

# 1. Analyse des algorithmes

## Temps d'exécution

- ▶ *Opérations élémentaires*: dépend du contexte, souvent comparaisons, affectations, arithmétique, accès, etc.
- ▶ *Pire cas*  $t_{\max}(n)$ : nombre maximum d'opérations élémentaires exécutées parmi les entrées de taille  $n$
- ▶ *Meilleur cas*  $t_{\min}(n)$ : même chose avec « minimum »
- ▶  $t_{\max}(m, n), t_{\min}(m, n)$ : même chose par rapport à  $m$  et  $n$

## Notation asymptotique

- ▶ *Déf.*:  $f \in \mathcal{O}(g)$  si  $n \geq n_0 \rightarrow f(n) \leq cg(n)$  pour certains  $c, n_0$
- ▶ *Signifie*:  $f$  croît moins ou aussi rapid. que  $g$  pour  $n \rightarrow \infty$
- ▶ *Transitivité*:  $f \in \mathcal{O}(g)$  et  $g \in \mathcal{O}(h) \rightarrow f \in \mathcal{O}(h)$
- ▶ *Règle des coeff.*:  $f_1 + \dots + f_k \in \mathcal{O}(c_1 \cdot f_1 + \dots + c_k \cdot f_k)$
- ▶ *Règle du max.*:  $f_1 + \dots + f_k \in \mathcal{O}(\max(f_1, \dots, f_k))$
- ▶ *Déf.*:  $f \in \Omega(g) \leftrightarrow g \in \mathcal{O}(f)$ ;  $f \in \Theta(g) \leftrightarrow f \in \mathcal{O}(g) \cap \Omega(g)$
- ▶ *Règle des poly.*:  $f$  polynôme de degré  $d \rightarrow f \in \Theta(n^d)$

## Notation asymptotique (suite)

- ▶ *Simplification*: lignes élem. comptées comme une seule opér.
- ▶ *Règle de la limite*:
 
$$\lim_{n \rightarrow +\infty} \frac{f(n)}{g(n)} = \begin{cases} 0 & f \in \mathcal{O}(g) \text{ et } g \notin \mathcal{O}(f) \\ +\infty & f \notin \mathcal{O}(g) \text{ et } g \in \mathcal{O}(f) \\ \text{const.} & \Theta(f) = \Theta(g) \end{cases}$$
- ▶ *Multi-params.*:  $\mathcal{O}, \Omega, \Theta$  étendues avec plusieurs seuils

## Correction et terminaison

- ▶ *Correct*: sur toute entrée  $x$  qui satisfait la pré-condition,  $x$  et sa sortie  $y$  satisfont la post-condition
- ▶ *Termine*: atteint instruction **retourner** sur toute entrée
- ▶ *Invariant*: propriété qui demeure vraie à chaque fois qu'une ou certaines lignes de code sont atteintes

## Exemples de complexité

$$\mathcal{O}(1) \subset \mathcal{O}(\log n) \subset \mathcal{O}(n) \subset \mathcal{O}(n \log n) \subset \mathcal{O}(n^2) \subset \mathcal{O}(n^2 \log n) \\ \subset \mathcal{O}(n^3) \subset \mathcal{O}(n^d) \subset \mathcal{O}(2^n) \subset \mathcal{O}(3^n) \subset \mathcal{O}(b^n) \subset \mathcal{O}(n!)$$

# 2. Tri

## Approche générique

- ▶ *Inversion*: indices  $(i, j)$  t.q.  $i < j$  et  $s[i] > s[j]$
- ▶ *Progrès*: corriger une inversion en diminue la quantité
- ▶ *Procédure*: sélectionner et corriger une inversion, jusqu'à ce qu'il n'en reste plus

## Algorithmes (par comparaison)

- ▶ *Insertion*: considérer  $s[1 : i-1]$  triée et insérer  $s[i]$  dans  $s[1 : i]$
- ▶ *Monceau*: transformer  $s$  en monceau et retirer ses éléments
- ▶ *Fusion*: découper  $s$  en deux, trier chaque côté et fusionner
- ▶ *Rapide*: réordonner autour d'un pivot et trier chaque côté

## Propriétés

- ▶ *Sur place*: n'utilise pas de séquence auxiliaire
- ▶ *Stable*: l'ordre relatif des éléments égaux est préservé

## Sommaire

Algorithme	Complexité (par cas)			Sur place	Stable
	meilleur	moyen	pire		
insertion	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	✓	✓
monceau	$\Theta(n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	✓	✗
fusion	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	✗	✓
rapide	$\Theta(n)$	$\Theta(n \log n)$	$\Theta(n^2)$	✓	✗

## Usage

- ▶ *Petite taille*: tri par insertion
- ▶ *Grande taille*: tri par monceau ou tri rapide
- ▶ *Grande taille + stabilité*: tri par fusion

## Tri sans comparaison

- ▶ *Par comparaison*: barrière théorique de  $\Omega(n \log n)$
- ▶ *Sans comparaison*: possible de faire mieux pour certains cas
- ▶ *Représentation binaire*: trier en ordonnant du bit de poids faible vers le bit de poids fort
- ▶ *Complexité*:  $\Theta(mn)$  où  $m$  = nombre de bits et  $n = |s|$

# 3. Graphes

## Graphes

- ▶ *Grphe*:  $\mathcal{G} = (V, E)$  où  $V$  = sommets et  $E$  = arêtes
- ▶ *Dirigé vs. non dirigé*:  $\{u, v\} \in E$  vs.  $(u, v) \in E$
- ▶ *Degré (cas non dirigé)*:  $\deg(u) = \#$  de voisins
- ▶ *Degré (cas dirigé)*:  $\deg^-(u) = \#$  préd.,  $\deg^+(u) = \#$  succ.
- ▶ *Taille*:  $|E| \in \Theta(\text{somme des degrés})$  et  $|E| \in \mathcal{O}(|V|^2)$
- ▶ *Chemin*: séq.  $u_0 \rightarrow \dots \rightarrow u_k$  (taille =  $k$ , simple si sans rép.)
- ▶ *Cycle*: chemin de  $u$  vers  $u$  (simple si sans rép. sauf début/fin)
- ▶ *Sous-graphe*: obtenu en retirant sommets et/ou arêtes
- ▶ *Composante*: sous-graphe max. où sommets access. entre eux

## Parcours

- ▶ *Profondeur*: explorer le plus loin possible, puis retour (pile)
- ▶ *Largeur*: explorer successeurs, puis leurs succ., etc. (file)
- ▶ *Temps d'exécution*:  $\mathcal{O}(|V| + |E|)$

## Représentation

		Mat.	Liste (non dirigé)	Liste (dir.)
	$u \rightarrow v?$	$\Theta(1)$	$\mathcal{O}(\min(\deg(u), \deg(v)))$	$\mathcal{O}(\deg^+(u))$
$a$	$\{v : u \rightarrow v\}$	$\Theta( V )$	$\mathcal{O}(\deg(u))$	$\mathcal{O}(\deg^+(u))$
	$\{u : u \rightarrow v\}$	$\Theta( V )$	$\mathcal{O}(\deg(v))$	$\mathcal{O}( V  +  E )$
$c$	Modif. $u \rightarrow v$	$\Theta(1)$	$\mathcal{O}(\deg(u) + \deg(v))$	$\mathcal{O}(\deg^+(u))$
	Mémoire	$\Theta( V ^2)$	$\Theta( V  +  E )$	

## Propriétés et algorithmes

- ▶ *Plus court chemin*: parcours en largeur + stocker préd.
- ▶ *Ordre topologique*:  $u_1 \preceq \dots \preceq u_n$  où  $i < j \implies (u_j, u_i) \notin E$
- ▶ *Tri topologique*: mettre sommets de degré 0 en file, retirer en mettant les degrés à jour, répéter tant que possible
- ▶ *Détec. de cycle*: tri topo. + vérifier si contient tous sommets
- ▶ *Temps d'exécution*: tous linéaires

## Arbres

- ▶ *Arbre*: graphe connexe et acyclique (ou prop. équivalentes)
- ▶ *Forêt*: graphe constitué de plusieurs arbres
- ▶ *Arbre couv.*: arbre qui contient tous les sommets d'un graphe

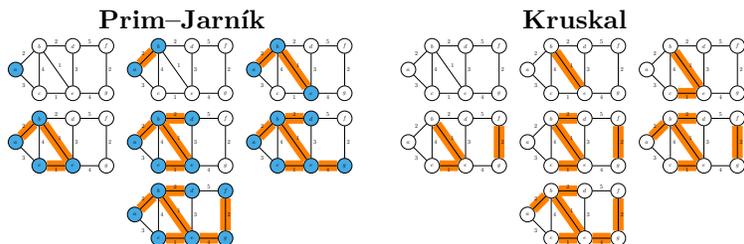
## 4. Algorithmes gloutons

### Arbres couvrants minimaux

- ▶ *Graphe pondéré*:  $\mathcal{G} = (V, E)$  où  $p[e]$  est le poids de l'arête  $e$
- ▶ *Poids d'un graphe*:  $p(\mathcal{G}) = \sum_{e \in E} p[e]$
- ▶ *Arbre couv. min.*: arbre couvrant de  $\mathcal{G}$  de poids minimal

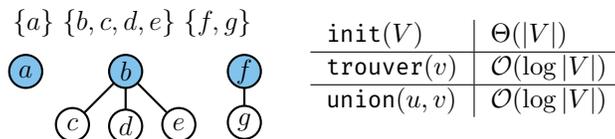
### Algorithmes

- ▶ *Prim-Jarník*: faire grandir un arbre en prenant l'arête min.
- ▶ *Complexité*:  $\mathcal{O}(|E| \log |V|)$  avec monceau
- ▶ *Kruskal*: connecter forêt avec l'arête min. jusqu'à un arbre
- ▶ *Complexité*:  $\mathcal{O}(|E| \log |V|)$  avec ensembles disjoints



### Ensembles disjoints

- ▶ *But*: manipuler une partition d'un ensemble  $V$
- ▶ *Représentation*: chaque ensemble sous une arborescence



### Algorithme glouton

- 1) Choisir un candidat  $c$  itérativement (*sans reconsidérer*)
- 2) Ajouter  $c$  à solution partielle  $S$  si *admissible*
- 3) Retourner  $S$  si *solution (complète)*, « impossible » sinon

### Problème du sac à dos

- ▶ *But*: choisir objets pour maxim. valeur sans excéder capacité
- ▶ *Algo. glouton*: trier en ordre décroissant par  $\text{val}[i]/\text{poids}[i]$
- ▶ *Fonctionne* seulement si on peut découper objets
- ▶ *Approxime* solution discrète à facteur  $1/2$

## 5. Algorithmes récursifs et approche diviser-pour-régner

### Diviser-pour-régner

- ▶ A) découper en sous-problèmes disjoints
- ▶ B) obtenir solutions récursivement
- ▶ C) s'arrêter aux cas de base (souvent triviaux)
- ▶ D) combiner solutions pour obtenir solution globale
- ▶ *Exemple*: tri par fusion  $\mathcal{O}(n \log n)$

### Récurrences linéaires

- ▶ *Cas homogène*:  $\sum_{i=0}^d a_i \cdot t(n-i) = 0$
- ▶ *Polynôme caractéristique*:  $\sum_{i=0}^d a_i \cdot x^{d-i}$
- ▶ *Forme close*:  $t(n) = \sum_{i=1}^d c_i \cdot \lambda_i^n$  où les  $\lambda_i$  sont les racines
- ▶ *Constantes  $c_i$* : obtenues en résolvant un sys. d'éq. lin.
- ▶ *Cas non homo.*: si  $= c \cdot b^n$ , on multiplie poly. par  $(x-b)$
- ▶ *Exemple*:
  - Récurrence:  $t(n) = 3 \cdot t(n-1) + 4 \cdot t(n-2)$
  - Poly. carac.:  $x^2 - 3x - 4 = (x-4)(x+1)$
  - Forme close:  $t(n) = c_1 \cdot 4^n + c_2 \cdot (-1)^n$

### Autres méthodes

- ▶ *Substitution*: remplacer  $t(n), t(n-1), t(n-2), \dots$  par sa déf. jusqu'à deviner la forme close
- ▶ *Arbres*: construire un arbre représentant la récursion et identifier le coût de chaque niveau

### Quelques algorithmes

- ▶ *Hanoi*:  $\text{src}[1:n-1] \rightarrow \text{tmp}, \text{src}[n] \rightarrow \text{dst}, \text{tmp}[1:n-1] \rightarrow \text{dst}$   $\mathcal{O}(2^n)$
- ▶ *Exp. rapide*: exploiter  $b^n = (b^{n \div 2})^2 \cdot b^{n \bmod 2}$   $\mathcal{O}(\log n)$
- ▶ *Mult. rapide*: calculer  $(a+b)(c+d)$  en 3 mult.  $\mathcal{O}(n^{\log 3})$
- ▶ *Horizon*: découper blocs comme tri par fusion  $\mathcal{O}(n \log n)$

### Théorème maître (allégé)

- ▶  $t(n) = c \cdot t(n \div b) + f(n)$  où  $f \in \mathcal{O}(n^d)$ :
  - $\mathcal{O}(n^d)$  si  $c < b^d$
  - $\mathcal{O}(n^d \cdot \log n)$  si  $c = b^d$
  - $\mathcal{O}(n^{\log_b c})$  si  $c > b^d$

## 6. Force brute

### Approche

- ▶ *Exhaustif*: essayer toutes les sol. ou candidats récursivement
- ▶ *Explosion combinatoire*: souvent  $\#$  solutions  $\geq b^n, n!, n^n$
- ▶ *Avantage*: simple, algo. de test, parfois seule option
- ▶ *Désavantage*: généralement très lent et/ou avare en mémoire

### Techniques pour surmonter explosion

- ▶ *Élagage*: ne pas développer branches inutiles
- ▶ *Contraintes*: élaguer si contraintes enfreintes
- ▶ *Bornes*: élaguer si impossible de faire mieux
- ▶ *Approximations*: débiter avec approx. comme meilleure sol.
- ▶ *Si tout échoue*: solveurs SAT ou d'optimisation

### Problème des $n$ dames

- ▶ *But*: placer  $n$  dames sur échiquier sans attaques
- ▶ *Algo.*: placer une dame par ligne en essayant colonnes dispo.

### Sac à dos

- ▶ *But*: maximiser valeur sans excéder capacité
- ▶ *Algo.*: essayer sans et avec chaque objet
- ▶ *Mieux*: élaguer dès qu'il y a excès de capacité
- ▶ *Mieux++*: élaguer si aucune amélioration avec somme valeurs

### Retour de monnaie

- ▶ *But*: rendre montant avec le moins de pièces
- ▶ *Algo.*: pour chaque pièce, essayer d'en prendre 0 à  $\#$  max.

## 7. Programmation dynamique

### Approche

- ▶ *Principe d'optimalité*: solution optimale obtenue en combinant solutions de sous-problèmes qui se chevauchent
- ▶ *Descendante*: algo. récursif + mémoïsation (ex. Fibonacci)
- ▶ *Ascendante*: remplir tableau itér. avec solutions sous-prob.

### Retour de monnaie

- ▶ *Sous-question*: # pièces pour rendre  $j$  avec pièces 1 à  $i$ ?
- ▶ *Identité*:  $T[i, j] = \min(T[i-1, j], T[i, j - s[i]] + 1)$
- ▶ *Exemple*: montant  $m = 10$  et pièces  $s = [1, 5, 7]$

	0	1	2	3	4	5	6	7	8	9	10
0	0	$\infty$									
1	0	1	2	3	4	5	6	7	8	9	10
2	0	1	2	3	4	1	2	3	4	5	2
3	0	1	2	3	4	1	2	1	2	3	2

### Sac à dos

- ▶ *Sous-question*: val. max. avec capacité  $j$  et les objets 1 à  $i$ ?
- ▶ *Identité*:  $T[i, j] = \max(T[i-1, j], T[i-1, j - p[i]] + v[i])$

### Plus courts chemins

- ▶ *Plus court chemin*: chemin simple de poids minimal
- ▶ *Bien défini*: si aucun cycle négatif
- ▶ *Approche générale*: raffiner distances partielles itérativement
- ▶ *Dijkstra*: raffiner en marquant sommet avec dist. min.
- ▶ *Floyd-Warshall*: raffiner via sommet intermédiaire  $v_k$
- ▶ *Bellman-Ford*: raffiner avec  $\geq 1, 2, \dots, |V| - 1$  arêtes
- ▶ *Sommaire*:

	Dijkstra	Bellman-Ford	Floyd-Warshall
Types de chemins	d'un sommet vers les autres	paire de sommets	paire de sommets
Poids négatifs?	✗	✓	✓
Temps d'exécution	$\mathcal{O}( V  \log  V  +  E )$	$\Theta( V  \cdot  E )$	$\Theta( V ^3)$
Temps ( $ E  \in \Theta(1)$ )	$\mathcal{O}( V  \log  V )$	$\Theta( V )$	$\Theta( V ^3)$
Temps ( $ E  \in \Theta( V )$ )	$\mathcal{O}( V  \log  V )$	$\Theta( V ^2)$	$\Theta( V ^3)$
Temps ( $ E  \in \Theta( V ^2)$ )	$\mathcal{O}( V ^2)$	$\Theta( V ^3)$	$\Theta( V ^3)$

## 8. Algorithmes et analyse probabilistes

### Modèle probabiliste

- ▶ *Modèle*: on peut tirer à pile ou face (non déterministe)
- ▶ *Aléa*: on peut obtenir une loi uniforme avec une pièce
- ▶ *Idéalisé*: on suppose avoir accès à une source d'aléa parfaite (en pratique: source plutôt pseudo-aléatoire)

### Algorithmes de Las Vegas

- ▶ *Temps*: varie selon les choix probabilistes
- ▶ *Valeur de retour*: toujours correcte
- ▶ *Exemple*: tri rapide avec pivot aléatoire
- ▶ *Temps espéré*: dépend de  $\mathbb{E}[Y_x]$  où  $Y_x = \#$  opér. sur entrée  $x$

### Algorithmes de Monte Carlo

- ▶ *Temps*: ne varie pas selon les choix probabilistes
- ▶ *Valeur de retour*: pas toujours correcte
- ▶ *Exemple*: algorithme de Karger
- ▶ *Prob. d'erreur*: dépend de  $\Pr(Y_x \neq \text{bonne sortie sur } x)$

### Coupe minimum: algorithme de Karger

- ▶ *Coupe*: partition  $(X, Y)$  des sommets d'un graphe non dirigé
- ▶ *Taille*: # d'arêtes qui traversent  $X$  et  $Y$
- ▶ *Coupe min.*: identifier la taille minimale d'une coupe
- ▶ *Algorithme*: contracter itérativement une arête aléatoire en gardant les multi-arêtes, mais pas les boucles



- ▶ *Prob. d'erreur*:  $\leq 1 - 1/|V|^2$  (Monte Carlo)
- ▶ *Amplification*: on peut réduire (augmenter) la prob. d'erreur (de succès) arbitrairement (en général: avec min., maj., etc.)

### Temps moyen

- ▶ *Temps moyen*:  $\sum$  temps instances de taille  $n$  / # instances
- ▶ *Attention*: pas la même chose que le temps espéré
- ▶ *Hypothèse*: entrées distribuées uniformément ( $\pm$  réaliste)
- ▶ *Exemple*:  $\Theta(n^2)$  pour le tri par insertion