Université Paris 7 - Licence d'informatique - L2

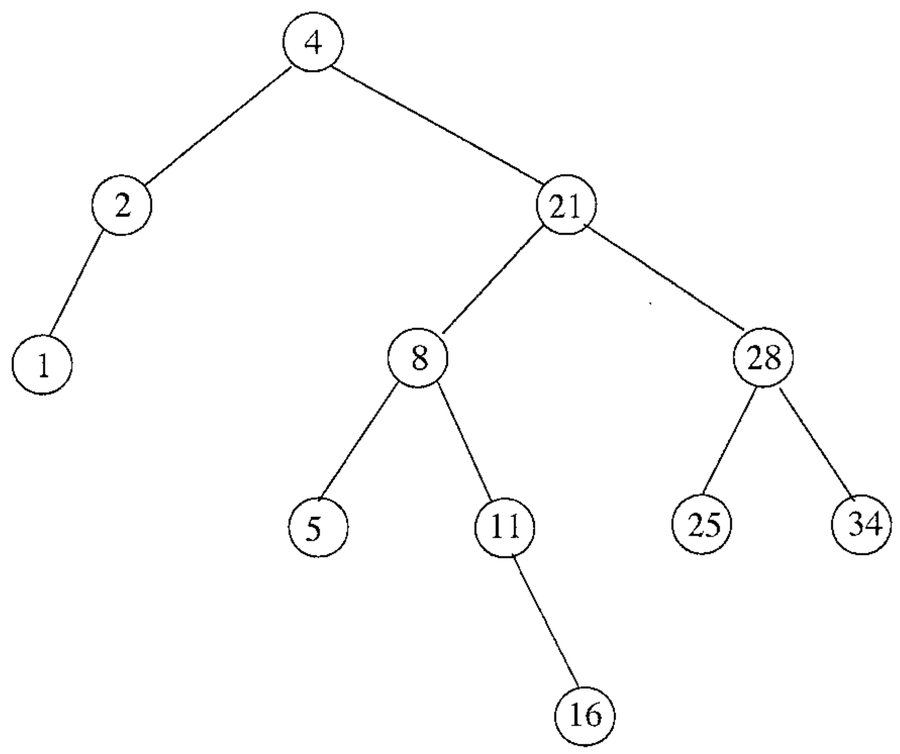
EA4 – Examen du 22 Mai 2013 - durée : 2h30

La qualité de la rédaction sera prise en compte dans la notation. Justifiez toutes vos réponses et expliquez les fondements de vos algorithmes en français avant de les rédiger en pseudo-code compréhensible et commenté. Le barème est donné seulement à titre indicatif.
Les algorithmes donnés sans commentaires ou sans explications ne seront pas pris en compte par le correcteur.
Documents autorisés : annotations personnelles ne dépassant pas une feuille A4.
Ne pas réécrire les algorithmes données en cours.

Exercice 1 (6 points).

Pour calculer le minimum de N entiers stockés dans un tableau on est obligé de consulter toutes les positions, ce qui implique une complexité temporelle en $O(N)$ dans le meilleur cas. On peut avoir une meilleure complexité en utilisant d'autres structures de données.

1. Donner un algorithme récursif $minimum(A : ABR)$ pour calculer l'adresse du minimum de N entiers stockés dans un **ABR**. Donner sa complexité temporelle dans les cas meilleur, pire et moyen (relativement à N) et justifier.
2. Donner une version itérative de cet algorithme.
3. Donner la règle qui permet de déterminer la position (l'adresse) du successeur d'un entier k (dont on connait l'adresse n) dans l'ABR. Vérifier votre règle sur l'ABR de la figure ci-dessous. On remarquera que :
   - le successeur de 21 est 25 ;
   - le successeur de 28 est 34 ;
   - le successeur de 2 est 4 ;
   - le successeur de 16 est 21.



4. Donner un algorithme $successeur(n : nœud)$ pour calculer le successeur d'un élément dont on connait l'adresse n dans l'ABR. Donner sa complexité temporelle dans les cas meilleur, pire et moyen (relativement à N) et justifier.

   Supposons que vous voulez lire n entiers et les **afficher** triés dans l'ordre croissant. Vous devez utiliser seulement les **AVL** comme structure de données avec les opérations indiquées.

5. Expliquer comment vous pouvez afficher ces n nombres avec une complexité temporelle en $O(n \log n)$ en utilisant les opérations suivantes : $minimum(A)$, $successeur(n)$, $insererAVL(k)$ et $rechercherAVL(k)$?
6. Même question avec les opérations : $minimum(A)$, $insererAVL(k)$, $supprimerAVL(k)$ et $rechercherAVL(k)$?
7. Même question avec les opérations : $insererAVL(k)$ et $parcoursInfixe(A)$?

Exercice 2 (4 points).

Soient les fonctions :

$$f_1(n) = n \quad f_2(n) = 2^n \quad f_3(n) = n^2 \quad f_4(n) = 2n$$
$$f_5(n) = n^n \quad f_6(n) = \log n \quad f_7(n) = n! \quad f_8(n) = n \log n$$

1. Pour chaque couple (i, j), $1 \leq i \leq 8$, $1 \leq j \leq 8$ dire si on a $f_i = O(f_j)$. Représentez vos réponses dans une table carrée 8x8.
2. Donnez une démonstration de vos réponses pour les 6 couples : $(2, 5)$, $(5, 2)$ $(2, 7)$, $(7, 2)$, $(5, 7)$ et $(7, 5)$.

Exercice 3 (5 points).

Si $T[1..n]$ est un tableau de n entiers positifs et négatifs, on appellera *sous-tableau de* T toute portion $T[i..j]$ du tableau pour $1 \leq i \leq j \leq n$. Si $i = j$, on a un sous-tableau de taille 1 qui contient un seul élément. On définit *poids* du sous-tableau $T[i..j]$ la somme $\sum_{k=i}^{j} T[k]$.

1. Donner un algorithme itératif qui pour un tableau T, retourne le poids maximal de tous ses sous-tableaux si ce poids est positif, et qui retourne 0 sinon.
2. Evaluer la complexité temporelle de votre algorithme et justifier.
3. Si on partage à sa moitié le tableau et on résout la question pour chacun des deux sous-tableaux ainsi obtenus, quelle(s) autre(s) information(s) on doit calculer pour répondre à la question pour le tableau tout entier ? Minimiser la quantité de cette information pour que l'algorithme demandé à la question suivante soit le plus efficace possible.
4. Donner un algorithme récursif basé sur la dichotomie qui calcule la même valeur que celui de la question 1. Evaluer la complexité temporelle de votre algorithme et justifier.