

Conception et analyse d'algorithmes

Gilles Schaeffer, Laboratoire d'informatique de l'X

Cours: 13h30–15h TD: 15h15-17h15

Examen final sur table

Support, corrigés, annales: [google search INF550](#)

Conception et analyse d'algorithmes

- **Conception:**

- reconnaître l'algo qui s'applique (95% des cas)
- inventer de nouveaux algorithmes (5% des cas)

⇒ importance de la boîte à outils et de la notion de réduction

- **Analyse:** (optimalité, efficacité)

- borner la complexité d'un algo: souvent faisable
- savoir si on peut espérer faire mieux... pas évident !

⇒ comprendre ce qui fait qu'un algo fonctionne pour étendre son champs d'application

Votre boîte à outils en arrivant ici...

- Tris; tableaux, listes, piles, files, tas, automates finis
- Structures de données pour les arbres, les graphes

cf par exemple poly INF421

- Exploration de graphes: BFS, DFS
- Plus courts chemins: Dijkstra, Floyd-Warshall, A^*
- Algorithmes gloutons: Prim, Kruskal
- Graphe acyclique: tri topologique

cf par exemple poly INF431

- Diviser pour régner: facteur logarithmique
- Récursivité, mémoïsation, programmation dynamique

cf poly chapitre 2 de l'ancien poly INF550

Plan du cours

- Compléter la boîte à outils:
 - problèmes de flots, de coupes, de couplages
 - programmation linéaire
- Reconnaître les problèmes difficiles:
 - réduction polynomiale
 - NP-complétude
- Contourner la difficulté:
 - optimalité et facteurs d'approximation
 - complexité paramétrique
 - mesures alternatives de complexité

Cours 1: Algorithmes gloutons

Aujourd'hui retour sur une famille d'algorithmes bien connus...

- Optimisation combinatoire
- Un algorithme glouton
- Arbres recouvrants et algorithme de Kruskal
- Optimalité du glouton, matroïde
- Algorithmes de Prim, Dijkstra et gloutoïde

Un problème d'optimisation combinatoire générique

Donnée: E un ensemble fini,

v une valuation des éléments de E , $v : E \rightarrow \mathbb{R}^+$,

\mathcal{F} une famille de parties de E .

Problème: Trouver $F \in \mathcal{F}$ qui maximise la quantité

$$\sum_{e \in F} v(e).$$

Un problème d'optimisation combinatoire générique

Donnée: E un ensemble fini,

v une valuation des éléments de E , $v : E \rightarrow \mathbb{R}^+$,

\mathcal{F} une famille de parties de E .

Problème: Trouver $F \in \mathcal{F}$ qui maximise la quantité

$$\sum_{e \in F} v(e).$$

Remarque: si $|E| = n$, $|\mathcal{F}|$ peut contenir jusqu'à 2^n éléments

Un problème d'optimisation combinatoire générique

Donnée: E un ensemble fini,

v une valuation des éléments de E , $v : E \rightarrow \mathbb{R}^+$,

\mathcal{F} une famille de parties de E .

Problème: Trouver $F \in \mathcal{F}$ qui maximise la quantité

$$\sum_{e \in F} v(e).$$

Remarque: si $|E| = n$, $|\mathcal{F}|$ peut contenir jusqu'à 2^n éléments

et jusqu'à $\binom{n}{n/2} \approx \frac{2^n}{\sqrt{n}}$ éléments maximaux pour l'inclusion

Un problème d'optimisation combinatoire générique

Donnée: E un ensemble fini,

v une valuation des éléments de E , $v : E \rightarrow \mathbb{R}^+$,

\mathcal{F} une famille de parties de E .

Problème: Trouver $F \in \mathcal{F}$ qui maximise la quantité

$$\sum_{e \in F} v(e).$$

Remarque: si $|E| = n$, $|\mathcal{F}|$ peut contenir jusqu'à 2^n éléments

et jusqu'à $\binom{n}{n/2} \approx \frac{2^n}{\sqrt{n}}$ éléments maximaux pour l'inclusion

\Rightarrow la recherche exhaustive peut nécessiter un temps exponentiel.

Exemples

- Remplissage d'un sac à dos.

Donnée: n objets de valeur v_i et poids p_i , $i = 1, \dots, n$, qu'on veut mettre dans un sac dont le poids ne doit pas dépasser P .

Problème: trouver l'ensemble d'objets de valeur maximale qui entre dans le sac.

$$\mathcal{F} = \left\{ F \mid p(F) = \sum_{i \in F} p_i < P \right\}, \text{ maximiser } \sum_{i \in F} v_i.$$

Exemples

- Remplissage d'un sac à dos.

Donnée: n objets de valeur v_i et poids p_i , $i = 1, \dots, n$, qu'on veut mettre dans un sac dont le poids ne doit pas dépasser P .

Problème: trouver l'ensemble d'objets de valeur maximale qui entre dans le sac.

$$\mathcal{F} = \left\{ F \mid p(F) = \sum_{i \in F} p_i < P \right\}, \text{ maximiser } \sum_{i \in F} v_i.$$

- Constitution d'une liste d'invités maximale.

Donnée: une liste \mathcal{I} de couples incompatibles

Problème: construire une liste d'invités sans incompatibilité.

$$\mathcal{F} = \left\{ F \mid \forall x, y \in F, (x, y) \notin \mathcal{I} \right\}, \text{ maximiser } |F|$$

Cours 1: Algorithmes gloutons

- Optimisation combinatoire
- L'algorithme glouton
- Arbres recouvrants et algorithme de Kruskal
- Optimalité du glouton, matroïde
- Algorithmes de Prim, Dijkstra et gloutoïde

Un algorithme glouton générique

Rappel: on cherche à déterminer $\operatorname{argmax}_{F \in \mathcal{F}} \sum_{e \in F} v(e)$

- Classer les éléments de E par valuations décroissantes:

$$v(e_1) \geq v(e_2) \geq \dots \geq v(e_n)$$

- Initialiser la recherche avec $F = \emptyset$
- Pour $i = 1$ à n faire:
 - si $F \cup \{e_i\} \in \mathcal{F}$, alors $F := F \cup \{e_i\}$.

Un algorithme glouton générique

Rappel: on cherche à déterminer $\operatorname{argmax}_{F \in \mathcal{F}} \sum_{e \in F} v(e)$

- Classer les éléments de E par valuations décroissantes:

$$v(e_1) \geq v(e_2) \geq \dots \geq v(e_n) \quad \text{choix du gain local maximum}$$

- Initialiser la recherche avec $F = \emptyset$
- Pour $i = 1$ à n faire:
 - si $F \cup \{e_i\} \in \mathcal{F}$, alors $F := F \cup \{e_i\}$.

Un algorithme glouton générique

Rappel: on cherche à déterminer $\operatorname{argmax}_{F \in \mathcal{F}} \sum_{e \in F} v(e)$

- Classer les éléments de E par valuations décroissantes:

$$v(e_1) \geq v(e_2) \geq \dots \geq v(e_n) \quad \text{choix du gain local maximum}$$

- Initialiser la recherche avec $F = \emptyset$
- Pour $i = 1$ à n faire:
 si $F \cup \{e_i\} \in \mathcal{F}$, alors $F := F \cup \{e_i\}$.

Plus généralement on est glouton lorsqu'on fait des choix de gain maximum à l'instant présent, et qu'on ne les remet plus en cause ensuite.

Un algorithme glouton générique

Rappel: on cherche à déterminer $\operatorname{argmax}_{F \in \mathcal{F}} \sum_{e \in F} v(e)$

- Classer les éléments de E par valuations décroissantes:

$$v(e_1) \geq v(e_2) \geq \dots \geq v(e_n) \quad \text{choix du gain local maximum}$$

- Initialiser la recherche avec $F = \emptyset$
- Pour $i = 1$ à n faire:
si $F \cup \{e_i\} \in \mathcal{F}$, alors $F := F \cup \{e_i\}$.

Plus généralement on est glouton lorsqu'on fait des choix de gain maximum à l'instant présent, et qu'on ne les remet plus en cause ensuite.

Bien sûr cette approche conduit le plus souvent à manquer l'optimum

Premier exemple: indépendant maximum

Donnée: un ensemble de n éléments avec une valeur v_i et une liste \mathcal{I} de paires d'éléments incompatibles.

Problème: trouver un sous-ensemble de valeur maximale parmi ceux qui ne contiennent pas de paire incompatible

Premier exemple: indépendant maximum

Donnée: un ensemble de n éléments avec une valeur v_i et une liste \mathcal{I} de paires d'éléments incompatibles.

Problème: trouver un sous-ensemble de valeur maximale parmi ceux qui ne contiennent pas de paire incompatible

On peut représenter \mathcal{I} par un graphe des incompatibilités. On cherche alors un sous-ensemble de sommets **indépendants** de valeur maximum.

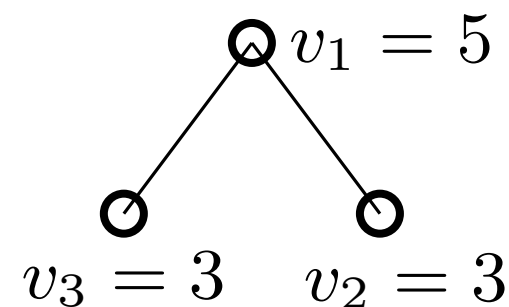
Premier exemple: indépendant maximum

Donnée: un ensemble de n éléments avec une valeur v_i et une liste \mathcal{I} de paires d'éléments incompatibles.

Problème: trouver un sous-ensemble de valeur maximale parmi ceux qui ne contiennent pas de paire incompatible

On peut représenter \mathcal{I} par un graphe des incompatibilités. On cherche alors un sous-ensemble de sommets **indépendants** de valeur maximum.

$$\mathcal{I} = \{\{1, 2\}, \{1, 3\}\}$$



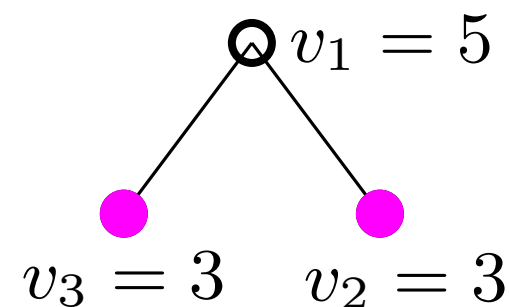
Premier exemple: indépendant maximum

Donnée: un ensemble de n éléments avec une valeur v_i et une liste \mathcal{I} de paires d'éléments incompatibles.

Problème: trouver un sous-ensemble de valeur maximale parmi ceux qui ne contiennent pas de paire incompatible

On peut représenter \mathcal{I} par un graphe des incompatibilités. On cherche alors un sous-ensemble de sommets **indépendants** de valeur maximum.

$$\mathcal{I} = \{\{1, 2\}, \{1, 3\}\}$$



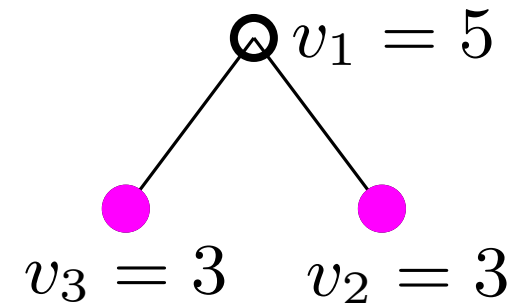
Premier exemple: indépendant maximum

Donnée: un ensemble de n éléments avec une valeur v_i et une liste \mathcal{I} de paires d'éléments incompatibles.

Problème: trouver un sous-ensemble de valeur maximale parmi ceux qui ne contiennent pas de paire incompatible

On peut représenter \mathcal{I} par un graphe des incompatibilités. On cherche alors un sous-ensemble de sommets **indépendants** de valeur maximum.

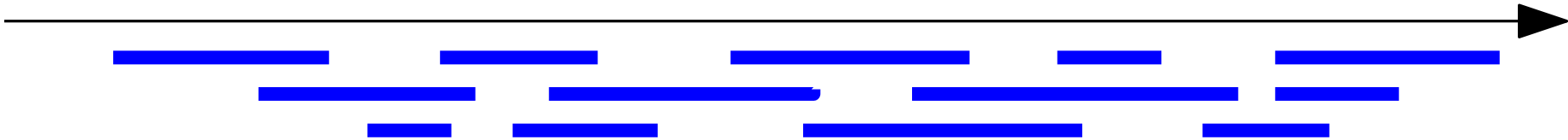
$$\mathcal{I} = \{\{1, 2\}, \{1, 3\}\}$$



L'optimum est la paire $\{2, 3\}$ de valeur totale 6 qui ne sera pas trouvé par l'algorithme glouton: ce dernier prend 1 et reste bloqué par les incompatibilités.

Cas particulier: un problème d'affectation simple

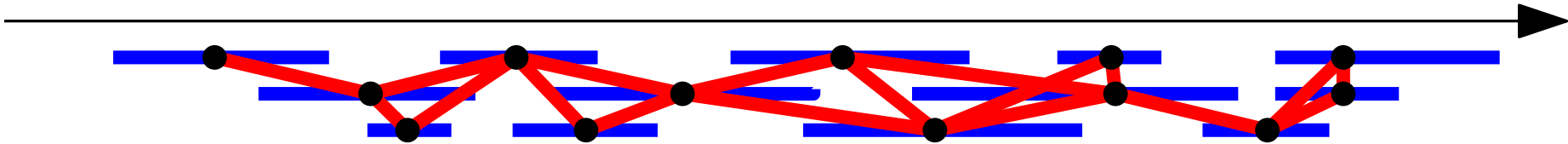
Donnée: n demandes d'affectation d'une ressource, avec pour chaque demande i , la date de début d_i et de fin f_i .



Problème: trouver un sous-ensemble F de demandes tel que $\forall i, j \in F, (d_i, f_i) \cap (d_j, f_j) = \emptyset$, et $|F|$ est maximal.

Cas particulier: un problème d'affectation simple

Donnée: n demandes d'affectation d'une ressource, avec pour chaque demande i , la date de début d_i et de fin f_i .



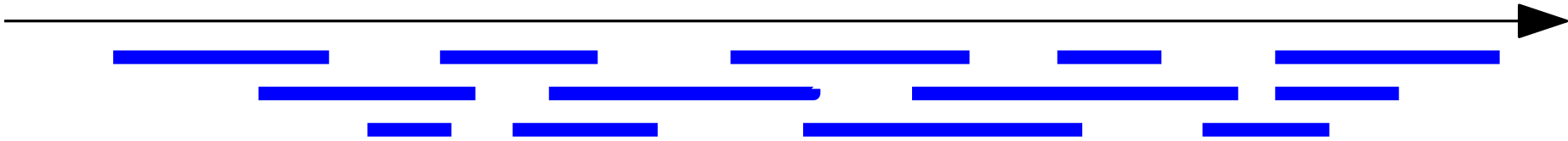
Problème: trouver un sous-ensemble F de demandes tel que $\forall i, j \in F, (d_i, f_i) \cap (d_j, f_j) = \emptyset$, et $|F|$ est maximal.

\Rightarrow un indépendant de cardinalité maximum où le graphe des incompatibilités est un graphe d'intervalle.

o

Cas particulier: un problème d'affectation simple

Donnée: n demandes d'affectation d'une ressource, avec pour chaque demande i , la date de début d_i et de fin f_i .



Problème: trouver un sous-ensemble F de demandes tel que $\forall i, j \in F, (d_i, f_i) \cap (d_j, f_j) = \emptyset$, et $|F|$ est maximal.

Les poids sont unitaires, on utilise un critère ad-hoc supplémentaire:

Cas particulier: un problème d'affectation simple

Donnée: n demandes d'affectation d'une ressource, avec pour chaque demande i , la date de début d_i et de fin f_i .



Problème: trouver un sous-ensemble F de demandes tel que $\forall i, j \in F, (d_i, f_i) \cap (d_j, f_j) = \emptyset$, et $|F|$ est maximal.

Les poids sont unitaires, on utilise un critère ad-hoc supplémentaire:
classer par date de fin croissante, $f_1 \leq f_2 \leq \dots \leq f_n$.

Cas particulier: un problème d'affectation simple

Donnée: n demandes d'affectation d'une ressource, avec pour chaque demande i , la date de début d_i et de fin f_i .



Problème: trouver un sous-ensemble F de demandes tel que $\forall i, j \in F, (d_i, f_i) \cap (d_j, f_j) = \emptyset$, et $|F|$ est maximal.

Les poids sont unitaires, on utilise un critère ad-hoc supplémentaire:
classer par date de fin croissante, $f_1 \leq f_2 \leq \dots \leq f_n$.

- Initialiser la recherche avec $F = \emptyset$
- Pour $i = 1$ à n faire:
 si $\max(f_j, j \in F) < d_i$, alors $F := F \cup \{i\}$.

Preuve d'optimalité du glouton pour l'affectation

- Soit F donné par le glouton.

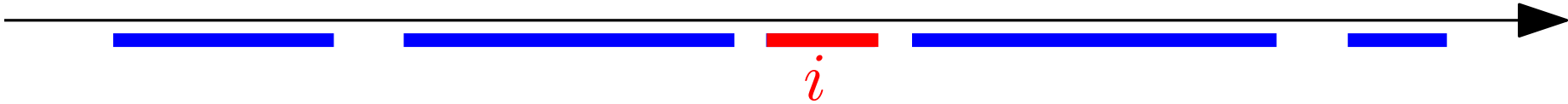


- Soit G un optimum.

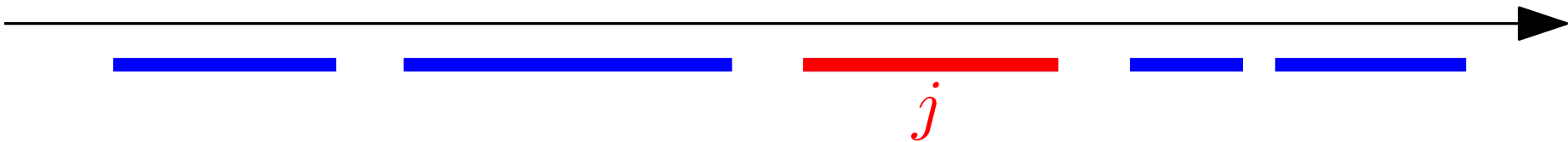


Preuve d'optimalité du glouton pour l'affectation

- Soit F donné par le glouton.



- Soit G un optimum.

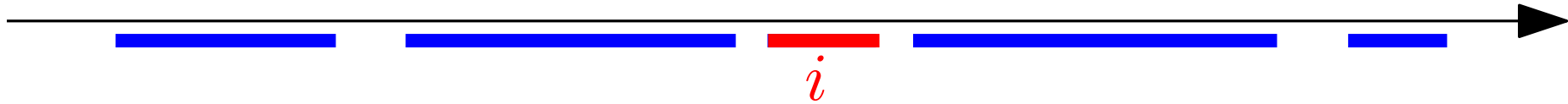


- Le premier élément de G qui n'est pas dans F peut être remplacé par le premier élément de F qui n'est pas dans G .

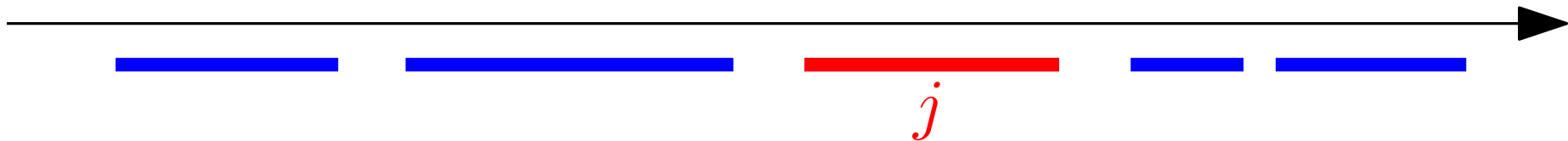
en effet i termine avant j vu le classement des demandes

Preuve d'optimalité du glouton pour l'affectation

- Soit F donné par le glouton.



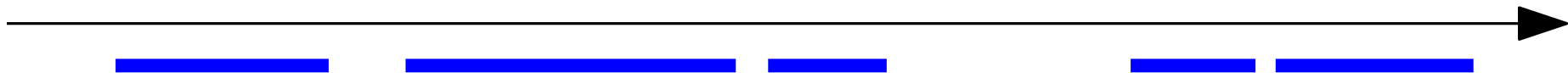
- Soit G un optimum.



- Le premier élément de G qui n'est pas dans F peut être remplacé par le premier élément de F qui n'est pas dans G .

en effet i termine avant j vu le classement des demandes

On obtient $G' = G - \{j\} + \{i\}$, toujours optimum et plus proche de F : $|F \cap G'| > |F \cap G|$.



Cours 1: Algorithmes gloutons

- Optimisation combinatoire
- L'algorithme glouton
- Arbres recouvrants et algorithme de Kruskal
- Optimalité du glouton, matroïde
- Algorithmes de Prim, Dijkstra et gloutoïde

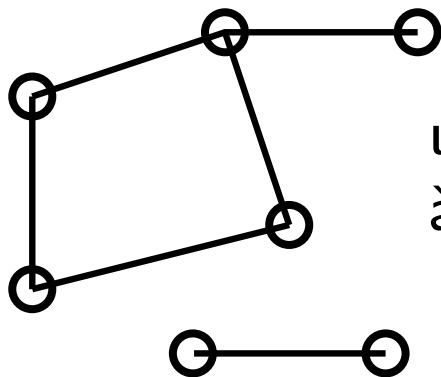
Rappels sur les graphes et les arbres

Un **graphe** $G = (X, E)$ est donné par un ensemble X de **sommets** et un ensemble E d'**arêtes**, une arête étant une paire de sommets.

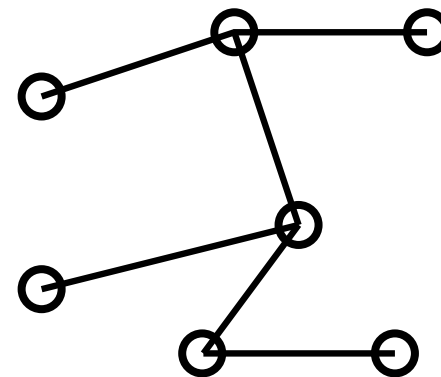
Un **chemin** est une suite $x_0, e_1, x_1, \dots, e_p, x_p$ avec $e_i = \{x_{i-1}, x_i\}$. C'est un **cycle** si $x_0 = x_p$, **simple** si $x_i \neq x_j$ et $e_i \neq e_j$, $0 < i < j \leq p$.

Un graphe est **connexe** si pour tout couple de sommets il existe un chemin qui les joint.

Un **arbre** est un graphe connexe sans cycle simple.



un graphe
à 7 sommets

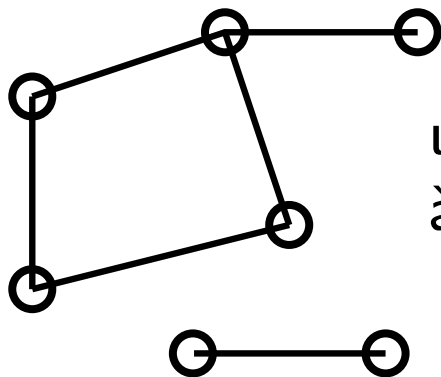


un arbre

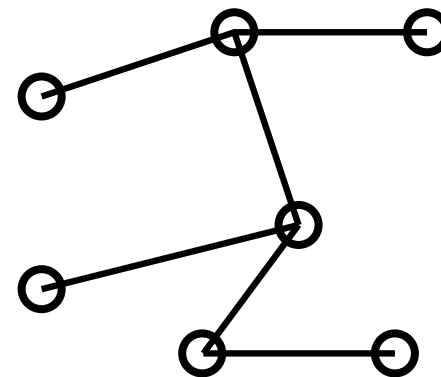
Rappels sur les graphes et les arbres

Les propriétés suivantes sont équivalentes pour un graphe à n sommets au fait d'être un arbre:

- G est connexe et toute suppression d'arête le déconnecte
- G est sans cycle simple et tout ajout d'arête crée un cycle
- G est connexe et possède $n - 1$ arêtes
- G est sans cycle simple et possède $n - 1$ arêtes
- Entre deux sommets de G il existe un unique chemin simple



un graphe
à 7 sommets

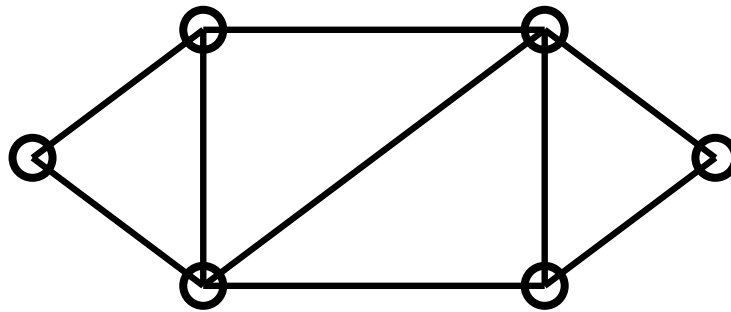


un arbre

Arbres recouvrants d'un graphe

Un **arbre recouvrant** de $G = (X, E)$ est un arbre ayant X pour ensemble de sommets et dont les arêtes sont incluses dans E .

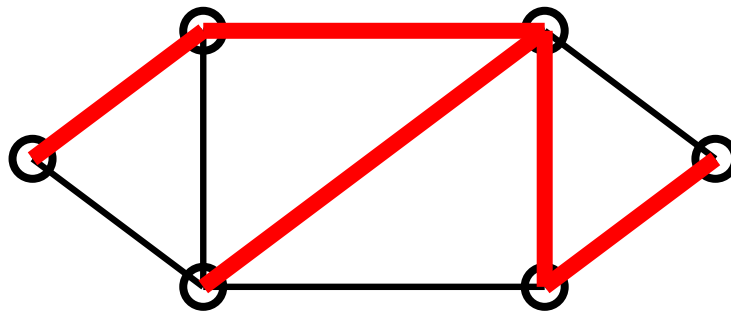
$$G = (X, E)$$
$$n = |X| = 6$$
$$m = |E| = 9$$



Arbres recouvrants d'un graphe

Un **arbre recouvrant** de $G = (X, E)$ est un arbre ayant X pour ensemble de sommets et dont les arêtes sont incluses dans E .

$$G = (X, E)$$
$$n = |X| = 6$$
$$m = |E| = 9$$



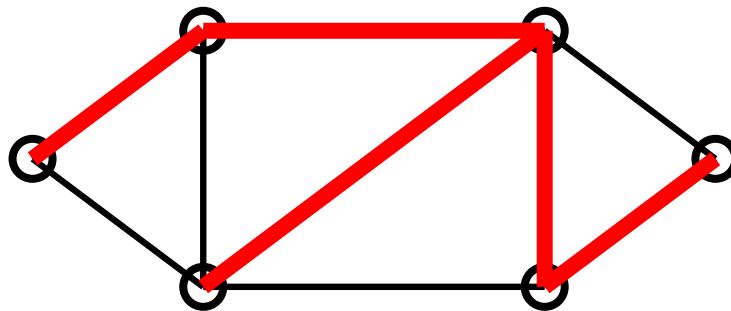
$$(X, T)$$

$$T \subset E$$

Arbres recouvrants d'un graphe

Un **arbre recouvrant** de $G = (X, E)$ est un arbre ayant X pour ensemble de sommets et dont les arêtes sont incluses dans E .

$$G = (X, E)$$
$$n = |X| = 6$$
$$m = |E| = 9$$



$$(X, T)$$

$$T \subset E$$

- D'après les remarques précédentes, $|T| = n - 1$.
- Toute arête $e \notin T$ crée avec T un unique cycle C_e .
- Le nombre de cycles qu'on peut ainsi former est $m - n + 1$.

Arbres recouvrants de fiabilité maximale

Donnée: Un graphe valué, chaque arête a ayant une fiabilité $p(a)$.

Problème: Trouver un arbre couvrant dont la somme des fiabilités des arêtes soit maximale.

\Rightarrow pb d'optimisation avec $\mathcal{F} = \{ \text{les parties de } E \text{ sans cycle simple} \}$.

Arbres recouvrants de fiabilité maximale

Donnée: Un graphe valué, chaque arête a ayant une fiabilité $p(a)$.

Problème: Trouver un arbre couvrant dont la somme des fiabilités des arêtes soit maximale.

\Rightarrow pb d'optimisation avec $\mathcal{F} = \{ \text{les parties de } E \text{ sans cycle simple} \}$.

Algorithme glouton \Rightarrow algorithme de Kruskal

- Classer les arêtes par fiabilité décroissante

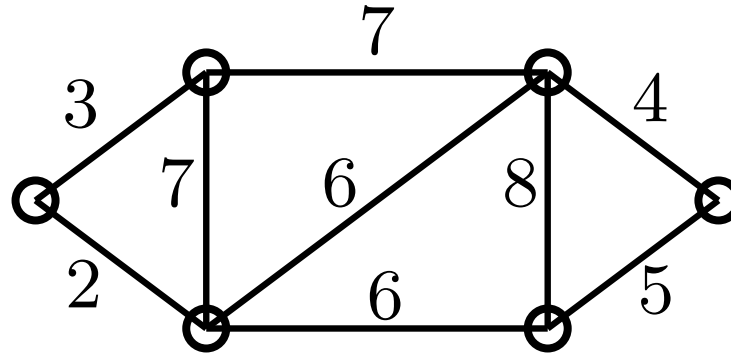
$$p(e_1) \geq p(e_2) \geq \dots \geq p(e_m).$$

- Initialiser avec $F = \emptyset$.

- Pour i de 1 à m faire:

Si $F \cup \{e_i\}$ sans cycle, alors $F := F \cup \{e_i\}$.

Arbres recouvrants de fiabilité maximale



Algorithme glouton = algorithme de Kruskal

- Classer les arêtes par fiabilité décroissante

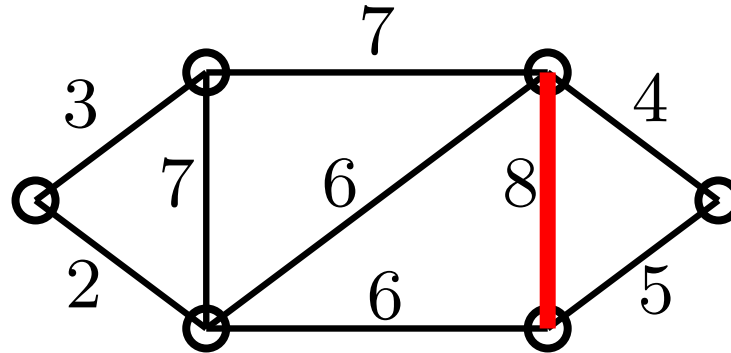
$$p(e_1) \geq p(e_2) \geq \dots \geq p(e_m).$$

- Initialiser avec $F = \emptyset$.

- Pour i de 1 à m faire:

Si $F \cup \{e_i\}$ sans cycle, alors $F := F \cup \{e_i\}$.

Arbres recouvrants de fiabilité maximale



Algorithme glouton = algorithme de Kruskal

- Classer les arêtes par fiabilité décroissante

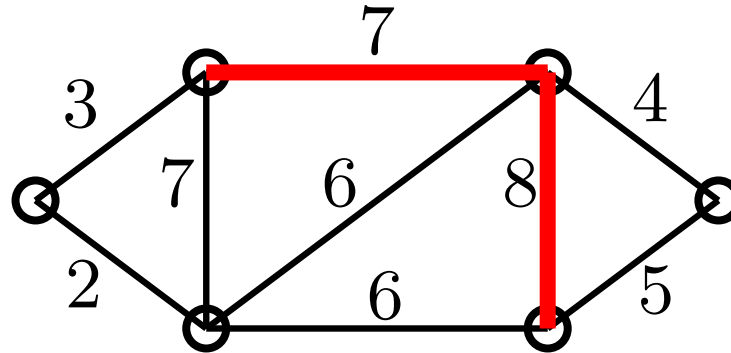
$$p(e_1) \geq p(e_2) \geq \dots \geq p(e_m).$$

- Initialiser avec $F = \emptyset$.

- Pour i de 1 à m faire:

Si $F \cup \{e_i\}$ sans cycle, alors $F := F \cup \{e_i\}$.

Arbres recouvrants de fiabilité maximale



Algorithme glouton = algorithme de Kruskal

- Classer les arêtes par fiabilité décroissante

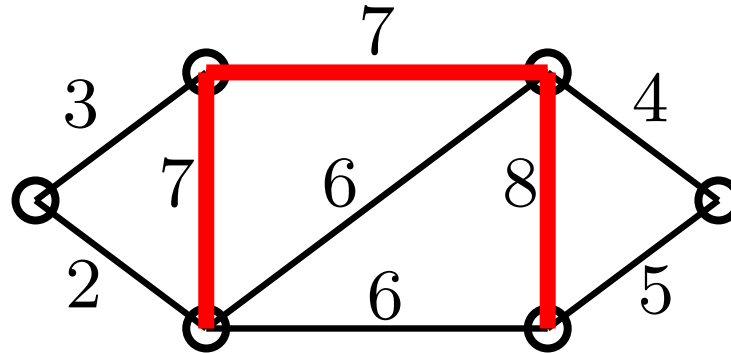
$$p(e_1) \geq p(e_2) \geq \dots \geq p(e_m).$$

- Initialiser avec $F = \emptyset$.

- Pour i de 1 à m faire:

Si $F \cup \{e_i\}$ sans cycle, alors $F := F \cup \{e_i\}$.

Arbres recouvrants de fiabilité maximale



Algorithme glouton = algorithme de Kruskal

- Classer les arêtes par fiabilité décroissante

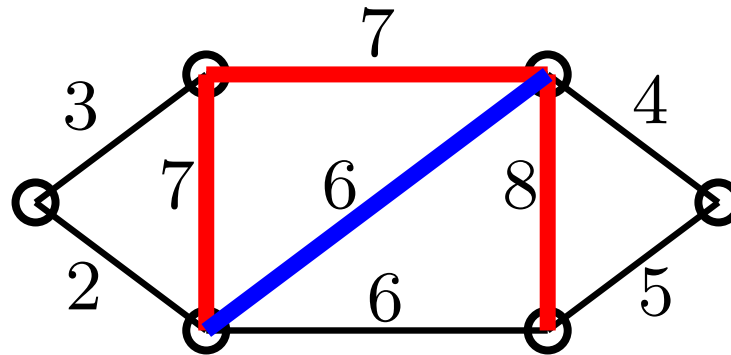
$$p(e_1) \geq p(e_2) \geq \dots \geq p(e_m).$$

- Initialiser avec $F = \emptyset$.

- Pour i de 1 à m faire:

Si $F \cup \{e_i\}$ sans cycle, alors $F := F \cup \{e_i\}$.

Arbres recouvrants de fiabilité maximale



Algorithme glouton = algorithme de Kruskal

- Classer les arêtes par fiabilité décroissante

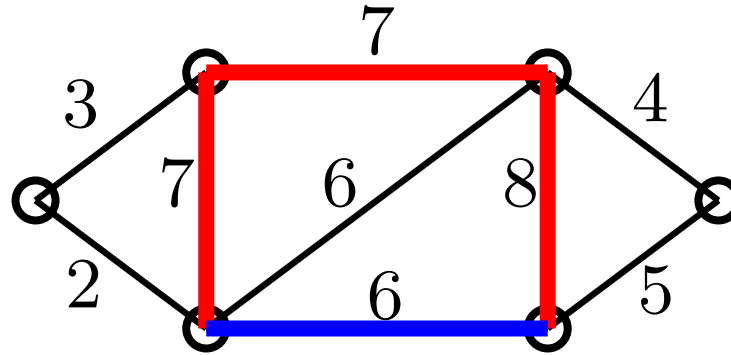
$$p(e_1) \geq p(e_2) \geq \dots \geq p(e_m).$$

- Initialiser avec $F = \emptyset$.

- Pour i de 1 à m faire:

Si $F \cup \{e_i\}$ sans cycle, alors $F := F \cup \{e_i\}$.

Arbres recouvrants de fiabilité maximale



Algorithme glouton = algorithme de Kruskal

- Classer les arêtes par fiabilité décroissante

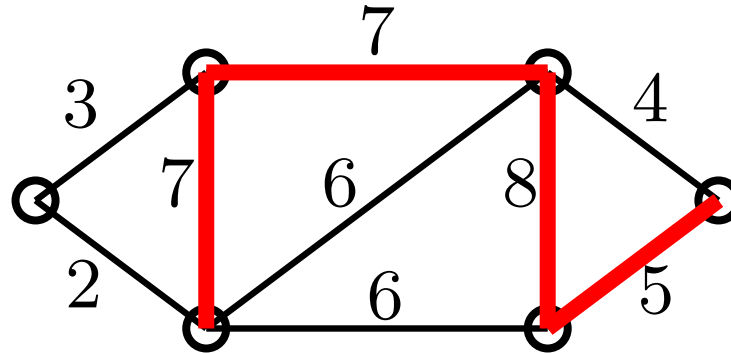
$$p(e_1) \geq p(e_2) \geq \dots \geq p(e_m).$$

- Initialiser avec $F = \emptyset$.

- Pour i de 1 à m faire:

Si $F \cup \{e_i\}$ sans cycle, alors $F := F \cup \{e_i\}$.

Arbres recouvrants de fiabilité maximale



Algorithme glouton = algorithme de Kruskal

- Classer les arêtes par fiabilité décroissante

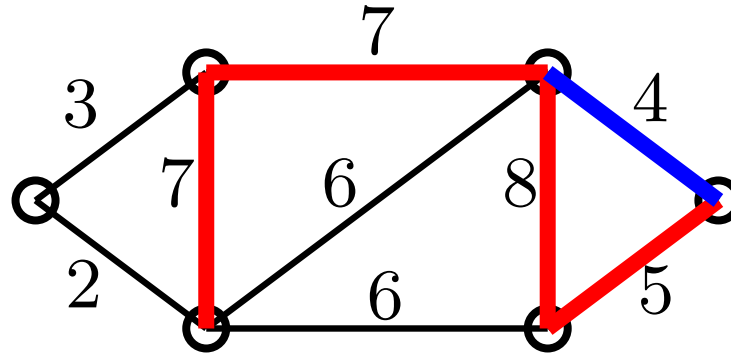
$$p(e_1) \geq p(e_2) \geq \dots \geq p(e_m).$$

- Initialiser avec $F = \emptyset$.

- Pour i de 1 à m faire:

Si $F \cup \{e_i\}$ sans cycle, alors $F := F \cup \{e_i\}$.

Arbres recouvrants de fiabilité maximale



Algorithme glouton = algorithme de Kruskal

- Classer les arêtes par fiabilité décroissante

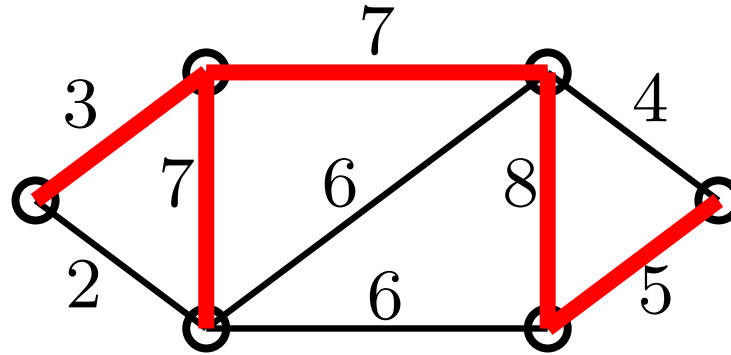
$$p(e_1) \geq p(e_2) \geq \dots \geq p(e_m).$$

- Initialiser avec $F = \emptyset$.

- Pour i de 1 à m faire:

Si $F \cup \{e_i\}$ sans cycle, alors $F := F \cup \{e_i\}$.

Arbres recouvrants de fiabilité maximale



Algorithme glouton = algorithme de Kruskal

- Classer les arêtes par fiabilité décroissante

$$p(e_1) \geq p(e_2) \geq \dots \geq p(e_m).$$

- Initialiser avec $F = \emptyset$.

- Pour i de 1 à m faire:

Si $F \cup \{e_i\}$ sans cycle, alors $F := F \cup \{e_i\}$.

Arbres recouvrants de fiabilité maximale

Preuve de l'optimalité: lemme d'échange

Soient T et U deux arbres recouvrants de G et $a \in U \setminus T$

Il existe $b \in T$ tel que $U \setminus \{a\} \cup \{b\}$ soit un arbre recouvrant.

($T \cup \{a\}$ contient un cycle, qui joint les 2 composantes de $U \setminus \{a\}$)

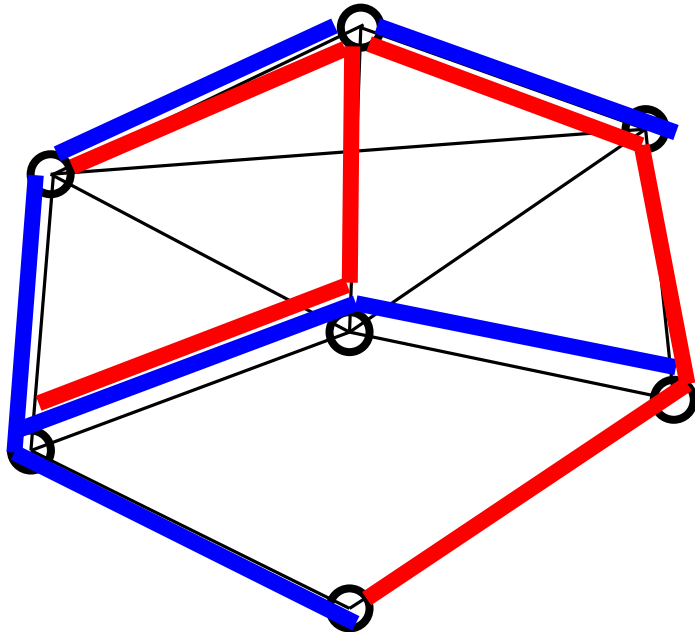
Arbres recouvrants de fiabilité maximale

Preuve de l'optimalité: lemme d'échange

Soient T et U deux arbres recouvrants de G et $a \in U \setminus T$

Il existe $b \in T$ tel que $U \setminus \{a\} \cup \{b\}$ soit un arbre recouvrant.

($T \cup \{a\}$ contient un cycle, qui joint les 2 composantes de $U \setminus \{a\}$)



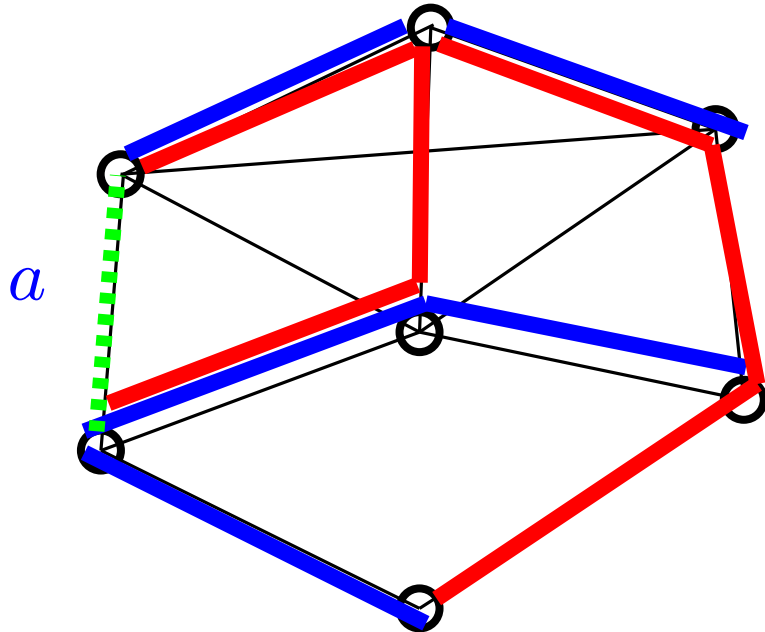
Arbres recouvrants de fiabilité maximale

Preuve de l'optimalité: lemme d'échange

Soient T et U deux arbres recouvrants de G et $a \in U \setminus T$

Il existe $b \in T$ tel que $U \setminus \{a\} \cup \{b\}$ soit un arbre recouvrant.

($T \cup \{a\}$ contient un cycle, qui joint les 2 composantes de $U \setminus \{a\}$)



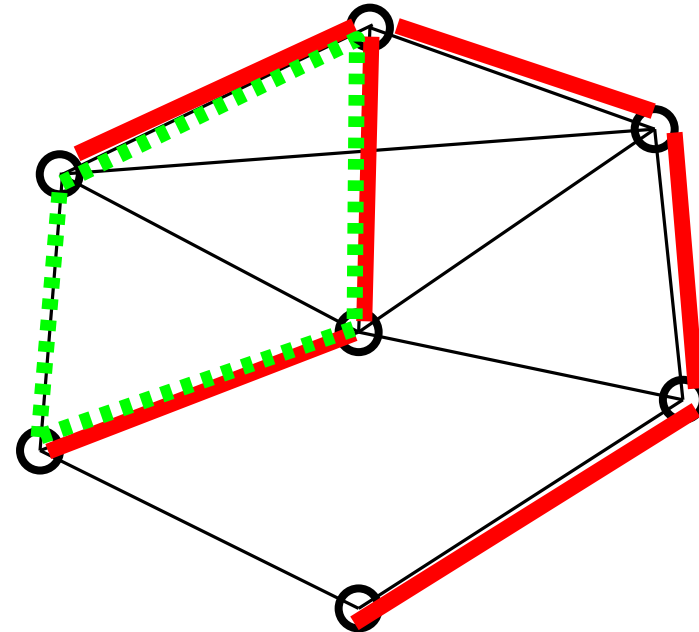
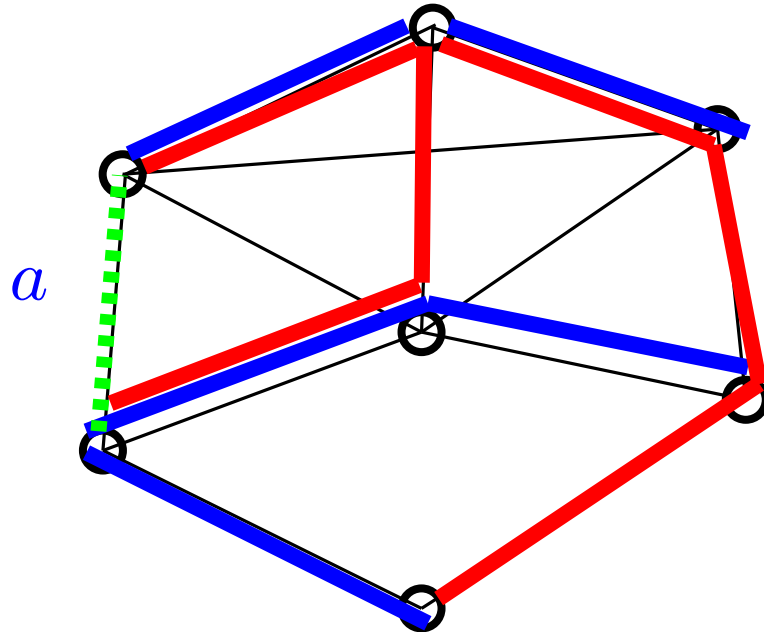
Arbres recouvrants de fiabilité maximale

Preuve de l'optimalité: lemme d'échange

Soient T et U deux arbres recouvrants de G et $a \in U \setminus T$

Il existe $b \in T$ tel que $U \setminus \{a\} \cup \{b\}$ soit un arbre recouvrant.

($T \cup \{a\}$ contient un cycle, qui joint les 2 composantes de $U \setminus \{a\}$)



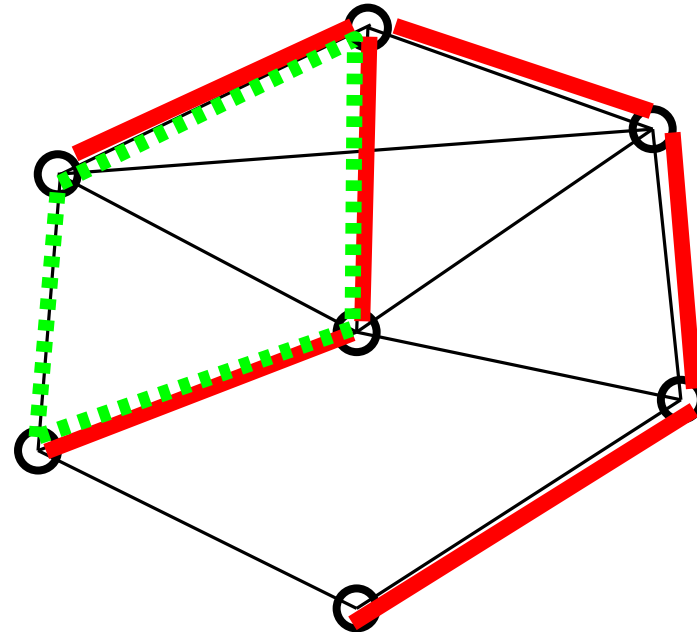
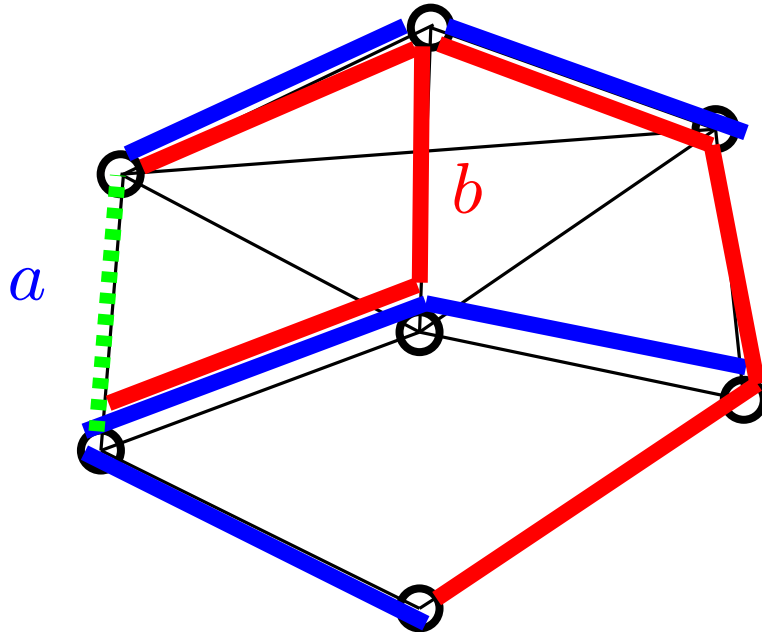
Arbres recouvrants de fiabilité maximale

Preuve de l'optimalité: lemme d'échange

Soient T et U deux arbres recouvrants de G et $a \in U \setminus T$

Il existe $b \in T$ tel que $U \setminus \{a\} \cup \{b\}$ soit un arbre recouvrant.

($T \cup \{a\}$ contient un cycle, qui joint les 2 composantes de $U \setminus \{a\}$)



Arbres recouvrants de fiabilité maximale

Preuve de l'optimalité: lemme d'échange

Soient T et U deux arbres recouvrants de G et $a \in U \setminus T$

Il existe $b \in T$ tel que $U \setminus \{a\} \cup \{b\}$ soit un arbre recouvrant.

($T \cup \{a\}$ contient un cycle, qui joint les 2 composantes de $U \setminus \{a\}$)

Preuve de l'optimalité: itération (similaire à la précédente)

Soient T l'arbre obtenu par l'algo glouton et U un optimum.

Soit a une arête de $U \setminus T$. On applique le lemme pour obtenir b dans T tel que $U' = U \setminus \{a\} \cup \{b\}$ soit un arbre

Le glouton n'a pas retenu a , donc les éléments du cycle de $T \cup \{a\}$ étaient déjà insérés, notamment b : $p(b) \geq p(a)$.

$p(U') = p(U)$ car U optimum, et U' plus proche de T que U .

Cours 1: Algorithmes gloutons

- Optimisation combinatoire
- L'algorithme glouton
- Arbres recouvrants et algorithme de Kruskal
- Optimalité du glouton, matroïde
- Algorithmes de Prim, Dijkstra et gloutoïde

Optimalité de l'algorithme glouton et matroïde

Théorème. Étant donnée une famille \mathcal{F} , l'algorithme glouton est optimal pour toute valuation si et seulement si \mathcal{F} est

stable par passage aux ss-ensembles: $G \subset F \in \mathcal{F} \Rightarrow G \in \mathcal{F}$, (i)

et satisfait l'axiome d'extension: $\forall F, G \in \mathcal{F}$ avec $|G| > |F|$,
il existe $x \in G \setminus F$ tq $F \cup \{x\} \in \mathcal{F}$. (ii)

Optimalité de l'algorithme glouton et matroïde

Théorème. Étant donnée une famille \mathcal{F} , l'algorithme glouton est optimal pour toute valuation si et seulement si \mathcal{F} est

stable par passage aux ss-ensembles: $G \subset F \in \mathcal{F} \Rightarrow G \in \mathcal{F}$, (i)

et satisfait l'axiome d'extension: $\forall F, G \in \mathcal{F}$ avec $|G| > |F|$,
il existe $x \in G \setminus F$ tq $F \cup \{x\} \in \mathcal{F}$. (ii)

Un **matroïde** est une famille non vide \mathcal{F} de parties de E qui satisfait les axiomes (i) et (ii). Les éléments de \mathcal{F} sont appelés les **ensembles indépendants** du matroïdes.

Optimalité de l'algorithme glouton et matroïde

Théorème. Étant donnée une famille \mathcal{F} , l'algorithme glouton est optimal pour toute valuation si et seulement si \mathcal{F} est

stable par passage aux ss-ensembles: $G \subset F \in \mathcal{F} \Rightarrow G \in \mathcal{F}$, (i)

et satisfait l'axiome d'extension: $\forall F, G \in \mathcal{F}$ avec $|G| > |F|$,
il existe $x \in G \setminus F$ tq $F \cup \{x\} \in \mathcal{F}$. (ii)

Un **matroïde** est une famille non vide \mathcal{F} de parties de E qui satisfait les axiomes (i) et (ii). Les éléments de \mathcal{F} sont appelés les **ensembles indépendants** du matroïdes.

Théorème. L'algorithme glouton trouve un optimum pour toute valuation si et seulement si la famille \mathcal{F} est un matroïde.

Preuve de l'équivalence

\mathcal{F} matroïde \Rightarrow optimalité du glouton

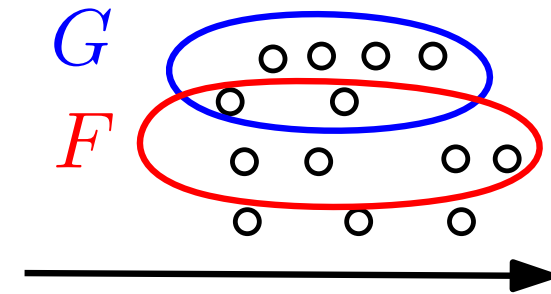
C'est toujours un peu la même preuve !

Preuve de l'équivalence

\mathcal{F} matroïde \Rightarrow optimalité du glouton

C'est toujours un peu la même preuve !

- Soit F obtenu par glouton et G un optimum.

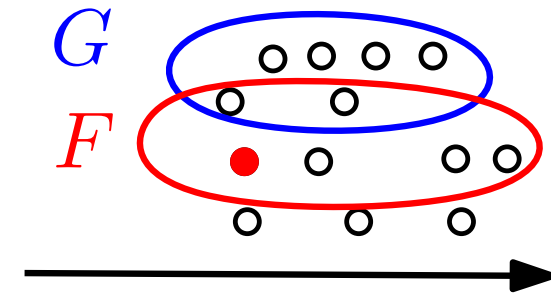


Preuve de l'équivalence

\mathcal{F} matroïde \Rightarrow optimalité du glouton

C'est toujours un peu la même preuve !

- Soit F obtenu par glouton et G un optimum.
- Dans $F \setminus G$, soit x le premier élément choisi par glouton

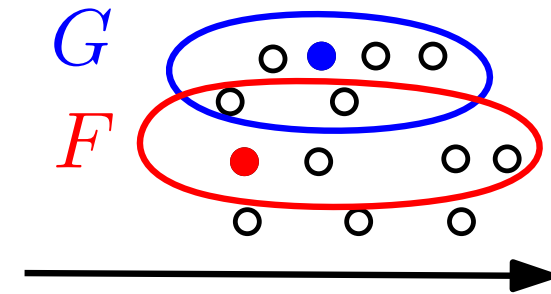


Preuve de l'équivalence

\mathcal{F} matroïde \Rightarrow optimalité du glouton

C'est toujours un peu la même preuve !

- Soit F obtenu par glouton et G un optimum.
- Dans $F \setminus G$, soit x le premier élément choisi par glouton
- Si $y \in G \setminus F$ alors $v(y) \leq v(x)$, sinon y aurait été pris

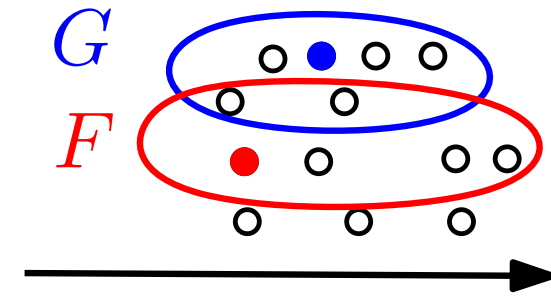


Preuve de l'équivalence

\mathcal{F} matroïde \Rightarrow optimalité du glouton

C'est toujours un peu la même preuve !

- Soit F obtenu par glouton et G un optimum.
- Dans $F \setminus G$, soit x le premier élément choisi par glouton
- Si $y \in G \setminus F$ alors $v(y) \leq v(x)$, sinon y aurait été pris
- Soit $F' = (F \cap G) \cup \{x\}$, puis compléter par extension avec G

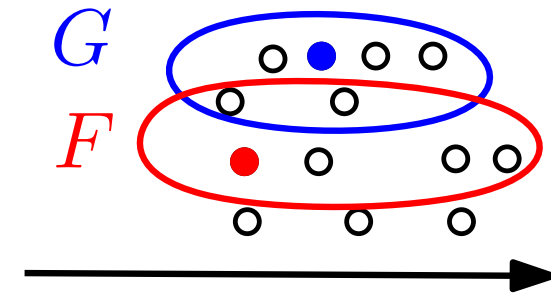


Preuve de l'équivalence

\mathcal{F} matroïde \Rightarrow optimalité du glouton

C'est toujours un peu la même preuve !

- Soit F obtenu par glouton et G un optimum.
- Dans $F \setminus G$, soit x le premier élément choisi par glouton



- Si $y \in G \setminus F$ alors $v(y) \leq v(x)$, sinon y aurait été pris
- Soit $F' = (F \cap G) \cup \{x\}$, puis compléter par extension avec G
- On obtient un certain $G' = G \setminus \{y\} \cup \{x\}$, toujours optimal, mais plus proche de F ... on peut itérer.

Preuve de l'équivalence

Glouton optimal pour toute valuation $\Rightarrow \mathcal{F}$ matroïde

Preuve de l'équivalence

Glouton optimal pour toute valuation $\Rightarrow \mathcal{F}$ matroïde

Il faut vérifier la stabilité de \mathcal{F} par passage aux sous-ensembles et l'axiome d'extension.

Preuve de l'équivalence

Glouton optimal pour toute valuation $\Rightarrow \mathcal{F}$ matroïde

Il faut vérifier la stabilité de \mathcal{F} par passage aux sous-ensembles et l'axiome d'extension.

- supposons qu'il existe $G \subset F \in \mathcal{F}$ avec $G \notin \mathcal{F}$;

on considère la valuation:

- pour $x \in F \setminus G$, $v(x) = a > 0$
- pour $x \in G$, $v(x) = b > a$
- pour $x \notin F \cup G$, $v(x) = 0$

Que fait l'algorithme glouton ?

Preuve de l'équivalence

Glouton optimal pour toute valuation $\Rightarrow \mathcal{F}$ matroïde

- Soient F et G tq $|G| > |F|$: on veut $x \in G \setminus F$ tq $x \cup F \in \mathcal{F}$.

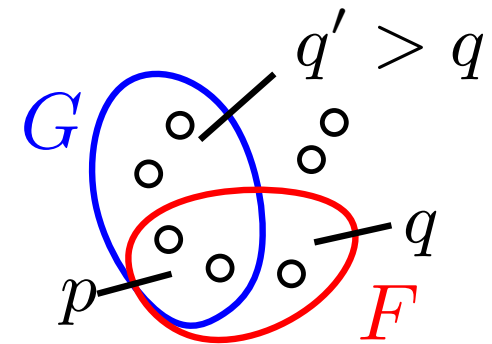
Preuve de l'équivalence

Glouton optimal pour toute valuation $\Rightarrow \mathcal{F}$ matroïde

- Soient F et G tq $|G| > |F|$: on veut $x \in G \setminus F$ tq $x \cup F \in \mathcal{F}$.

On construit une valuation, avec $a > 0$, $b > 0$

- si $e \in F$, alors $v(e) = a + b$,
- si $e \in G \setminus F$, alors $v(e) = a$,
- Si $e \notin F \cup G$, alors $v(e) = 0$,



tq le glouton construisse un optimum T contenant au moins F .

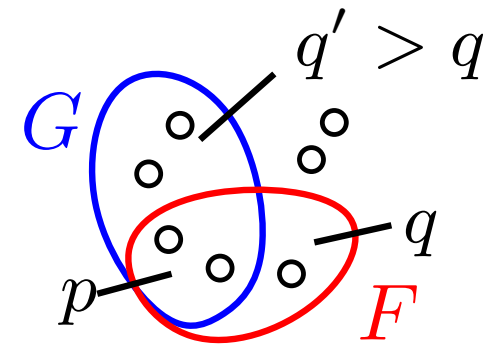
Preuve de l'équivalence

Glouton optimal pour toute valuation $\Rightarrow \mathcal{F}$ matroïde

- Soient F et G tq $|G| > |F|$: on veut $x \in G \setminus F$ tq $x \cup F \in \mathcal{F}$.

On construit une valuation, avec $a > 0$, $b > 0$

- si $e \in F$, alors $v(e) = a + b$,
- si $e \in G \setminus F$, alors $v(e) = a$,
- Si $e \notin F \cup G$, alors $v(e) = 0$,



tq le glouton construisse un optimum T contenant au moins F .

- $v(F) = (p + q)(a + b) = (p + q)a + pb + qb$,
- $v(G) \geq p(a + b) + (q + 1)a = (p + q)a + pb + a$.

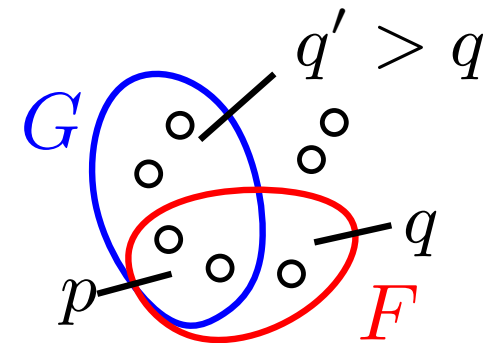
Preuve de l'équivalence

Glouton optimal pour toute valuation $\Rightarrow \mathcal{F}$ matroïde

- Soient F et G tq $|G| > |F|$: on veut $x \in G \setminus F$ tq $x \cup F \in \mathcal{F}$.

On construit une valuation, avec $a > 0$, $b > 0$

- si $e \in F$, alors $v(e) = a + b$,
- si $e \in G \setminus F$, alors $v(e) = a$,
- Si $e \notin F \cup G$, alors $v(e) = 0$,



tq le glouton construise un optimum T contenant au moins F .

- $v(F) = (p + q)(a + b) = (p + q)a + pb + qb$,
- $v(G) \geq p(a + b) + (q + 1)a = (p + q)a + pb + a$.
- Si $a > qb$, alors $v(G) > v(F)$ et il y a d'autres éléments dans T : l'un quelconque de ces éléments est le x recherché.

Exemples de matroïdes

- Les sous-ensembles d'arêtes sans cycle d'un graphe (aussi appelés forêts couvrantes).
- Les ensembles de vecteurs libres d'un espace vectoriel
- Les ensembles d'arcs sans origine commune d'un graphe orienté
- Les transversaux d'une famille \mathcal{C} de parties d'un ensemble E : un transversal est un sous-ensemble $R = \{e_1, \dots, e_p\}$ de E tel qu'il existe des éléments distincts C_1, C_2, \dots, C_p de \mathcal{C} avec $e_i \in C_i$.

Cours 1: Algorithmes gloutons

- Optimisation combinatoire
- L'algorithme glouton
- Arbres recouvrants et algorithme de Kruskal
- Optimalité du glouton, matroïde
- Algorithmes de Prim, Dijkstra et gloutoïde

Algorithme de Prim

Un autre algorithme classique pour l'arbre couvrant de poids max:

- Initialiser l'ensemble des sommets atteints $S := \{x\}$ à un sommet arbitraire et l'arbre partiel à $A := \emptyset$.
- Itérer tant que $S \neq X$
 - soit $e = \{x, y\}$ une arête de poids maximal joignant $x \in S$ à $y \notin S$, faire $A := A \cup \{e\}$, $S := S \cup \{y\}$.

Algorithme de Prim

Un autre algorithme classique pour l'arbre couvrant de poids max:

- Initialiser l'ensemble des sommets atteints $S := \{x\}$ à un sommet arbitraire et l'arbre partiel à $A := \emptyset$.
- Itérer tant que $S \neq X$
 - soit $e = \{x, y\}$ une arête de poids maximal joignant $x \in S$ à $y \notin S$, faire $A := A \cup \{e\}$, $S := S \cup \{y\}$.

Cet algorithme glouton construit aussi un arbre couvrant de poids max.

Algorithme de Prim

Un autre algorithme classique pour l'arbre couvrant de poids max:

- Initialiser l'ensemble des sommets atteints $S := \{x\}$ à un sommet arbitraire et l'arbre partiel à $A := \emptyset$.
- Itérer tant que $S \neq X$
 - soit $e = \{x, y\}$ une arête de poids maximal joignant $x \in S$ à $y \notin S$, faire $A := A \cup \{e\}$, $S := S \cup \{y\}$.

Cet algorithme glouton construit aussi un arbre couvrant de poids max.

Cependant l'ensemble des arêtes utilisables à chaque étape évolue de manière non monotone au cours de l'exécution: l'algo n'entre pas dans le cadre précédent.

Algorithme de Prim

Un autre algorithme classique pour l'arbre couvrant de poids max:

- Initialiser l'ensemble des sommets atteints $S := \{x\}$ à un sommet arbitraire et l'arbre partiel à $A := \emptyset$.
- Itérer tant que $S \neq X$
 - soit $e = \{x, y\}$ une arête de poids maximal joignant $x \in S$ à $y \notin S$, faire $A := A \cup \{e\}$, $S := S \cup \{y\}$.

Cet algorithme glouton construit aussi un arbre couvrant de poids max.

Cependant l'ensemble des arêtes utilisables à chaque étape évolue de manière non monotone au cours de l'exécution: l'algo n'entre pas dans le cadre précédent.

En particulier l'ensemble des sous-arbres issus de x ne forme pas un matroïde (pas de stabilité des sous-ensembles).

Gloutoïdes (en anglais Greedoids)

L'exemple de Prim montre que la condition de stabilité par passage aux sous-ensembles des matroïdes est trop restrictive.

Gloutoïdes (en anglais Greedoids)

L'exemple de Prim montre que la condition de stabilité par passage aux sous-ensembles des matroïdes est trop restrictive.

Un **gloutoïde** est une famille non vide \mathcal{F} de parties de E qui satisfait les axiomes (i') et (ii) suivants:

axiome d'accessibilité: $\forall F \in \mathcal{F}$ il existe $x \in F$ tq $F \setminus \{x\} \in \mathcal{F}$.

axiome d'extension: $\forall F, G \in \mathcal{F}$ avec $|G| > |F|$,
il existe $x \in G \setminus F$ tq $F \cup \{x\} \in \mathcal{F}$.

Gloutoïdes (en anglais Greedoids)

L'exemple de Prim montre que la condition de stabilité par passage aux sous-ensembles des matroïdes est trop restrictive.

Un **gloutoïde** est une famille non vide \mathcal{F} de parties de E qui satisfait les axiomes (i') et (ii) suivants:

axiome d'accessibilité: $\forall F \in \mathcal{F}$ il existe $x \in F$ tq $F \setminus \{x\} \in \mathcal{F}$.

axiome d'extension: $\forall F, G \in \mathcal{F}$ avec $|G| > |F|$,
il existe $x \in G \setminus F$ tq $F \cup \{x\} \in \mathcal{F}$.

L'ensemble des sous-arbres issus d'un sommet x_0 d'un graphe G forme un gloutoïde (the branching greedoid of G in x_0).

Gloutoïdes (en anglais Greedoids)

Une **base** de $X \subset E$ est un indépendant inclus dans X et maximal parmi ceux-ci. Le **rang** de X est le cardinal d'une de ses bases.

Un **circuit** est un sous-ensemble non indépendant minimal. Un sous-ensemble X est **fermé** s'il n'existe pas de circuit dont l'intersection avec le complément de X soit réduite à un seul élément.

Théorème. L'algorithme glouton est optimal pour toute valuation sur un gloutoïde si et seulement si tout ensemble fermé dans le gloutoïde l'est aussi dans le matroïde induit $\{G \mid G \subset F \in \mathcal{F}\}$ (clôture par inclusion).

Application à Prim: Les indépendants du matroïde induit sont les ensembles d'arêtes sans cycles. La fermeture d'un sous-arbre est le sous-graphe induit par ses sommets, et le reste dans le matroïde.

\Rightarrow Prim trouve l'arbre couvrant de poids max.

Comparaison entre algorithmes de Prim et Dijkstra

Dijkstra: trouver les **plus courts chemins** à partir d'une source x_0 .

À chaque sommet x on associe une variable qui contiendra au cours de l'exécution la distance de x à x_0 via les sommets déjà visités.

- Initialiser à ∞ la variable de chaque sommet.
- Le sommet x_0 est déclaré visité, sa variable est mise à 0 et on met à jour la variable des voisins accessibles depuis x_0 .
- Tant qu'il reste des sommets non visités, on visite le plus proche et on met à jour la variable des voisins accessibles depuis lui.

Comparaison entre algorithmes de Prim et Dijkstra

Dijkstra: trouver les **plus courts chemins** à partir d'une source x_0 .

À chaque sommet x on associe une variable qui contiendra au cours de l'exécution la distance de x à x_0 via les sommets déjà visités.

- Initialiser à ∞ la variable de chaque sommet.
- Le sommet x_0 est déclaré visité, sa variable est mise à 0 et on met à jour la variable des voisins accessibles depuis x_0 .
- Tant qu'il reste des sommets non visités, on visite le plus proche et on met à jour la variable des voisins accessibles depuis lui.

Cet algorithme est identique à l'algorithme de Prim, sauf le critère de choix du sommet visité: le plus proche, au lieu de l'arête de poids maximale.

Comparaison entre algorithmes de Prim et Dijkstra

Dijkstra: trouver les **plus courts chemins** à partir d'une source x_0 .

À chaque sommet x on associe une variable qui contiendra au cours de l'exécution la distance de x à x_0 via les sommets déjà visités.

- Initialiser à ∞ la variable de chaque sommet.
- Le sommet x_0 est déclaré visité, sa variable est mise à 0 et on met à jour la variable des voisins accessibles depuis x_0 .
- Tant qu'il reste des sommets non visités, on visite le plus proche et on met à jour la variable des voisins accessibles depuis lui.

Cet algorithme est identique à l'algorithme de Prim, sauf le critère de choix du sommet visité: le plus proche, au lieu de l'arête de poids maximale.

L'optimalité de Dijkstra découle d'une variante du théorème précédent pour les objectifs non nécessairement linéaires mais vérifiant une condition de compatibilité avec le gloutoïde.

Cours 1: Algorithmes gloutons

- Optimisation combinatoire
- L'algorithme glouton
- Arbres recouvrants et algorithme de Kruskal
- Optimalité du glouton, matroïde
- Algorithmes de Prim, Dijkstra et gloutoïde

⇒ A retenir avant tout: La méthode de preuve d'optimalité!

La semaine prochaine, de nouveaux outils:
algorithmes de flots et couplages