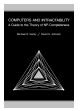
GRAPHES ET COMPLEXITE

Johanne Cohen

LRI-CNRS,
Chargée de Recherche au CNRS.

Références du cours

1. *Computers and Intractability : A Guide to the Theory of NP-Completeness*, M. R. Garey, D. S. Johnson, 1976.



2. *Algorithm Design*, Jon Kleinberg, Eva Tardos, Addison-Wesley, 2006.

3. *The Algorithm Design Manual*, Steven Skiena, Springer 2014.

Déroulement du cours

- 5 séances de cours :
  1. Introduction à la complexité
  2. La complexité : les classes P et NP.
  3. La NP-complétude et les entiers
  4. Algorithmes d'approximation
  5. Au delà de la classe NP.

- Chaque séance est divisée en deux de durée 1H30 chacune : une partie cours et une partie d'exercices.

- Evaluation du cours :

  Il aura lieu le 19 décembre (10h à 12h)
  Les documents sont autorisés.

Introduction à la complexité : algorithmes gloutons

Johanne Cohen

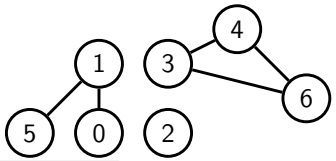
LRI-CNRS,
Chargée de Recherche au CNRS.

Plus précisément

Rappel sur la théorie des graphes
Les graphes
Les arbres

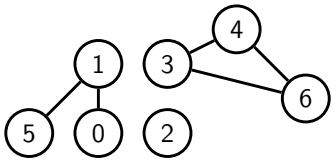
Un graphe

- Un **graphe**[1] donné par un couple $G = (V, E)$, où
  - V est un ensemble.
  - $E \subset V \times V$ est un ensemble de paires $\{u, v\}$ avec $u, v \in V$.
- Les éléments de V sont appelés des **sommets** (ou **nœuds**).
- Les éléments de E sont appelés des **arêtes**.
- La paire $\{u, v\}$ peut être représentée par (u, v) ou (v, u).
  Autrement dit, (u, v) et (v, u) dénotent la même arête.
- Exemple :
  - $V = \{0, 1, \ldots, 6\}$
  - $E = \{(0, 1), (3, 4), (5, 1), (6, 3), (6, 4)\}$.



1. Lorsqu'on ne précise pas, par défaut, un graphe est non-orienté.

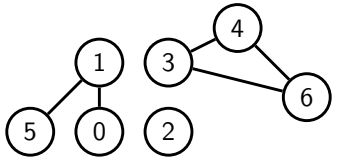
Vocabulaire



- u et v sont dits **voisins** s'il y a une arête entre u et v.
- Le **degré** de u est le nombre de voisins de u.

- Remarque : (sauf autre convention explicite)
  - Les boucles ne sont pas autorisées.

Vocabulaire : chemins et cycles



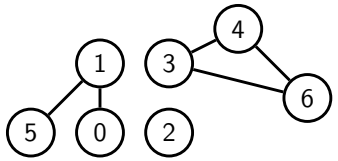
- Un **chemin du sommet** s **vers le sommet** t est une suite $e_0, e_1, \cdots, e_n$ de sommets telle que

  $$e_0 = s,\ e_n = t,\ (e_{i-1}, e_i) \in E, \text{ pour tout } 1 \leq i \leq n.$$

  - n est appelé la **longueur** du chemin,
  - on dit que t **est joignable** à partir de s.
  - Le chemin est dit **simple** si les e_i sont distincts deux-à-deux.
- Un **cycle** est un chemin de longueur non-nulle avec $e_0 = e_n$.
- s est dit **à distance** n de t s'il existe un chemin de longueur n entre s et t, mais aucun chemin de longueur inférieure.

Vocabulaire : composantes connexes

- La relation “être joignable” est une relation d’équivalence.
- Les classes d’équivalence sont appelées les **composantes connexes**.



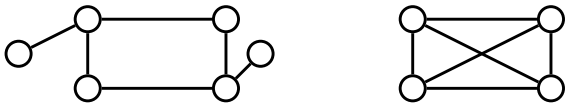
- Un graphe est dit **connexe** s’il n’y a qu’une seule classe d’équivalence.
  - Autrement dit, tout sommet est joignable à partir de tout sommet.

Les graphes sont partout !

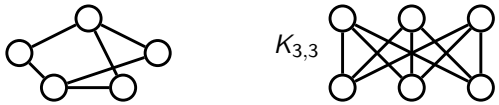
- Beaucoup de problèmes se modélisent par des objets et des relations entre objets.
- Exemples :
  - Le graphe routier.
  - Les réseaux informatiques.
  - Le graphe du web.
- Beaucoup de problèmes se ramènent à des problèmes sur les graphes.
- Théorie des graphes :
  - Euler, Hamilton, Kirchhoff, König, Edmonds, Berge, Lovász, Seymour,...
- Les graphes sont omniprésents en informatique.

Type de graphes

- Un graphe est dit **planaire** s’il peut se représenter sur un plan sans qu’aucune arête n’en croise une autre.

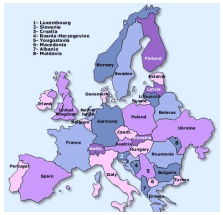


- Un graphe est dit **biparti** si l’ensemble V des sommets est partitionné en deux sous-ensembles A et B telle que chaque arête ait une extrémité dans A et l’autre dans B.



Exemple : Coloriage de graphe.

- Allouer des fréquences GSM correspond à colorier les sommets d’un graphe (planaire).
  - sommets : des émetteurs radio.
  - arête entre u et v : le signal de u perturbe v ou réciproquement.
  - couleur : fréquence radio.
- Le problème de **coloriage d’un graphe** : colorier les sommets d’un graphe de telle sorte qu’il n’y ait aucune arête entre deux sommets d’une même couleur.



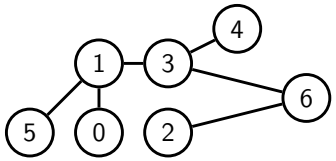
Un coloriage avec 4 couleurs

Plus précisément

Rappel sur la théorie des graphes
Les graphes
Les arbres

Les arbres sont partout !

- Un graphe connexe sans cycle est appelé un **arbre**.
- Un graphe sans-cycle est appelé une **forêt** :
  - chacune de ses composantes connexes est un arbre.
- Dès qu'on a des objets, des relations entre objets, et pas de cycle, on a donc un arbre ou une forêt.

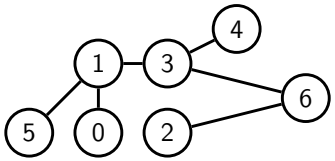


- Les arbres sont omniprésents en informatique.

Quelques propriétés

Soit $G = (V, E)$ un graphe.
Les propriétés suivantes sont équivalentes :

- G est un arbre ;
- G est connexe, mais ne l'est plus si on enlève n'importe laquelle de ses arêtes ;
- G est connexe et $|E| = |V| - 1$;
- G est sans cycle et $|E| = |V| - 1$;



Plus précisément

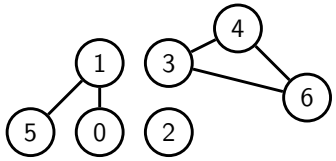
Représentation des graphes
Matrice d'adjacences
Liste de successeurs

Représentation des graphes : matrice d'adjacence

Soit $G = (V, E)$ avec $V = \{1, 2, \ldots, n\}$,

- G peut être représenté par une matrice M $n \times n$.

$$M_{i,j} = \begin{cases} 1 & \text{si } (i,j) \in E \\ 0 & \text{sinon} \end{cases}$$



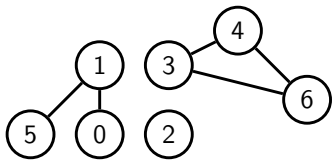
$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

Plus précisément

Représentation des graphes
Matrice d'adjacences
Liste de successeurs

Représentation des graphes : liste de successeurs

- On associe à chaque sommet i, la liste des sommets j tels que $(i,j) \in E$.



- $L[0] = (1)$
- $L[1] = (0, 5)$
- $L[2] = ()$
- $L[3] = (4, 6)$
- $L[4] = (3, 6)$
- $L[5] = (1)$
- $L[6] = (3, 4)$

Meilleure représentation ?

- Matrice : mémoire $O(n^2)$
- Listes : mémoire $O(n + m)$
  où n nombre de sommets, m nombre d'arêtes.

- Quelle est la meilleure représentation ?
  Cela dépend du contexte.

	La méthode plus efficace est
Tester si (u, v) est dans le graphe.	matrice d'adjacences
Tester si calculer le degré de v	liste de successeurs
Stocker des graphes denses	matrice d'adjacences
Stocker des graphes creux	liste de successeurs
Insérer ou supprimer des arêtes	matrice d'adjacences

Un premier exemple : problème du stockage.

Considérons n programmes $P_1, P_2, \ldots, P_n$ qui peuvent être stoquer sur un disque dur de capacité D gigabytes.

- Chaque programme P_i a besoin s_i gigabytes pour être stocké.
- Tous les programmes ne peuvent pas être stockés sur le disque : $(\sum_{i=1}^{n} s_i > D)$

Objectif :

Concevoir un algorithme qui permet de maximiser le nombre de programmes stockés sur le disque.

Problème d'optimisation

Données :

$\mathcal{X}$ un ensemble fini,
v une valuation des éléments de $\mathcal{X}$: $\mathcal{X} \to \mathbb{R}^+$,
$\mathcal{F}$ un ensemble de parties de $\mathcal{X}$ (ensemble des solutions faisables)

Objectif : Trouver $S \in \mathcal{F}$ qui maximise la quantité $\sum_{e \in S} v(e)$

Remarque : Si $\mathcal{F}$ est l'ensemble des parties de $\mathcal{X}$, alors l'algorithme "*force brute*" nécessite au moins $\mathcal{O}(2^{|\mathcal{X}|})$ opérations.

Ordre de grandeur

Pour un processeur capable d'effectuer un million d'instructions élémentaires par seconde :

taille	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	4 s
$n = 30$	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	18 m	10^{25} a
$n = 50$	< 1 s	< 1 s	< 1 s	< 1 s	11 m	36 a	∞
$n = 10^2$	< 1 s	< 1 s	< 1 s	1s	12.9 a	10^{17} a	∞
$n = 10^3$	< 1 s	< 1 s	1s	18 m	∞	∞	∞
$n = 10^4$	< 1 s	< 1 s	2 m	12 h	∞	∞	∞
$n = 10^5$	< 1 s	2 s	3 h	32 a	∞	∞	∞
$n = 10^6$	1s	20s	12 j	31710 a	∞	∞	∞

Notations :

$s =$ seconde, $m =$ minute, $h =$ heure, $a =$ an,
$\infty =$ le temps dépasse 10^{25} années.

Un algorithme résolvant le problème du stockage.

Rappel : Trouver un sous-ensemble de $\{P_1, \ldots, P_n\}$ de cardinal maximum qui peut être stockés sur le disque de capacité D.

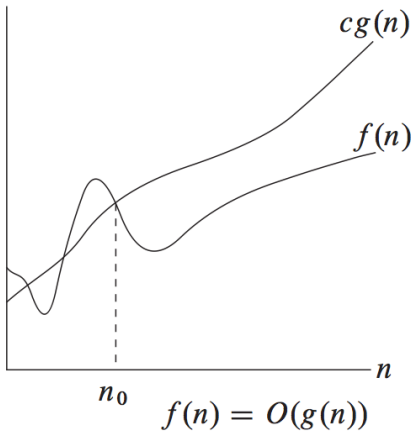
1. Classer les programmes $P_1, P_2, \ldots, P_n$ en fonction de la taille du programme s_i : $\mathcal{O}(n \log n)$ opérations
   $s_1 \leq s_2 \leq \cdots \leq s_n$
2. Initialiser la recherche avec $S = \emptyset$ $\mathcal{O}(1)$ *opérations*
3. Pour $i = 1$ à n faire : *la boucle est exécutée n fois*
   Si $\underbrace{\sum_{P_j \in S} s_j + s_i \leq D}_{\mathcal{O}(1) \text{ opérations}}$ alors $\underbrace{S \leftarrow S \cup \{P_i\}}_{\mathcal{O}(1) \text{ opérations}}$
4. Retourner S

Complexité (au pire) : $\mathcal{O}(n \log n)$

Exemple : Considérons qu'on dispose 4 programmes de tailles 3, 5, 7, 10, et un disque dur de capacité $D = 17$.

L'algorithme retourne $S = \{P_1, P_2, P_3\}$.

Notation $\mathcal{O}$



(image scannée dans le livre *Introduction to Algorithms* de Thomas H. Cormen Charles E. Leiserson Ronald L. Rivest Clifford Stein)

Théorème

Théorème

L'algorithme retourne une solution optimale.

Preuve :

- Il suffit de prouver par récurrence qu'une solution optimale S contient les $|S|$ premiers programmes ayant les plus petites tailles.
- Soit S une solution optimale
  - si S contient le programme P_1, alors c'est bon.
  - si S ne contient pas le programme P_1, alors

en remplacant un élément de S par P_1,

la solution obtenue reste optimale.

- et ainsi de suite en considérant les programmes $P_2, P_3 \ldots, P_{|S|}$.

Un algorithme glouton : principe général

- Pour un problème d'optimisation,
  on construit la solution de façon séquentielle, en faisant à chaque étape le meilleur choix local.
- Pas de retour en arrière : on va directement vers une solution.
- Progression descendante = choix puis résolution d'un problème plus petit.

Un algorithme glouton générique

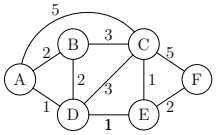
1. Classer les éléments de $\mathcal{X}$ par valuations décroissantes :
   $v(e_1) \geq v(e_2) \geq \cdots \geq v(e_n)$
   $\mathcal{O}(n \log n)$ opérations
2. Initialiser la recherche avec $S = \emptyset$; $\mathcal{O}(1)$ *opérations*
3. Pour $i = 1$ à n faire : *la boucle est exécutée n fois*
   Si $\underbrace{S \cup \{e_i\} \in \mathcal{F}}_{\mathcal{O}(T) \text{ opérations}}$ alors $\underbrace{S \leftarrow S \cup \{e_i\}}_{\mathcal{O}(1) \text{ opérations}}$;
4. Retourner S

Complexité : $\mathcal{O}(n \log n + nT)$

où T correspond au coût du test $(S \cup \{e_i\} \in \mathcal{F})$

Graphes pondérés

- Un graphe **pondéré** est un graphe où chaque arête e a un poids $w(e)$.



- Par abus de notation, nous étendrons la définition de poids aux ensembles A d'arêtes :

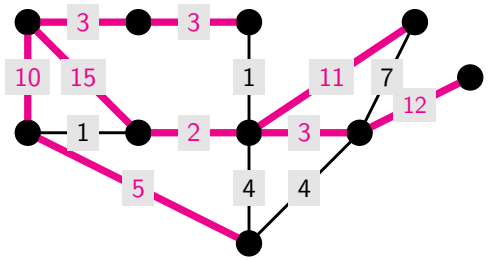
$$w(A) = \sum_{e \in A} w(e)$$

Arbres couvrants de poids maximum

- Le problème de **l'arbre couvrant de poids maximun** : construire un arbre couvrant dont la somme des poids des arêtes est maximun.

  Données : $\begin{cases} \text{Un graphe } G = (V, E) \text{ ;} \\ \text{Une fonction de poids } w : E \to \mathbb{R}^{\geq 0} \end{cases}$

  Objectif : Trouver un arbre couvrant T qui maximise la quantité $w(T) = \sum_{e \in T} w(e)$.



Algorithme de Kruskal

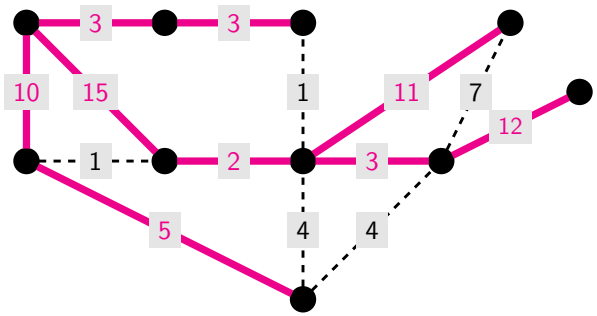
Entrée : $\begin{cases} \text{Un graphe } G = (V, E) \text{ ;} \\ \text{Une fonction de poids } w : E \to \mathbb{R}^{\geq 0}. \end{cases}$

Sortie : Un ensemble T d'arêtes formant un arbre couvrant.

1. Trier les éléments de E par ordre de poids décroissants ;
   $$w(e_1) \geq w(e_2) \geq \ldots \geq w(e_{|E|})$$
2. $T := \emptyset$;
3. Pour $i := 1$ à $|E|$
   3.1 Si $T \cup \{e_i\}$ est acyclique alors $T := T \cup \{e_i\}$;
4. Retourner T.

Remarque : à chaque étape de l'algorithme, T forme une forêt.

Exemple



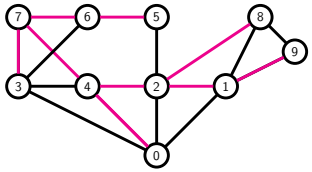
$15 > 12 > 11 > 10 > 7 > 5 > 4 \geq 4 > 3 \geq 3 \geq 3 > 3 > 1 \geq 1$

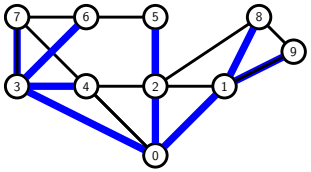
Lemme

Lemme
Soit T et U deux arbres couvrants de G. Soit a une arête telle que $a \in U$, $a \notin T$.

> Il existe $b \in T$ tel que $U \backslash \{a\} \cup \{b\}$ soit un arbre couvrant.
>
> De plus, b peut être choisi dans le cycle formé par des arêtes de T et de a.

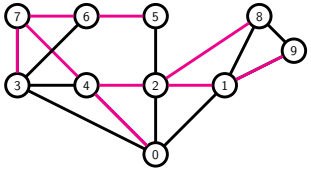
Preuve :

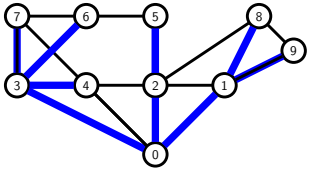




Preuve du lemme

- la suppression de a dans U divise l'arbre en 2 composantes connexes U_1 et U_2.
- Le cycle γ crée par a dans T 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.
- On vérifie qu'une telle arête b satisfait bien aux conditions du lemme.





Théorème

Théorème
L'algorithme retourne bien une solution optimale.

Preuve :

- Soit T l'arbre donné par l'algorithme glouton.
- Si T n'est pas l'arbre couvant de poids maximun,
  alors il existe des arbres couvrants de poids supérieur.
- Notons U celui parmi ceux-ci dont le nombre d'arêtes communes avec T est plus grand possible.
- Soit a l'arête de U de plus grand poids qui n'est pas dans T, et b l'arête donnée par le lemme (précédent) pour ce cas.

Fin de la preuve du théorème

- Comme a n'a pas été choisie par l'algorithme glouton, cela signifie
  1. qu'elle crée un cycle avec les éléments de T,
  2. que ce cycle $\mathcal{C}$ est formé d'arêtes toutes de poids supérieur ou égal à $w(a)$.
- Puisque b est dans ce cycle $\mathcal{C}$, on a $w(b) \geq w(a)$.
- Ainsi $w(U \backslash \{a\} \cup \{b\}) \geq w(U) > w(T)$.
- Mais $U \backslash \{a\} \cup \{b\}$ est un arbre qui a une arête commune de plus avec T que U.
- Ceci contredit le choix de U.

□

Remarques

- L'algorithme fonctionne si G n'est pas connexe.
- On peut aussi obtenir l'arbre de poids minimun :

  considérer une nouvelle fonction de poids w' telle que $w'(e) = M - w(e)$ avec $M = \max\{w(e) | e \in E\}$.

Complexité au pire : $O(|E| \log |E|)$.

Aujourd'hui

- Rappels sur la terminologie des graphes
- Problèmes d'optimisation
- Algorithmes gloutons
- Arbres recouvrants et algorithme de Kruskal

La semaine prochaine :

introduction à la complexité et à la définition de *problèmes difficiles*.