TD d'algorithmique avancée
Corrigé du TD 1 : recherche par rang

Jean-Michel Dischler et Frédéric Vivien

Recherche du maximum

1. Concevez un algorithme de recherche du maximum dans un ensemble à n éléments (vous disposez en tout et pour tout d'une fonction de comparaison).

   MAXIMUM(A)
   max $\leftarrow A[1]$
   pour $i \leftarrow 2$ **à** n **faire**
   si max ¡ $A[i]$ **alors** max $\leftarrow A[i]$
   renvoyer max

2. Quelle est la complexité de votre algorithme en nombre de comparaisons ?
   Réponse : $n-1$.

3. Montrez qu'il est optimal.
   Tout élément hormis le maximum doit avoir perdu une comparaison, sinon, on ne peut pas savoir qu'il n'est pas le maximum. Il y a $n-1$ *tels éléments. Tout algorithme de recherche du maximum doit donc faire au moins* $n-1$ *comparaisons.*

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 1.

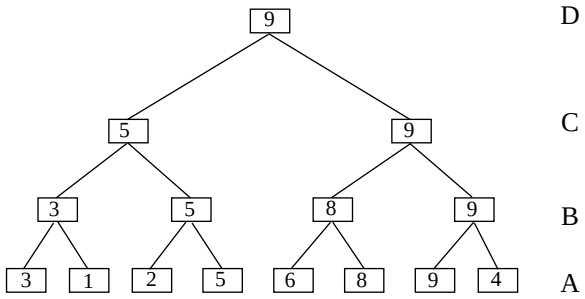

FIG. 1 – Méthode du tournoi pour la détermination du maximum : A : les éléments sont comparés par paires ; B : les plus grands éléments de la phase A sont comparés entre eux, par paires ; C : les éléments « vainqueurs » à la phase B sont comparés entre eux ; D : il ne reste plus qu'un élément, c'est l'élément maximal.

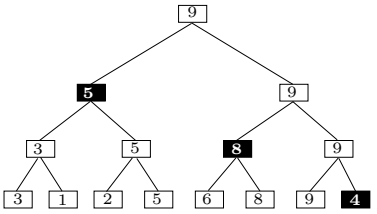

FIG. 2 – Le deuxième plus grand élément a nécessairement été battu par le plus grand élément (et que par lui). Il figure donc parmi les éléments comparés à l'élément maximal. Ces éléments apparaissent ici sur fond noir.

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 2.

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.

Recherche du maximum et du minimum

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

1. Proposez un algorithme naïf de recherche du maximum et du minimum d'un ensemble de n éléments.

[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.

MAXIMUM-ET-MINIMUM(A)

max $\leftarrow A[1]$
pour $i \leftarrow 2$ **à** n **faire**
  si max ¡ $A[i]$ **alors** max $\leftarrow A[i]$
min $\leftarrow A[1]$
pour $i \leftarrow 2$ **à** n **faire**
  si min ¿ $A[i]$ **alors** min $\leftarrow A[i]$
renvoyer max **et** min

2. Quelle est sa complexité en nombre de comparaisons ?
   Cet algorithme effectue $2n - 2$ *comparaisons.*
3. Proposez un algorithme plus efficace.
   Indication : dans une première phase les éléments sont comparés par paire.
   L'algorithme se décompose en trois phases :
   (a) *On compare par paire les éléments de l'ensemble. On met d'un côté les plus grands éléments (ici dans les cases paires du tableau) — c'est-à-dire les éléments qui sont sortis « vainqueurs » de leur comparaison — et de l'autre les plus petits (ici dans les cases impaires).*
   (b) *On recherche le minimum parmi tous les plus petits éléments (si on a un nombre impair d'éléments, il ne faut pas oublier le* n^e *élément qui n'a été comparé avec personne dans la première phase).*
   (c) *On recherche le maximum parmi tous les plus grands éléments.*

MAXIMUM-ET-MINIMUM(A)

Pour $i \leftarrow 1$ **à** $n - 1$ **faire par pas de** 2
  si $A[i] > A[i+1]$ **alors** échanger $A[i]$ et $A[i+1]$
min $\leftarrow A[1]$
Pour $i \leftarrow 3$ **à** n **faire par pas de** 2
  si $A[i] <$ min **alors** min $\leftarrow A[i]$
max $\leftarrow A[2]$
Pour $i \leftarrow 4$ **à** n **faire par pas de** 2
  si $A[i] >$ max **alors** max $\leftarrow A[i]$
si n est impair **alors si** $A[n] >$ max **alors** max $\leftarrow A[n]$
renvoyer max **et** min

4. Quelle est sa complexité en nombre de comparaisons ?
   Regardons indépendamment le coût des trois phases :
   (a) *On peut former* $\lfloor \frac{n}{2} \rfloor$ *paires, on effectue donc* $\lfloor \frac{n}{2} \rfloor$ *comparaisons.*
   (b) *Parmi* n *éléments on a* $\lceil \frac{n}{2} \rceil$ *éléments de rangs impairs. Dans cette phase on effectue donc* $\lceil \frac{n}{2} \rceil - 1$ *comparaisons.*
   (c) *Ici aussi on effectue aussi* $\lceil \frac{n}{2} \rceil - 1$ *comparaisons.*
   D'où une complexité totale en :

$$T(n) = \left\lfloor \frac{n}{2} \right\rfloor + 2\left(\left\lceil \frac{n}{2} \right\rceil - 1\right) = n + \left\lceil \frac{n}{2} \right\rceil - 2$$

5. Montrez que cet algorithme est optimal.
   Indication : on appelle *unité d'information* :
   – l'information « l'élément x ne peut pas être le plus grand élément » ;
   – l'information « l'élément x ne peut pas être le plus petit élément ».
   (a) Quel est le nombre minimal d'unités d'information qu'un algorithme de recherche du maximum et du minimum doit produire pour nous garantir la validité de son résultat ?

Pour être sûr qu'un élément est bien le maximum (respectivement le minimum) il faut que l'on sache que les $n-1$ autres ne peuvent pas être le maximum (respectivement le minimum) ce qui représente $n-1$ unités d'informations. L'algorithme doit donc produire au moins $2n-2$ unités d'information.

(b) Combien d'unités d'information sont produites par la comparaison de deux éléments (distinguez des cas, suivant que l'on a ou non des unités d'informations sur ces valeurs).

i. *Si on n'a d'unités d'information pour aucun des deux éléments, la comparaison nous fait gagner deux unités d'information : le plus petit des deux ne peut pas être le maximum, ni le plus grand le minimum.*

ii. *Si on a la même unité d'information pour les deux éléments (par exemple, aucun des deux ne peut être le plus grand), la comparaison nous procure une unité d'information (dans notre exemple, le plus grand des deux ne peut pas être le plus petit).*

iii. *Si on a une unité d'information pour chacun des deux éléments, mais des unités de type différent : si celui qui peut être le minimum est plus grand que celui qui peut être le maximum, on gagne deux unités d'information, et sinon zéro.*

iv. *Si on a une unité d'information pour un des éléments (par exemple, ne peut pas être le plus petit) et zéro pour l'autre, la comparaison peut nous donner une unité d'information (celui sans information est plus petit que l'autre dans notre exemple et il ne peut pas être le maximum) ou deux (celui sans information est plus grand, ne peut donc plus être le minimum, et l'autre ne peut plus être le maximum).*

v. *Si on a deux unités d'information pour un des éléments et zéro pour l'autre, la comparaison nous donne une unité d'information (par exemple, si l'élément sans information est plus grand, il ne peut plus être le minimum).*

(c) Concluez.

Il nous faut donc $2n-2$ unités d'information pour pouvoir conclure. Les comparaisons de type 5(b)i nous donnent toujours deux unités d'information chacune, or on peut au plus effectuer $\left\lfloor \frac{n}{2} \right\rfloor$ comparaisons de ce type (autant que l'on peut former de paires). Les autres comparaisons nous donnent, dans le pire des cas, une seule unité d'information chacune. Donc il nous faudra au moins effectuer $2n-2-2\left(\left\lfloor \frac{n}{2} \right\rfloor\right) = 2\left\lceil \frac{n}{2} \right\rceil - 2$ telles comparaisons. Dans le pire des cas, il nous faudra donc effectuer au moins :

$$\left\lfloor \frac{n}{2} \right\rfloor + 2\left\lceil \frac{n}{2} \right\rceil - 2 = n + \left\lceil \frac{n}{2} \right\rceil - 2$$

comparaisons, d'où l'optimalité de notre algorithme !