X Informatique MP/PC 2012 — Corrigé

Ce corrigé est proposé par Céline Chevalier (Enseignant-chercheur à l'université) ; il a été relu par Guillaume Batog (Professeur agrégé) et Arnaud Borde (École Polytechnique).

Ce sujet s'intéresse au problème dit du « sandwich au jambon », aussi appelé théorème de Stone-Tukey, qui s'énonce de la manière suivante : « Un ensemble de n points en dimension d peut toujours être séparé en 2 parties de cardinal au plus $\lfloor n/2 \rfloor$ par un hyperplan de dimension $d-1$ ». Il procède pour cela en trois temps.

- Une courte première partie, très facile, permet d'obtenir deux algorithmes de base sur les tableaux, l'un pour renvoyer l'indice d'une case contenant le plus grand élément du tableau, l'autre pour renvoyer le nombre d'éléments du tableau qui sont plus petits qu'une valeur donnée.
- La deuxième partie a pour objectif de calculer l'élément médian d'un tableau. L'énoncé demande en fait une généralisation de ce problème, à savoir la recherche du k-ième élément. Cet algorithme repose sur une partition des données du tableau, en les comparant à une valeur particulière appelée « pivot ». Afin d'améliorer la complexité de la recherche, la dernière question permet de déterminer le pivot à utiliser dans cette partition.
- Enfin, la troisième partie se place en dimension 2 et utilise les algorithmes précédents pour généraliser le problème du médian. L'objectif est de séparer un ensemble de n points par deux droites, de telle sorte que chaque cadran contienne moins de $n/4$ points.

Au final, ce problème est plutôt facile. Comme souvent, il est regrettable que le jury ait fait le choix de détailler précisément plusieurs algorithmes (questions 4 et 10), ne demandant aux candidats que de traduire ces lignes dans le langage de leur choix. Ce sujet permet malgré tout de bien réviser les opérations classiques sur les tableaux.

Indications

Partie 1

1 Utiliser une variable temporaire `resultat` pour stocker la valeur de l'indice de la plus grande case du tableau. Parcourir le tableau en comparant pour chaque indice `i` la valeur de la case `tab[i]` et celle de `tab[resultat]`. Si la première est plus grande, mettre à jour la variable `resultat`.

2 Effectuer un parcours du tableau en comparant à chaque étape `i` la valeur de la case `tab[i]` et `val`. Le cas échéant, incrémenter la variable résultat.

Partie 2

3 Parcourir le tableau en utilisant deux tableaux annexes pour stocker les éléments plus petits (dans `tab_inf`) et plus grands (dans `tab_sup`) que le pivot. Les compter à l'aide de deux variables temporaires et compter les éléments égaux au pivot à l'aide d'une variable supplémentaire `nb_pivot`. Finalement, écraser le tableau initial avec les éléments de `tab_inf` puis `nb_pivot` éléments égaux au pivot et enfin les éléments de `tab_sup`.

6 Calculer l'indice médian de chaque sous-tableau et enregistrer les résultats dans un tableau `indicemedian`. Décaler ensuite les éléments correspondants au début du tableau.

Partie 3

8 Créer un tableau pour contenir les angles entre (x, y) et les éléments dont l'ordonnée est supérieure à y. Renvoyer alors l'indice médian de ce tableau.

9 Comme à la question précédente, enregistrer les angles entre (x, y) et les éléments dont l'ordonnée est strictement inférieure à y. Parcourir ensuite ce tableau pour compter le nombre d'angles inférieurs ou supérieurs à `theta`.

Sandwich au jambon

Les programmes de ce corrigé sont écrits en Maple. L'énoncé demande des modifications des tableaux donnés en argument, ce qui n'est pas possible avec la structure de liste. Afin de respecter cette demande tout en proposant des codes qui fonctionnent, les programmes de ce corrigé utilisent la structure `array`, ce qui crée quelques complications dans leur écriture. Un jour de concours, une structure de liste aurait parfaitement fait l'affaire.

L'énoncé fait l'hypothèse de l'existence d'une primitive `allouer(m,c)`, qui crée un tableau de taille `m` dont chaque case contient `c` à l'origine, ainsi que d'une primitive `taille(t)` qui renvoie la taille d'un tableau `t`. On donne ici deux instanciations possibles de ces primitives et les exemples associés :

```
allouer := proc(m,c)
    local t,i;
    t := array(1..m);
    for i from 1 to m do t[i] := c; od;
    RETURN(t);
end;
```

```
t := allouer(7,4); print(t);
```

$$t := t$$
$$[4, 4, 4, 4, 4, 4, 4]$$

```
taille := proc(t)
    RETURN(nops([entries(t)]));
end;
```

```
taille(t);
```

$$7$$

Enfin, l'énoncé suppose l'existence d'une fonction `floor(x)` qui renvoie la partie entière $\lfloor x \rfloor$ pour tout réel $x \geqslant 0$. Cette fonction a bien ce nom en Maple.

I. Grand, petit et médian

1 La fonction `calculeIndiceMaximum` prend en argument un tableau `tab` et deux indices `a` et `b`, et doit renvoyer un indice `a<=i<=b` du tableau tel que l'élément `tab[i]` soit le plus grand du tableau. Pour cela, elle initialise le résultat `resultat` à `a`. Elle parcourt ensuite les cases du tableau à l'aide d'une boucle `for`. Pour tout `i` de `a+1` jusqu'à `b`, elle teste la valeur de `tab[i]`. Si cette valeur est supérieure à `tab[resultat]`, alors la variable `resultat` prend la valeur `i`, sinon, la fonction passe à la case d'indice `i+1`. Enfin, elle renvoie la variable `resultat`.

```
calculeIndiceMaximum := proc(tab,a,b)
    local resultat, i;
    resultat := a;
```

```
    for i from a+1 to b do
        if tab[i] > tab[resultat] then resultat := i; fi;
    od;
    RETURN(resultat);
end;
```

Sur l'exemple demandé, l'application de ce programme donne

```
tableau := array(1..11,[3,2,5,8,1,34,21,6,9,14,8]);
```

$$tableau := [3, 2, 5, 8, 1, 34, 21, 6, 9, 14, 8]$$

```
calculeIndiceMaximum(tableau,1,11);
```

6

Notons que l'énoncé demande de renvoyer l'indice d'une case, mais que ce dernier n'est pas unique (si plusieurs cases contiennent le plus grand élément du tableau). On peut donc choisir de renvoyer le premier (comme ici, en utilisant un supérieur strict dans le test), le dernier (il faudrait alors utiliser un supérieur ou égal), ou encore n'importe quel autre.

Remarquons également que l'énoncé ne demande jamais de vérifier si le calcul demandé est licite, à savoir si `1<=a<=b<=taille(tab)`. On a pris le parti dans ce corrigé de ne pas tester ces inégalités, mais on pourrait rajouter, au début de chaque programme demandé, des lignes du type

```
if not(1<=a and a<=b and b<=taille(tab)) then
    ERROR('Les indices demandés ne sont pas compatibles
    avec le tableau tab.');
fi;
```

2 La fonction `nombrePlusPetit` prend en argument un tableau `tab`, deux indices `a` et `b` et un nombre `val`, et doit renvoyer le nombre d'éléments du tableau `tab[a..b]` dont la valeur est inférieure ou égale à `val`. Pour cela, elle initialise ce nombre `nombre` à 0 et parcourt le tableau à l'aide d'une boucle `for`. À l'étape `i`, si l'élément `tab[i]` est inférieur ou égal à la valeur `val`, la variable `nombre` est augmentée de 1. Sinon, on passe directement à l'élément `i+1`. Cela donne le code suivant :

```
nombrePlusPetit := proc(tab,a,b,val)
    local nombre, i;
    nombre := 0;

    for i from a to b do
        if tab[i] <= val
            then nombre := nombre + 1;
        fi;
    od;

    RETURN(nombre);
end;
```

On a bien le résultat demandé par l'énoncé :

```
nombrePlusPetit(tableau,1,11,5);
```

4