

- Notes de cours -

Présentation du module

Objectifs du cours. Le but du cours est d'étudier des outils et des méthodes pour concevoir et analyser des algorithmes. Les problèmes algorithmiques que nous étudierons viendront de différents domaines : algèbre, combinatoire, recherche opérationnelle, logique...

Intervenants. Cours/td/tp : Stéphane Bessy (stephane.bessy@umontpellier.fr)
Tp : Bruno Grenet (bruno.grenet@umontpellier.fr)

Planning. Les cours (10 séances) ont lieu le mercredi matin de 8h00 à 9h30, les tds (12 séances) à la suite de 9h45 à 11h15 et les tps (9 séances) le mercredi après-midi de 15h00 à 16h30. Le planning suivant peut-être amené à changer...

N.	Date	Contenu
01	Me 12/09	<i>Cours 1</i> : Introduction, rappels
02	Me 19/09	<i>Cours 2</i> : Nouvelle structure de données : le tas <i>Td 1</i> : Complexité, premiers exemples
03	Me 26/09	<i>Cours 3</i> : Algorithmes gloutons <i>Td 2</i> : Tas
04	Me 3/10	<i>Cours 4</i> : Diviser pour régner <i>Td 3</i> : Tas et algos gloutons <i>Tp 1</i> : Tas
05	Me 10/10	<i>Cours 5</i> : Diviser pour régner (suite) <i>Td 4</i> : Algo gloutons <i>Tp 2</i> : Glouton
06	Me 17/10	<i>Cours 6</i> : Nouvelle structure de données : arbre binaire de recherche <i>Td 5</i> : Diviser pour régner <i>Tp 3</i> : Glouton et D&C
07	Me 24/10	<i>Cours 7</i> : Programmation dynamique <i>Td 6</i> : Diviser pour régner <i>Tp 4</i> : D&C
xx	Me 31/10	VACANCES TOUSSAINT
08	Me 7/11	<i>Cours 8</i> : Programmation dynamique (suite) <i>Td 7</i> : ABR <i>Tp 5</i> : ABR
09	Me 14/11	<i>Sur le créneau de cours</i> : Contrôle continu-type exam <i>Td 8</i> : ABR et Prog dyn <i>Tp 6</i> : ABR
10	Me 21/11	<i>Cours 9 (dernier cours !)</i> : Analyse amortie <i>Td 9</i> : Prog dyn <i>Tp 7</i> : Prog dyn
11	Me 28/11	<i>Td 10</i> : Prog dyn <i>Tp 8</i> : Prog dyn
12	Me 5/12	<i>Td 11</i> : Prog dyn et analyse amortie <i>Tp 9 (denier tp !)</i> : Contrôle continu de tp
12	Me 12/12	<i>Td 12 (dernier td !)</i> : Analyse amortie

Modalité de contrôle de connaissance. L'évaluation comportera un contrôle continu et un examen. Le contrôle continu est composé d'une partie type exam (sur 15 pts, vraisemblablement le 14 Novembre) et d'une partie tp (sur 5 pts, vraisemblablement le 5 Décembre). Le contrôle continu compte pour 30% dans la note finale avec la 'règle du max', c'est-à-dire que la note finale sera $\max\{exam; 0.7 \times exam + 0.3 \times cc\}$. Une deuxième session d'examen est prévue, pas de contrôle continu. **Les évaluations 'type exam' sont sans document.**

Prérequis. D'un point de vue algorithmique, sont attendues des connaissances sur les instructions classiques en pseudo-code, la récursivité, un algorithme de tri au moins, la capacité à déterminer la complexité en temps d'algorithmes simples et la connaissance des structures de données de tableaux, piles, files, liste chaînées.

En programmation, les tp se feront en C(++) . Il faut connaître pour cela les instructions classiques, l'usage des fonctions et les pointeurs.

Ressource. Les ressources pédagogiques seront disponibles sur le moodle de l'Université. Les ouvrages suivant contiennent l'essentiel du cours (et même plus...) :

- T.H. Cormen, C.E. Leiserson, R. Rivest and C. Stein. **Introduction to Algorithms**, 3rd Edition, *MIT Press*, 2009 (une version française existe).
- S. Dasgupta, C. Papadimitriou and U. Vazirani. **Algorithms**, *McGraw-Hill Higher Education*, 2006.

1 Introduction, rappels

1.1 Exemple introductif

1.2 Modèle

- Le **pseudo-code** d'un algorithme comprend des **opérations élémentaires** : déclaration de variable, affectation, lecture, écriture de variables, opération arithmétique : $+$, $-$, $\times$, $\div$, test élémentaire et appel de fonction, ainsi que des **boucles** : *pour* et *tant que*.
- Dans le modèle étudié (WORD-RAM), on considère que chaque **opération élémentaire** prend un **temps constant** et que chaque **déclaration de variable (simple) et appel de fonction** consomme un **espace machine constant**.
- Dans ce modèle-là, on va compter (ou majorer) le **nombre d'opérations élémentaires** (pour établir la **complexité en temps** de l'algo), compter (ou majorer) le **nombre de déclarations de variables et d'appels de fonctions** (pour établir la **complexité en espace** de l'algo), exprimer ces valeurs **en fonction des paramètres d'entrée** de l'algorithme, de manière asymptotique et dans le **pire des cas**.

1.3 Conception et analyse d'un algorithme

- Pour concevoir et analyser un algorithme, on doit : **écrire le pseudo-code** de l'algorithme, choisir les **structures de données** à utiliser pour les variables puis **analyser l'algorithme**.
- Pour analyser un algorithme, on étudie sa **terminaison**, on établit sa **complexité en temps et en espace** et enfin sa **validité**. Pour ce dernier point, on cherche souvent un **invariant de l'algorithme** que l'on prouve généralement par **récurrence**.

1.4 Outils mathématiques, notations de Landau

- **Une notation de Landau** : Soient $f, g : \mathbb{N} \rightarrow \mathbb{R}^+$. On dit que $f = O(g)$ si il existe une constante $c > 0$ et un rang $n_0 \in \mathbb{N}$ tels que : $\forall n \geq n_0$ on ait $f(n) \leq c.g(n)$

Lemme 1 (Calcul des O) On a les propriétés suivantes : $O(f) + O(g) = O(f + g)$, si $h = O(f)$ alors $f + h = O(f)$, $O(f) \times O(g) = O(f \times g)$ et pour $\lambda \in \mathbb{R}^+$ on a $O(\lambda f) = O(f)$.

Lemme 2 (O et limites) Pour $f : \mathbb{N} \rightarrow \mathbb{R}$ et $g : \mathbb{N} \rightarrow \mathbb{R}^{+*}$, si il existe une constante $c > 0$ telle que $\frac{f(n)}{g(n)} \xrightarrow{n \rightarrow +\infty} c$ alors on a $f = O(g)$. Et si $\frac{f(n)}{g(n)} \xrightarrow{n \rightarrow +\infty} +\infty$ alors on a $f \neq O(g)$.

- **Autre notation** : Soient $f, g : \mathbb{N} \rightarrow \mathbb{R}^+$. On dit que $f = \Omega(g)$ si il existe une constante $c > 0$ et un

rang $n_0 \in \mathbb{N}$ tels que : $\forall n \geq n_0$ on ait $f(n) \geq c.g(n)$ (c'est-à-dire si $g = O(f)$).

Lemme 3 (Limites comparées) - Pour $\alpha, \beta > 0$ on a :

$\frac{(\log n)^\alpha}{n^\beta} \xrightarrow{n \rightarrow +\infty} 0$ $\frac{n^\alpha}{(2^n)^\beta} \xrightarrow{n \rightarrow +\infty} 0$ $\frac{(2^n)^\alpha}{n!} \xrightarrow{n \rightarrow +\infty} 0$
- Soit $u(n)$ est une fonction telle que $u(n) \xrightarrow{n \rightarrow +\infty} \infty$ (typiquement, $u(n) = n$ ou $u(n) = \log n$ ou $u(n) = 2^n$).
Si $a.X^p$ et $b.X^q$ avec $a > 0$ et $b > 0$ sont respectivement les termes de plus haut degré de deux polynômes P et Q alors $\lim_{n \rightarrow \infty} P(u(n))/Q(u(n))$ vaut : 0 si $q > p$, a/b si $q = p$ et $+\infty$ si $p > q$.

Lemme 4 (Règles de calcul pour le log) Pour $a, b \in \mathbb{R}^{+*}$ et $c \in \mathbb{R}$, on a :

$\log 0$ est non défini $\log 1 = 0$ $\log 2 = 1$
 $\log(a \times b) = \log a + \log b$ $\log(\frac{a}{b}) = \log a - \log b$ $\log(a^c) = c \times \log a$

Lemme 5 (Règles de calcul pour l'exp) Pour $a, b \in \mathbb{R}^{+*}$, on a :

$2^0 = 1$ $2^{a+b} = 2^a \times 2^b$ $2^{a \times b} = (2^a)^b$
 $2^{\log a} = a$ $\log 2^a = a$

Théorème 1 (Formule de Stirling) Pour $n \in \mathbb{N}^*$, $n!$ vaut $n \times (n-1) \times \dots \times 2 \times 1$ et vérifie

$$n! \underset{n \rightarrow +\infty}{\sim} \sqrt{2\pi n} \left(\frac{n}{e}\right)^n.$$

- Pour $x \in \mathbb{R}$, on définit $\lfloor x \rfloor$ comme le plus grand entier k vérifiant $k \leq x$. De même, $\lceil x \rceil$ est le plus petit entier k vérifiant $x \leq k$. On a $x-1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x+1$.
Pour n entier, si n est pair on a $\lfloor n/2 \rfloor = \lceil n/2 \rceil = n/2$ et si n est impair, on a $\lfloor n/2 \rfloor = (n-1)/2$ et $\lceil n/2 \rceil = (n+1)/2$. Toujours pour $n \in \mathbb{N}$, on a toujours $\lfloor n/2 \rfloor + \lceil n/2 \rceil = n$.

Lemme 6 (Sommes arithmétique et géométrique) Pour $a, b \in \mathbb{N}$ avec $a \leq b$ et $x \in \mathbb{R} \setminus \{1\}$, on a :

$$\sum_{i=a}^b i = (b-a+1) \cdot \frac{b+a}{2} \quad \text{et} \quad \sum_{i=a}^b x^i = \frac{x^{b+1} - x^a}{x-1}$$

2 Nouvelle structure de données : le tas

2.1 Arbres binaires

- Un **nœud** non vide, x , possède une valeur, $\text{val}(x) \in \mathbb{R}$ et un lien vers 3 autres nœuds : **pere**(x), **filsg**(x) et **filsd**(x).
- Un arbre binaire A est une structure de données contenant un ensemble de nœuds, dont un est spécial : la **racine** de A , notée **rac**(A), et est défini récursivement par : un arbre réduit à sa racine est un arbre binaire et si A est un arbre binaire dont un nœud x a un fils (G ou D) vide et si y est un nœud n'appartenant pas à A alors la structure obtenu en faisant **pere**(y) $\leftarrow x$ et **filsg**(ou **filsd**)(x) $\leftarrow y$ est un arbre binaire.
- La **hauteur** d'un nœud x , notée **h**(x) est donnée récursivement par : $h(\text{rac}(A)) = 0$ et si $x \neq \text{rac}(A)$ alors $h(x) = h(\text{pere}(x)) + 1$.
La **hauteur de** A est **h**(A) = $\max\{h(x) : x \in A\}$. Pour $0 \leq k \leq h(A)$, le **kième niveau de** A est l'ensemble $\{x \in A : h(x) = k\}$, on le note N_k .

Lemme 7 (Nombre de nœuds) Pour $0 \leq k \leq h(A)$, on a $|N_k| \leq 2^k$.

Pour tout arbre binaire A , on a $h(A) \leq n(A) \leq 2^{h(A)-1} + 1$.

- On numérote les sommets de A par une fonction **num** : $A \rightarrow \{0, \dots, n(A)-1\}$, en commençant par le niveau N_0 , puis N_1 , etc. Dans chaque niveau les sommets sont numérotés de 'gauche à droite'.

2.2 Arbres binaires quasi-complet

- un **arbre binaire quasi-complet** est un arbre binaire dont tous les niveaux sont complets sauf potentiellement le dernier dont tous les nœuds sont 'rangés le plus à gauche possible'.

Lemme 8 (Hauteur d'un quasi-complet) Si A est quasi-complet alors on a $2^{h(A)} \leq n(A) \leq 2^{h(A)-1} + 1$, c'est-à-dire $h(A) = \lfloor \log n(A) \rfloor$

Lemme 9 (Num des liens) Dans A quasi-complet, si $\text{num}(x) = i$ pour un nœud x alors si $\text{pere}(x) \neq \emptyset$ on a $\text{num}(\text{pere}(x)) = \lfloor (i-1)/2 \rfloor$, si $\text{filsG}(x) \neq \emptyset$ on a $\text{num}(\text{filsG}(x)) = 2i+1$ et si $\text{filsD}(x) \neq \emptyset$ on a $\text{num}(\text{filsD}(x)) = 2i+2$.

- Du coup, on encode les nœuds un arbre binaire quasi-complet directement par leur numéros.

2.3 Tas

- Un **tas** est un arbre binaire quasi-complet A vérifiant pour tout $i = 1, \dots, n(A) - 1$ (c-à-d pour tout nœud sauf la racine) $A[\text{pere}(i)] \geq A[i]$ (la '**propriété de tas**').
- Pour un nœud i d'un arbre quasi-complet A , si les sous-arbres enracinés sur les fils droit et gauche de i sont des tas alors après appel de l'algorithme $\text{ENTASSER}(A, i)$, le sous-arbre enraciné sur i est un tas.

Lemme 10 (Complexité de ENTASSER) L'appel $\text{ENTASSER}(A, i)$ fait au plus $\log n(A)$ appels récursifs et s'exécute donc en $O(\log n(A))$.

2.4 Applications

- l'algo TRI-PAR-TAS permet de trier un tableau de taille n en temps $O(n \log n)$.

Lemme 11 (Borne inf pour le tri) Tout algo de tri ne faisant que des comparaisons effectue $\Omega(n \log n)$ comparaisons dans le pire des cas pour trier n nombres quelconques.

- Une **file de priorité** est une structure de donnée contenant des éléments ayant des priorités et possédant des fonctions permettant : d'ajouter un élément, d'extraire un élément de priorité maximale et d'augmenter ou de diminuer la priorité d'un élément. Toutes ses opérations s'effectuant en temps $O(\log n)$ pour une file de priorité à n éléments.

3 Algorithmes gloutons

3.1 Heuristiques gloutonnes

- Une **stratégie gloutonne** consiste, à chaque étape, à faire un choix qui semble optimal à ce moment-là. Les choix sont localement optimaux dans l'espoir d'obtenir une solution globalement optimale.

3.2 Exemple 1 : choix d'activités

- On considère n activités. Pour tout $i = 1, \dots, n$, l'activité i a une date de début d_i et une date de fin f_i . L'activité j est compatible avec l'activité i si $d_j > f_i$. Le but du problème **choix d'activité** est de choisir un plus grand nombre d'activités deux-à-deux compatibles.

Lemme 12 (Sous-solution optimale) Il existe une solution optimale au problème de choix d'activité qui est formée de l'activité s_1 se terminant le plus tôt et d'une solution optimale du problème restreint aux activités i avec $d_i > f_{s_1}$.

Théorème 2 (Choix d'activités) *L'algorithme CHOIX-GLOUTON-D-ACTIVITES résout le problème du choix d'activités de manière optimale, et en temps $O(n \log n)$ pour n activités.*

3.3 Exemple 2 : problème du sac à dos fractionnaire

- Dans le **problème du sac à dos** $(W, (v_1, p_1), \dots, (v_n, p_n))$, on dispose d'un sac à dos ayant un poids maximum de charge W et d'un ensemble d'objets $O_1, \dots, O_n$, chaque objet O_i ayant une valeur v_i et un poids p_i . Le but est de charger le sac à dos en maximisant la valeur emportée. Plus précisément, pour chaque objet O_i on pose $x_i = 0$ si on ne prend pas O_i et $x_i = 1$ si on le prend. Ainsi, on veut trouver $(x_1, \dots, x_n)$ tel que $\sum_{i=1}^n x_i p_i \leq W$ et que $\sum_{i=1}^n x_i v_i$ soit maximale.
Dans la version fractionnaire du sac à dos, on peut prendre la portion que l'on souhaite de chaque objet, c'est-à-dire qu'on autorise x_i à être un réel de $[0, 1]$.

Lemme 13 (Sous-solution optimale) *Il existe une solution optimale au problème de sac à dos fractionnaire $(W, (v_1, p_1), \dots, (v_n, p_n))$ avec $v_1/p_1 \geq \dots \geq v_n/p_n$ qui est donnée par :*
 - $x_1 = 1$ si $p_1 \leq W$, et $x_1 = W/p_1$ sinon
 - une solution optimale $(x_2, \dots, x_n)$ du problème $(W - p_1, (v_2, p_2), \dots, (v_n, p_n))$

Théorème 3 (Sac à dos fractionnaire) *L'algorithme CHOIX-GLOUTON-SAD-FRACTIONNAIRE résout le problème du sac à dos fractionnaire de manière optimale, et en temps $O(n \log n)$ pour n objets.*

3.4 Idées de stratégies gloutonnes

- Un algorithme glouton marche bien quand il existe une solution optimale du problème formée du choix glouton d'un élément et d'une solution optimale du sous-problème restant. C'est le cas des problèmes dont la structure combinatoire sous-jacente est celle d'un *matroïde*.

3.5 Exemple 3 : approximation de SET-COVER dans le plan

- Une variante du problème de SET-COVER dans le plan correspond à la question suivante. On se donne un ensemble de n maison dans le plan. Si on place un émetteur sur une maison, alors toutes les maisons situées à moins de 500m sont couvertes par cet émetteur. Le but est de placer le moins d'émetteurs possibles pour couvrir toutes les maisons.
- Le choix glouton proposé est de placer à chaque étape l'émetteur qui couvre le plus de maisons non déjà couvertes.

Lemme 14 (Approximation de SET-COVER) *Si k_{opt} désigne le nombre d'émetteurs à placer dans une solution optimale, alors le choix glouton proposé place au plus $k_{\text{opt}} \cdot \log n$ émetteurs.*

4 Paradigme 'diviser pour régner'

5 Programmation dynamique

6 Structure de données pour dictionnaire : arbre binaire de recherche et table de hachage