UFR d'Informatique

Algorithmique L3

2019–2020

F. Laroussinie

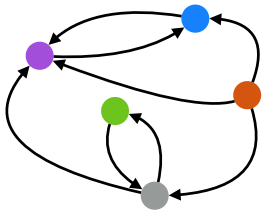
objectifs

Apprendre à manipuler les algorithmes.

- concevoir
- analyser
- chercher dans la littérature
- comprendre
- modifier

Contenu

→ Les algorithmes dans les graphes.



- Parcours
- Composantes (fortement) connexes
- Plus courts chemins
- Arbres couvrants minimaux
- Flots

Prerequis

- Structure de données:
  → piles, files, files de priorités (tas).
- Algorithmes classiques:
  - recherche dichotomique,
  - tri,
  - backtracking,
  - algorithmes sur les arbres,
  - arbres binaires de recherche…

EA3, EA4

Fonctionnement du cours

Cours et TD

Cours: vendredi 10h45 → 12h45, amphi 1A
TD1: lundi 8h30 →10h30, 2027 SG;
TD2: mercredi 10h45 →12h45, 2027 SG;
TD3: jeudi 8h30→10h30, 2027 SG;
TD4: vendredi 8h30→10h30, 2027 SG;
TD MI: jeudi 15h30→17h30, 2027 ? SG.

Équipe enseignante: François Laroussinie, Anne Micheli, Valia Mitsou, Roberto Mantaci, Pierre Charbit, Fabien de Montgolfier.

francois.laroussinie@univ-paris-diderot.fr
http://www.irif.fr/~francoisl/l3algo.html

Fonctionnement du cours

Des notes de cours seront distribuées vendredi 21 septembre.

Avec:
- des définitions sur les graphes,
- les principaux algorithmes, et
- leurs preuves.

En cours: présentation des algorithmes de base, exemples, discussion sur les preuves + des développements.
Le programme de l'examen et du partiel = tout.

Attention au mode d'emploi du poly ! c'est en plus du cours et des TD.

Poly - mode d'emploi

Les notes de cours sont essentiellement limitées aux algorithmes et leurs preuves.
→ A lire après chaque cours.

Pourquoi insister sur les preuves ?

- important d'avoir des algorithmes (et des programmes) corrects !
- souvent ce n'est pas le cas !

→ il faut être capable de trouver des erreurs.
→ il faut être capable de s'assurer qu'un algo est correct.

Quand la complexité dérape…

Déroulement du semestre

Début des cours: semaine du 9 septembre.
Début des TD: semaine du 16 septembre.

Un examen (60%) + du contrôle continu (40%)

Examen, partiel: aucun document autorisé.

Ce qu'il ne faut pas faire

Ecrire algorythme (cf. présentation de M. Habib)

Croire que cela vient du grec *algos* qui signifie douleur

Ecrire algorithmie

Aller en cours et pas en TD

Aller en TD et pas en cours

N'aller ni en cours ni en TD

(Arriver en retard, partir en avance, entrer et sortir pendant le cours.)

Ce qu'il faut faire

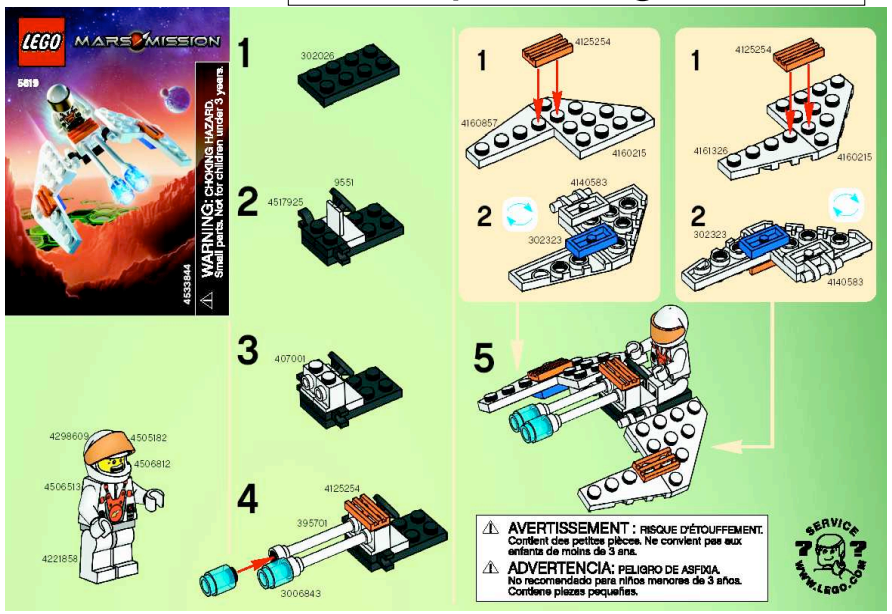
Venir en cours et en en TD, arriver à l'heure et partir à la fin.

Travailler sur le poly et les notes prises en cours et en TD.

Et programmer les algorithmes !

C'est parti !

Exemple d'algorithme



Exemple d'algorithme

les données

Résultat

484 Émincé de rouget au pistou

24 heures à l'avance
Préparation : 40 mn - Cuisson : 30 à 35 mn

1 kg 500 rougets.
400 g pâtes.
500 g fenouil.
2 dl huile olive.
0 dl 5 vinaigre xérès.
Coulis de tomates.
1 citron.
Basilic.
Herbes de Provence.
Thym.
Vinaigrette.
Sel.
Poivre.

Écailler les rougets, les laver, puis soulever délicatement les filets et enlever les arêtes qui restent avec une pince à épiler. Mettre les filets dans un plat creux ; arroser avec de l'huile d'olive ; saupoudrer d'herbes de Provence et laisser mariner au frais, pendant 24 heures, le tout couvert par un torchon.
Cuire le fenouil à l'eau bouillante salée, citronnée et parfumée à l'huile d'olive et avec une brindille de thym (15 à 20 mn). Égoutter.
Cuire les pâtes à l'eau bouillante salée, huilée, pendant 12 mn. Égoutter. Rafraîchir. Tenir au chaud.
Pendant ces cuissons préparer le coulis de tomates (41). Assaisonner le fenouil coupé en tranches avec la vinaigrette. Tenir au chaud. Assaisonner les pâtes avec le basilic (à volonté) haché et un mélange d'huile d'olive et de vinaigre de xérès. Disposer les pâtes sur le plat, recouvrir avec la fondue de fenouil.
Rapidement, cuire à la poêle Téfal, à feu très vif, les filets marinés pour qu'ils deviennent dorés et croustillants. Saler. Poivrer et disposer en étoile les filets de rougets, sur le plat de légumes. Servir avec le coulis de tomates en saucière.

Un algorithme est une procédure (une "recette") pour obtenir un résultat en un nombre fini d'opérations.

Un algorithme résout un *problème*.

Les problèmes algorithmiques

- Données: deux nombres *a* et *b*
  Résultat: *a+3b+ab*
- Données: un nombre *a*
  Résultat: la somme *1+2+3+...+a*
- Données: un ensemble de pièces de monnaie, une valeur *S*
  Résultat: un ensemble de pièces de somme *S*
- Données: une liste de mots
  Résultat: la liste triée dans l'ordre alphabétique
- Données: deux textes
  Résultat: la liste des mots communs aux deux textes
- Données: une carte, deux villes *A* et *B*
  Résultat: un chemin le plus court entre *A* et *B*
- Données: un programme *P*, une donnée *a*
  Résultat: réponse à "est-il vrai que le résultat de *P* appliqué à *a* vaut *8a+5*" ?

problème / instance

Un problème:

- Données: une liste de mots
  Résultat: la liste triée dans l'ordre alphabétique

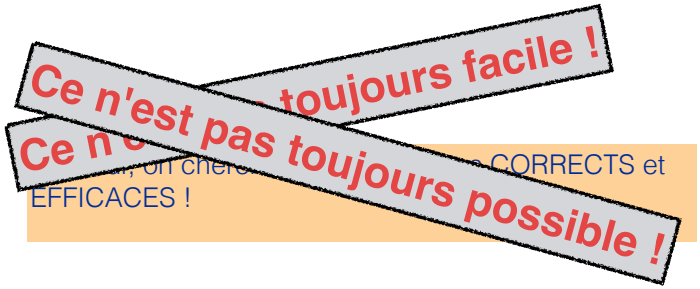
Un algorithme doit résoudre toutes les instances d'un problème

Des in

- table, vélo, chaise, pain, chocolat, tram, ciel, pomme
- Noémie, Pierre, Alain, Célia, Juan, Leila, John
- pomme, oiseau, vélo, fgsdhgf, hjdshq, sqdkjhd
-
- bip, bop, bip, tip

Problèmes et algorithmes

Pour un problème, il y a parfois plusieurs algorithmes très différents...



Algorithmes *corrects* : y en-a-t-il ?

Algorithmes *efficaces* : notions de complexité (pire cas, en moyenne, amortie…)

Algorithmes *optimaux*: peut-on faire mieux ?

Evaluer l'efficacité d'un algorithme

Pire cas, en moyenne, amortie…

$C_A(x)$: nombre d'opérations élémentaires nécessaires pour l'exécution de l'algorithme A sur la donnée x.

Complexité (coût) dans le pire cas:

$$C_A(n) = \max_{x.|x|=n} C_A(x)$$

Complexité en moyenne:

$$C_A^{moy}(n) = \sum_{x.|x|=n} p(x).C_A(x)$$

distribution de probabilités sur les données de taille n

Complexité amortie:
Evaluation du coût cumulé de *n* opérations (dans le pire cas).

→ complexité en temps.
[on peut aussi considérer sa complexité en espace mémoire.]

Complexité des algorithmes

Obj: avoir un ordre de grandeur du nombre d'opérations…

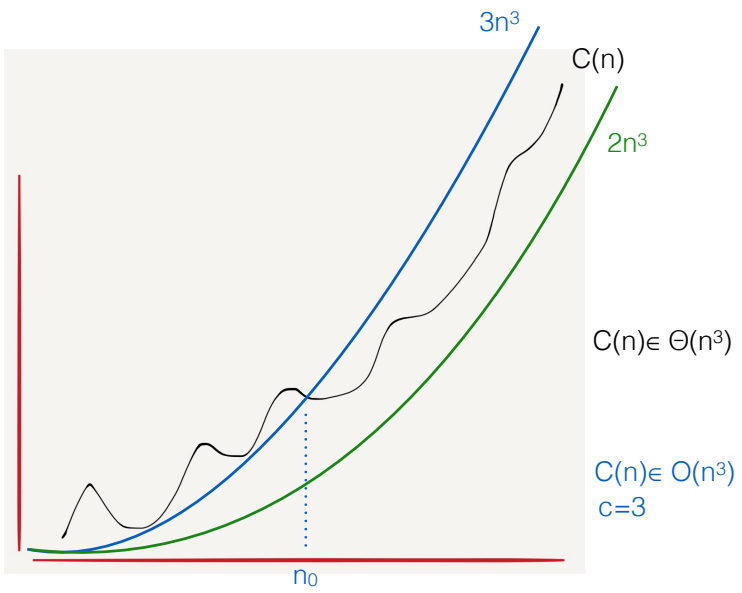
Notations: O( ), Ω( ) et Θ( ):

$O(g(n)) = \{f(n) \mid \exists c>0, n_0 \geq 0 \text{ tq } 0 \leq f(n) \leq c.g(n) \ \forall\ n \geq n_0\}$
→ ensemble des fonctions *majorées* par c.g(n)

$\Omega(g(n)) = \{f(n) \mid \exists c>0, n_0 \geq 0 \text{ tq } 0 \leq c.g(n) \leq f(n) \ \forall\ n \geq n_0\}$
→ ensemble des fonctions *minorées* par c.g(n)

$\Theta(g(n)) = \{f(n) \mid \exists c_1, c_2>0, n_0 \geq 0 \text{ tq } 0 \leq c_1.g(n) \leq f(n) \leq c_2.g(n) \ \forall\ n \geq n_0\}$
→ ensemble des fonctions *encadrées* par c_1.g(n) et c_2.g(n)

Notations: O,Ω,Θ



Notations: O,Ω,Θ

- $3n^2+4n+5 \in O(n^2)$ et $3n^2+4n+5 \in O(n^5)$ …
- $3n^2+4n+5 \in \Theta(n^2)$ mais $3n^2+4n+5 \notin \Theta(n^5)$
- $8n^3-3n \in \Theta(n^3)$
- $2^n+2n^3 \in \Theta(2^n)$
- $|\sin(n)| \in \Theta(1)$
- $2n+1 \in \Theta(n)$
- $3^n \notin O(2^n)$
- $(n+1)! \notin O(n!)$
- $n! \in O(n^n)$

Les algorithmes efficaces… et les autres.

On s'intéresse à des grandes familles de fonctions:

- les algorithmes sous-linéaires.
  Par ex. en O(log n)
- les algorithmes linéaires: O(n)
  ou "quasi-linéaires" comme O(n.log n)
- les algorithmes polynomiaux $O(n^k)$

- les algorithmes exponentiels: $O(2^n)$
- …

Les algorithmes efficaces… et les autres.

fct $\backslash n$	10	50	100	300	1000
$5n$	50	250	500	1500	5000
$n \cdot \log_2(n)$	33	282	665	2469	9966
n^2	100	2500	10000	90000	$10^6(7c)$
n^3	1000	125000	$10^6(7c)$	$27 \cdot 10^6(8c)$	$10^9(10c)$
2^n	1024	$\ldots(16c)$	$\ldots(31c)$	$\ldots(91c)$	$\ldots(302c)$
$n!$	$3.6 \cdot 10^6(7c)$	$\ldots(65c)$	$\ldots(161c)$	$\ldots(623c)$	…!!!
n^n	$10 \cdot 10^9(11c)$	$\ldots(85c)$	$\ldots(201c)$	$\ldots(744c)$	…!!!!

notation: (Xc) -> "s'écrit avec X chiffres en base 10"

(voir « Algorithmics, the spirit of computing », D. Harel)

Les algorithmes efficaces… et les autres.

Avec un ordinateur exécutant 10^9 instructions par seconde…

Fonc.\ n	20	40	60	100	300
n^2	1/2500 milliseconde	1/625 milliseconde	1/278 milliseconde	1/100 milliseconde	1/11 milliseconde
n^5	1/300 seconde	1/10 seconde	78/100 seconde	10 secondes	40,5 minutes
2^n	1/1000 seconde	18,3 minutes	36,5 année	400.10^9 siècles	(72c) siècles
n^n	$3,3.10^9$ années	(46c) siècles	(89c) siècles	(182c) siècles	(725c) siècles

On situe le big-bang à environ $13,8.10^9$ années !

(voir « Algorithmics, the spirit of computing », D. Harel)

Les algorithmes efficaces… et les autres.

En 10 ans, la vitesse des ordinateurs a été multipliée par 50...

Supposons qu'aujourd'hui, on puisse résoudre un problème de taille K en une heure...

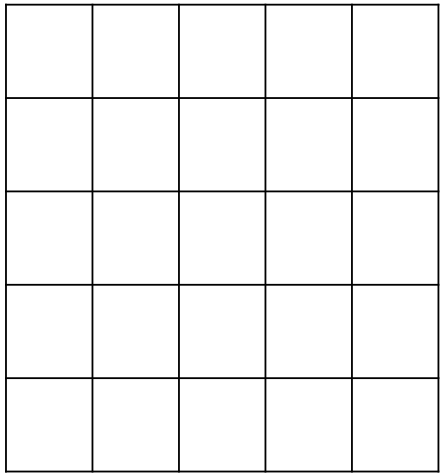
Si l'algorithme a une complexité n, alors...
- un ordinateur 100 fois plus rapide, résoudra des pb de taille 100xK.
- un ordinateur 1000 fois plus rapide, résoudra des pb de taille 1000xK.

Si l'algorithme a une complexité n^2, alors...
- un ordinateur 100 fois plus rapide, résoudra des pb de taille 10xK.
- un ordinateur 1000 fois plus rapide, résoudra des pb de taille 32xK.

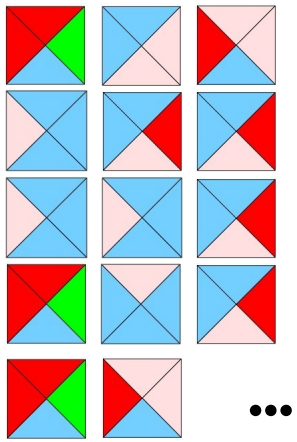
Si l'algorithme a une complexité 2^n, alors...
- un ordinateur 100 fois plus rapide, résoudra des pb de taille 7+K.
- un ordinateur 1000 fois plus rapide, résoudra des pb de taille 10+K.

Un puzzle...

Une grille 5x5



25 dominos



Puzzle

On essaie toutes les possibilités ?

25 cartes à placer sur 25 cases:

- 25 possibilités pour la première,
- 24 pour la seconde,
- 23 pour la troisième,
- ...

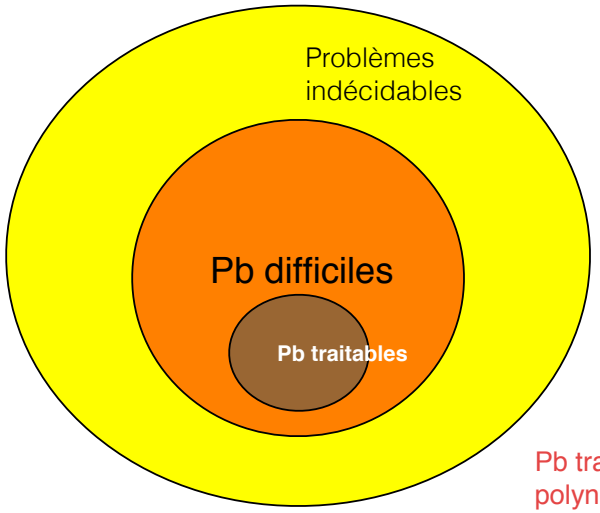
25x24x23x...x2 possibilités !

Avec un ordinateur qui évalue 1 milliard de cas par seconde, il faudrait ...

490 millions d'années !

(25! s'écrit avec 26 chiffres...)

Vue globale



Pb traitables = algo. polynomial

Comparaison d'algorithmes « efficaces »

$n^2 \neq 2^n$ OK !

Mais n^3, n^2, $n.log\ n$, ou n ce n'est pas non plus « pareil » !

Comparaison d'algorithmes « efficaces »

Exemple

Voir J. Bentley «Pearls of programming»

suite de cases consécutives

Un problème:
Etant donné un tableau de nombres (positifs ou négatifs) de taille n, calculer la somme maximale des éléments d'un sous-tableau.

8	-10	10	4	-19	40	0	5	-9	14	2	3	78	7	-24	6	9	-18	7	2

somme max:= 140

(≠ somme de tous les éléments = 115)

Comparaison d'algorithmes « efficaces »

Un exemple: le calcul de la somme maximale des éléments d'un sous-tableau

Donnée: un tableau T[0..n-1]

Résultat: Max { sum[i,j] | $0 \leq i,j \leq n-1$ }

$$sum[i,j] = \sum_{k \in [i;j]} T[k]:$$

intervalle: i,i+1,...,j

Comparaison d'algorithmes « efficaces »

Un exemple: le calcul de la somme maximale des éléments d'un sous-tableau

On peut facilement trouver plusieurs algorithmes pour problème:

- un algorithme naïf (3 boucles « for » imbriquées) en $O(n^3)$
- une version améliorée en $O(n^2)$
- une version « diviser pour régner » en $O(n.\log n)$
- une version en $O(n)$

Tous polynomiaux… mais sensiblement différents !

J. Bentley «Pearls of programming», Addison-Wesley (1986)

pb: calcul de la somme maximale des éléments d'un sous-tableau

TABLE I. Summary of the Algorithms

Algorithm		1	2	3	4
Lines of C Code		8	7	14	7
Run time in microseconds		$3.4N^3$	$13N^2$	$46N \log N$	$33N$
Time to solve problem of size	10^2	3.4 secs	130 msecs	30 msecs	3.3 msecs
	10^3	.94 hrs	13 secs	.45 secs	33 msecs
	10^4	39 days	22 mins	6.1 secs	.33 secs
	10^5	108 yrs	1.5 days	1.3 min	3.3 secs
	10^6	108 mill	5 mos	15 min	33 secs
Max problem solved in one	sec	67	280	2000	30,000
	min	260	2200	82,000	2,000,000
	hr	1000	17,000	3,500,000	120,000,000
	day	3000	81,000	73,000,000	2,800,000,000
If N multiplies by 10, time multiplies by		1000	100	10+	10
If time multiplies by 10, N multiplies by		2.15	3.16	10–	10

1984... machine: VAX-11/750

CRAY-1 vs TRS 80 ?

TABLE II. The Tyranny of Asymptotics

N	Cray-1, FORTRAN, Cubic Algorithm	TRS-80, BASIC, Linear Algorithm
10	3.0 microsecs	200 millisecs
100	3.0 millisecs	2.0 secs
1000	3.0 secs	20 secs
10,000	49 mins	3.2 mins
100,000	35 days	32 mins
1,000,000	95 yrs	5.4 hrs

J. Bentley «Pearls of programming», Addison-Wesley (1986)

Et aujourd'hui ?

2012: Macbook Air, i7...

(en secondes)

	$O(n^3)$	$O(n^2)$	$O(n \cdot log\ n)$	$O(n)$
	Algo 1	Algo 2	Algo 3	Algo 4
100	0,0213	0,0018	0,0006	0,0001
500	2,0193	0,0423	0,0031	0,0003
1 000	16,2401	0,1736	0,007	0,0005
2 000	125,9673	0,7059	0,0154	0,0011
10 000		17,5162	0,0788	0,0052
30 000		158,801	0,2664	0,0174
100 000			1,0107	0,0657
1 000 000			10,1928	0,5407

rappel 1984: 1h 22min 15min 33sec

Remarque n°1

La notion de complexité (ou de coût, d'efficacité…) d'un algorithme est robuste…

Remarque n°2

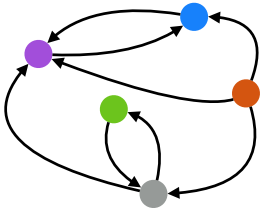
Il y a de fortes différences en pratique entre ces algorithmes !

Et pourtant... ce sont tous des algorithmes dits «*efficaces*» !

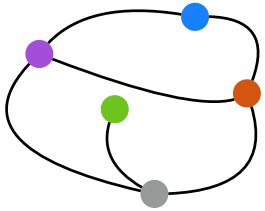
il y a beaucoup de problèmes pour lesquels
- il n'y a que des algorithmes très inefficaces… ou
- il n'y a même pas d'algorithme !

Les algorithmes dans les graphes

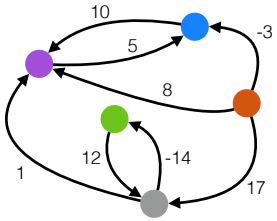
Les graphes



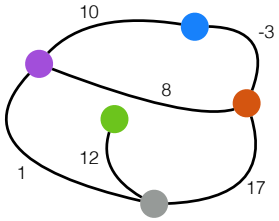
graphe orienté



graphe non orienté



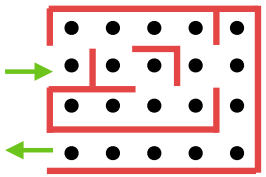
graphe orienté valué

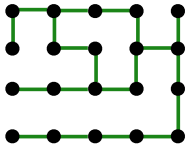


graphe non orienté valué

pour quoi faire ?

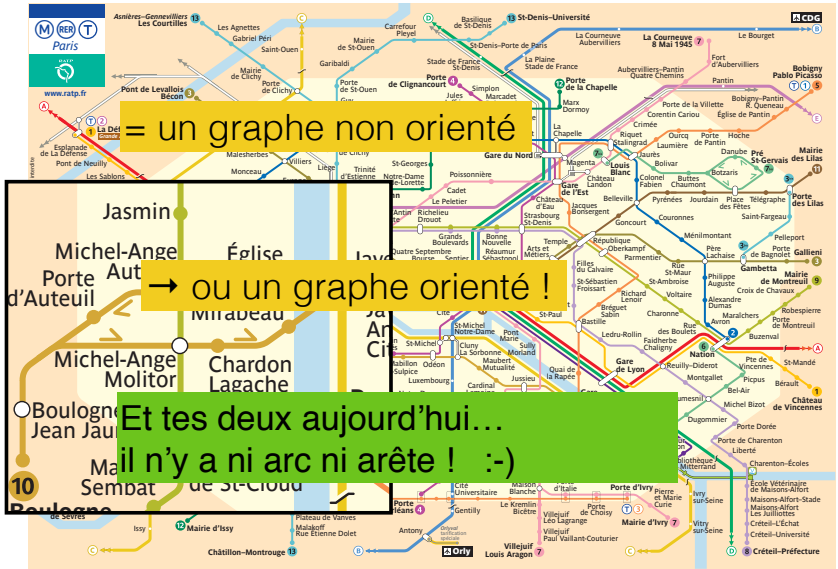
Un labyrinthe…





= un graphe non orienté !

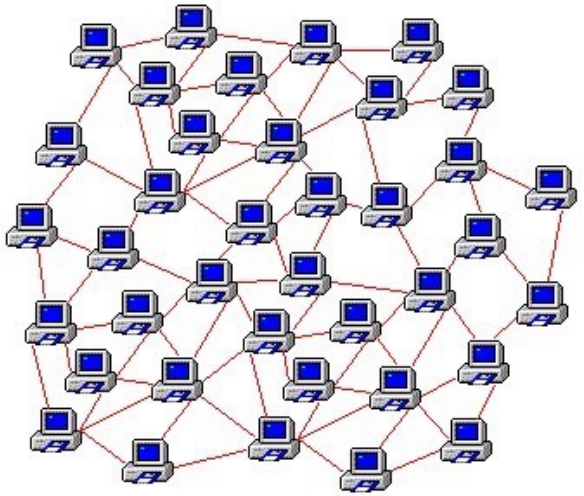
Le réseau de métro…



Une carte routière

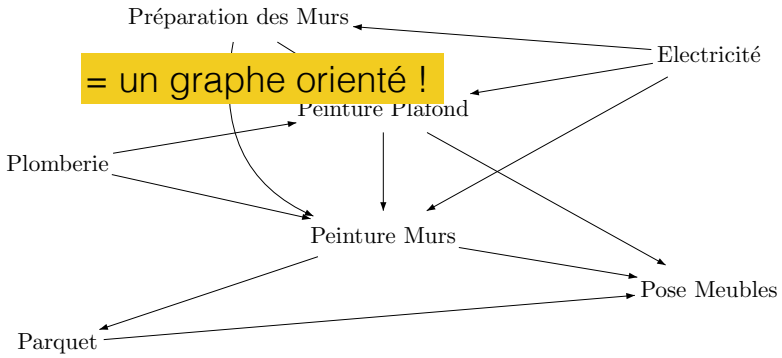
= un graphe non orienté valué !

Un réseau d'ordinateurs



Des relations binaires

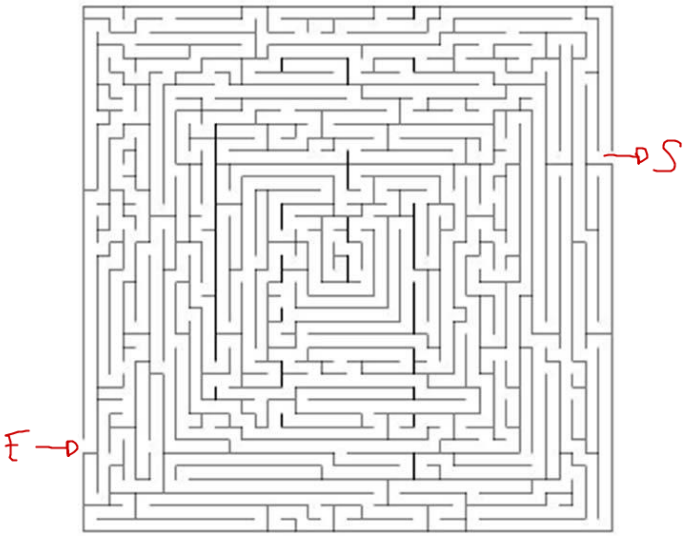
La peinture se fait après l'électricité,
la peinture du plafond se fait avant celle des murs, …



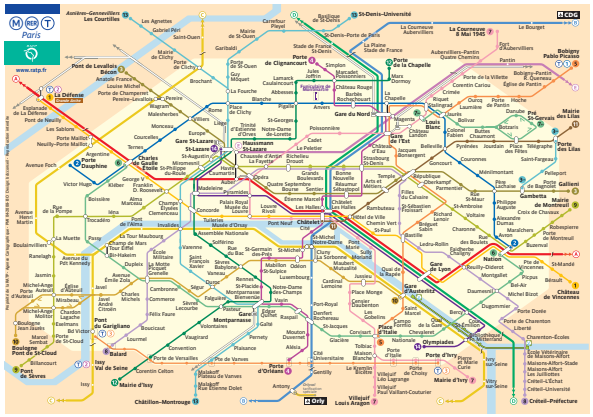
= un graphe orienté !

Les problèmes algorithmiques classiques sur les graphes

Comment sortir ?



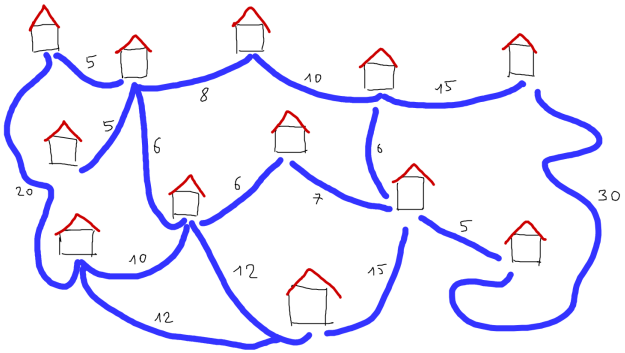
Plus courts chemins



Comment aller de Cité Universitaire à Richard Lenoir le plus rapidement possible ?

Le problème de la ville embourbée

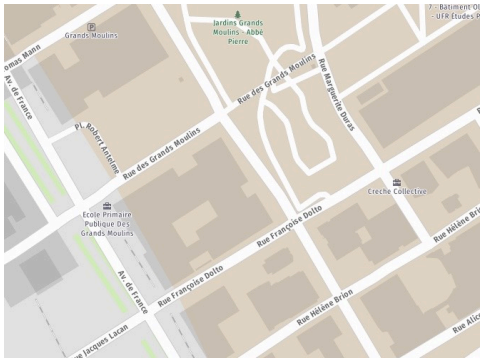
Problème : Paver suffisamment de rues pour que tous les habitants puissent se rendre n'importe où les pieds au sec... mais en utilisant le moins de pavés possible.



(recherche d'un arbre couvrant minimal)

le problème de la goudronneuse

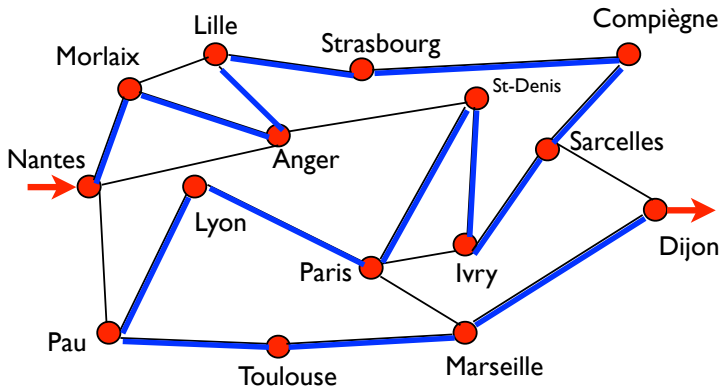
Problème : On veut faire passer une goudronneuse une et une seule fois par chaque rue.
(on s'autorise à repasser aux croisements...)



(recherche d'un chemin eulérien)

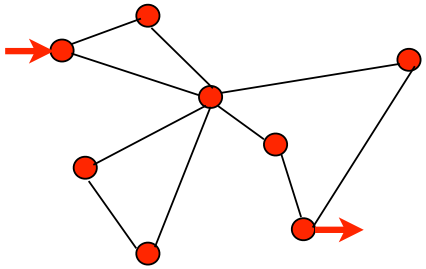
Le parcours des bandits

Des bandits veulent “visiter” toutes les villes d’une carte sans jamais repasser par une ville...



Le parcours des bandits

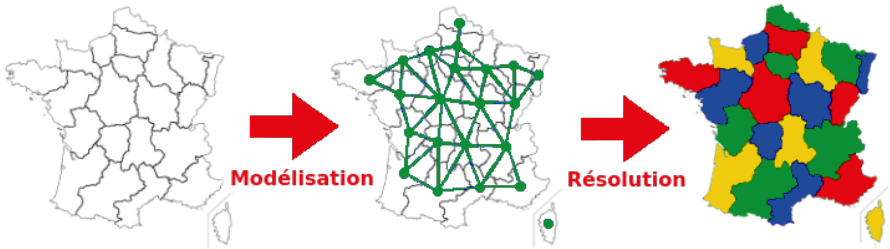
Et ici ?



(recherche de chemins hamiltoniens)

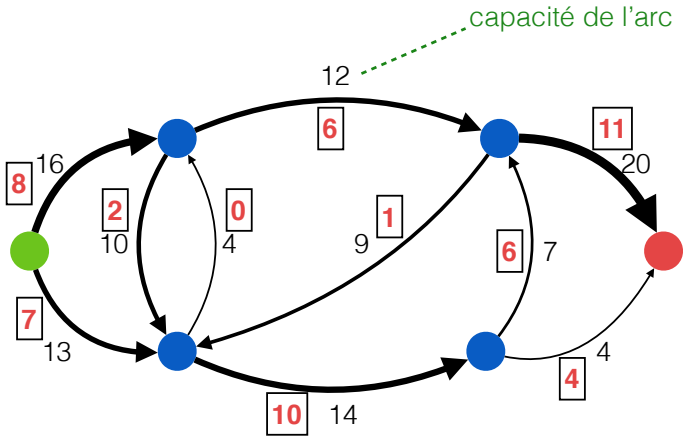
⚠ problème difficile !!!

Colorier une carte



(image de paco-solver.fr)

Flots et réseaux de transport

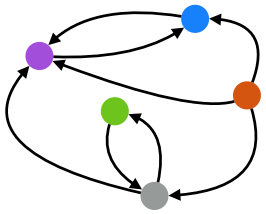


La valeur du « flot » est 15.

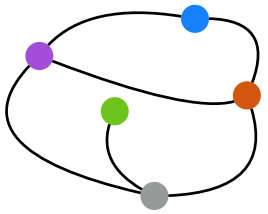
Quel est le flot maximal ?

Graphes
définitions

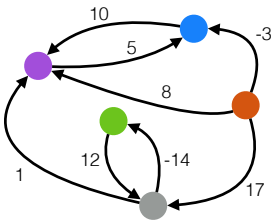
Les graphes



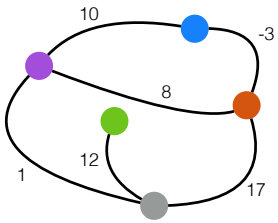
graphe orienté



graphe non orienté

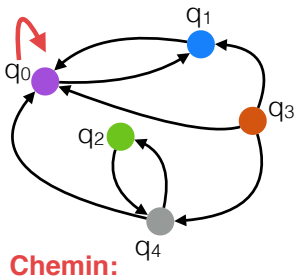


graphe orienté valué



graphe non orienté valué

Les graphes orientés



G=(S,A)
S = ensemble fini de sommets
A ⊆ SxS = ensemble de couples (paires ordonnées), appelés les **arcs** (ou transition)

S={q_0,q_1,q_2,q_3,q_4}
A = { (q_0,q_1), (q_1,q_0), (q_2,q_4), (q_4,q_2), (q_4,q_0), (q_3,q_0), (q_3,q_4), (q_3,q_1)}

Chemin:
$q_3q_4q_0$, $q_3q_4q_2q_4q_0$, $q_3q_4q_2q_4q_2q_4q_0q_1$…

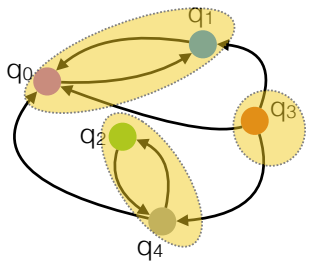
Def: une suite de sommets de S (v_0,v_1,..,v_k) telle que (v_i,v_{i+1})∈A ∀ i<k.

Notation: $v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_k$
On note: $q \rightarrow^* q'$ lorsqu'il existe un chemin reliant q à q'.

Un chemin est élémentaire si tous ses sommets sont distincts.

Circuit: Un circuit est un chemin (v_0,v_1,..,v_k) avec $v_0=v_k$ et comprenant au moins 1 arc.
Boucle: un circuit avec un seul arc.

Les graphes orientés



G=(S,A)
S = ensemble fini de sommets
A ⊆ SxS = ensemble de couples (paires ordonnées), appelés les **arcs** (ou transition)

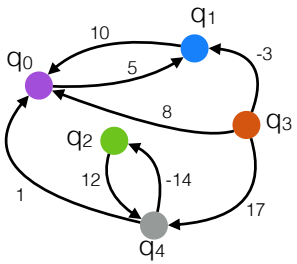
$q \rightarrow^* q'$ lorsqu'il existe un chemin reliant q à q'.

Une composante fortement connexe C est un ensemble maximal de sommets tel que si u,v ∈ C, alors $u \rightarrow^* v$ et $v \rightarrow^* u$.

[ou: les CFC sont les classes d'équivalence de la relation ~ définie par $u \sim v$ ssi $u \rightarrow^* v$ et $v \rightarrow^* u$.]

Un graphe G est **fortement connexe** ssi il n'a qu'une seule CFC.

Les graphes orientés valués



G=(S,A,w) avec
S = ensemble fini de sommets
A ⊆ SxS = ensemble de couples (paires ordonnées), appelés les **arcs** (ou transition), et w : A → **R:** w associe un poids, une distance, un coût… à chaque arc de A.

Mêmes notions de chemin, circuit, boucle…
Mais un chemin a désormais un poids (distance, coût) en plus de sa taille:

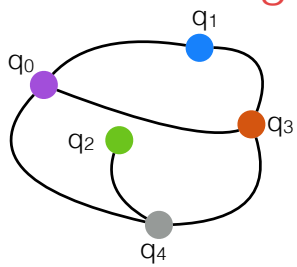
Chemin:

$q_3q_4q_0$, $q_3q_4q_2q_4q_0$, $q_3q_4q_2q_4q_2q_4q_0q_1$…

Poids de $q_3q_4q_0$ = 17+1 = 18, (taille = 2)
Poids de $q_3q_4q_2q_4q_0$ = 17 -14 + 12 + 1 = 16 (taille = 4)

$w(v_0,v_1,..,v_k) = \sum_{i=0..k-1} w(v_i,v_{i+1})$

Les graphes non orientés



G=(S,A)
S = ensemble fini de sommets
A ⊆ $\mathcal{P}_2$(S) = ensemble de paires de sommets, appelées les **arêtes**.

$S=\{q_0,q_1,q_2,q_3,q_4\}$
$A = \{ \{q_0,q_1\}, \{q_2,q_4\}, \{q_4,q_0\}, \{q_3,q_0\}, \{q_3,q_4\}, \{q_3,q_1\} \}$

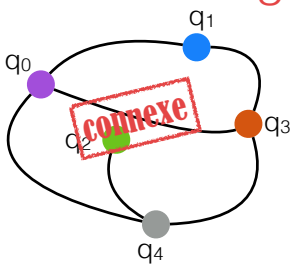
Chemin: …

On note: $q \rightarrow^* q'$ lorsqu'il existe un chemin reliant q à q'.
Et bien sûr $q \rightarrow^* q' \Leftrightarrow q' \rightarrow^* q$, on note souvent: $q \leftrightarrow^* q'$.

Cycle: Un circuit est un chemin $(v_0,v_1,..,v_k)$ avec $v_0=v_k$ et $k\geq 3$ et $v_1,v_2,\ldots,v_k$ sont distincts.

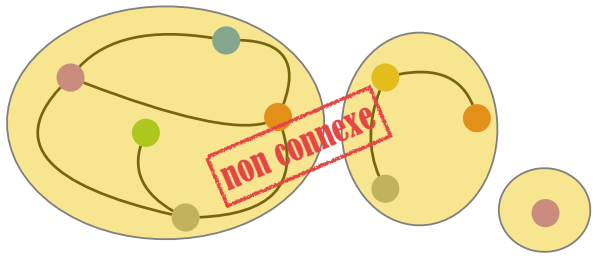
Si G ne contient pas de cycle, on dit qu'il est acyclique.

Les graphes non orientés

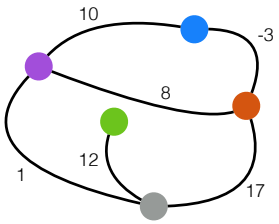


Les composante connexe (CC) sont les classes d'équivalence de la relation $\leftrightarrow^*$.

Un graphe G est **connexe** ssi il n'a qu'une seule CC.



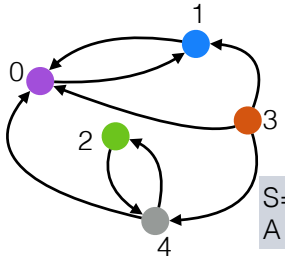
Les graphes non orientés valués



G=(S,A,w) avec
A ⊆ $\mathcal{P}_2$(S) et w : A → **R**
w associe un poids, une distance, un coût… à chaque arête de A.

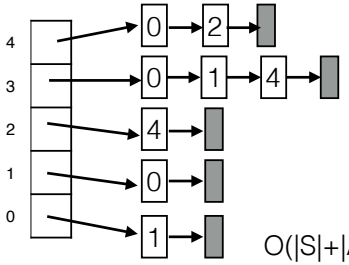
Graphes
représentation

Représentation des graphes orientés



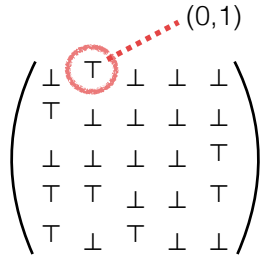
G=(S,A)
S = ensemble fini de sommets
A ⊆ SxS = ensemble de couples (paires ordonnées), appelés les **arcs** (ou transition)

S={0, 1, …, 4}
A = { (0,1), (1,0), (3,1), (3,0), (2,4), (4,2), (3,4), (4,0) }



O(|S|+|A|)

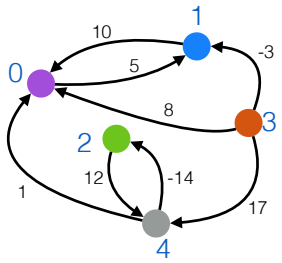
par liste d'adjacence.



O(|S|²)

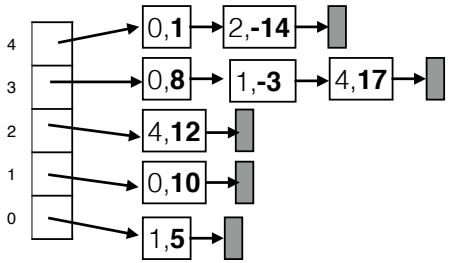
par matrice

Représentation des graphes orientés valués

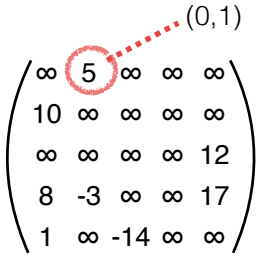


G=(S,A,w)
S = ensemble fini de sommets
A ⊆ SxS
w : A → **R**

S={0, 1, …, 4}
A = { (0,1), (1,0), (3,1), (3,0), (2,4), (4,2), (3,4), (4,0) }
w(0,1) = 5, w(1,0)=10, …

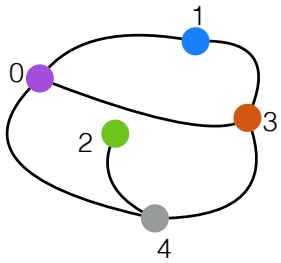


par liste d'adjacence.

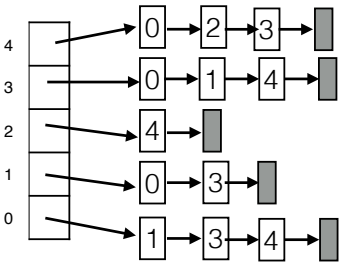


par matrice

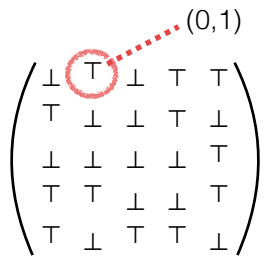
Représentation des graphes non orientés



G=(S,A)
S = ensemble fini de sommets
A ⊆ SxS = ensemble de couples (paires ordonnées), appelés les **arcs** (ou transition)



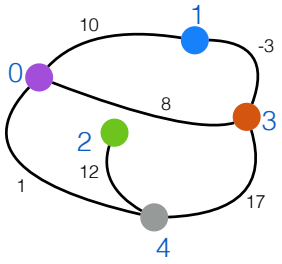
par liste d'adjacence. O(|S|+|A|)



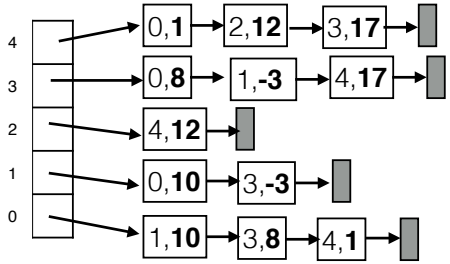
O(|S|²)

par matrice (symétrique)

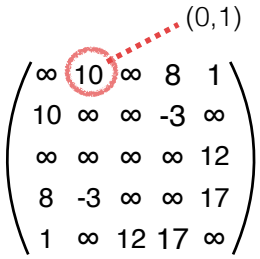
Représentation des graphes non orientés valués



G=(S,A,w)
S = ensemble fini de sommets
A ⊆ SxS
w : A → **R**



par liste d'adjacence.



par matrice

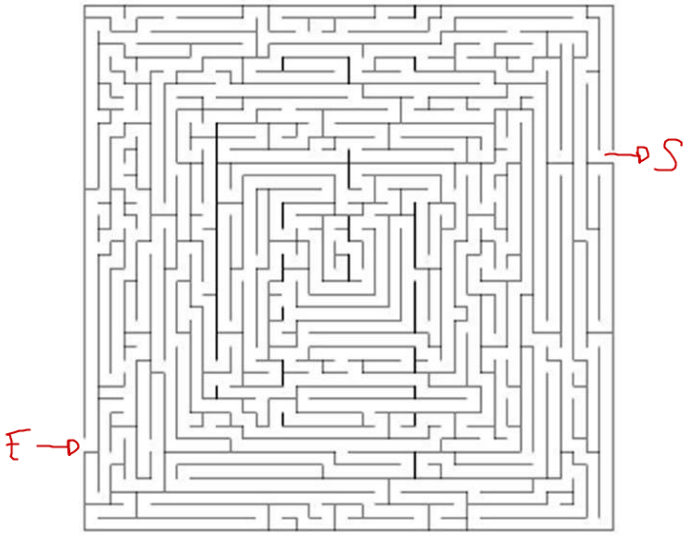
Représentation des graphes et complexité

	Matrice d'adjacence	Liste d'ajacence
Espace mémoire	$O(\|S\|^2)$	$O(\|S\| + \|A\|)$
Tester si $(u, v) \in A$	$O(1)$	$O(\mathsf{deg}^-(u)) = O(\|S\|)$
Enumérer les voisins de $u \in S$	$O(\|S\|)$	$O(\mathsf{deg}^-(u)) = O(\|S\|)$
Enumérer tous les arcs de A	$O(\|S\|^2)$	$O(\|S\| + \|A\|)$
Ajouter un arc (u, v) dans A	$O(1)$	$O(1)$

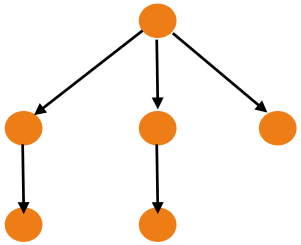
→ le coût d'un algorithme dépend de la représentation choisie !

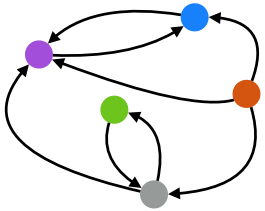
Algorithmes de parcours

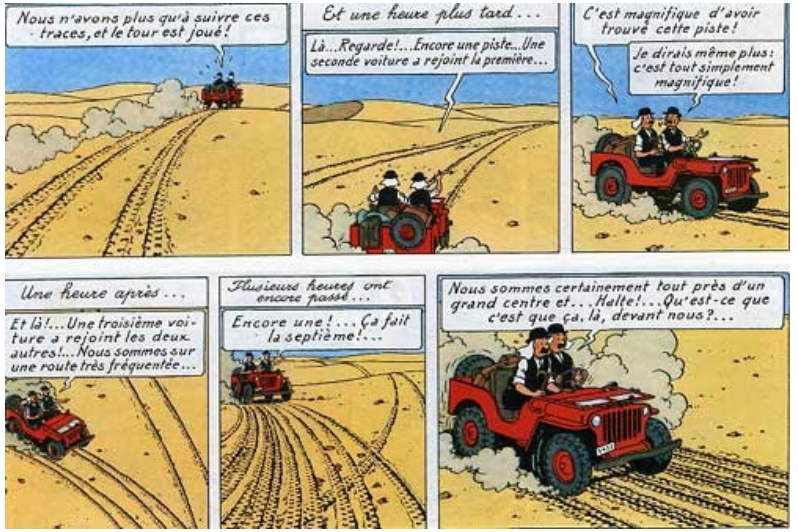
Comment sortir ?



Parcourir un arbre ≠ parcourir un graphe







Il y a longtemps déjà…