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 $n \geq n_0 \to f(n) \leq c \cdot g(n)$ pour certains c, n_0
- *Signifie*: f croît moins ou aussi rapid. que g pour $n \to \infty$
- *Transitivité*: $f \in \mathcal{O}(g)$ et $g \in \mathcal{O}(h) \to f \in \mathcal{O}(h)$
- *Règle des coeff.*: $f_1 + \ldots + f_k \in \mathcal{O}(c_1 \cdot f_1 + \ldots + c_k \cdot f_k)$
- *Règle du max.*: $f_1 + \ldots + f_k \in \mathcal{O}(\max(f_1, \ldots, f_k))$
- *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 $d \to f \in \Theta(n^d)$

Notation asymptotique (suite)

- *Simplification*: lignes élem. comptées comme une seule opér.
- *Règle de la limite*:

$$\lim_{n \to +\infty} \frac{f(n)}{g(n)} = \begin{cases} 0 & f \in \mathcal{O}(g) \text{ et } g \notin \mathcal{O}(f) \\ +\infty & f \notin \mathcal{O}(g) \text{ et } g \in \mathcal{O}(f) \\ \text{const.} & \Theta(f) = \Theta(g) \end{cases}$$

- *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) meilleur	moyen	pire	Sur place	Stable
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 (de façon stable) 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

- *Graphe*: $\mathcal{G} = (V, E)$ 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. $u_0 \to \cdots \to u_k$ (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

$$\begin{array}{c} \\ a \\ b \\ c \end{array} \begin{array}{c} \begin{array}{ccc} a & b & c \end{array} \\ \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \end{array} \quad \begin{array}{l} [a \mapsto [b,c], \\ b \mapsto [a], \\ c \mapsto [b]] \end{array}$$

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

Propriétés et algorithmes

- *Plus court chemin*: parcours en largeur + stocker préd.
- *Ordre topologique*: $u_1 \preceq \cdots \preceq u_n$ 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 avec tous sommets d'un graphe $\mathcal{G}$ non dirigé; possible ssi $\mathcal{G}$ connexe; se trouve avec parcours en prof.

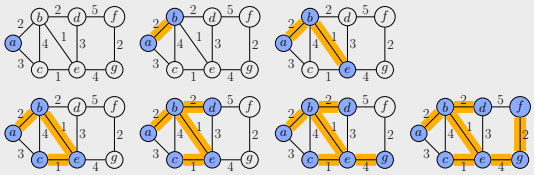
4. Algorithmes gloutons

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

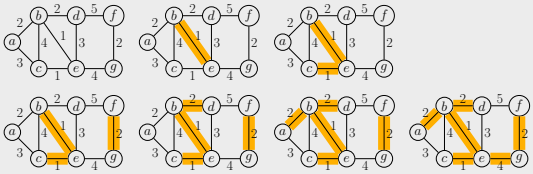
Algorithme de Prim–Jarník

- *Approche*: faire grandir un arbre en prenant l'arête min.
- *Complexité*: $\mathcal{O}(|E| \log |V|)$ avec monceau



Algorithme de Kruskal

- *Approche*: 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 $val[i]/poids[i]$
- *Fonctionne* si on peut découper objets (sinon approx. à ½)