Algorithmique avancée

Corrigé du TP1

J.M. Dischler

Diviser pour régner : algorithmes de tri

Ecrire une fonction permettant de remplir un tableau A de n entiers relatifs sélectionnés au hasard en utilisant *Math.random()*. Implémenter les deux algorithmes de tri par insertion et tri rapide. Comparer ces deux algorithmes en traceant une courbe de temps de calcul mesurée pour une valeur de n croissante.

Diviser pour régner : intervalle de plus grande somme

Nous avons un tableau A de n entiers relatifs. Nous recherchons un sous-tableau de A dont la somme des éléments soit maximale. Autrement dit, nous recherchons un couple d'entiers i et j, $1 \leq i \leq j \leq n$ tel que $\sum_{k=i}^{j} A[k]$ soit maximale. La figure 1 montre un exemple de tableau pour lequel les valeurs cherchées sont $i = 4$ et $j = 5$.

2	5	-8	6	5	-9	3	4

FIGURE 1 – Tableau dont l'intervalle de plus grande somme est défini par $i = 4$ et $j = 5$.

1. Proposez un algorithme naïf.

```
PGS-NAÏF(A)
  d ← 1
  f ← 1
  max ← A[1]
  pour i ← 1 à n faire
  s ← 0
    pour j ← i à n faire
      s ← s + A[j]
      si s > max alors
        d ← i
        f ← j
        max ← s
```

2. Quelle est sa complexité ?

   La première boucle comporte n itérations, pour i allant de 1 à n. La deuxième boucle comporte $n - i + 1$ itérations. Toutes les autres opérations sont de coût constant. Le coût global est donc :

$$\sum_{i=1}^{n}(n - i + 1) = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}.$$

   D'où une complexité en $\Theta(n^2)$.

3. Proposez un algorithme « diviser pour régner » découpant le tableau en deux moitiés.

   Trois cas sont possibles : l'intervalle de plus grande somme est inclus dans la première moitié du tableau ; l'intervalle de plus grande somme est inclus dans la deuxième moitié du tableau ; l'intervalle de plus grande somme empiète sur les deux moitiés du tableau. Les deux premiers cas sont réglés par

un appel récursif sur chacune des deux moitiés. Pour régler le troisième cas, on calcule l'intervalle de plus grande somme dont la limite droite est le dernier élément de la première moitié et l'intervalle de plus grande somme dont la limite gauche est le premier élément de la deuxième moitié. Ensemble, ces deux intervalles nous fournissent l'intervalle de plus grande somme empiétant sur les deux moitiés. On peut envisager deux méthodes pour réaliser ce calcul.

Première méthode

Ici tout est recalculé à chaque fois, sans tenir compte d'éventuels résultats précédents.

PGS(A, d, f)
  si $d = f$ **renvoyer** (d, d, $A[d]$)
  $milieu \leftarrow \lfloor \frac{f-d}{2} \rfloor$
  $(début_1, fin_1, somme_1) \leftarrow$ PGS(A, d, $milieu$)
  $(début_2, fin_2, somme_2) \leftarrow$ PGS(A, $milieu + 1$, f)
  $DemiSommeGauche \leftarrow A[milieu]$
  $LimiteGauche \leftarrow milieu$
  $s \leftarrow A[milieu]$
  pour $i \leftarrow milieu - 1$ **à** d **faire**
    $s \leftarrow s + A[i]$
    si $s > DemiSommeGauche$ **alors** $DemiSommeGauche \leftarrow s$
      $LimiteGauche \leftarrow i$
  $DemiSommeDroite \leftarrow A[milieu + 1]$
  $LimiteDroite \leftarrow milieu + 1$
  $s \leftarrow A[milieu + 1]$
  pour $i \leftarrow milieu + 2$ **à** f **faire**
    $s \leftarrow s + A[i]$
    si $s > DemiSommeDroite$ **alors** $DemiSommeDroite \leftarrow s$
      $LimiteDroite \leftarrow i$
  $(début_3, fin_3, somme_3) \leftarrow (LimiteGauche, LimiteDroite, DemiSommeGauche + DemiSommeDroite)$
  si $somme_3 \geq somme_2$ et $somme_3 \geq somme_1$
    alors renvoyer ($début_3$, fin_3, $somme_3$)
    sinon si $somme_2 \geq somme_3$ et $somme_2 \geq somme_1$
      alors renvoyer ($début_2$, fin_2, $somme_2$)
      sinon renvoyer ($début_1$, fin_1, $somme_1$)

Deuxième méthode

On évite ici de parcourir entièrement chaque moitié de tableau pour calcule l'intervalle de plus grande somme dont la limite droite est le dernier élément de la première moitié et l'intervalle de plus grande somme dont la limite gauche est le premier élément de la deuxième moitié. Pour ce faire on calcule à chaque fois trois valeurs : l'intervalle de plus grande somme dont la limite droite est le dernier élément du tableau, l'intervalle de plus grande somme dont la limite gauche est le premier élément du tableau, la somme de tous les éléments du tableau.

PGS(A, d, f)
  si $d = f$ **renvoyer** (d, $A[d]$, d, $A[d]$, $A[d]$, d, d, $A[d]$)
  $milieu \leftarrow \lfloor \frac{f-d}{2} \rfloor$
  $(début_1, somme\_début_1, fin_1, somme\_fin_1, somme_1, d_1, f_1, s_1) \leftarrow$ PGS(A, d, $milieu$)
  $(début_2, somme\_début_2, fin_2, somme\_fin_2, somme_2, d_2, f_2, s_2) \leftarrow$ PGS(A, $milieu + 1$, f)
  si $somme\_début_1 \geq somme_1 + somme\_début_2$
    alors
      $somme\_début \leftarrow somme\_début_1$

$début \leftarrow début_1$
sinon
$somme\_début \leftarrow somme_1 + somme\_début_2$
$début \leftarrow début_2$
si $somme\_fin_2 \geq somme_2 + somme\_fin_1$
alors
$somme\_fin \leftarrow somme\_fin_2$
$fin \leftarrow fin_2$
sinon
$somme\_fin \leftarrow somme_2 + somme\_fin_1$
$fin \leftarrow fin_1$
$somme \leftarrow somme_1 + somme_2$
$d_3 \leftarrow fin_1$
$f_3 \leftarrow début_2$
$s_3 \leftarrow somme\_fin_1 + somme\_début_2$
si $s_3 \geq s_2$ et $s_3 \geq s_1$
alors renvoyer $(début, somme\_début, fin, somme\_fin, somme, d_3, f_3, s_3)$
sinon si $s_2 \geq s_3$ et $s_2 \geq s_1$
alors renvoyer $(début, somme\_début, fin, somme\_fin, somme, d_2, f_2, s_2)$
sinon renvoyer $(début, somme\_début, fin, somme\_fin, somme, d_1, f_1, s_1)$

L'appel initial est alors de la forme :

$$(début, somme\_début, fin, somme\_fin, somme, i, j, somme\_maximale) = \mathrm{PGS}(A, 1, n).$$

4. Quelle est sa complexité ?

Première méthode

Un appel de PGS *sur un tableau de taille n génère deux appels récursifs sur des tableaux de taille $n/2$. La division a un coût constant (calcul du milieu). Par contre, la combinaison des résultats nécessite la recherche de l'intervalle de plus grande somme se terminant en l'extrémité droite (resp. gauche) du premier (resp. deuxième) demi-tableau. Cette double recherche a un coût en $\Theta(n)$ (parcours de tout le tableau). D'où l'équation définissant la complexité :*

$$T(n) = T\left(\frac{n}{2}\right) + \Theta(n).$$

Nous avons donc ici : $a = 2$, $b = 2$ et $f(n) = \Theta(n) = \Theta(n^{\log_2 2})$. Nous sommes donc dans le cas 2 du théorème et donc :

$$T(n) = \Theta(n \log n).$$

Deuxième méthode

Un appel de PGS *sur un tableau de taille n génère deux appels récursifs sur des tableaux de taille $n/2$. La division a un coût constant (calcul du milieu) tout comme la combinaison des résultats. D'où l'équation définissant la complexité :*

$$T(n) = T\left(\frac{n}{2}\right) + \Theta(1).$$

Nous avons donc ici : $a = 2$, $b = 2$ et $f(n) = \Theta(1) = O(n^{1-1}) = O(n^{(\log_2 2)-1})$. Nous sommes donc dans le cas 1 du théorème et donc :

$$T(n) = \Theta(n^{\log_2 2}) = \Theta(n).$$

Recherche du deuxième plus grand élément

Nous supposerons ici que l'ensemble considéré ne contient pas deux fois la même valeur.

1. Proposez un algorithme simple de recherche du deuxième plus grand élément.

   DEUXIÈME-PLUS-GRAND(A)

   rang_max $\leftarrow 1$
   pour $i \leftarrow 2$ **à** n **faire si** A[rang_max] ¡ $A[i]$ **alors** rang_max $\leftarrow$ i
   si rang_max $\neq 1$ **alors** rang_second $\leftarrow 1$
   sinon rang_second $\leftarrow 2$
   pour $i \leftarrow 2$ **à** n **faire si** $i \neq$ rang_max **et** A[rang_second] ¡ $A[i]$ **alors** rang_second $\leftarrow$ i
   renvoyer A[rang_second]

2. Quel est sa complexité en nombre de comparaisons ?
   La recherche du maximum coûte $n-1$ comparaisons. La boucle qui recherche le deuxième plus grand élément une fois que le maximum a été trouvé effectue $n-2$ comparaisons. D'où un coût total de $2n-3$ comparaisons.

3. Récrivez votre algorithme de recherche du maximum sous la forme d'un tournoi (de tennis, de foot, de pétanque ou de tout autre sport). Il n'est pas nécessaire de formaliser l'algorithme ici, une figure explicative sera amplement suffisante.
   Les comparaisons sont organisées comme dans un tournoi :
   – *Dans une première phase, les valeurs sont comparées par paires. Dans chaque paire, il y a bien sûr un plus grand élément (le « vainqueur ») et un plus petit élément (le « vaincu »).*
   – *Dans la deuxième phase, les valeurs qui étaient plus grand élément de leur paire à la phase précédente sont comparées entres elles deux à deux.*
   – *On répète ce processus jusqu'au moment où il n'y a plus qu'**un** plus grand élément.*
   Ce procédé est illustré par la figure 2.

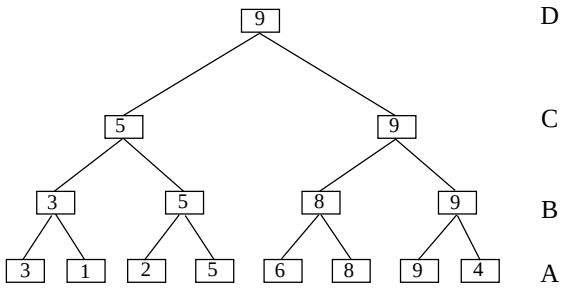

FIGURE 2 – Méthode du tournoi pour la détermination du maximum : A : les éléments sont comparés par paires ; B : les plus grands éléments de la phase A sont comparés entre eux, par paires ; C : les éléments « vainqueurs » à la phase B sont comparés entre eux ; D : il ne reste plus qu'un élément, c'est l'élément maximal.

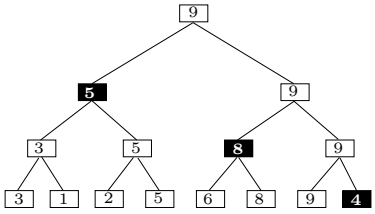

FIGURE 3 – Le deuxième plus grand élément a nécessairement été battu par le plus grand élément (et que par lui). Il figure donc parmi les éléments comparés à l'élément maximal. Ces éléments apparaissent ici sur fond noir.

4. Dans combien de comparaisons, le deuxième plus grand élément de l'ensemble a-t-il été trouvé être le plus petit des deux éléments comparés ?
   Le deuxième plus grand n'est plus petit que devant le plus grand élément. Il n'a donc « perdu » que dans une comparaison, celle avec le plus grand élément.

5. Proposez un nouvel algorithme de recherche du deuxième plus grand élément.
   Le deuxième plus grand élément est donc un des éléments qui ont été battus par le plus grand élément. L'algorithme a lieu en deux phases :
   (a) On recherche tout d'abord le plus grand élément suivant la méthode du tournoi.

(b) On obtient le deuxième plus grand élément en recherchant l'élément maximal parmi ceux qui ont été éliminés du tournoi lors d'une comparaison avec l'élément maximal.

Voir la figure 3.

6. Quelle est sa complexité en nombre de comparaisons ?

   La recherche de l'élément maximal coûte $n-1$ comparaisons, comme d'habitude. Ensuite la recherche du deuxième plus grand élément nous coûte $m-1$ comparaisons, où m est le nombre d'éléments à qui l'élément maximal a été comparé. Dans le pire cas[1]*, m est égal à la hauteur de l'arbre moins 1 (un arbre réduit à sa racine étant de hauteur un). Or un arbre binaire presque parfait à n feuilles est de hauteur $\lceil \log_2 n \rceil$. D'où la complexité :*

$$T(n) = n + \lceil \log_2 n \rceil - 2$$

   Note : cet algorithme est optimal.

1. Quand n n'est pas une puissance de deux, la complexité peut varier d'une comparaison suivant la place initiale dans l'arbre du maximum.