CONCOURS CENTRALE•SUPÉLEC

Informatique

MP

4 heures Calculatrices autorisées

2012

Les candidats indiqueront en tête de leur copie le langage de programmation choisi (Pascal ou Caml). Les candidats ayant choisi Caml devront donner le type de chaque fonction écrite, lorsque celui-ci n'est pas imposé. Les candidats travaillant en Pascal pourront écrire des fonctions ou des procédures.

Choix du pivot dans le tri rapide

L'objet de ce problème est le choix d'un pivot dans l'algorithme du tri rapide. On s'intéresse au tri en ordre croissant d'un tableau d'entiers qui pourront toujours être supposés distincts.

En Pascal, on dispose du type tableau suivant :

```
const long_tab = 1000;
type tableau = array[0..long_tab-1] of integer;
```

La longueur t des tableaux avec lesquels on travaille devra donc être majorée par `long_tab` et les données sont alors présentes dans la zone indexée par $0 \leqslant i \leqslant t-1$. Cette longueur utile t devra donc être éventuellement donnée en paramètre.

En Caml, on rappelle que la longueur d'un tableau est donnée par `vect_length` en Caml light et `Array.length` en Ocaml.

I Tri rapide d'un tableau

I.A – Écrire une fonction/procédure `echange` réalisant l'échange de deux éléments d'un tableau.

```
echange : int -> int -> int vect -> unit = <fun>
procedure echange(i, j : integer ; var t : tableau);
```

I.B – Décrire un algorithme simple de tri ; évaluer sa complexité dans le pire des cas en termes de comparaisons. Le programmer.

I.C – L'algorithme du tri rapide sur un tableau consiste en une première étape où on permute les éléments du tableau de sorte que les éléments plus petits (respectivement plus grands) qu'un pivot p soient placés avant (respectivement après) cet élément p dans le tableau. On exécute ensuite récursivement le tri rapide sur les deux parties du tableau ainsi séparées par p.

Par exemple, le tableau $[|\mathbf{19}; 7; 17; 14; 22; 5; 26; 21; 2; 12|]$ devient après la première étape (si on choisit 19 comme pivot) : $[|2; 7; 17; 14; 12; 5; \mathbf{19}; 21; 26; 22|]$. Notons que les éléments inférieurs (respectivement supérieurs) à 19 peuvent être permutés entre eux. Le point crucial est qu'ils soient situés avant (respectivement après) 19 dans le tableau.

Un des intérêts importants de cet algorithme est qu'il peut être réalisé en place : le tableau ne sera jamais recopié. Pour cela, les appels récursifs prendront comme arguments les indices délimitant la partie à trier.

I.C.1) Expliquer comment on peut réaliser en place la phase de séparation du tableau en temps $O(n)$, avec n la longueur (nombre d'éléments) du (sous-)tableau à traiter.

I.C.2) Écrire une fonction `separation` prenant en entrée un vecteur v et deux indices i_1 et i_2 (avec $0 \leqslant i_1 < i_2 \leqslant t-1$, condition que le programme n'aura pas à vérifier), ayant pour fonction de séparer le sous-tableau $v[i_1+1..i_2]$ selon le pivot $p = v(i_1)$, et retournant l'indice du tableau correspondant à la position de p dans v après séparation. Dans l'exemple précédent, l'appel `separation v0 0 9` (Caml) ou `separation(v0,0,9)` (Pascal) retourne l'indice 6 et le vecteur v_0 a été modifié.

```
separation: 'a vect -> int -> int -> int = <fun>
function separation(var v: tableau ; i1, i2: integer): integer;
```

I.C.3) Écrire une fonction `tri_rapide` réalisant le tri d'un vecteur/tableau en appliquant l'algorithme décrit plus haut.

II Étude de complexité

On s'intéresse maintenant à la complexité du tri rapide dans deux cas particuliers.

II.A – Donner un ordre de grandeur du nombre de comparaisons effectuées lorsque le tableau est déjà trié dans l'ordre croissant (respectivement décroissant). Sans forcément donner un équivalent de ce nombre de comparaison $C(n)$, on donnera (en justifiant) une valeur simple $v(n)$ telle que $C(n) = O(v(n))$ et $v(n) = O(C(n))$.

II.B – On suppose ici que lors d'une exécution du tri rapide, chaque séparation du tableau a coupé le tableau en deux parts égales.

II.B.1) Montrer qu'un majorant raisonnable du nombre de comparaisons effectuées pour trier un tableau de taille n vérifie une relation de la forme $M(2n) = 2M(n) + \alpha n$.

II.B.2) Donner (en fonction de n) la valeur de $M(n)$ lorsque n est de la forme 2^k avec $k \in \mathbb{N}$.

II.B.3) On ne suppose plus, ici, que n est de la forme 2^k. Montrer que, si on suppose que $(M(n))_{n\in\mathbb{N}}$ est croissante, alors $M(n) = O(n \ln n)$.

III Recherche d'une pseudo médiane

Ce qui précède suggère que le choix d'un pivot « proche de la médiane » permet d'améliorer les performances du tri rapide. Il existe un algorithme permettant de trouver cette médiane en un temps linéaire en la taille du tableau, mais cet algorithme difficile ne sera pas discuté ici. Une façon simple pour lutter contre le pire des cas consiste à choisir le pivot de façon aléatoire, mais le gain obtenu est relativement subtil et le pire des cas reste quadratique. Une méthode intermédiaire consiste à rechercher une « pseudo médiane ».

Lorsque $\alpha \in]0, 1[$, une *α-pseudo médiane* d'un tableau est une valeur présente dans le tableau telle qu'au moins $K_1 n^\alpha$ (respectivement $K_2 n^\alpha$) éléments du tableau lui sont inférieurs (respectivement supérieurs), avec K_1 et K_2 deux constantes strictement positives.

Pour un tableau de taille $n = 3^k$, on utilisera l'algorithme de recherche d'une pseudo médiane suivant :
- si $k = 0$, on retourne directement le seul élément considéré ;
- sinon, on regroupe les éléments du tableau par 3, on calcule les 3^{k-1} médianes de ces groupes, puis on applique récursivement l'algorithme à ces 3^{k-1} valeurs.

Dans les questions **III.A** et **III.B**, on admet que cet algorithme permet le calcul d'une pseudo médiane et on propose de le mettre en œuvre de deux manières différentes.

On rappelle que les différents éléments du tableau pourront être supposés distincts.

III.A – Dans un tableau, en place

III.A.1) Écrire une fonction prenant en entrée un tableau et trois indices distincts et retournant la position de la médiane, parmi les trois éléments du tableau dont on a donné les indices.

III.A.2) Écrire une fonction calculant une pseudo médiane. Cette fonction travaillera obligatoirement dans le tableau initial, sans en créer de nouveau, et en maintenant globalement invariant l'ensemble des valeurs présentes dans le tableau.

> On pourra supposer le tableau de taille 3^k, placer dans une première étape les médianes de blocs de trois en positions $3i$, puis prendre les médianes de ces médianes et les placer en position $9i$, etc.

III.B – À l'aide d'un arbre ternaire

On propose de construire un arbre ternaire : les feuilles sont étiquetées par les entrées du tableau et chaque nœud interne est la médiane de ses fils. La **figure 1** montre l'arbre construit sur le tableau $[7; 1; 4; 9; 8; 5; 3; 2; 6]$.

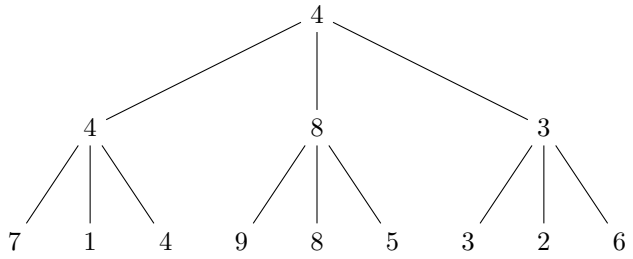

Figure 1

En Caml :

```
type ternaire=F of int | N of int*ternaire*ternaire*ternaire;;
let racine=function F(x)->x | N(x,_,_,_)->x;;
```

En Pascal :

```
type
   Arbre = ^Noeud;
   Noeud = record
              racine: integer;
              fg, fm, fd: Arbre;
           end;
```

III.B.1) Écrire une fonction calculant la médiane de trois entiers distincts.

```
mediane3 : int * int * int -> int = <fun>
function mediane3(i1, i2, i3 : integer) : integer;
```

III.B.2) Écrire une fonction prenant en entrée trois arbres ternaires et retournant l'arbre ternaire dont la racine est étiquetée par la médiane des trois racines des arbres donnés en entrée et a pour fils ces trois arbres.

III.B.3) Écrire une fonction récursive construisant l'arbre ternaire associé à un sous-tableau de taille 3^k. En Caml, `construire t i j` retourne l'arbre du sous-tableau $t[i..j]$. Même chose en Pascal avec `construire(t,i,j)`

```
construire : int vect -> int -> int -> ternaire = <fun>
function construire(t: tableau ; i, j: integer): Arbre;
```

III.B.4) Écrire une fonction calculant, à l'aide d'un arbre ternaire, une pseudo médiane d'un tableau (dont la longueur pourra être supposée de la forme 3^k).

III.C – Étude théorique de l'algorithme

III.C.1) Donner un ordre de grandeur du temps d'exécution de l'algorithme de calcul d'une pseudo médiane.

III.C.2) Si $k = 1$, la valeur retournée est exactement la médiane du tableau. Montrer que pour $k \geqslant 2$, il existe au moins 2^k éléments du tableau qui sont majorés (au sens large) par la valeur retournée.

III.C.3) Montrer que pour tout $k \geqslant 2$, il existe un tableau pour lequel il y a exactement 2^k éléments du tableau qui sont majorés (au sens large) par la valeur retournée.

III.C.4) Prouver que cet algorithme permet de calculer une α-pseudo médiane, avec $\alpha = \ln 2 / \ln 3$.

III.C.5) Expliquer comment adapter l'implémentation de l'algorithme si le tableau a une longueur qui n'est pas une puissance de 3 ?

III.D – Extensions du principe de l'algorithme

III.D.1) Si on modifie l'algorithme en considérant des blocs de 5 éléments plutôt que 3, que dire (en supposant que la longueur du tableau est une puissance de 5) du résultat retourné et du temps de calcul de ce nouvel algorithme ?

III.D.2) Montrer que pour tout $\varepsilon > 0$, il existe un algorithme s'exécutant en un coût linéaire et permettant de calculer une $(1 - \varepsilon)$-pseudo médiane d'un tableau.

IV Gain apporté par la pseudo médiane

On s'intéresse enfin au gain qu'apporte l'utilisation de pseudo médianes dans le tri rapide. On suppose ici (sauf à la dernière question) que le tri rapide est exécuté en utilisant à chaque étape une 1/2-pseudo médiane. Ici encore, une analyse précise et rigoureuse de la complexité est délicate, mais on souhaite obtenir une évaluation de façon raisonnablement convaincante.

On note $C(n)$ le temps de calcul dans le pire des cas pour appliquer le tri rapide avec une 1/2-pseudo médiane. Dans cette évaluation en première approximation, on s'autorise à écrire $C(x)$ même lorsque x n'est pas entier : ce sera un raccourci pour $C(\lceil x \rceil)$, avec $\lceil x \rceil$ la « partie entière supérieure de x ».

IV.A – Justifier qualitativement le fait que C vérifie une inégalité de la forme

$$C(n) \leqslant C(n - \sqrt{n}) + Kn$$

IV.B – Montrer que $C(n) = C(n/2) + O(n^{3/2})$.

On pourra étudier la suite définie par $\alpha_0 = n$ et $\alpha_{k+1} = \alpha_k - \sqrt{\alpha_k}$ si $\alpha_k \geqslant 1$ (et $\alpha_{k+1} = 0$ sinon) : établir par exemple que $\alpha_k \leqslant n/2$ si $k \geqslant \sqrt{n/2}$.

IV.C – Conclure.

IV.D – Que peut-on raisonnablement espérer comme complexité dans le pire des cas, pour un tri rapide effectué en calculant une $\ln 2 / \ln 3$-pseudo médiane avec l'algorithme de la **partie III** ?

• • • FIN • • •