

Résumé d'algorithmique

Auteurs : Samuel Robyr, Daniel Peppicelli, 3 juillet 2005

Merci à tous pour l'aide donnée, en particulier à Antoine Jimod !

Dernière version disponible à l'url <http://epfl1.neoch.net/>

Rappels

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$
$$\sum_{i=1}^n i = \frac{n(n+1)}{2} = \frac{n^2 + n}{2}$$
$$\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6} = \frac{2n^3 + 3n^2 + n}{6}$$
$$\sum_{i=1}^n i^3 = \frac{n^4 + 2n^3 + n^2}{4}$$

Théorie des ensembles et définitions :

- $A \setminus B = \{x \in A \mid x \notin B\}$: la différence de A et B
- $A \times B$ (produit cartésien de A et B) :
- tous les couples (a, b) où $a \in A$ et $b \in B$
- $A \triangle B = (A \setminus B) \cup (B \setminus A)$
- $A^n = A \times A \times \dots \times A$
- $\text{Pot}\{1, 2\} = \{\emptyset, \{1\}, \{2\}, \{1, 2\}\}$ (ensemble des parties)
- $\text{Pot}^*(X)$ (ensembles des parties finies de X) :
- ensemble de tous les sous ensembles finis de X .
- Si X est fini $\text{Pot}(X) = \text{Pot}^*(X)$
- $R \subseteq A \times A$ une relation sur A
- R symétrique $\Rightarrow R^{-1} = R$
- $F : A \rightarrow B$ est surjective si $\forall b \in B \exists a \in A \mid b = F(a)$
- $\text{Pot}^*(X)$ (ensembles des parties finies de X) :
- ensemble de tous les sous ensembles finis de X .
- Si X est fini $\text{Pot}(X) = \text{Pot}^*(X)$
- $R \subseteq A \times A$ une relation sur A
- R symétrique $\Rightarrow R^{-1} = R$
- $F : A \rightarrow B$ est surjective si $\forall b \in B \exists a \in A \mid b = F(a)$
- $F : A \rightarrow B$ est injective si $F(a) = F(a') \Rightarrow a = a'$
- Un problème computationnel est spécifié de façon abstraite par un triplet d'ensemble $P = (I, O, R)$ où I (inputs) et O (outputs) sont des ensembles non vides et $R \subseteq I \times O$ est une relation.

1 Induction

Induction simple. Une affirmation $A(n)$ est vraie ssi les deux conditions suivantes sont vraies :

1. $A(k)$ (normalement on pose $k = 1$) ;
2. $\forall n \geq k : A(n) \Rightarrow A(n+1)$.

Induction descendente. Une affirmation $A(n)$ est vraie $\forall n$ ssi les deux conditions suivantes sont vraies :

1. $A(k)$ est vraie pour une infinité de n ;
2. $\forall n \geq 2 : A(n) \Rightarrow A(n-1)$.

Induction forte. L'affirmation $A(n)$ est vraie ssi les deux conditions suivantes sont vraies :

1. $A(k)$ (normalement on pose $k = 1$) ;
2. $\forall n \in \mathbb{N} : A(k) \wedge \dots \wedge A(n) \Rightarrow A(n+1)$.

2 Analyse d'algorithmes

O. Pour des fonctions f et g nous dirons que $f = O(g)$ si

$$\exists c > 0, \exists n_0 > 0, \forall n \geq n_0 : f(n) \leq cg(n)$$

Alors f croît au plus aussi vite que g . Les constantes multiplicatives ne jouent pas de rôle : $a \cdot f(n) = O(b \cdot f(n))$.

o. Pour des fonctions f et g nous écrivons $f = o(g)$ si

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$$

Alors f croît plus lentement que g . Si $f = o(g)$ alors on a aussi $f = O(g)$.

Ω et Θ . Pour des fonctions f et g nous disons que

- $f = \Omega(g)$ si $g = O(f)$
- $f = \Theta(g)$ si $f = O(g)$ et $f = \Omega(g)$

Si $f = \Theta(g)$ alors f et g ont le même ordre de magnitude¹.

Exemples

$$\begin{cases} - \log(n) + n^2 = O(n^2) \\ - \log n = O(\log n) \\ - n^2 = O(n^3) \\ - an = O(bn), \forall a, b \in \mathbb{R}^+ \\ - 1000 \cdot n^{1000} = O(2^n) \\ - 100 \log n = O(n) \\ - O(m+n) = O(\max\{m, n\}) \\ - p(n) (\text{degré} < d) : p(n) = o(n^d) \\ - p(n) (\text{degré} d) : p(n) \neq o(n^d) \\ - p(n) (\text{degré} d) : p(n) = \Theta(n^d) \end{cases}$$
$$\begin{cases} - \log(n) + n^2 = O(n^2) \\ - n^2 = \Omega(n^3) \\ - 100n^2 = \Theta(10n^3) \\ - n + n^2 = O(n^3) \neq o(n^2) \\ - n + n^2 = O(n^2) = \Theta(n^2) \\ - n^{1001} = o(1.001^n) \end{cases}$$

Fonctions classées dans leur ordre selon O : 144, $\log_2(1000n)$, $1001n^2(n)$, $99n + 101$, $\frac{1}{1000} \log_{10}(n)$, n^6 , $\frac{1}{3} 3^n$, $n!$

L'algorithme de Karatsuba. Soit $f(x) = \sum_{i=0}^{2n-1} a_i x^i$ et $g(x) = \sum_{i=0}^{2n-1} b_i x^i$ écrits comme suit :

$$f(x) = \underbrace{(a_0 + \dots + a_{n-1} x^{n-1})}_{=f_0(x)} + x^n \underbrace{(a_n + \dots + a_{2n-1} x^{n-1})}_{=f_1(x)}$$
$$g(x) = \underbrace{(b_0 + \dots + b_{n-1} x^{n-1})}_{=g_0(x)} + x^n \underbrace{(b_n + \dots + b_{2n-1} x^{n-1})}_{=g_1(x)}$$

Algorithme 1 KARATSUBA($f(x), g(x)$) rec : $O(n^{\log_2(3)})$

```
1:  $h_0(x) \leftarrow f_0(x) \cdot g_0(x)$ 
2:  $h_2(x) \leftarrow f_1(x) \cdot g_1(x)$ 
3:  $u(x) \leftarrow f_0(x) + f_1(x)$ 
4:  $v(x) \leftarrow g_0(x) + g_1(x)$ 
5:  $A(x) \leftarrow u(x) \cdot v(x)$ 
6:  $h_1(x) \leftarrow A(x) - h_0(x) - h_2(x)$ 
7:  $h(x) \leftarrow h_0(x) + x^n h_1(x) + x^{2n} h_2(x)$ 
8: return  $h(x)$ 
```

L'algorithme a besoin de $6n^2 + 1$ opérations (l'algorithme naïf en utilise $8n^2 - 4n + 1$). Pour les puissances de 2 on a les récurrences suivantes :

- $M(2n) = 3M(n)$, $M(1) = 1$
- $A(2n) = 3A(n) + 8n - 4$, $A(1) = 0$
- $T(2n) = M(2n) + A(2n) = 3T(n) + 8n - 4$, $T(1) = 1$

Relations de récurrence. Soit $T : \mathbb{N} \rightarrow \mathbb{R}$ une fonction telle qu'il existe $b, c, d \in \mathbb{R}_{>0}$ et $a \in \mathbb{N}$ avec

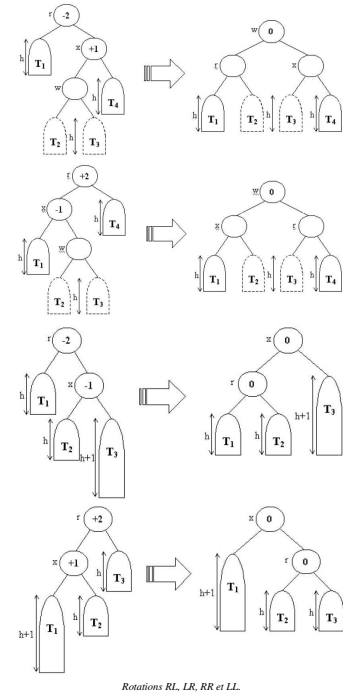
1. $T(n) \leq T(n+1) \forall n \geq 1$
2. $T(an) \leq cT(n) + dn^b$

On a alors

1. Si $a^b < c$, alors $T(n) = O(n^{\log_a(c)})$
2. Si $a^b = c$, alors $T(n) = O(n^{\log_a(c)} \cdot \log_a(n))$
3. Si $a^b > c$, alors $T(n) = O(n^b)$

¹ f et g croissent à la même vitesse.

Page 1



Rotations RL, LR, RR et LL.

3.3 Hashing

Définition La fonction de hachage est une application $f : \mathbb{K} \rightarrow \{0, 1, \dots, m-1\}$ qui fait correspondre à chaque clé k un indice $h(k)$, appelé l'adresse de hachage. Si, pour $k, k' \in \mathbb{K}$, $k \neq k'$ et $h(k) = h(k')$ on a une collision d'adresse. k et k' sont synonymes.

La méthode de division avec reste Si m est la taille de la table de hachage alors la fonction de hachage sera :

$$h(k) := k \bmod m \quad h(k) \in \{0, 1, \dots, m-1\}$$

La méthode multiplicative La clé est multipliée par un nombre irrationnel ϕ , puis on regarde la partie fractionnaire (après la virgule) du résultat. Le nombre d'or $\frac{1+\sqrt{5}}{2}$ est un bon facteur de multiplication. $h(k) := \lfloor m(k\phi^{-1} - \lfloor k\phi^{-1} \rfloor) \rfloor$

Birthday Lemma Si $q \gtrsim 1.78\sqrt{|M|}$, alors la probabilité qu'une fonction aléatoire uniforme $f : \{1, 2, \dots, q\} \rightarrow M$ soit injective est inférieur à $1/2$.

Page 3

3 Graphes et Arbres

Définitions Un graphe (V, E) est un ensemble fini de sommets V et une relation $E \subseteq V \times V$ (appelée ensemble d'arcs). Il est connexe si tous les sommets sont atteignables entre eux. Il est non-orienté si E est symétrique.

Quelques définitions liées aux arbres et aux graphes

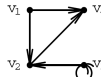
- le **indégré** d'un sommet a est le nombre d'arêtes qui vont vers a . le **outdégré** le nombre d'arêtes qui en sortent.
- la longueur du chemin d'un sommet v à la racine est la **profondeur** de v .
- le **degré** d'un sommet est son nombre de fils.
- la **hauteur** d'un arbre est sa plus grande profondeur.
- le maximum $\max_{v \in V} \text{deg } v$ s'appelle le **degré** de l'arbre.
- **bal**(v_k) = hauteur de l'arbre de gauche - hauteur de l'arbre de droite.
- un graphe est dit **connexe** si tous les sommets sont reliés.
- un arbre est un graphe acyclique et connexe.

Parcourir des arbres Il y a trois méthodes de parcourir des arbres binaires :

- Parcours *preorder* : racine, arbre gauche, arbre droite.
- Parcours *inorder* : arbre gauche, racine, arbre droite ;
- Parcours *postorder* : arbre gauche, arbre droite, racine.

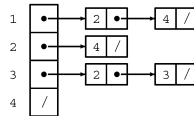
Représentation des graphes Pour représenter un graphe $G = (V, E)$ où $V = \{v_1, \dots, v_n\}$ on peut utiliser une matrice d'adjacence $A = (a_{ij})_{1 \leq i, j \leq n}$ telle que pour tout i et j nous avons $a_{ij} \in \{0, 1\}$, définis comme suit :

$$a_{ij} := \begin{cases} 1 & \text{si } (v_i, v_j) \in E \\ 0 & \text{sinon} \end{cases}$$



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

On peut aussi utiliser des listes d'adjacences :



Représentation d'arbres Il y a plusieurs manières de représenter les arbres :

- **par Structure de pointeurs** n : chaque n contient la valeur du noeud, un pointeur sur chaque fils et un sur son père.
- **par Tableaux** : 4 entrées dans les tableaux : l'indice du noeud (i), sa valeur (val) et les indices des sous arbres de gauche et de droite.
- **implicite** : Les sommets sont stockés dans un array, les arêtes s'obtiennent de manière implicite par la position des sommets dans l'array : si x est stocké dans $A[i]$, alors les enfants y_1, \dots, y_d de x sont stockés dans $A[di]$, $A[di+1], \dots, A[di+d-1]$, où d est le degré ($d \geq 2$).

4.4 L'algorithme du Knapsack 0/1

KNAPSACK(w, v, W) prend une liste de poids w , une liste de valeurs v associées et un poids maximum W comme entrées. Il renvoie une matrice C_{iW} qui indique le poids max que l'on peut avoir avec les n premiers objets en ne dépassant pas le poids max W .

Algorithme 5	KNAPSACK(w, v, W)	$O(nW)$
Input: $w = (w_1, \dots, w_n), v = (v_1, \dots, v_n), W \in \mathbb{N}$		
Output: Valeur V telle que $V = \max \sum_{i \in S} v_i$ pour tout $S \subseteq \{1, \dots, n\}$ avec $\sum_{i \in S} w_i \leq W$		
1:	for $w = 0 \dots W$ do	
2:	$c_{0,w} \leftarrow 0$	
3:	end for	
4:	for $i = 1 \dots n$ do	
5:	$c_{i,0} \leftarrow 0$	
6:	for $w = 1 \dots W$ do	
7:	if $w_i \leq w$ then	
8:	if $v_i + c_{i-1, w-w_i} > c_{i-1, w}$ then	
9:	$c_{i,w} = v_i + c_{i-1, w-w_i}$	
10:	else	
11:	$c_{i,w} = c_{i-1, w}$	
12:	end if	
13:	else	
14:	$c_{i,w} = c_{i-1, w}$	
15:	end if	
16:	end for	
17:	end for	
18:	return $c_{n,W}$	

L'algorithme suivant permet de retrouver une solution optimale :

Algorithme 6	GETOPTIMALCHOICE(W, n, v_i, C)
Input: L'input de KNAPSACK et le C calculé	
Output: Un choix optimal	
1:	for $i = n, \dots, 1$ do
2:	if $c_{i-1, W} = c_{i, W}$ then
3:	print "On ne prend pas l'objet" i
4:	else
5:	print "On prend l'objet" i
6:	$W \leftarrow W - v_i$
7:	end if
8:	end for

4.5 Le problème LCS

Définitions Un sous-ensemble $C \subseteq M$ ne contenant que des éléments comparables s'appelle une chaîne.

Une antichaine $A \subseteq M$ est un sous-ensemble dans lequel il n'y a pas deux éléments qui sont comparables.

Soit $X(x_1, \dots, x_m)$ une suite. Alors $Z = (z_1, \dots, z_k)$ est une sous-suite de X s'il existe des indices $1 \leq i_1 < \dots < i_k \leq m$ tels que tout $j \in [1, k]$ nous avons $z_j = x_{i_j}$. Le problème LCS (Longest Common Subsequence) consiste à trouver la plus longue sous-suite commune à deux suites X et Y .

Dans un ensemble fini et partiellement ordonné M , la longueur p d'une chaîne maximale est égale au nombre minimal d'antichaines d'une décomposition de M .

3.1 Arbres binaires de recherche

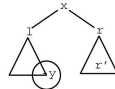
Search. $O(N)$

- S'il s'agit de l'arbre vide, la recherche échoue (x ne se trouve pas dans l'arbre).
- Sinon on compare x avec la clé r de la racine :
 - Si $x = r$, on a fini : on a trouvé le noeud r cherché.
 - Si $x < r$, on cherche pour x dans le sous-arbre de gauche.
 - Si $x > r$, on cherche dans le sous-arbre droit.

Insert. $O(N)$

On commence en faisant une recherche de x . Si on trouve x alors qu'il est déjà dans l'arbre et on ne doit donc plus rien faire. Si on ne le trouve pas, l'algorithme se terminera sur une feuille. Appelons b la clé de cette feuille.

- Si $x < b$, on ajoute un sommet gauche de cette feuille, et on le munir de la clé x .
- Sinon, on ajoute un sommet droit à cette feuille, et on le munir de la clé x .



Delete. $O(N)$ On commence comme pour Search(x). Si la recherche échoue, x ne se trouve pas dans l'arbre, nous n'avons plus rien à faire. Si x se trouve dans l'arbre, nous déterminons ainsi sa position. Nous aimerions enlever x sans perdre la propriété de l'arbre de recherche. Dans le cas général si x a deux fils, on remplace x par y et on remplace y par son sous-arbre de gauche.

Temps de parcours

1. meilleur : $\lfloor \log_2(N) \rfloor$;
2. moyen : $O(\log(N))$;
3. pire : $N - 1$,

3.2 Arbres AVL

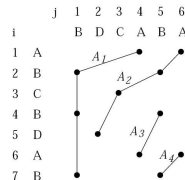
Un arbre binaire est dit équilibré (ou arbre AVL) si pour tout sommet k $\text{bal}(k) < 2$

La hauteur d'un arbre AVL avec n sommets vérifie l'inégalité : $h < 1.4404 \log(n) + 0.672$. La hauteur est donc $O(\log(N))$.

Insertion. On commence par insérer le sommet de la même manière que dans un arbre de recherche binaire. Ensuite, l'algorithme remonte du sommet inséré jusqu'à la racine en vérifiant l'équilibre à chaque sommet du chemin. Si un sommet non-équilibré est rencontré, alors une rotation est effectuée.

Effacement. On commence par effacer le sommet, on le remplace ensuite par le plus petit élément de son sous-arbre de droite ou par le plus grand élément de son sous-arbre de gauche, et pour finir on rééquilibre un utilisant des rotations.

Page 2



Algorithme 7 LCS(X, Y) $O(mn)$

Input: Suites X et Y

Output: Une décomposition de M en antichaines

```
1: for  $j = 1 \dots m$  do
2:    $s[j] \leftarrow n+1$ 
3:    $A[j] \leftarrow \emptyset$ 
4: end for
5: for  $i = 1 \dots m$  do
6:    $l \leftarrow \min\{k \mid (i, k) \in M\}$ 
7:    $j \leftarrow l$ 
8:   while  $l \leq n$  do
9:     if  $l \leq s[j]$  then
10:       $A[j] \leftarrow A[j] \cup \{(i, k) \in M \mid l \leq k \leq s[j]\}$ 
11:      temp  $\leftarrow s[j]$ 
12:       $s[j] \leftarrow l$ 
13:       $l \leftarrow \min\{k \mid (i, j) \in M \text{ AND } k > \text{temp}\}$ 
14:    end if
15:     $j \leftarrow j+1$ 
16:  end while
17: end for
18: return  $A[j]$  où les antichaines sont stockées et  $p = \max\{j \mid s[j] \leq n\}$  la longueur de la lcs( $X, Y$ )
```

4.6 Floyd-Warshall

L'algorithme de Floyd-Warshall permet de trouver pour toutes les paires de sommets du graphe $G = (G, E)$ la longueur de chemin la plus courte entre eux. Soit $V = \{1, \dots, n\}$, alors l'algorithme se base sur l'idée suivante : Fixons $k \leq n$. Pour toute paire de points i et j , trouver le plus court chemin entre i et j qui n'utilise que les sommets $1, \dots, k$. Commencer avec $k = 1$ et continuer jusqu'à ce que $k = n$.

$$C_{ij}^0 = \begin{cases} w(i, j) & \text{si } (i, j) \in E \\ \infty & \text{sinon} \end{cases}$$

$$C_{ij}^k = \min\{C_{ij}^{k-1}, C_{ik}^{k-1} + C_{kj}^{k-1}\}$$

5 Algorithmes gloutons

Définition Un algorithme glouton est un algorithme qui maximise son profit à chaque étape.

5.1 L'algorithme Greedy Selector

GREEDYSELECTOR(s, f) prend en entrée un tableau s contenant les horaires de début de mobilisation d'une ressource par une personne i et f un tableau contenant les horaires de fin. Les tableaux sont triés selon f . Il renvoie A une listes d'indices des ressources tel que le nombre maximal de préférences soit satisfait.

Page 4

