Algorithmique ENS Lyon L3 - TD5 - Corrigé

Anne Benoit

25 octobre 2005

Exercice 1. *Hors d'œuvre*

On considère un ensemble S de $n \geq 2$ entiers distincts stockés dans un tableau (S n'est pas supposé trié). Résoudre les questions suivantes :

1 - Proposer un algorithme en $\mathcal{O}(n)$ pour trouver deux éléments x et y de S tels que $|x-y| \geq |u-v|$ pour tout $u, v \in S$.

2 - Proposer un algorithme en $\mathcal{O}(n \log n)$ pour trouver deux éléments x et y de S tels que $x \neq y$ et $|x-y| \leq |u-v|$ pour tout $u, v \in S$, $u \neq v$.

3 - Soit m un entier arbitraire (pas nécessairement dans S), proposer un algorithme en $\mathcal{O}(n \log n)$ pour déterminer s'il existe deux éléments x et y de S tels que $x + y = m$.

4 - Proposer un algorithme en $\mathcal{O}(n)$ pour trouver deux éléments x et y de S tels que $|x-y| \leq \frac{1}{n-1}(\max(S) - \min(S))$.

Correction.

1 - Pour que $|a-b|$ soit maximum il faut que a (Resp b) soit maximum et que b (Resp a) soit minimum. Le problème se résume donc à une recherche du min et du max qui se fait en $O(n)$.

2 - Si $|c-d|$, est minimum pour c et d appartenant à S alors il n'existe pas e appartenant à S tel que $a > e > b$ ou $a < e < b$: sinon $|c-e| < |c-d|$. On trie donc S en $O(n \times logn)$ et on cherche ensuite le min des $s_i - s_{i+1}$, (s_i et s_{i+1} étant deux entier consécutifs de S) en $O(n)$.On obtient donc un algorithme en $O(n \times logn)$.

3 - On commence par trier S en $O(n \times logn)$.Ensuite pour tout y appartenant à S on cherche si $m - y$ appartient à S par dichotomie (S est trié), la dichotomie se faisant en $O(logn)$ cette opération ce fait en $O(n \times logn)$. A la fin on obtient bien un algorithme en $O(n \times logn)$.

4 - Il existe deux méthodes pour résoudre cette question.

4.1 - Première méthode

Cette méthode est basée sur la médiane. On note S_1 l'ensemble des élément de S inféreurs à la médiane et S_2 les éléments de S supérieurs à la médiane. Il est à noter que $\|S_1\| = \|S_2\| = \lceil \frac{\|S\|}{2} \rceil$.

début
  Calculer le min et le max de S ;
  si $n = 2$ **alors** retourner (Min,Max) ;
  sinon
    extraire la médiane de S ; calculer S_1 S_2 ;
    relancer l'algo sur S_1 ou S_2 suivant le cas.
fin

Preuve de l'algorithme : on pose $M(S)$ la moyenne pondérée de S, $n = \| S \|$, a le min de S, b le max de S, et m la médianne de S, deux cas sont alors possibles :

– n impair :

$$n = 2k+1, \text{alors} \|S_1\| = \frac{n+1}{2} = k+1$$

$$\text{donc :} M(S_1) = \frac{m-a}{k+1-1} = \frac{m-a}{k}$$

$$\text{et :} \|S_2\| = \frac{n+1}{2} = k+1$$

$$\text{donc :} M(S_2) = \frac{b-m}{k+1-1} = \frac{b-m}{k}$$

$$\text{donc :} \frac{M(S_1)+M(S_2)}{2} = (\frac{m-a}{k} + \frac{b-m}{k}) \times \frac{1}{2} = \frac{b-a}{2k}$$

$$\text{or :} \frac{b-a}{2k} = \frac{b-a}{n-1} = M(S)$$

$$\text{donc :} 2 \times M(S) = M(S_1) + M(S_2)$$

$$\text{finalement :} M(S_1) \leq M(S) \text{ou} M(S_2) \leq M(S).$$

– n pair :

$$n = 2k, \text{alors} \|S_1\| = \frac{n+1}{2} = k$$

$$\text{donc :} M(S_1) = \frac{m-a}{k-1}$$

$$\text{et :} \|S_2\| = \frac{n+1}{2}$$

$$\text{donc :} M(S_2) = \frac{b-m}{k-1}$$

$$\text{donc :} \frac{M(S_1)+M(S_2)}{2} = (\frac{m-a}{k-1} + \frac{b-m}{k-1}) \times \frac{1}{2} = \frac{b-a}{2k-2}$$

$$\text{or :} \frac{b-a}{2k-2} \leq \frac{b-a}{n-1} = M(S)$$

$$\text{donc :} 2 \times M(S) \geq M(S_1) + M(S_2)$$

$$\text{finalement :} M(S_1) \leq M(S) \text{ou} M(S_2) \leq M(S).$$

Comme on sait que $M(S_1) \leq M(S)$ ou $M(S_2) \leq M(S)$ cela nous permet d'enclencher l'induction : soit sur S_1, soit sur S_2. Rechercher la moyenne se fait en $O(n)$, chercher le min et le max d'un ensemble aussi, donc :

$$C(n) = C(\lceil \frac{n}{2} \rceil) + O(n)$$

Ce qui donne par master théorème

$$C(n) = O(n)$$

4.2 - Deuxième méthode

On divise S en $(n-1)$ boites.

début
  si *il n'y a qu'une boite* **alors** on renvoie deux de ses éléments;
  sinon
    si *le nombre d'éléments des* $\lfloor \frac{n-1}{2} \rfloor$ *premières boites est supérieur à* $\lfloor \frac{n-1}{2} \rfloor$. **alors** la moyenne pondérée sur les $\lfloor \frac{n-1}{2} \rfloor$ premières boites est inférieure à celle de S, et on relance l'algorithme sur ces boites. ;
    sinon
      le nombre d'éléments des $\lceil \frac{n+1}{2} \rceil$ dernières boites est supérieur à $\lceil \frac{n+1}{2} \rceil$, et alors la moyenne pondérée sur les $\lceil \frac{n+1}{2} \rceil$ dernières boites est inférieure à celle de S, et on relance l'algorithme sur celles-ci.
fin

Quant à la complexité elle est en $O(n)$:

$$C(n) = C(\lceil \frac{n}{2} \rceil + 1) + O(n)$$

Ce qui donne par master théorème

$$C(n) = O(n)$$

Exercice 2. *Arbres binaires de recherche optimaux*

Un *arbre binaire de recherche* est une structure de données permettant de stocker un ensemble de clés ordonnées $x_1 < x_2 < \ldots < x_n$, pour ensuite effectuer des opérations du type *rechercher*, *insérer* ou *supprimer* une clé. Il est défini par les propriétés suivantes :
(i) C'est un arbre où chaque nœud a 0, 1 ou 2 fils, et où chaque nœud stocke une des clés.
(ii) Etant donné un nœud avec sa clé x, alors les clés de son sous-arbre gauche sont strictement inférieures à x et celles de son sous-arbre droit sont strictement supérieures à x.

La figure 1 représente un arbre binaire de recherche pour les clés $x_1 < x_2 < x_3 < x_4 < x_5 < x_6 < x_7 < x_8 < x_9$.

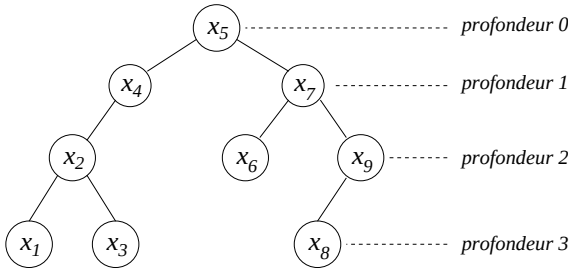


FIG. 1 – Exemple d'arbre binaire de recherche.

Les requètes auxquelles on s'intéresse ici sont les *recherches* de clés. Le coût de la recherche d'une clé x correspond au nombre de tests effectués pour la retrouver dans l'arbre en partant de la racine, soit exactement la profondeur de x dans l'arbre, plus 1 (la racine est de profondeur 0).

Pour une séquence fixée de recherches, on peut se demander quel est l'arbre binaire de recherche qui minimise la somme des coûts de ces recherches. Un tel arbre est appelé *arbre binaire de recherche optimal* pour cette séquence.

1 - Pour un arbre binaire de recherche fixé, le coût de la séquence ne dépend clairement que du nombre de recherches pour chaque clé, et pas de leur ordre. Pour $n = 4$ et $x_1 < x_2 < x_3 < x_4$, supposons que l'on veuille accèder une fois à x_1, 9 fois à x_2, 5 fois à x_3 et 6 fois à x_4. Trouver un arbre binaire de recherche optimal pour cet ensemble de requètes.

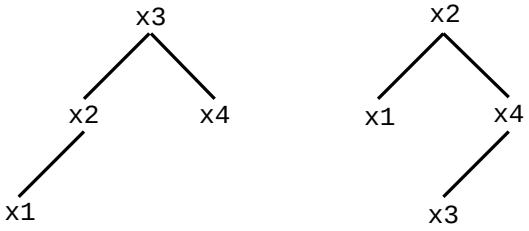
2 - Donner un algorithme en temps $\mathcal{O}(n^3)$ pour construire un arbre binaire de recherche optimal pour une séquence dont les nombres d'accès aux clés sont $c_1, c_2, \ldots, c_n$ (c_i est le nombre de fois que x_i est recherché). Justifier sa correction et sa complexité.

Indication : pour $i \leq j$, considérer $t[i,j]$ le coût d'un arbre de recherche optimal pour les clés $x_i < ... < x_j$ accédées respectivement $c_i, \ldots, c_j$ fois.

Correction.

1 - Arbres binaires optimaux

il suffit de dessiner l'ensemble des graphes possibles (il y en a 14) pour trouver les deux graphes minimaux.



2 - Algorithme

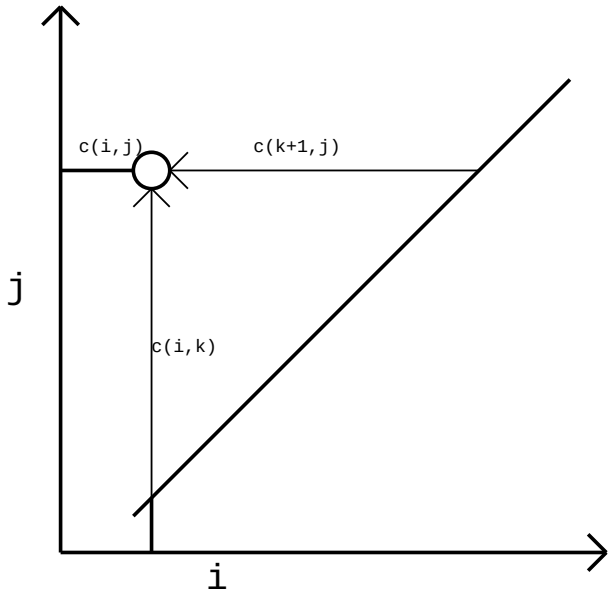
Le poids d'un arbre est le poid de ses fils, plus le poids de sa racine, plus le poids de tous les noeuds de ses fils (on leur rajoute un niveau). On appelle $t[i,j]$ le poid de l'arbre, de poids minimum contenant les sommets de i à j, et C_k le poid du sommet p. donc :

$$t[i,j] = min_{i \leq k \leq j}(t[i,k-1] + C_k + t[k+1,j] + \sum_{p=i}^{k-1}(C_p) + \sum_{p=k+1}^{j}(C_p))$$

soit :

$$t[i,j] = min_{i \leq k \leq j}(t[i,k-1] + t[k+1,j]) + \sum_{p=i}^{j}(C_p)$$

On initialise l'algo avec $t[i,i-1] = 0$.



La complexité est en $O(n^3)$: une boucle sur les i une autre sur les j puis une dernière sur les k.

Exercice 3. *Un nouvel algorithme de tri*

Soit T un tableau contenant n entiers distincts, la procédure `Nouveau_Tri` est définie de la manière suivante :

`Nouveau_Tri` (T,i,j)
| `si` $T[i] > T[j]$ `alors` échanger les valeurs $T[i]$ et $T[j]$;
| `si` $i \leq j-2$ `alors faire`
| $k \leftarrow \lfloor (j-i+1)/3 \rfloor$;
| `Nouveau_Tri`$(T, i, j-k)$;
| `Nouveau_Tri`$(T, i+k, j)$;
| `Nouveau_Tri`$(T, i, j-k)$;

1 - Montrer que `Nouveau_Tri`$(T, 1, n)$ trie correctement le tableau T.

2 - Donner la complexité en nombre de comparaisons de cet algorithme. Comparer avec les complexités des algorithmes de tri vus en cours (*Rappel :* $\ln(2) \simeq 0,7$ et $\ln(3) \simeq 1,1$).

Correction.

1 - L'algorithme trie correctement.
- Pour une longueur inférieure à deux l'algo est réduit à l'échange ce qui donne un tableau trié.
- Pour i supérieur à trois la première ligne ne sert à rien. On notera $X \leq Y$ si $\forall x \in X$ $\forall y \in Y : x \leq y$. On pose $l = \|T\|$,on pose A l'ensemble des $(\lfloor \frac{l+1}{3} \rfloor)$ premier élément de T, C l'ensemble des $(\lceil \frac{l}{3} \rceil)$ élément suivants , on pose B l'ensemble des $(\lfloor \frac{l}{3} \rfloor)$ dernier élément. On commence par trier les éléments de $A \cup C$ par induction : comme $\|A \cup C\| = (\lceil \frac{2l}{3} \rceil) < l$. Donc on obtient $A < C$. On trie maintenant $C \cup B$ par induction : comme $\|A \cup B\| \leq (\lceil \frac{2l}{3} \rceil) < l$.Alors $\|B\| \leq \|C\|$ ce qui implique que si un élément b de B avant le tri se retrouve dans B après, c'est qu'il existe un élément c dans C avant le tri tel que $c < b$ or comme c est supérieur à tous les éléments de A b aussi donc après le tri $C \leq B$ et $A \leq B$ ou encore $C \cup A \leq B$. Finalement on retrie $A \cup C$ et on obtient $A < B < C$ avec A, B, C, trié donc T est trié.

2 - Complexité

Pour trier un ensemble de n (n supérieur à trois) éléments on trie trois ensembles à $\lceil \frac{2n}{3} \rceil$ éléments et on fait un échange. Pour trier un ensemble de 2 éléments on fait un échange. Donc :

$$C(n) = 1 + 3C(\lceil \tfrac{2n}{3} \rceil) \text{ et } C(1) = 2$$

donc par master théorème :

$$C(n) = O(n^{log_{\frac{3}{2}}(3)}) \simeq O(n^{2,75})$$

C'est mauvais.

Exercice 4. *Bricolage*

Dans une boite à outils, vous disposez de n écrous de diamètres tous différents et des n boulons correspondants. Mais tout est mélangé et vous voulez appareiller chaque écrou avec le boulon qui lui correspond. Les différences de diamètre entre les écrous sont tellement minimes

qu'il n'est pas possible de déterminer à l'œil nu si un écrou est plus grand qu'un autre. Il en va de même avec les boulons. Par conséquent, le seul type d'opération autorisé consiste à essayer un écrou avec un boulon, ce qui peut amener trois réponses possibles : soit l'écrou est strictement plus large que le boulon, soit il est strictement moins large, soit ils ont exactement le même diamètre.

1 - Ecrire un algorithme simple en $\mathcal{O}(n^2)$ essais qui appareille chaque écrou avec son boulon.

2 - Supposons qu'au lieu de vouloir appareiller tous les boulons et écrous, vous voulez juste trouver le plus petit écrou et le boulon correspondant. Montrer que vous pouvez résoudre ce problème en moins de $2n - 2$ essais.

3 - Prouver que tout algorithme qui appareille tous les écrous avec tous les boulons doit effectuer $\Omega(n \log n)$ essais dans le pire des cas.

Problème ouvert : proposer un algorithme en $o(n^2)$ essais pour résoudre ce problème.

Correction.

1 - Algorithme

Pour appareiller les boulons et les écrous, il suffit de prendre un boulon arbitrairement, de le tester avec tous les écrous.On trouve alors le bon en au plus n tests et il suffit alors de recommencer avec tous les autres boulons.On obtient donc le résultat en au plus $\frac{n(n-1)}{2} = O(n^2)$ tests.

2 - Le plus petit écrou et son boulon

Le principe est de numéroter les boulons et les écrous de 1 à n, de manière arbitraire, et de faire progresser des compteurs (par exemple i et j) pour marquer le minimun courant dans l'une des catégories, et la structure en cours de test dans l'autre.

début
  while *(i ≤ n) && (j ≤ n)* **do**
    si *i=j=n* **alors** sortir de la boucle ;
    si *ecrou.i = boulon.j* **alors** s'en souvenir et faire $i := i + 1$;
    si *ecrou.i < boulon.j* **alors** $j := j + 1$; $min = ecrou$;
    si *ecrou.i > boulon.j* **alors** $i := i + 1$; $min = boulon$;
fin

A la fin de cette boucle,

- si min=écrou, l'écrou i est le plus petit.
- si min=boulon, le boulon j est le plus petit. Et dans les deux cas, on sait déjà quel est l'élément qui correspond : en effet, le compteur de l'autre catégorie vaut n+1 (ou n dans un cas spécial),et on a donc déjà rencontré le minimum. la seule raison possible pour laquelle il n'a pas été conservé est donc qu'il ait été testé avec l'autre minimum. Pour le cas spécial ou i=j=n, le dernier test est inutile : en effet, on sait que l'un des deux, est minimum, et que une fois le boulon minimum atteind, j reste fixe : d'où le boulon n est minimum, et son homologue a soit déjà été trouvé, soit est l'écrou restant.

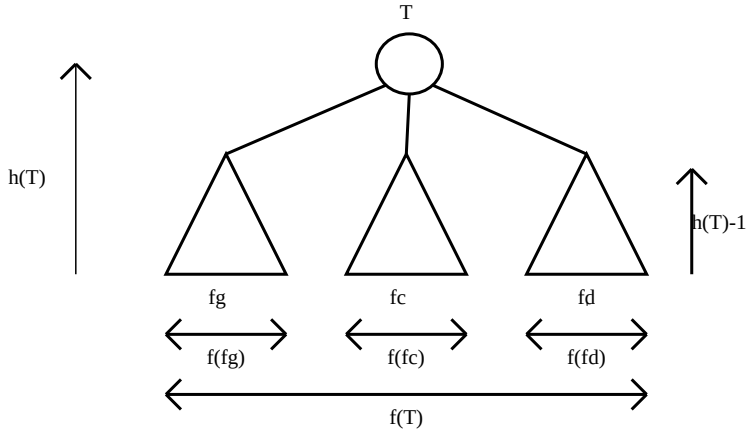
Cette boucle effectue donc au plus $2 * n - 2$ tests.

3 - Algorithme d'appareillage des écrous et de leur boulon

On note $f(T)$ le nombre de feuilles de l'arbre et $h(T)$ sa hauteur. Si T est un arbre ternaire $h(T) \geq \lceil log_3 f(T) \rceil$. Par induction sur la hauteur de T :

- Pour $h(T) = 0$ l'arbre à une feuille donc on a bien $h(T) \geq \lceil log_3 f(T) \lceil$.
- Pour $h(T) > 0$ un arbre ternaire a trois fils, le fils gauche fg, le fils central fc, et le fils droit fd de hauteur inférieur à $h(T) - 1$.On suppose que $h(T) \geq \lceil log_3 f(T) \rceil$ est vrai

pour $h(T) \leq k$ k étant fixé on veut démontrer que cette propriété est vrai aussi pour $h(T) = k + 1$. Les feuilles d'un arbre sont aussi celles de ses fils. Donc



$$f(T) = f(fd) + f(fg) + f(fd)$$

de plus :

$$h(T) \leq h(fd) + 1$$
$$h(T) \leq h(fg) + 1$$
$$h(T) \leq h(fc) + 1$$

or par induction comme $h(T) = k + 1$, $h(fg) \leq k$, $h(fg) \leq k$,et $h(fg) \leq k$ on a :

$$k = h(fc) \leq \lceil log_3 f(fc) \rceil$$

$$h(fd) \leq \lceil log_3 f(fd) \rceil$$
$$h(fg) \leq \lceil log_3 f(fg) \rceil$$

Donc $f(fc), f(fg), f(fd) \leq 3^k$ or :

$$f(T) = f(fc) + f(fg) + f(fd)$$

donc :

$$f(T) \leq 3 \times 3^k = 3^{k+1}$$

D' où on en déduit que :

$$h(T) \geq log_3 f(T)$$

Il y a $!n$ agencements boulons-écrous possibles donc l'arbre de décision, qui est ternaire a pour hauteur : $log_3 !n \sim n log_3 n$, donc la complexité dans le pire cas de l'algo est de : $\Omega(n \times logn)$.