}$, mais un ensemble de triplets.
  - Intuitivement, lors d'une exécution la machine a la possibilité de choisir n'importe quel triplet.

- On peut passer de la configuration C à la configuration successeur C' si et seulement si on peut passer de C à C' (ce que nous notons $C \vdash C'$) avec les définitions du cours 4, mais en remplaçant $\delta(q,a) = (q',a',m')$ par $((q,a),(q',a',m')) \in \delta$.

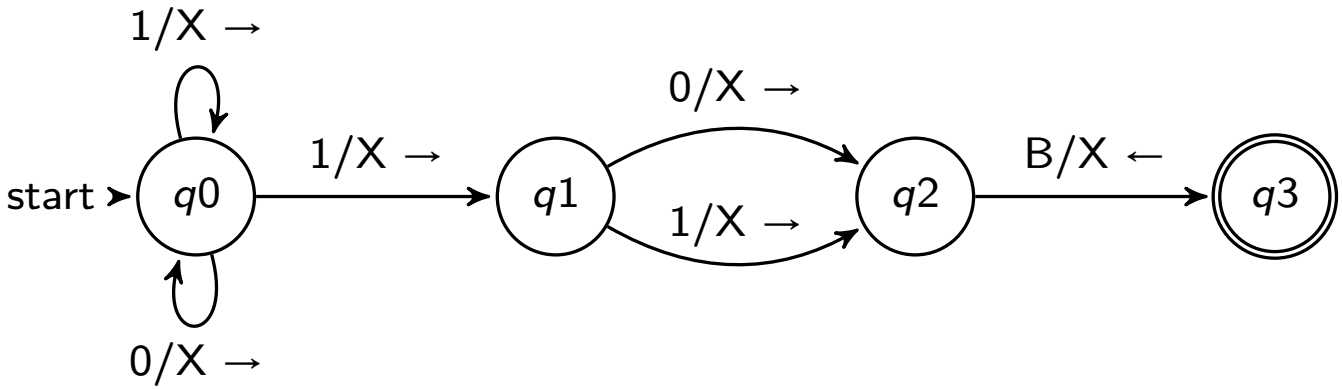
- La différence est qu'une machine de Turing non-déterministe n'a pas une exécution unique sur une entrée w, mais éventuellement plusieurs :

- On peut passer de la configuration C à la configuration successeur C' si et seulement si on peut passer de C à C' (ce que nous notons $C \vdash C'$) avec les définitions du cours 4, mais en remplaçant $\delta(q,a) = (q',a',m')$ par $((q,a),(q',a',m')) \in \delta$.
  - Les autres définitions sont alors essentiellement inchangées, et comme pour les machines de Turing déterministes.
- La différence est qu'une machine de Turing non-déterministe n'a pas une exécution unique sur une entrée w, mais éventuellement plusieurs :

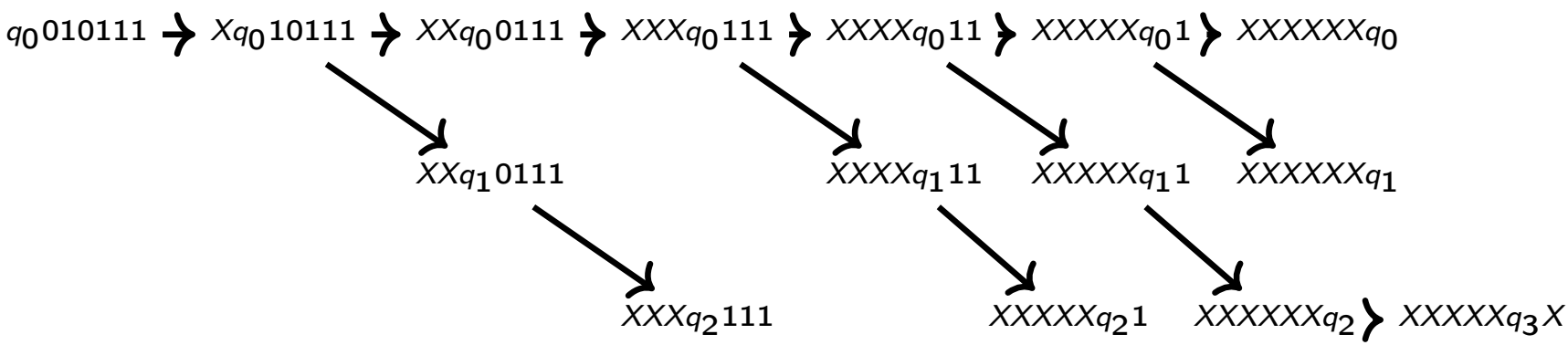
- On peut passer de la configuration C à la configuration successeur C' si et seulement si on peut passer de C à C' (ce que nous notons $C \vdash C'$) avec les définitions du cours 4, mais en remplaçant $\delta(q,a) = (q',a',m')$ par $((q,a),(q',a',m')) \in \delta$.
  - Les autres définitions sont alors essentiellement inchangées, et comme pour les machines de Turing déterministes.
- La différence est qu'une machine de Turing non-déterministe n'a pas une exécution unique sur une entrée w, mais éventuellement plusieurs :
  - en fait, les exécutions de la machine sur un mot w donnent lieu à un arbre de possibilités (arbre de dérivations), et l'idée est qu'on accepte un mot si l'une des branches de cet arbre contient une configuration acceptante.

Une façon de voir les exécutions

- Exemple :



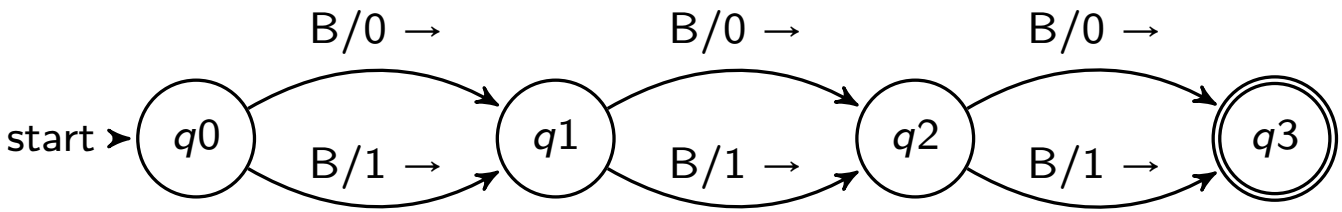
- Exécutions sur $w = 010111$:



- On accepte un mot si et seulement s'il y a une exécution sur ce mot partant de l'état initial qui termine en l'état acceptant.

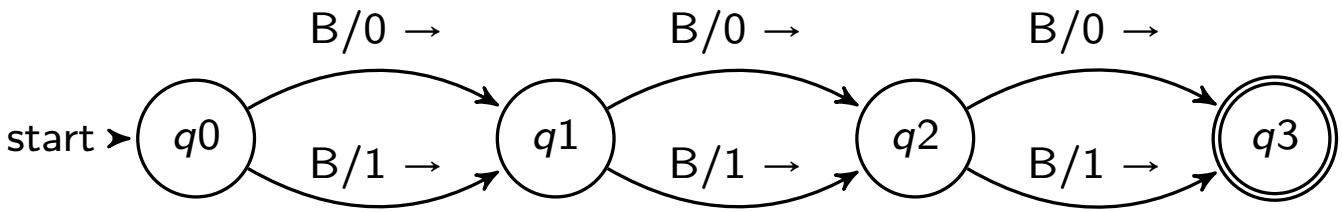
Générer un mot de longueur k

- Que peut-contenir le ruban de la machine de Turing non-déterministe suivante ?



Générer un mot de longueur k

- Que peut-contenir le ruban de la machine de Turing non-déterministe suivante ?



  - Réponse : tous les mots de longueur 3 sur l'alphabet $\Sigma = \{0, 1\}$.

Exercice : Généralisations

- On fixe un polynome $p(n)$. Produire une machine de Turing non-déterministe qui sur un mot w accepte, et dont le ruban au moment d'accepter contient n'importe quel mot sur l'alphabet $\Sigma = \{0, 1\}$ de longueur $p(|w|)$.

- Même chose, mais pour les mots de longueur $\leq p(|w|)$.

Du point de vue de la calculabilité

Théorème

Une machine de Turing non-déterministe peut être simulée par une machine de Turing déterministe :

- *un langage L est accepté par une machine de Turing non-déterministe si et seulement si il est accepté par une machine de Turing (déterministe).*

Du point de vue de la calculabilité

Théorème

Une machine de Turing non-déterministe peut être simulée par une machine de Turing déterministe :

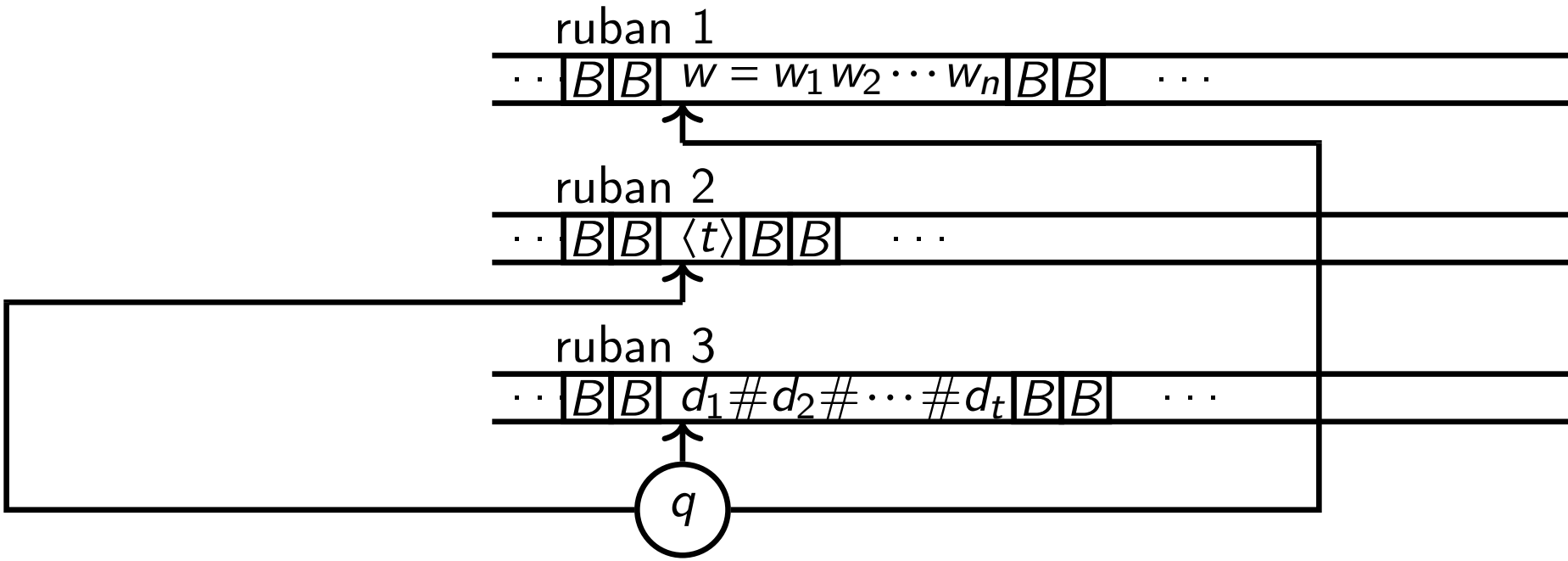
- *un langage L est accepté par une machine de Turing non-déterministe si et seulement si il est accepté par une machine de Turing (déterministe).*

- Du point de vue de la calculabilité, les machines de Turing non-déterministes ne sont donc qu'un modèle de plus vérifiant la thèse de Church.

Illustration graphique de la démonstration :

Soit r le degré de non-déterminisme de la machine :

$$r = max_{q \in Q, a \in \Gamma} |\{((q,a),(q',a',m)) \in \delta\}|.$$



Pour simuler la machine non-déterministe M, faire :

- pour $t = 1, 2, \cdots$
  - pour $d_1, d_2, \cdots, d_t \in \{1, 2, \cdots, r\}$, où
    - Simuler t étapes de la machine non déterministe sur l'entrée w en fixant les choix non-déterministes selon $d_1, d_2, \cdots, d_t$.
    - Si cette simulation accepte alors accepter et terminer.

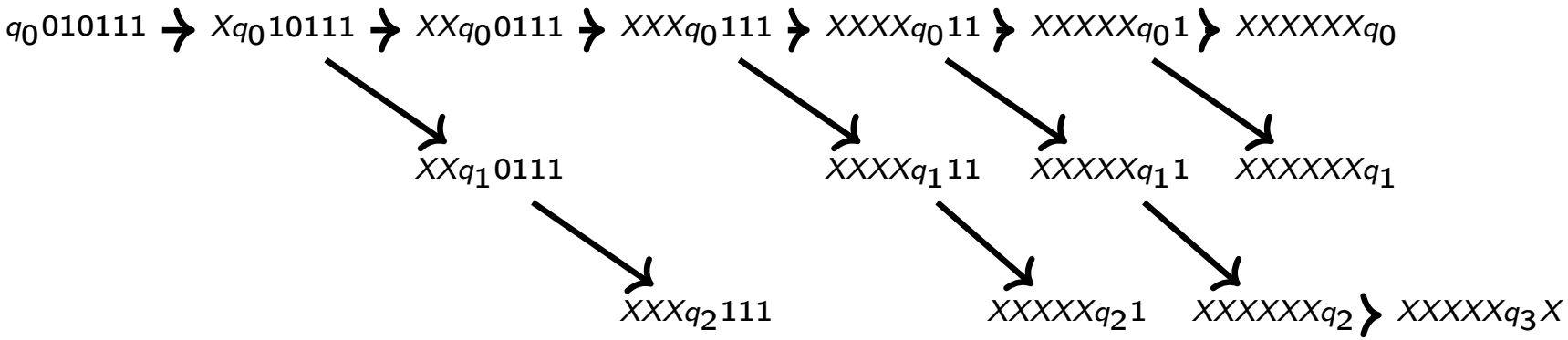
- Démonstration formelle :
  - la relation de transition de la machine non déterministe M est nécessairement de **degré de non determinisme** borné : c'est-à-dire, le nombre

$$r = max_{q \in Q, a \in \Gamma} |\{((q,a),(q',a',m)) \in \delta\}|$$

    est fini ($|.|$ désigne le cardinal).
  - Supposons que pour chaque paire (q,a) on numérote les choix parmi la relation de transition de la machine non déterministe M de 1 à (au plus) r :
    - Pour décrire les choix non déterministes effectués dans un calcul jusqu'au temps t, il suffit de donner une suite $d_1, d_2, \cdots, d_t$ de t nombres entre 1 et (au plus) r.
  - On peut alors construire une machine de Turing (déterministe) qui simule la machine M de la façon suivante : (on peut voir cela comme effectuer un parcours en largeur des possibilités en mémorisant la pile)
    - pour $t = 1, 2, \cdots$, elle énumère toutes les suites $d_1, d_2, \cdots, d_t$ de longueur t d'entiers entre 1 et r.
    - Pour chacune de ces suites $d_1, d_2, \cdots, d_t$, elle simule t étapes de la machine M en utilisant les choix donnés par la suite pour résoudre chaque choix non déterministe de M.
    - La machine s'arrête dès qu'elle trouve t et une suite telle que M atteint une configuration acceptante.

Mesurer la complexité ?

- On dit que la machine **accepte** w **en temps** t si t est le minimum du nombre d'étapes que la machine utilise sur w sur chacune des branches de son arbre de dérivations pour atteindre l'état accepteur.



Vers la classe NP

- Soit $t : \mathbb{N} \to \mathbb{N}$ une fonction.

- On note NTIME(t(n)) pour la classe des problèmes (langages) décidés par une machine de Turing non-déterministe en temps $\mathcal{O}(t(n))$, où n est la taille de l'entrée.

Vers la classe NP

- Soit $t : \mathbb{N} \to \mathbb{N}$ une fonction.
- On note NTIME(t(n)) pour la classe des problèmes (langages) décidés par une machine de Turing non-déterministe en temps $\mathcal{O}(t(n))$, où n est la taille de l'entrée.
  - Si on préfère, $L \in$ NTIME(t(n)) s'il y a une machine de Turing M non-déterministe telle que
    - M décide L :
    - En un temps en $\mathcal{O}(t(n))$, où $n = |w|$ est la longueur de w :

Vers la classe NP

- Soit $t : \mathbb{N} \to \mathbb{N}$ une fonction.

- On note NTIME(t(n)) pour la classe des problèmes (langages) décidés par une machine de Turing non-déterministe en temps $\mathcal{O}(t(n))$, où n est la taille de l'entrée.

  - Si on préfère, $L \in$ NTIME(t(n)) s'il y a une machine de Turing M non-déterministe telle que

    - M décide L : pour tout mot w,
      si $w \in L$ alors M possède **un** calcul acceptant sur w ;
      si $w \notin L$ alors **tous** les calculs de M sont refusants ;
    - En un temps en $\mathcal{O}(t(n))$, où $n = |w|$ est la longueur de w :

Vers la classe NP

- Soit $t : \mathbb{N} \to \mathbb{N}$ une fonction.

- On note NTIME(t(n)) pour la classe des problèmes (langages) décidés par une machine de Turing non-déterministe en temps $\mathcal{O}(t(n))$, où n est la taille de l'entrée.

  - Si on préfère, $L \in$ NTIME(t(n)) s'il y a une machine de Turing M non-déterministe telle que

    - M décide L :

    - En un temps en $\mathcal{O}(t(n))$, où $n = |w|$ est la longueur de w : il y a des entiers n_0 et c tels que pour tout mot w avec $|w| \geq n_0$ si $w \in L$ alors M possède **un** calcul acceptant sur w en moins de $c * t(n)$ étapes ;
      si $w \notin L$ alors **tous** les calculs de M refusent en moins de $c * t(n)$ étapes.

NP = Temps polynomial non-déterministe

Théorème

Un problème est dans NP *si et seulement s'il est décidé par une machine de Turing non déterministe en temps polynomial.*

NP = Temps polynomial non-déterministe

Théorème

Un problème est dans NP *si et seulement s'il est décidé par une machine de Turing non déterministe en temps polynomial.*

- Autrement dit :

$$\mathsf{NP} = \bigcup_{k \in \mathbb{N}} \mathsf{NTIME}(n^k).$$

NP = Temps polynomial non-déterministe

Théorème

Un problème est dans NP *si et seulement s'il est décidé par une machine de Turing non déterministe en temps polynomial.*

- Autrement dit :

$$\mathsf{NP} = \bigcup_{k \in \mathbb{N}} \mathsf{NTIME}(n^k).$$

- Autrement dit,
  - la question de savoir si $\mathsf{P} = \mathsf{NP}$ est la question de savoir si tout langage accepté par une machine de Turing non-déterministe en temps polynomial est accepté par une machine de Turing classique (déterministe) en temps polynomial.

Démonstration :

- Considérons un problème A de NP.
  - Soit V le vérificateur associé, qui fonctionne en temps polynomial $p(n)$.
  - On construit une machine de Turing M non déterministe qui, sur un mot w, va produire de façon non déterministe un mot u de longueur $\leq p(n)$ et simuler V sur (w, u) : si V accepte, alors M accepte. Si V refuse, alors M refuse.
- Réciproquement,
  - soit A un problème décidé par une machine de Turing non déterministe M en temps polynomial $p(n)$ de degré de non-déterminisme r.
  - Une suite d'entiers de longueur $p(n)$ entre 1 et r est un certificat valide pour un mot w : étant donné w et un mot u codant une telle suite, un vérificateur V peut facilement vérifier en temps polynomial si la machine M accepte w avec ces choix non déterministes.

Plus précisément

Compléments sur la NP-complétude

- Non-déterminisme
- Théorème de Cook-Levin
- Digression
- Preuve du théorème de Cook-Levin
- Décision vs Construction

Théorème (Cook-Levin)

- *Le problème* SAT *est* NP*-complet.*
- *Le problème* 3-SAT *est* NP*-complet.*

Plus précisément

Compléments sur la NP-complétude

- Non-déterminisme
- Théorème de Cook-Levin
- Digression
- Preuve du théorème de Cook-Levin
- Décision vs Construction

Digression : Exercice

- Produire un problème RE-complet (pour $\leq_m$ ou $\leq$).

Digression : Exercice

- Produire un problème RE-complet (pour $\leq_m$ ou $\leq$).
  - Problème

    L_{univ}

    :

    Donnée: • Le codage $\langle M \rangle$ d'une machine de Turing M
    • et un mot w.

    Réponse: Décider si la machine M accepte le mot w.

Digression : Exercice

- Produire un problème RE-complet (pour $\leq_m$ ou $\leq$).
  - Problème

    L_{univ}

    :

    Donnée: • Le codage $\langle M \rangle$ d'une machine de Turing M
    • et un mot w.

    Réponse: Décider si la machine M accepte le mot w.

  - Il est dans RE.

Digression : Exercice

- Produire un problème RE-complet (pour $\leq_m$ ou $\leq$).
  - Problème

    L_{univ}

    :

    Donnée: • Le codage $\langle M \rangle$ d'une machine de Turing M
    • et un mot w.

    Réponse: Décider si la machine M accepte le mot w.
  - Il est dans RE.
  - Il est RE-difficile :
    - Soit A un problème de RE : il est reconnu par une machine de Turing M.
    - La fonction f qui à w associe $(\langle M \rangle, w)$ vérifie

      $$w \in A \text{ ssi } f(w) \in L_{univ}.$$

    - La fonction f est calculable (en temps polynomial).

Digression : Exercice

- Produire un problème RE-complet (pour $\leq_m$ ou $\leq$).
  - Problème

    L_{univ}

    :

    Donnée: • Le codage $\langle M \rangle$ d'une machine de Turing M
    • et un mot w.

    Réponse: Décider si la machine M accepte le mot w.
  - Il est dans RE.
  - Il est RE-difficile :
    - Soit A un problème de RE : il est reconnu par une machine de Turing M.
    - La fonction f qui à w associe $(\langle M \rangle, w)$ vérifie

      $$w \in A \text{ ssi } f(w) \in L_{univ}.$$

    - La fonction f est calculable (en temps polynomial).
    - On a donc bien $A \leq L_{univ}$.

Digression : Exercice

- Produire un problème NP-complet (pour $\leq$).

Digression : Exercice

- Produire un problème NP-complet (pour $\leq$).

  - Problème *ARTIFICIEL*:

Donnée:
- Le codage $\langle M \rangle$ d'une machine de Turing non déterministe M
- 1^t, un entier écrit en unaire
- et un mot w.

Réponse: Décider si la machine M accepte le mot w en temps t.

Digression : Exercice

- Produire un problème NP-complet (pour $\leq$).
  - Problème *ARTIFICIEL*:

Donnée: • Le codage $\langle M \rangle$ d'une machine de Turing non déterministe M
• 1^t, un entier écrit en unaire
• et un mot w.

Réponse: Décider si la machine M accepte le mot w en temps t.

  - Il est dans NP.

Digression : Exercice

- Produire un problème NP-complet (pour $\leq$).

  - Problème *ARTIFICIEL*:

Donnée: • Le codage $\langle M \rangle$ d'une machine de Turing non déterministe M
• 1^t, un entier écrit en unaire
• et un mot w.

Réponse: Décider si la machine M accepte le mot w en temps t.

  - Il est dans NP.

  - Il est NP-difficile :

    - Soit A un problème de NP : il est reconnu par une machine de Turing non-déterministe M en temps $p(n)$.
    - La fonction f qui à w associe $\langle \langle M \rangle, 1^{p(|w|)}, w \rangle$ vérifie $w \in A$ ssi $f(w) \in \text{ARTIFICIEL}$.
    - La fonction f est calculable en temps polynomial.

Digression : Exercice

- Produire un problème NP-complet (pour $\leq$).

  - Problème *ARTIFICIEL*:

Donnée: • Le codage $\langle M \rangle$ d'une machine de Turing non déterministe M
• 1^t, un entier écrit en unaire
• et un mot w.

Réponse: Décider si la machine M accepte le mot w en temps t.

  - Il est dans NP.

  - Il est NP-difficile :

    - Soit A un problème de NP : il est reconnu par une machine de Turing non-déterministe M en temps $p(n)$.
    - La fonction f qui à w associe $\langle \langle M \rangle, 1^{p(|w|)}, w \rangle$ vérifie $w \in A$ ssi $f(w) \in \text{ARTIFICIEL}$.
    - La fonction f est calculable en temps polynomial.

    - On a donc bien $A \leq \text{ARTIFICIEL}$.

- Remarque :
  - la fonction f utilisée à chaque fois dépend du problème A (de M).
  - on a besoin seulement de la dépendance de f en w.

- Remarque :
  - la fonction f utilisée à chaque fois dépend du problème A (de M).
  - on a besoin seulement de la dépendance de f en w.
- Le vrai intérêt du Théorème de Cook-Levin :

- Remarque :
  - la fonction f utilisée à chaque fois dépend du problème A (de M).
  - on a besoin seulement de la dépendance de f en w.
- Le vrai intérêt du Théorème de Cook-Levin :
  - Le problème *ARTIFICIEL* n'est pas très naturel

- Remarque :
  - la fonction f utilisée à chaque fois dépend du problème A (de M).
  - on a besoin seulement de la dépendance de f en w.
- Le vrai intérêt du Théorème de Cook-Levin :
  - Le problème *ARTIFICIEL* n'est pas très naturel
    - (et la preuve de sa NP-complétude ressemble plutôt à un jeu sur les définitions)

- Remarque :
  - la fonction f utilisée à chaque fois dépend du problème A (de M).
  - on a besoin seulement de la dépendance de f en w.
- Le vrai intérêt du Théorème de Cook-Levin :
  - Le problème *ARTIFICIEL* n'est pas très naturel
    - (et la preuve de sa *NP*-complétude ressemble plutôt à un jeu sur les définitions)
  - Le théorème de Cook-Levin démontre qu'il existe au moins un problème *NP*-complet naturel.

Plus précisément

Compléments sur la NP-complétude

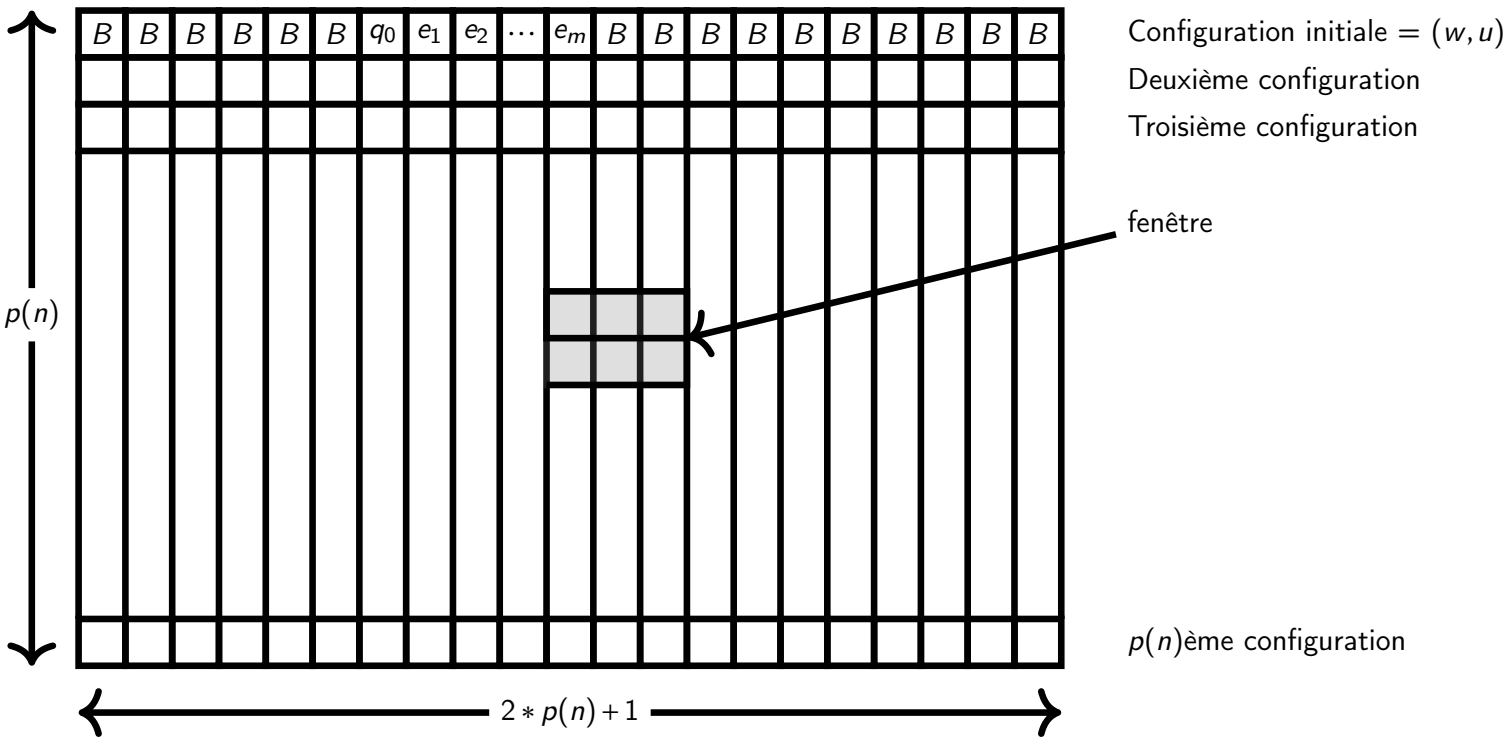
- Non-déterminisme
- Théorème de Cook-Levin
- Digression
- Preuve du théorème de Cook-Levin
- Décision vs Construction

Preuve du théorème de Cook-Levin

- On ne peut pas utiliser la stratégie habituelle, car au moment où l'on prouve ce théorème, aucun autre problème NP-complet n'est connu.
- On revient à la définition de la NP-complétude :
  - On a déjà vu que SAT $\in$ NP.
  - On va montrer que $A \leq \text{SAT}$ pour tout problème $A \in$ NP.
    - Soit A un problème de NP et V un vérificateur associé.
    - L'idée est de construire une formule propositionnelle $\gamma = \gamma_{V,w}(u)$ qui code l'existence d'un calcul accepteur de V sur $w\#u$ (c-à-d. un diagramme espace-temps acceptant), où u est un certificat.
    - En observant que $\gamma = \gamma_{V,w}(u)$ peut bien s'obtenir par un algorithme polynomial à partir de w, on aura prouvé le théorème :

      en effet, on aura $w \in A$ si et seulement si $\gamma_{V,w}(u) \in \text{SAT}$.

- Par hypothèse, V fonctionne en temps polynomial $p(n)$ en la taille n de w.
  - En ce temps là, V ne peut pas déplacer sa tête de lecture de plus de $p(n)$ cases vers la droite, ou de $p(n)$ cases vers la gauche.
  - On peut donc se restreindre à considérer un sous-rectangle de taille $(2 * p(n) + 1) \times p(n)$ du diagramme espace-temps du calcul de V sur $w\#u$:



- Les cases du tableau $T[i,j]$ correspondant au diagramme espace temps sont des éléments de l'ensemble fini $C = \Gamma \cup Q$:
  - pour chaque $1 \leq i \leq p(n)$ et $1 \leq j \leq 2 * p(n) + 1$ et pour chaque $s \in C$, on définit une variable propositionnelle $x_{i,j,s}$.
  - Si $x_{i,j,s}$ vaut 1, cela signifie que la case $T[i,j]$ du tableau contient s.

- La formule γ est la conjonction de 4 formules

$$\mathrm{CELL} \wedge \mathrm{START} \wedge \mathrm{MOVE} \wedge \mathrm{HALT}.$$

- La formule CELL permet de garantir qu'il y a bien un symbole et un seul par case.

$$\mathrm{CELL} = \bigwedge_{1 \leq i \leq p(n), 1 \leq j \leq 2p(n)+1} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{s,t \in C, s \neq t} (\neg x_{i,j,s} \vee \neg x_{i,j,t}) \right) \right].$$

- Si on note $w\#u = w_1 w_2 \dots w_n \# u_1 u_2 \dots u_m$, la formule START permet de garantir que la première ligne correspond bien à la configuration initiale du calcul de V sur $w\#u$.

$$\mathrm{START} = \mathrm{START}(u) =$$

$$x_{1,1,B} \wedge x_{1,2,B} \wedge \dots \wedge x_{1,p(n)+1,q_0} \wedge x_{1,p(n)+2,w_1} \wedge \dots \wedge x_{1,p(n)+n+1,w_n} \wedge x_{1,p(n)+n+2,\#}$$

$$\wedge x_{1,p(n)+n+m+3,B} \wedge \dots \wedge x_{1,2p(n)+1,B}$$

$$\wedge \bigwedge_{1 \leq j \leq m} ((x_{1,p(n)+n+2+i,1} \wedge u_i) \vee (x_{1,p(n)+n+2+i,0} \wedge \neg u_i))$$

- La formule HALT permet de garantir qu'une des lignes correspond bien à une configuration acceptante

$$\text{HALT} = \bigvee_{1 \le i \le p(n), 1 \le j \le 2p(n)+1} x_{i,j,q_a}.$$

- Enfin la formule MOVE écrit que tous les sous-rectangles 3×2 du tableau T sont des fenêtres légales :

$$\text{MOVE} = \bigwedge_{1 \le i \le p(n), 1 \le j \le 2p(n)+1} \text{LEGAL}_{i,j},$$

où $\text{LEGAL}_{i,j}$ est une formule propositionnelle qui exprime que le sous-rectangle 3×2 à la position i,j est une fenêtre légale :

$$\text{LEGAL}_{i,j} = \bigvee_{(a,b,c,d,e,f) \in \text{WINDOW}} (x_{i,j-1,a} \wedge x_{i,j,b} \wedge x_{i,j+1,c} \wedge x_{i+1,j-1,d} \wedge x_{i+1,j,e} \wedge x_{i,j+1,f}),$$

où WINDOW est l'ensemble des 6-uplets (a,b,c,d,e,f) tels que

a	b	c
d	e	f

est une fenêtre légale.

- La formule $\gamma = \gamma_{V,w}(u)$ conjonction de

$$\mathrm{CELL} \wedge \mathrm{START} \wedge \mathrm{MOVE} \wedge \mathrm{HALT}.$$

  s'obtient bien par un algorithme polynomial à partir de w.

- On l'a construit exactement pour que :
  - $w \in A$ si et seulement si $\gamma_{V,w}(u)$ est satisfiable.

- Ce qui prouve bien que

$$A \leq \mathrm{SAT},$$

  via la fonction f calculable en temps polynomial qui à w associe $\gamma = \gamma_{V,w}(u)$.

- Fin de la preuve.

Plus précisément

Compléments sur la NP-complétude

- Non-déterminisme
- Théorème de Cook-Levin
- Digression
- Preuve du théorème de Cook-Levin
- Décision vs Construction

Décision vs Construction

- Nous avons jusque-là parlé uniquement de problèmes de **décision**, c'est-à-dire dont la réponse est soit vrai ou soit faux
  - par exemple : "étant donnée une formule F, décider si la formule F est satisfiable".
- en opposition aux problèmes qui consisteraient à **produire un objet avec une propriété**.
  - par exemple : "étant donnée une formule F, produire une affectation des variables qui la satisfait s'il en existe une."

- Si $P \neq NP$, alors aucun des deux problèmes n'admet une solution en temps polynomial.
  - car le premier problème est NP-complet.
  - et si l'on sait résoudre le second, alors on sait résoudre le premier.

- Et si $P = NP$?

- Si $P \neq NP$, alors aucun des deux problèmes n'admet une solution en temps polynomial.
  - car le premier problème est NP-complet.
  - et si l'on sait résoudre le second, alors on sait résoudre le premier.
  - **ATTENTION : $\leq$ ou $\leq_m$ ne comparent que des problèmes de décision.**
- Et si $P = NP$?

- Si $P \neq NP$, alors aucun des deux problèmes n'admet une solution en temps polynomial.
  - car le premier problème est NP-complet.
  - et si l'on sait résoudre le second, alors on sait résoudre le premier.
  - **ATTENTION : $\leq$ ou $\leq_m$ ne comparent que des problèmes de décision.**

- Et si $P = NP$?

Théorème
Supposons $P = NP$.

- *Soit L un problème de* NP *et V un vérificateur associé. On peut construire une machine de Turing qui sur toute entrée $w \in L$ produit en temps polynomial un certificat u pour w pour le vérificateur V.*

Démonstration : Le cas de SAT.

- Commençons par le prouver pour L correspondant au problème de la satisfaction de formules (problème SAT).
  - Supposons $\mathrm{P} = \mathrm{NP}$: on peut donc tester si une formule propositionnelle F à n variables est satisfiable ou non en temps polynomial.
  - Si elle est satisfiable, alors on peut fixer sa première variable à 0, et tester si la formule obtenue F_0 est satisfiable.
  - Si elle l'est, alors on écrit 0 et on recommence récursivement avec cette formule F_0 à $n-1$ variables.
  - Sinon, nécessairement tout certificat doit avoir sa première variable à 1, on écrit 1, et on recommence récursivement avec la formule F_1 dont la première variable est fixée à 1, qui possède $n-1$ variables.
  - Puisqu'il est facile de vérifier si une formule sans variable est satisfiable, par cette méthode, on aura écrit un certificat.
  - Cette procédure est bien de complexité polynomiale.

- Démonstration : Le cas de L un langage quelconque de NP
  - La réduction produite par la preuve du théorème de Cook-Levin est en fait une réduction de **Levin** :
    - on peut aussi retrouver un certificat pour w à partir d'un certificat de la satisfiabilité de la formule $f(w)$.
  - On peut donc utiliser l'algorithme précédent pour retrouver un certificat pour L, lorsque $\mathrm{P} = \mathrm{NP}$.

Digression : Retour à un peu de logique

- On peut assez facilement se persuader que le problème de décision

  $\text{THEOREMS} = \{(\phi, 1^n) | \phi \text{ possède une preuve de longueur } \leq n\}$

  est NP-complet.
  - (et ce dans n'importe quel système de preuve usuel).

- Si P = NP, cela veut donc dire :
  - qu'il doit exister une machine de Turing qui soit capable de produire une preuve mathématique de tout énoncé ϕ prouvable en un temps polynomial en la longueur de la preuve.

Digression : Retour à un peu de logique

- On peut assez facilement se persuader que le problème de décision

  $\text{THEOREMS} = \{(\phi, 1^n) | \phi \text{ possède une preuve de longueur } \leq n\}$

  est NP-complet.
  - (et ce dans n'importe quel système de preuve usuel).

- Si P = NP, cela veut donc dire :
  - qu'il doit exister une machine de Turing qui soit capable de produire une preuve mathématique de tout énoncé ϕ prouvable en un temps polynomial en la longueur de la preuve.

- Un argument de plus pour penser que P $\neq$ NP ...

Au menu

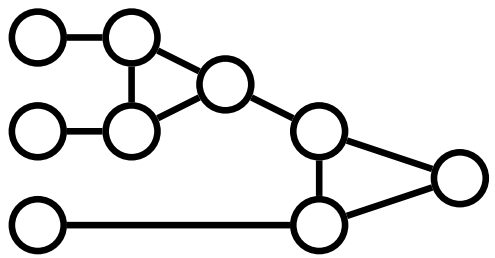
Exemples de preuve de NP-complétude

Quelques autres problèmes NP-complets célèbres

Compléments sur la NP-complétude

Gérer la NP-complétude

- Comment gérer la NP-complétude ?
  - En cherchant à contourner la/les difficultés.



  - En relachant les contraintes :
    - par exemple, en autorisant des solutions approchées.

Plus précisément

Gérer la NP-complétude

- Un exemple : Le problème du voyageur de commerce
- Plus généralement ?

Exemple : Le problème du voyageur de commerce

Un représentant doit visiter n villes. Le représentant souhaite faire une tournée en visitant chaque ville au moins et exactement une fois, et en terminant à sa ville de départ.

Sous forme de graphe

- On se donne un graphe connexe $G = (V, E)$, avec un poids $w(e)$ pour chaque arête $e = (u, v) \in E$.
- On veut déterminer un **cycle hamiltonien** (un cycle simple qui passe par tous les sommets) de G de poids minimal.

Sous forme de graphe

- On se donne un graphe connexe $G = (V, E)$, avec un poids $w(e)$ pour chaque arête $e = (u, v) \in E$.
- On veut déterminer un **cycle hamiltonien** (un cycle simple qui passe par tous les sommets) de G de poids minimal.
- On ne connaît aucun algorithme pour résoudre ce problème qui fonctionne en temps polynomial (par tout ce qui précède, ce problème est équivalent à la question $\mathrm{P} = \mathrm{NP}$?).

Sous forme de graphe

- On se donne un graphe connexe $G = (V, E)$, avec un poids $w(e)$ pour chaque arête $e = (u, v) \in E$.
- On veut déterminer un **cycle hamiltonien** (un cycle simple qui passe par tous les sommets) de G de poids minimal.
- On ne connaît aucun algorithme pour résoudre ce problème qui fonctionne en temps polynomial (par tout ce qui précède, ce problème est équivalent à la question $\mathrm{P} = \mathrm{NP}$?).
- Même si l'on se limite au cas Euclidien (les poids vérifient l'inégalité triangulaire), le problème du voyageur de commerce reste NP-complet.

Le cas euclidien est 2-approximable

- On se place dans le cas suivant :
  - le graphe est complet : pour toute paire de sommets u, v, il y a une arête (u, v).
  - les poids vérifient l'inégalité triangulaire :

    pour 3 sommets u, v, w arbitraires de V,

    $$w(u, v) \leq w(u, w) + w(w, v).$$

- On ne connaît pas d'algorithme en temps polynomial qui produit une solution optimale (sauf si P = NP).
- Mais on connaît un algorithme en temps polynomial qui produit une solution dont le poids est inférieur au double de l'optimal.

  $Poids$(Solution de l'algorithme) $\leq 2 * Poids$(Cycle optimal).

Le cas euclidien est 2-approximable

- On se place dans le cas suivant :
  - le graphe est complet : pour toute paire de sommets u, v, il y a une arête (u, v).

    Quitte à ajouter des arêtes de poids infini (très grand), on peut toujours s'y ramener.
  - les poids vérifient l'inégalité triangulaire :

    pour 3 sommets u, v, w arbitraires de V,

    $$w(u, v) \leq w(u, w) + w(w, v).$$
- On ne connaît pas d'algorithme en temps polynomial qui produit une solution optimale (sauf si $\mathrm{P} = \mathrm{NP}$).
- Mais on connaît un algorithme en temps polynomial qui produit une solution dont le poids est inférieur au double de l'optimal.

  $Poids$(Solution de l'algorithme) $\leq 2 * Poids$(Cycle optimal).

L'algorithme

Algorithme :

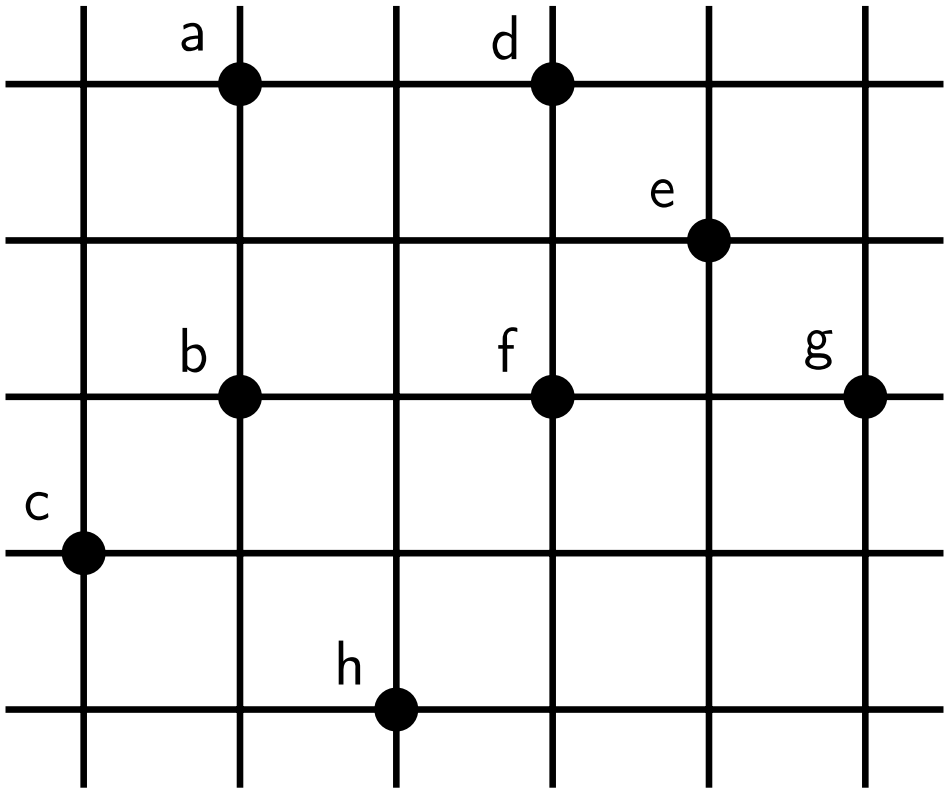
- Construire un arbre couvrant de poids minimal T de G.

▸ Détails

- Soit L la liste des sommets de G dans l'ordre où ils sont visités dans un parcours préfixe de T.
- Renvoyer le cycle hamiltonien qui visite les sommets dans l'ordre de L.

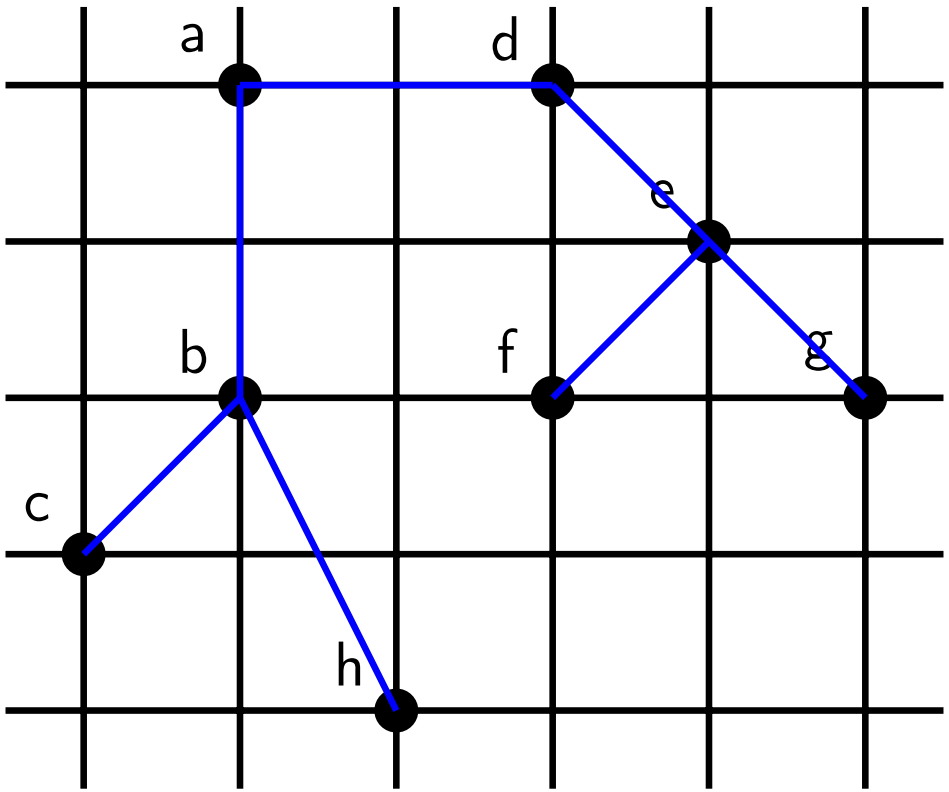
Exemple

Graphe initial (les poids sont les distances euclidiennes).



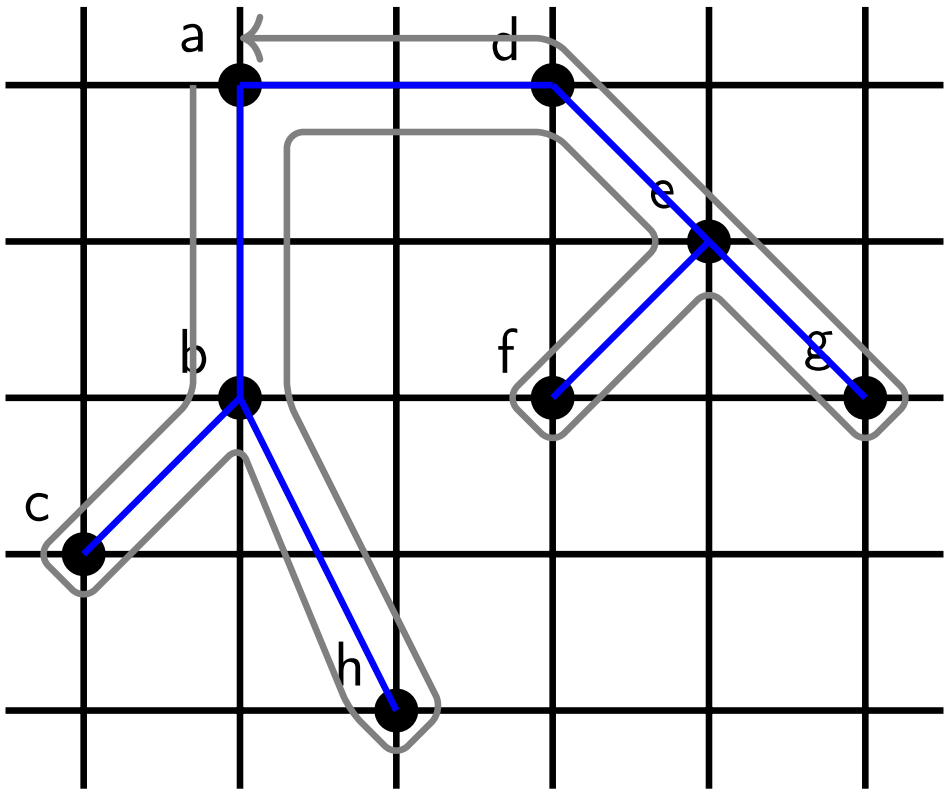
Exemple

Arbre couvrant T de poids minimal.

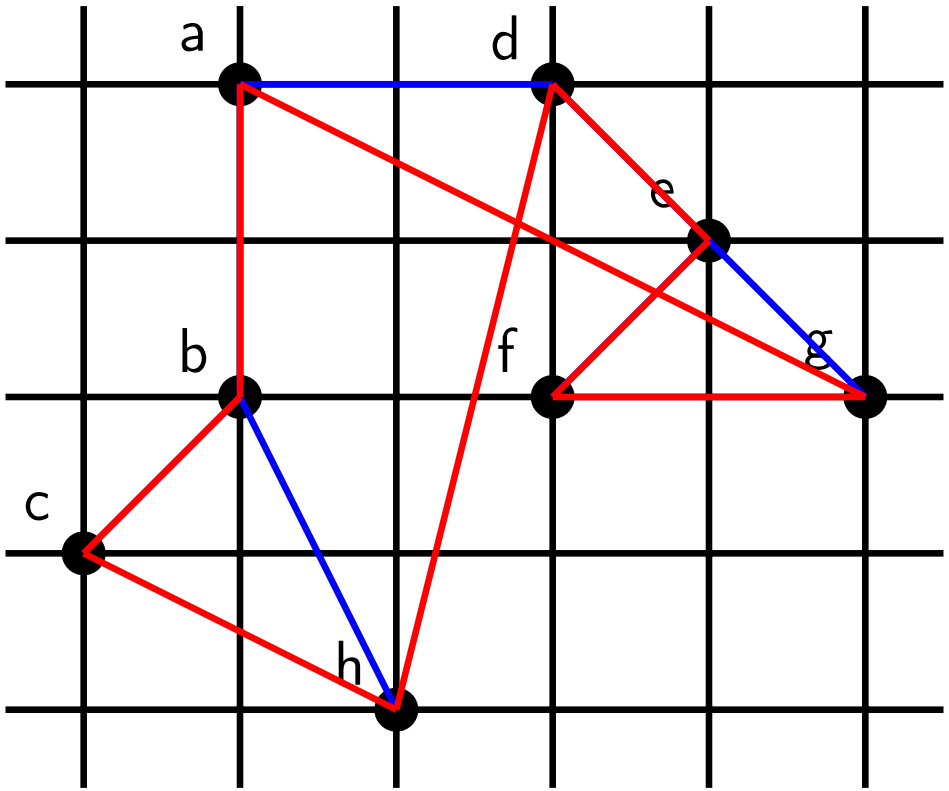


Exemple

Parcours préfixe de T : *abchdefg*.

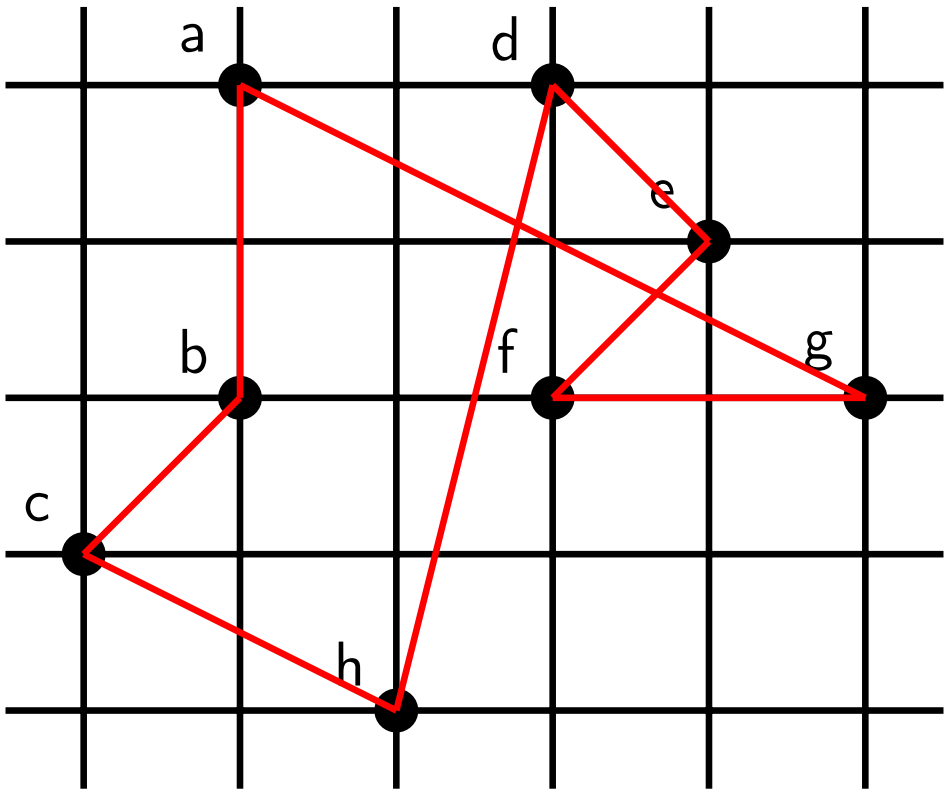


Exemple



Exemple

Résultat

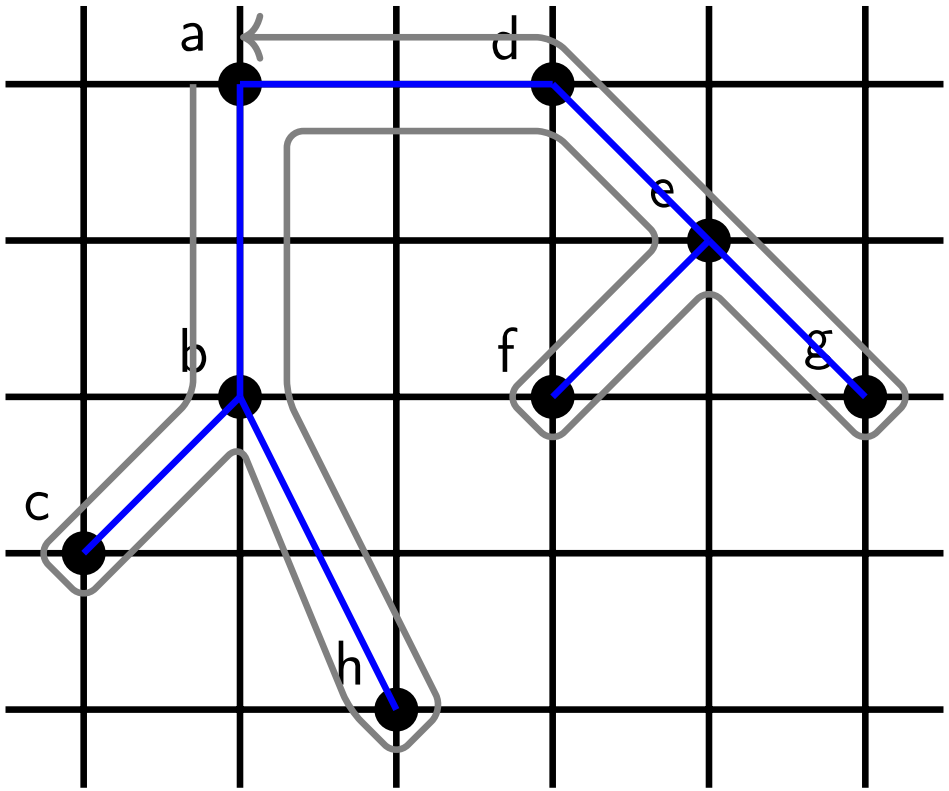


Preuve du facteur 2 d'approximation de l'algorithme

- Soit H^* une tournée optimale, et H la tournée retournée par l'algorithme.
- Il reste donc à prouver que $poids(H) \leq 2 * poids(H^*)$.
- Soit T un arbre couvrant minimal : $poids(T) \leq poids(H^*)$, car le cycle H^* privé d'une arête est un arbre.
- Soit L le résultat d'un parcours de l'arbre T, en suivant l'ordre préfixe et avec les répétitions.

Preuve du facteur 2 d'approximation (suite)

Sur l'exemple : parcours $L = a, b, c, b, h, b, a, d, e, f, e, g, e, d, a$.



$$poids(L) = 2 * poids(T),$$

car chaque arête de T est visitée 2 fois.

Preuve du facteur 2 d'approximation (suite)

- Donc $poids(L) \leq 2 * poids(H^*)$.

- On peut supprimer les sommets en double de L un par un sans augmenter le coût, grace à l'inégalité triangulaire.

  - exemple : plutôt que de passer une deuxième fois par b dans $L = a, b, c, b, h, b, a, d, e, f, e, g, e, d, a$, on peut faire $L = a, b, c, h, b, a, d, e, f, e, g, e, d, a$: le poids d'aller de c à h est inférieur ou égal au poids d'aller de c à b puis de b à h par l'inégalité triangulaire.

- Donc $poids(H) \leq poids(L) \leq 2 * poids(H^*)$.

Preuve du facteur 2 d'approximation (suite)

- Donc $poids(L) \leq 2 * poids(H^*)$.
- On peut supprimer les sommets en double de L un par un sans augmenter le coût, graçe à l'inégalité triangulaire.
  - exemple : plutôt que de passer une deuxième fois par b dans $L = a, b, c, b, h, b, a, d, e, f, e, g, e, d, a$, on peut faire $L = a, b, c, h, b, a, d, e, f, e, g, e, d, a$: le poids d'aller de c à h est inférieur ou égal au poids d'aller de c à b puis de b à h par l'inégalité triangulaire.
- Donc $poids(H) \leq poids(L) \leq 2 * poids(H^*)$.
- L'ensemble de l'algorithme reste de complexité polynomiale
  - En effet, l'algorithme de Kruskal pour calculer un arbre couvrant de poids minimal est bien de complexité polynomiale (voir annexe ou cours d'algorithmique à venir).

Plus précisément

Gérer la NP-complétude

- Un exemple : Le problème du voyageur de commerce
- Plus généralement ?

- Malheureusement, la théorie de l'approximation est essentiellement au cas par cas ...
  - La possibilité d'approximer un problème NP-complet ne dit rien en général sur les autres problèmes NP-complets.

ANNEXES

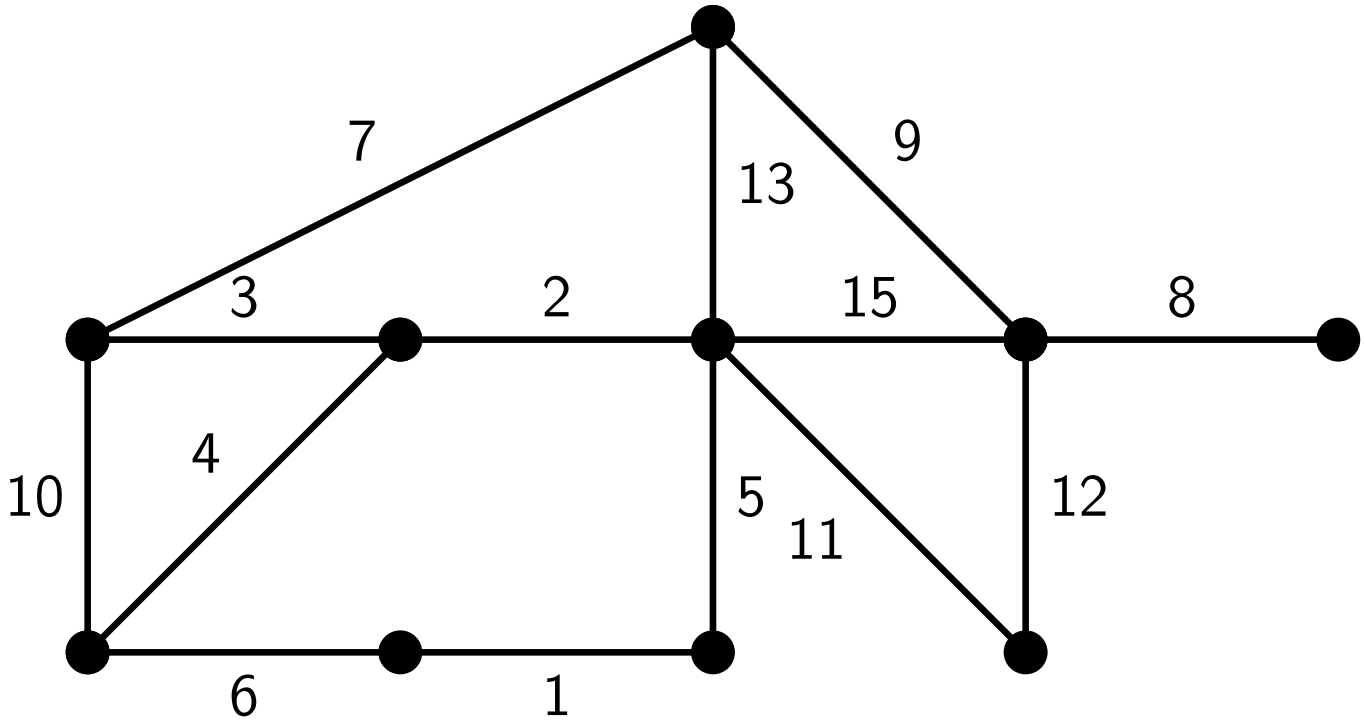
Algorithme de Kruskal pour calculer un arbre couvrant de poids minimal

Algorithme $Kruskal(G, w)$

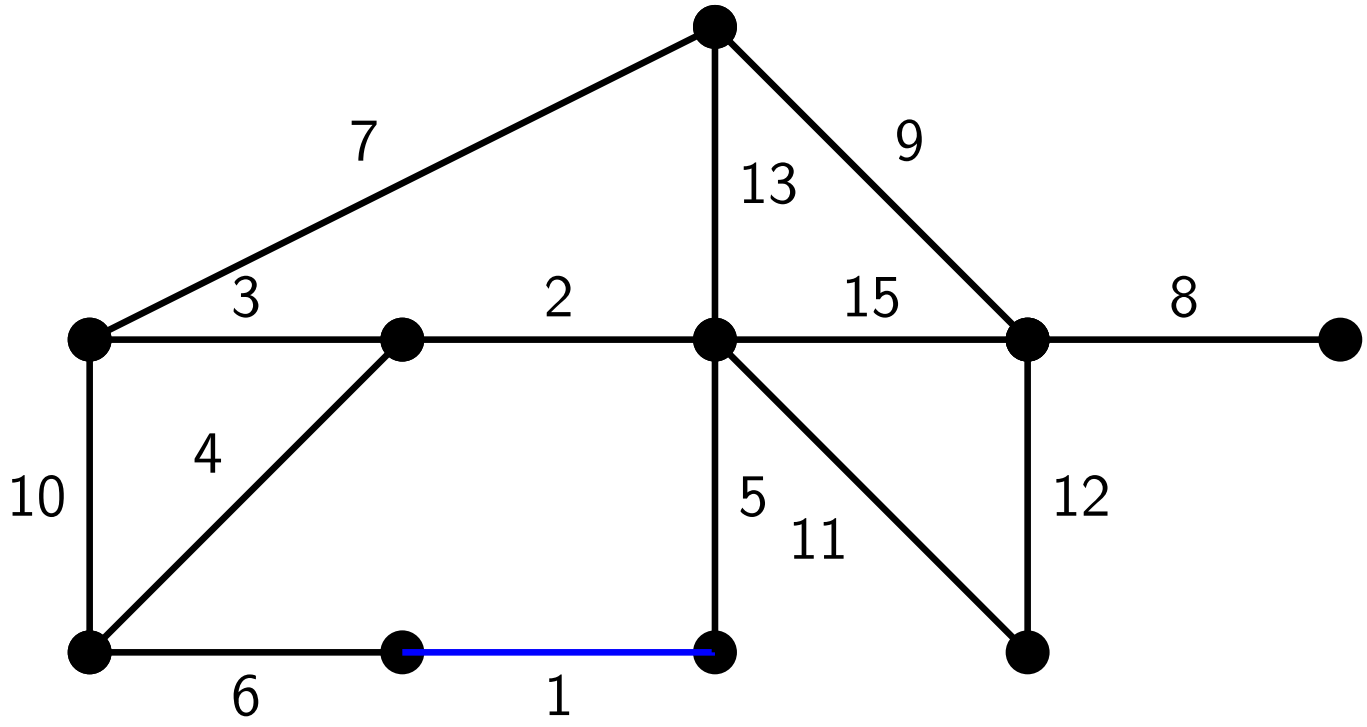
- Trier les arêtes par ordre de poids croissants $w(e_1) \leq w(e_2) \leq \ldots \leq w(e_{|E|})$
- $T := \emptyset$
- Pour $i := 1$ à $|E|$
  - si $T \cup \{e_i\}$ est acyclique alors
    - $T := T \cup \{e_i\}$
- Renvoyer T.

Remarque : à chaque étape T est une forêt.

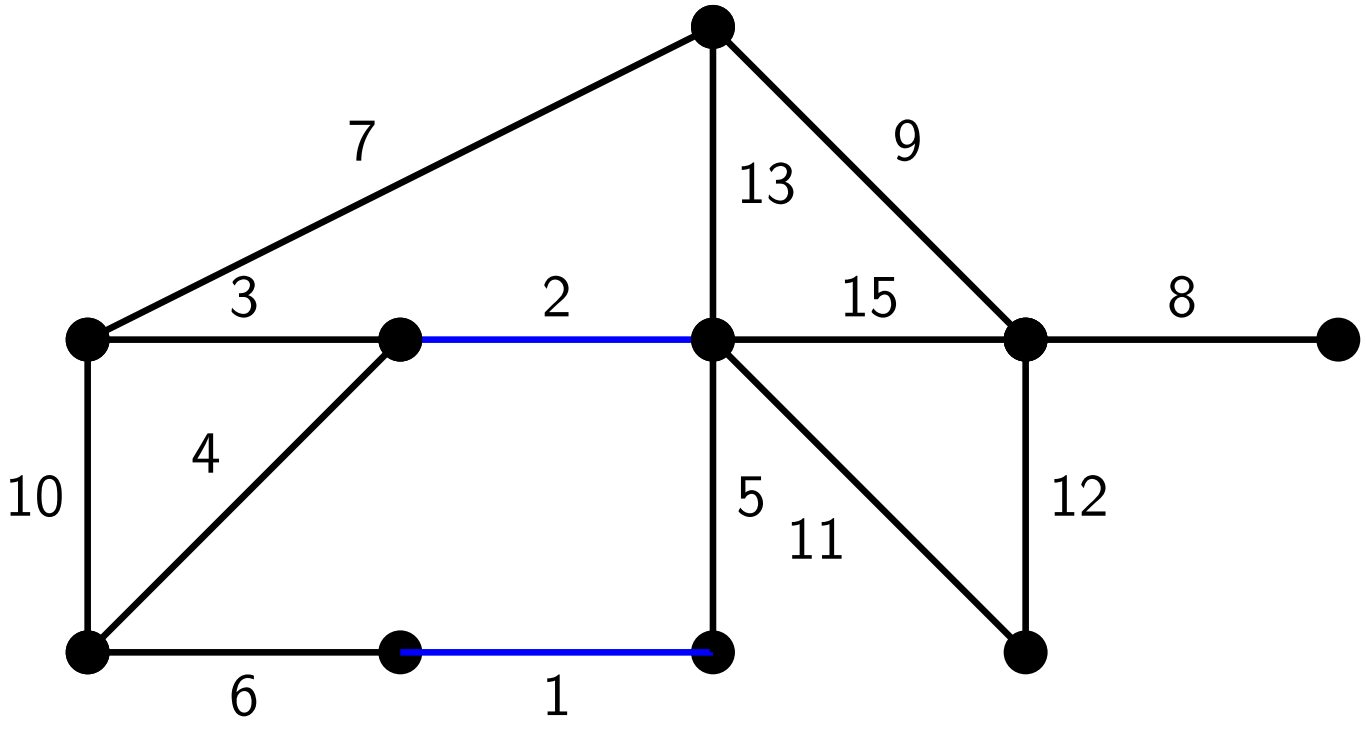
Exemple d'exécution de Kruskal



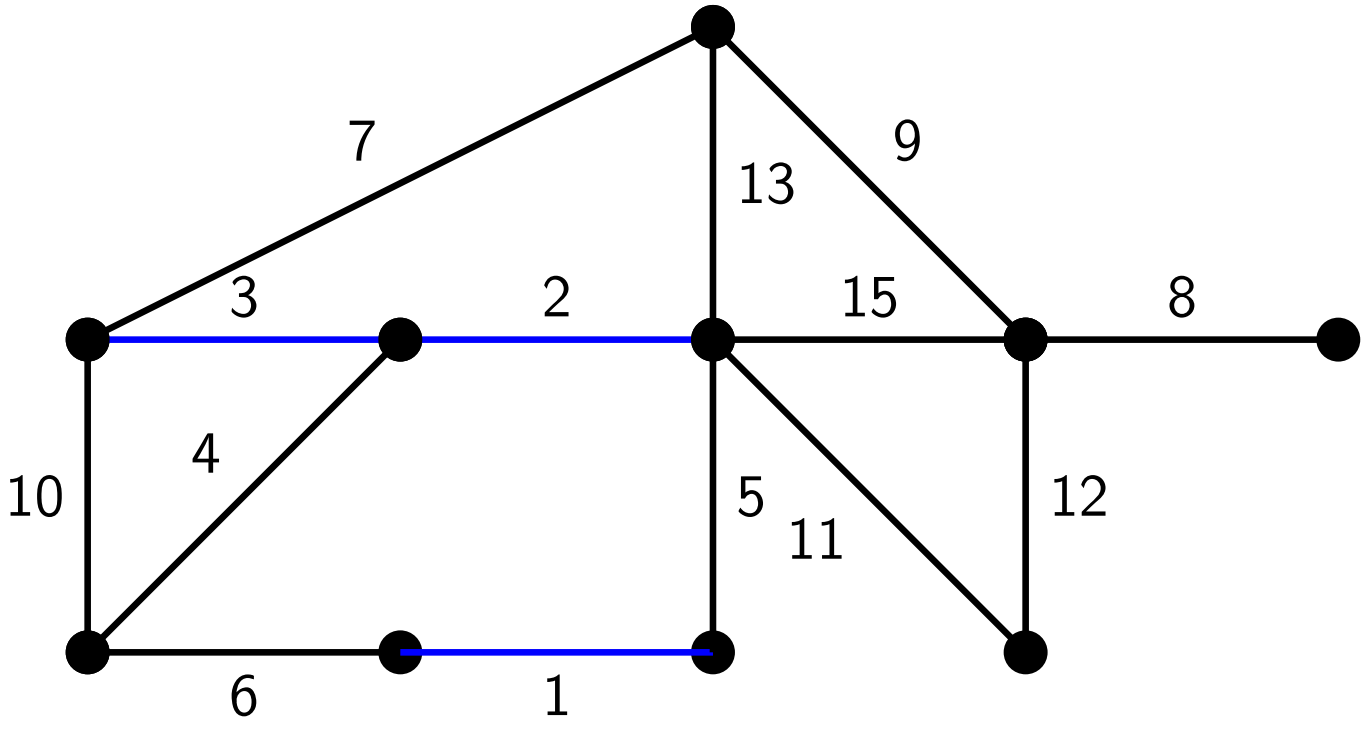
Exemple d'exécution de Kruskal



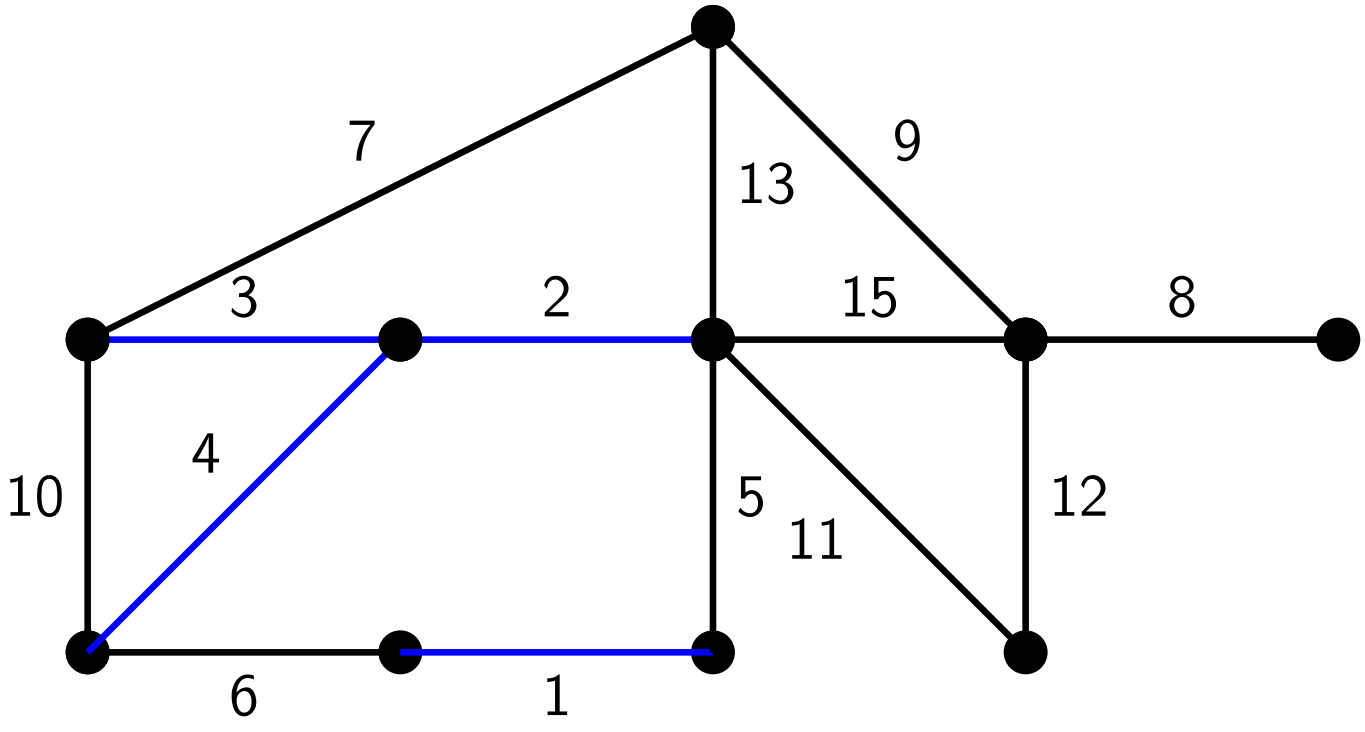
Exemple d'exécution de Kruskal



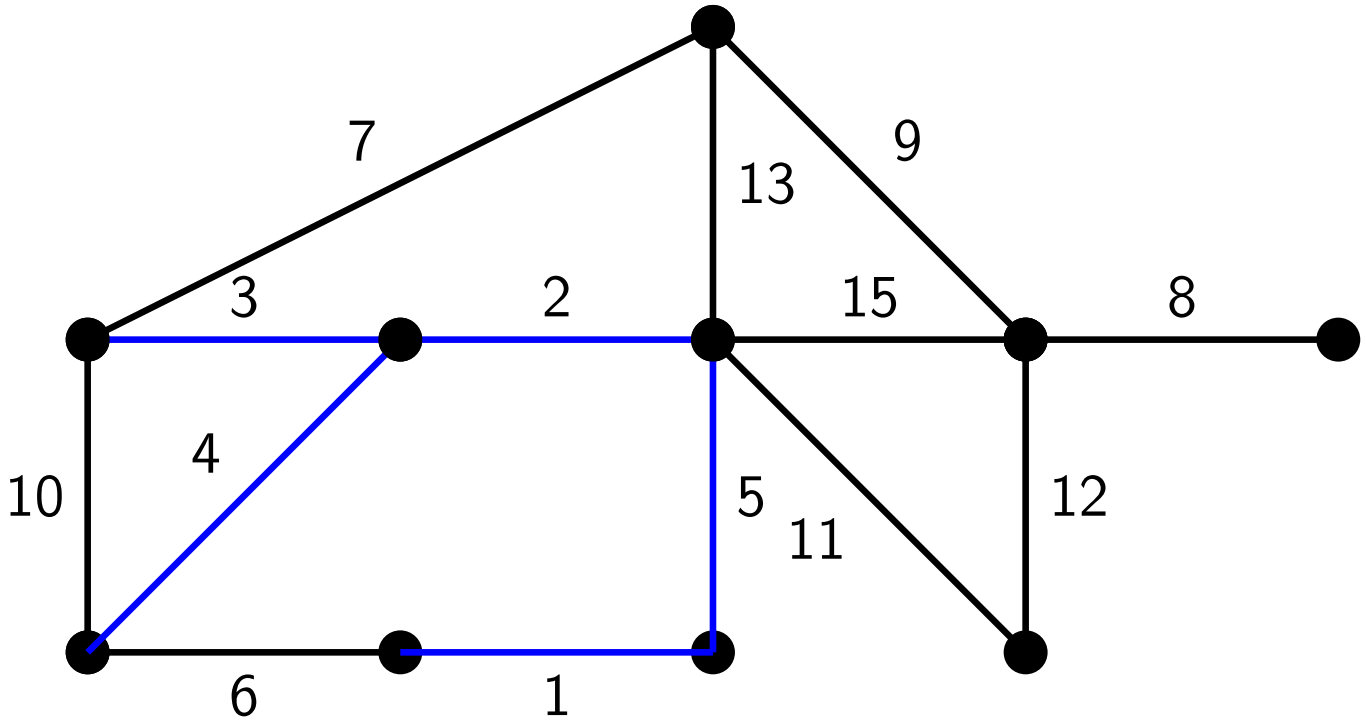
Exemple d'exécution de Kruskal



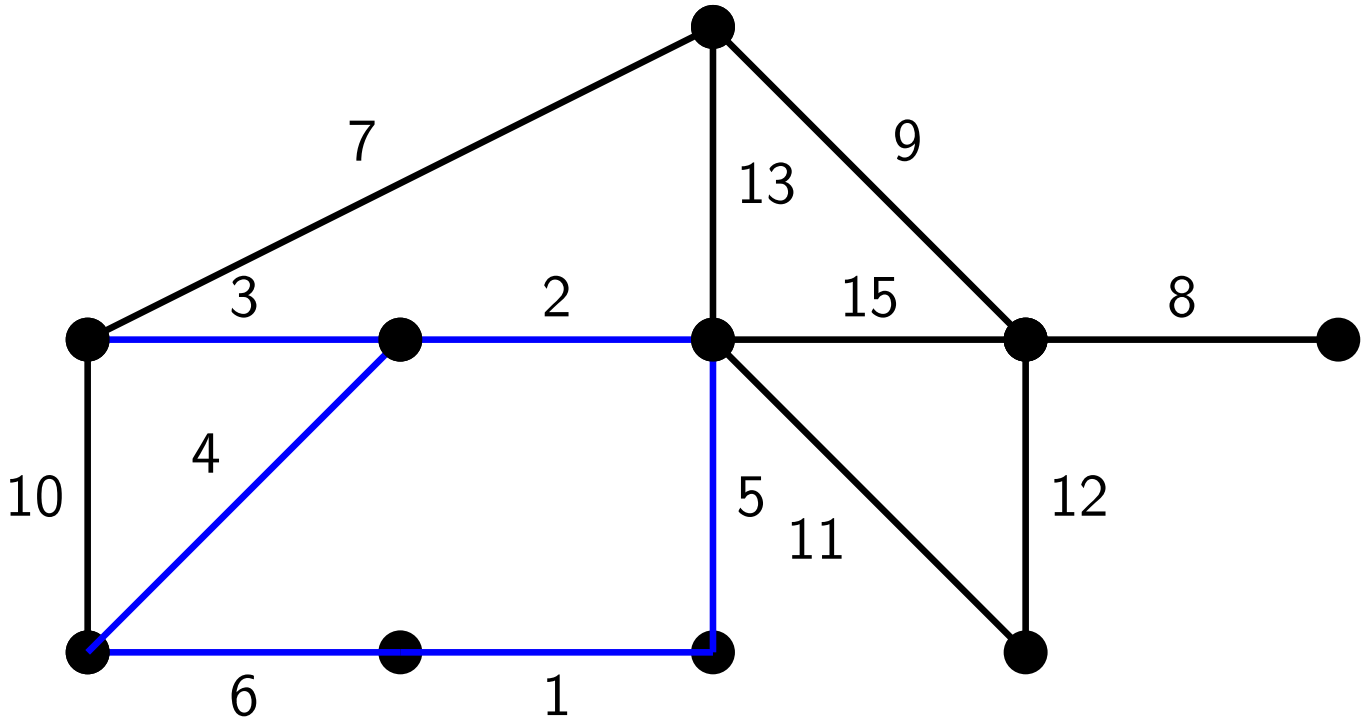
Exemple d'exécution de Kruskal



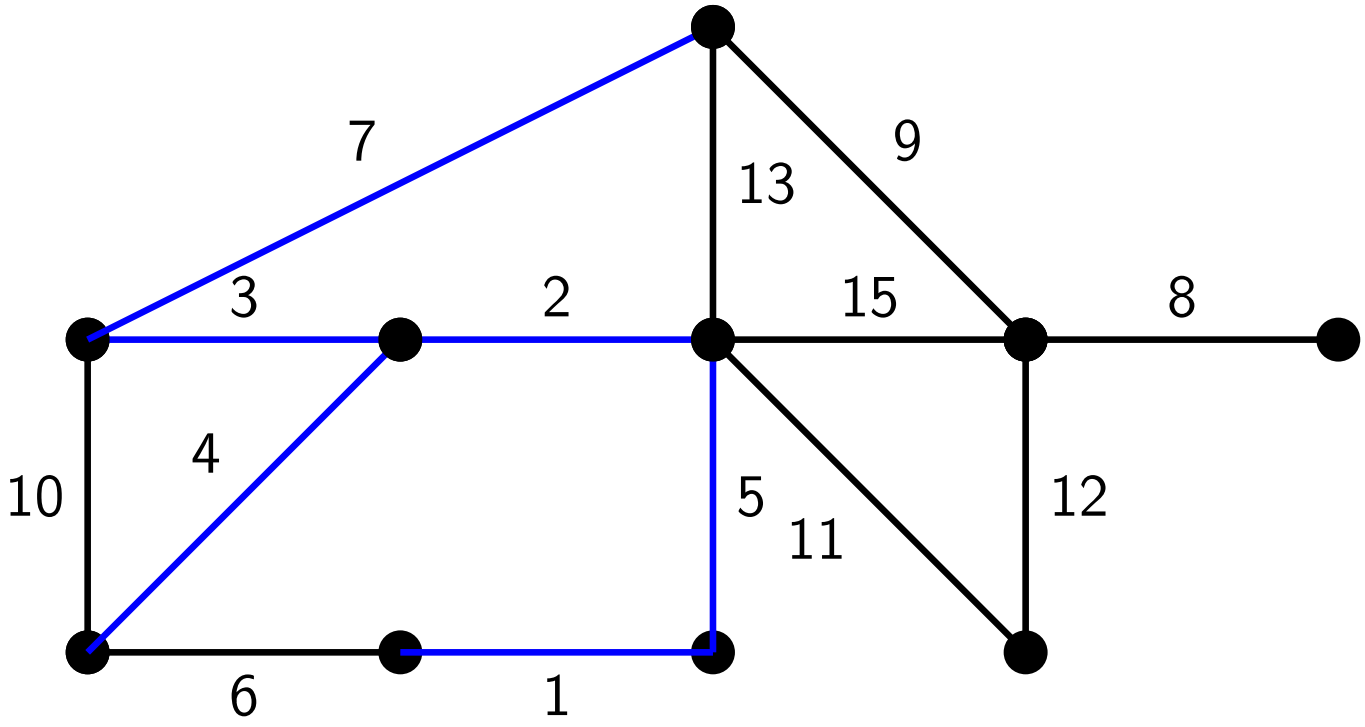
Exemple d'exécution de Kruskal



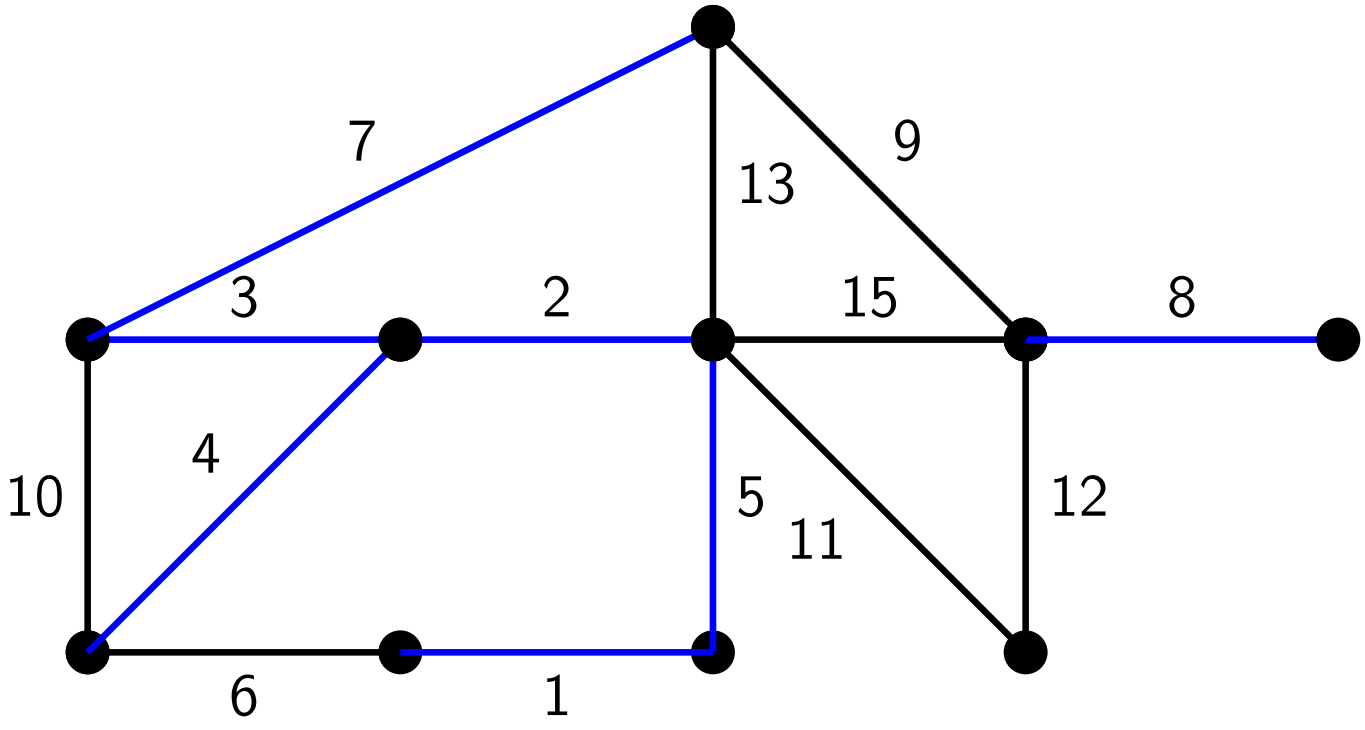
Exemple d'exécution de Kruskal



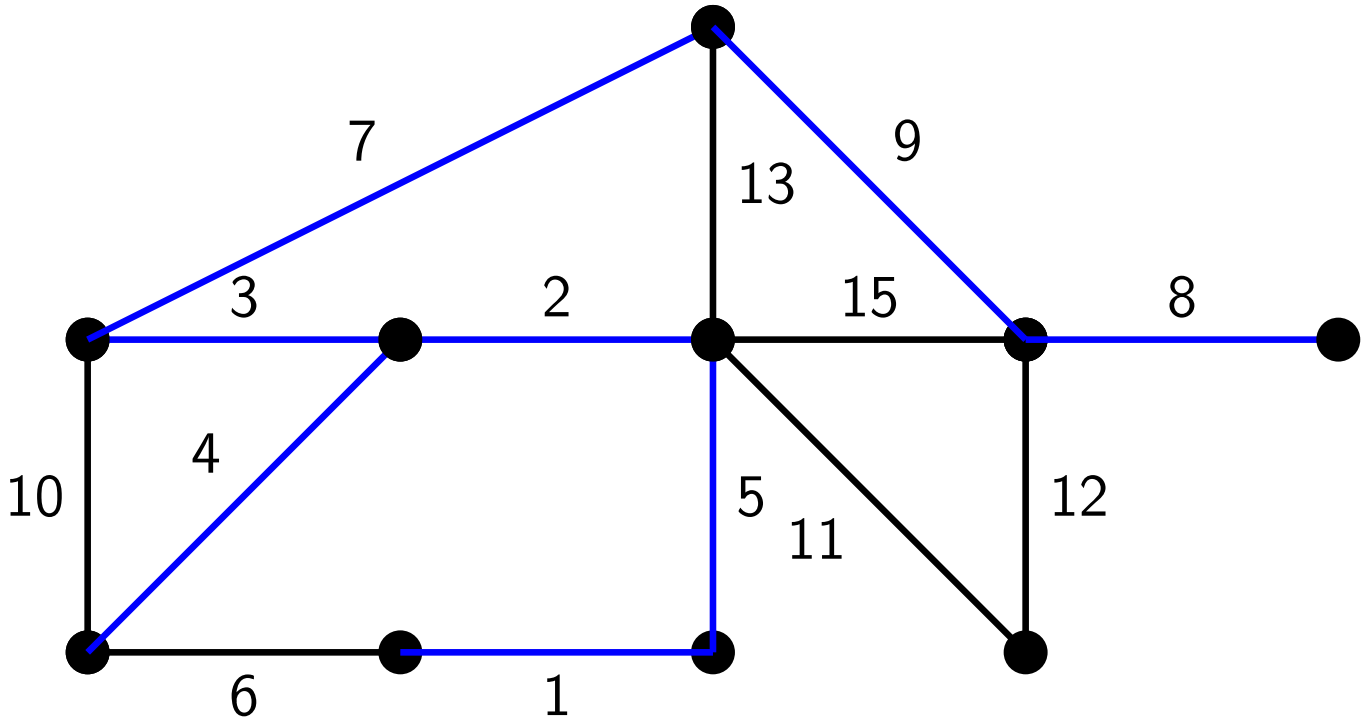
Exemple d'exécution de Kruskal



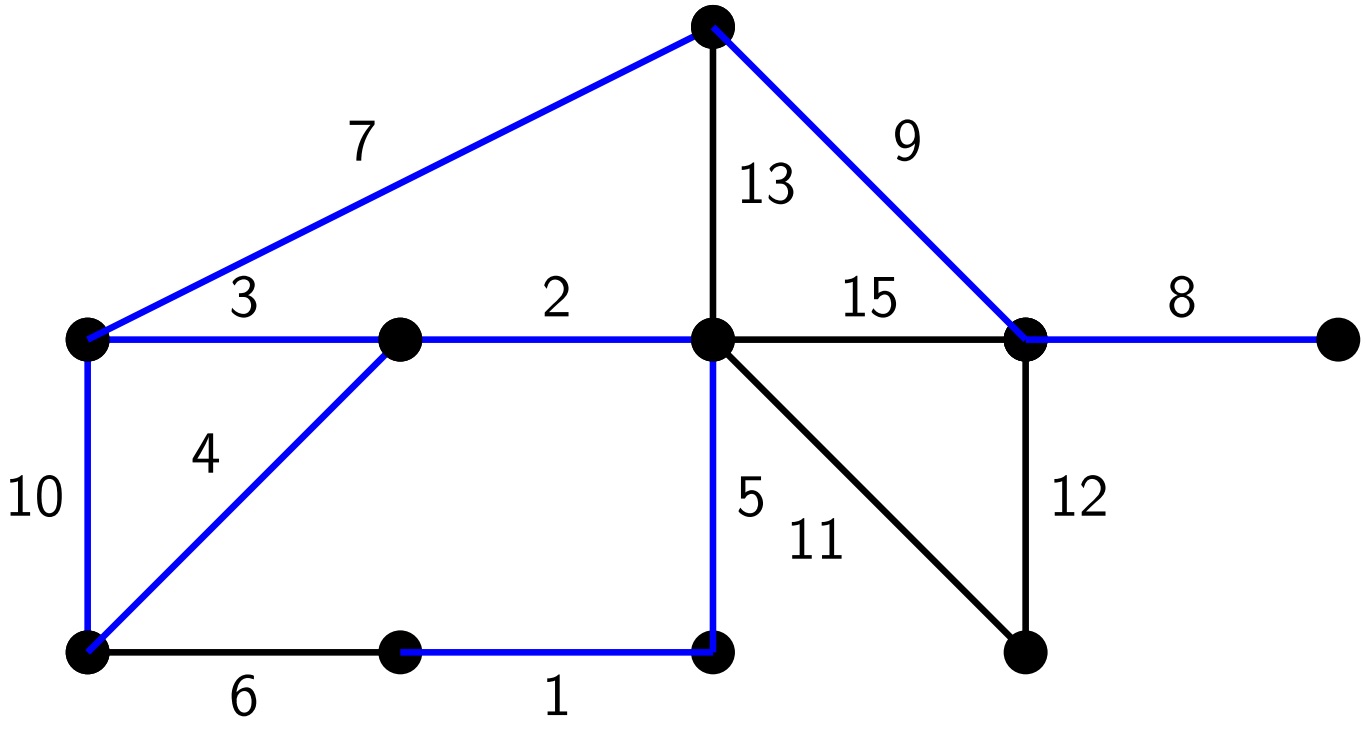
Exemple d'exécution de Kruskal



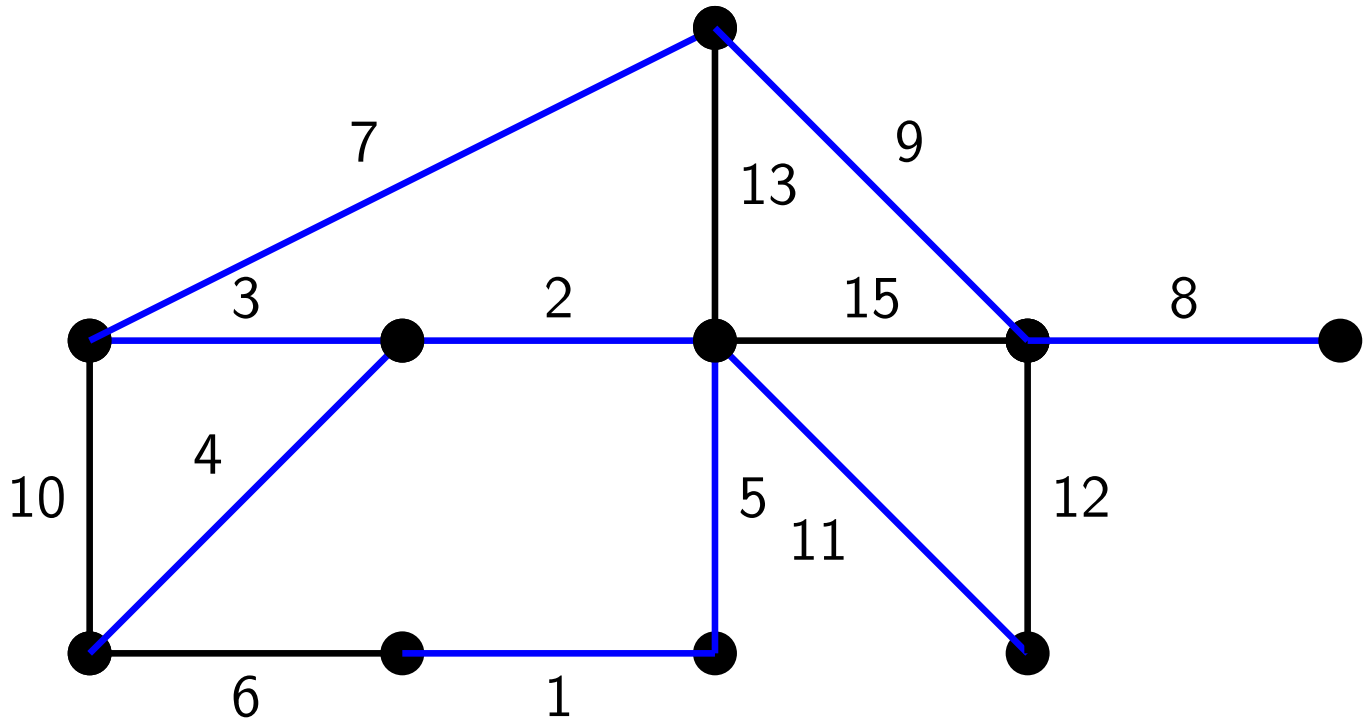
Exemple d'exécution de Kruskal



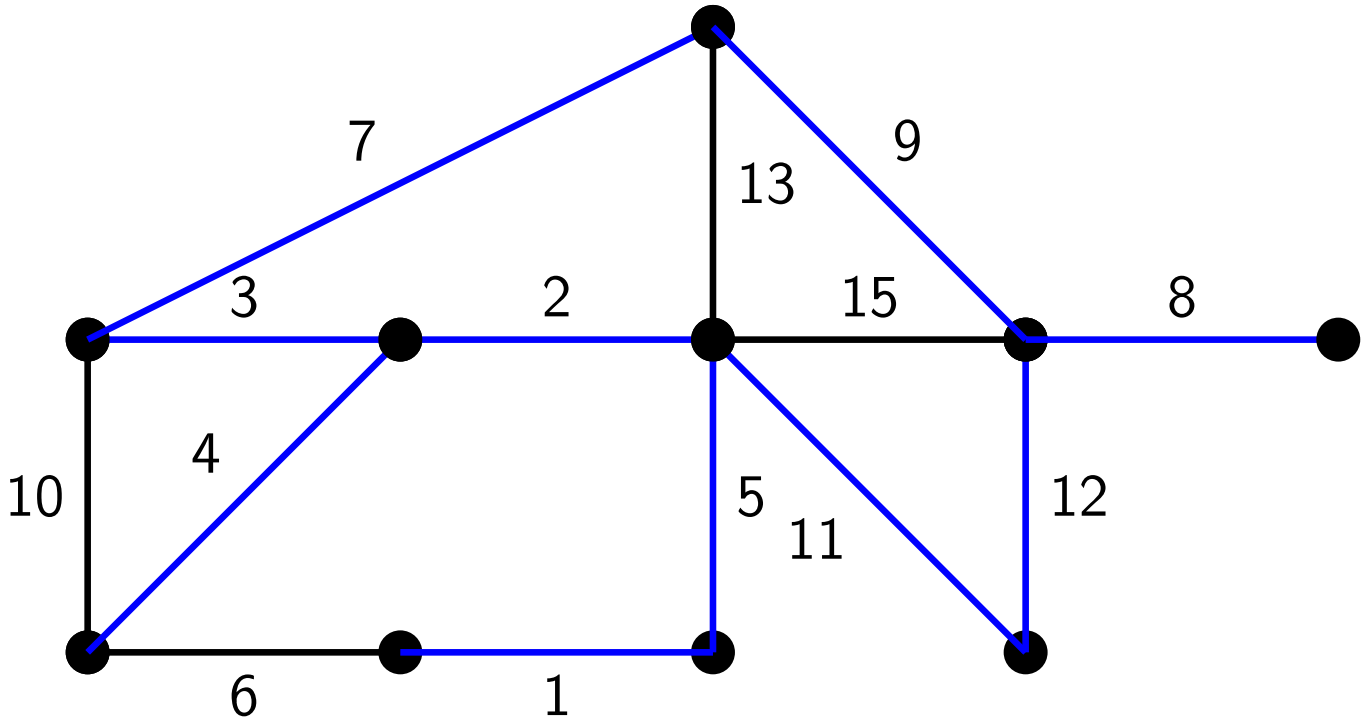
Exemple d'exécution de Kruskal



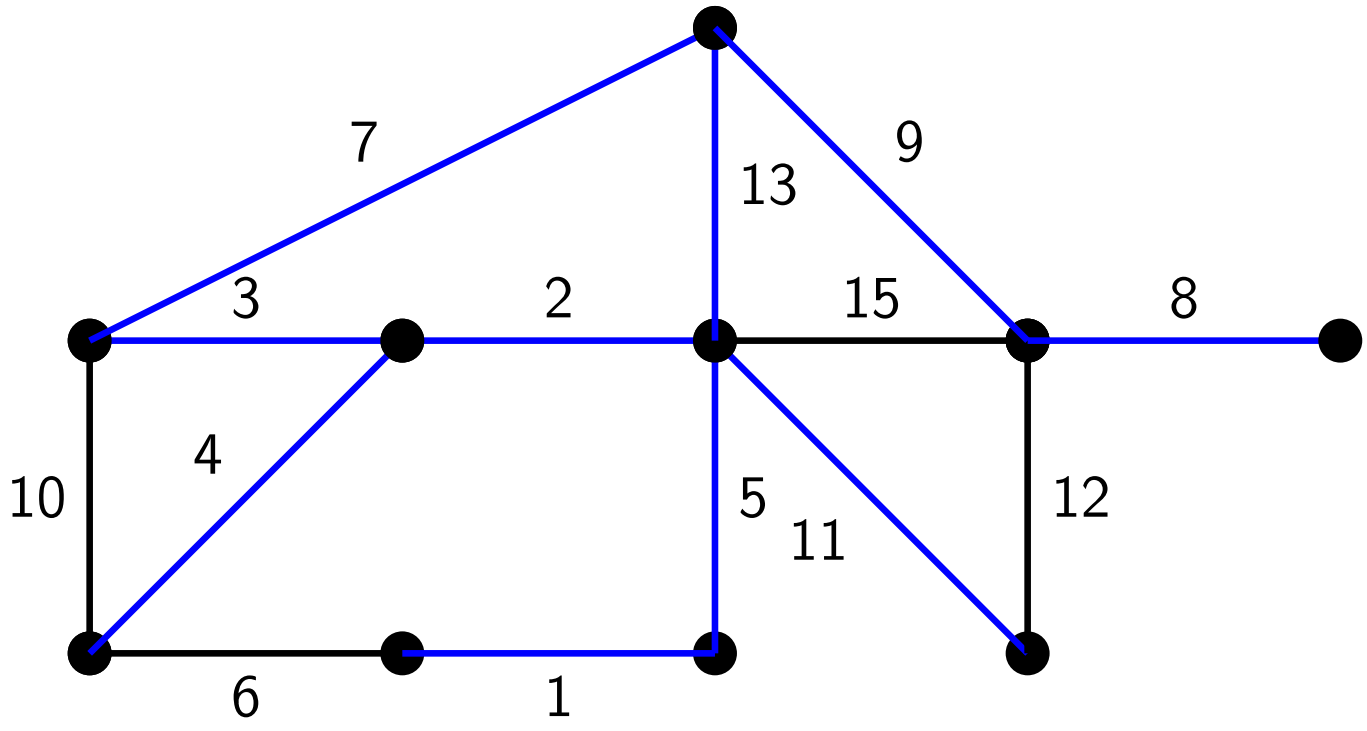
Exemple d'exécution de Kruskal



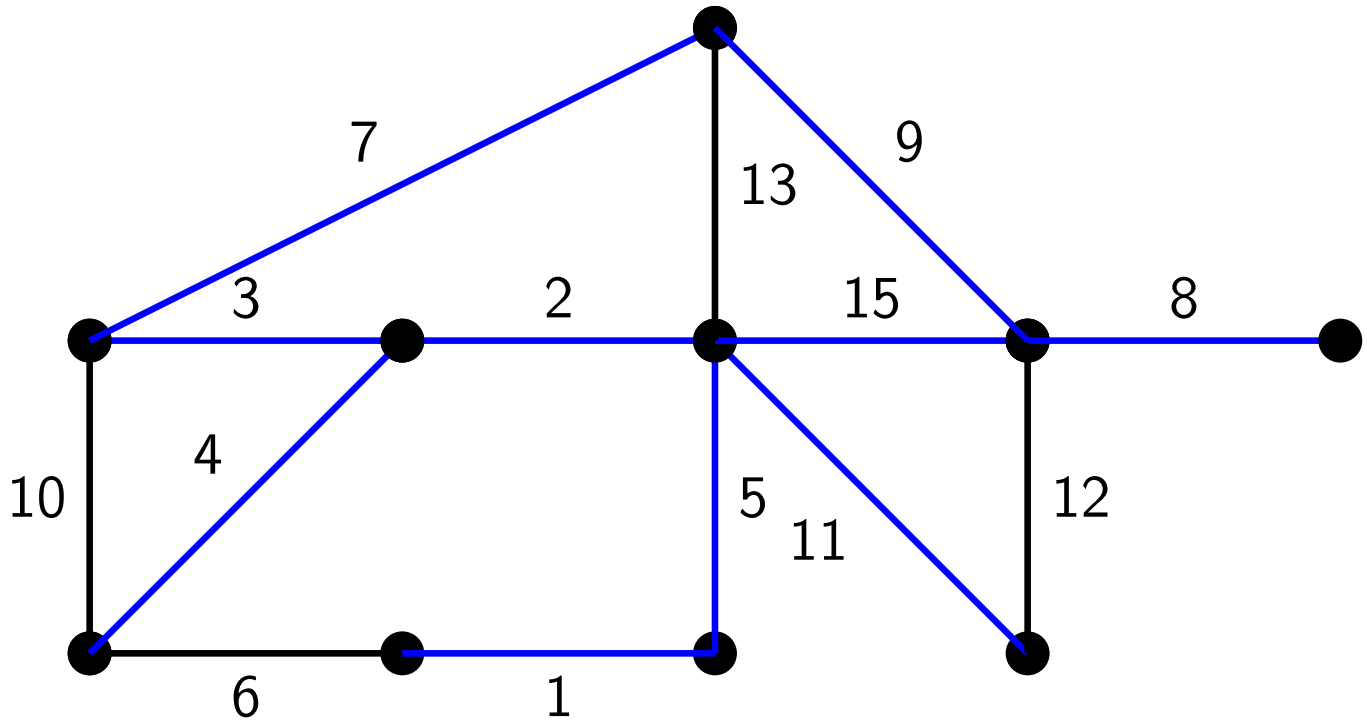
Exemple d'exécution de Kruskal



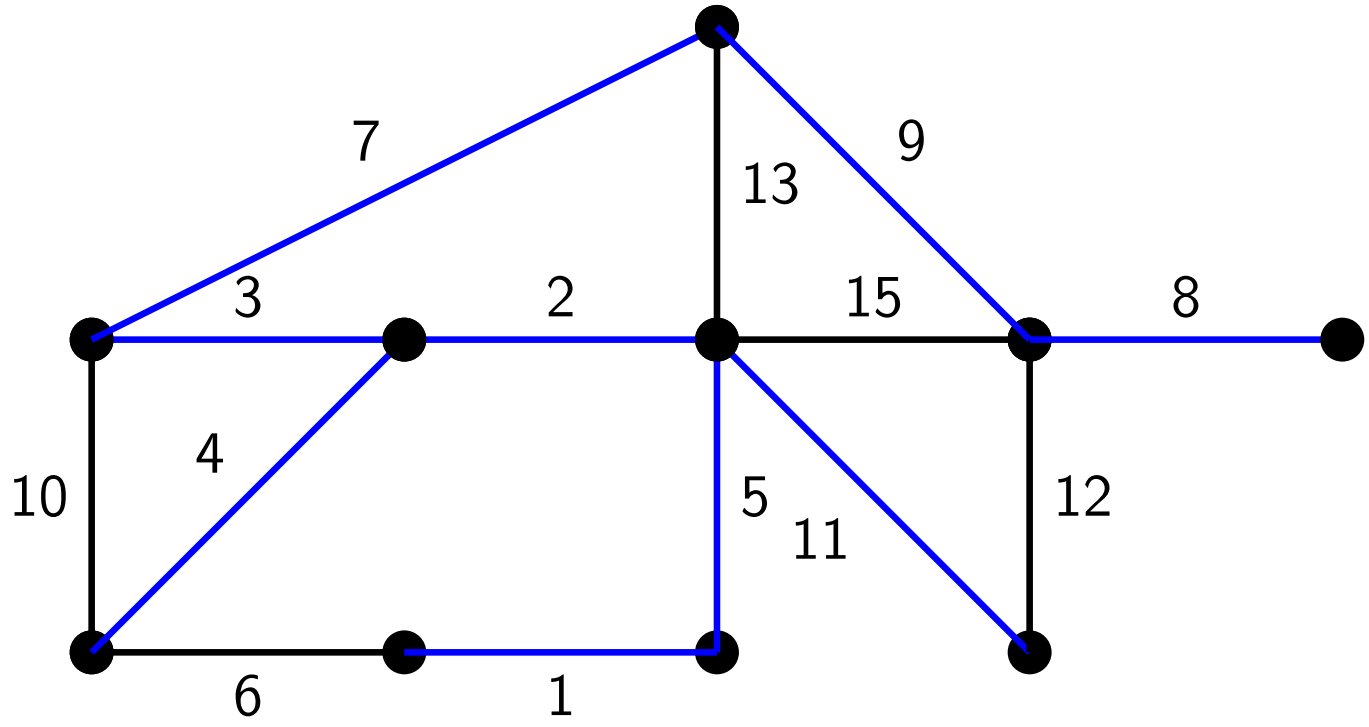
Exemple d'exécution de Kruskal



Exemple d'exécution de Kruskal



Exemple d'exécution de Kruskal



Résultat

L'algorithme de Kruskal est correct

- Il construit bien un arbre couvrant :
  - à chaque étape, T est une forêt.
  - chaque sommet a au moins une des arêtes qui lui est adjacente dans le T final.
- Il reste à montrer que l'arbre couvrant produit est de poids minimal.

Retour

Lemme

Lemma

Soient T et U deux arbres couvrants de G. Soit a une arête telle que $a \in U$, $a \notin T$. Alors, il existe une arête $b \in T$ telle que $U - a + b$ soit un arbre couvrant.
De plus, b peut être choisi dans le cycle formé par les arêtes de T et de a.

- Preuve :
  - la suppression de a dans U divise l'arbre en 2 composantes connexes U_1 et U_2.
  - Le cycle γ créé par a dans $T + a$ contient nécessairement un nombre pair d'arêtes reliant ces deux composantes.
  - Comme a est une telle arête, il en existe au moins une autre b dans T.
  - Cette arête b satisfait bien aux conditions du lemme.

L'arbre couvrant produit est optimal

- Soit T l'arbre donné par l'algorithme de Kruskal.
- Si l'algorithme ne donne pas l'optimal c'est qu'il existe un arbre couvrant de poids inférieur. Choisissons en un U dont le nombre d'arêtes communes avec T est maximal.
- Soit a l'arête de U de plus petit poids qui n'est pas dans T, et b l'arête donnée par le lemme pour ce cas.
- Comme a n'a pas été choisie par l'algorithme de Kruskal, c'est qu'elle crée un cycle avec les éléments de T, et que ce cycle est formé d'arètes toutes de poids inférieur ou égal à $w(a)$.
- Puisque b est dans ce cycle, on a $w(b) \leq w(a)$.
- $Poids(U - a + b) \leq Poids(U) < Poids(T)$, mais $U - a + b$ est un arbre couvrant qui a une arête commune avec T de plus que U.
- Ceci contredit le choix de U.

◂ Retour