TD2 : Structures de données arborescentes

Arbres binaires

Exercice 1. *Opérations sur les arbres binaires*

Étant donné un arbre binaire A, donner le pseudo-code des fonctions suivantes :

1. SUPP(x, A) qui teste si le nœud x est une feuille de A qui n'est pas la racine et le supprime de A dans ce cas.
2. AJOUTFG(y, x) qui teste si le nœud y a un fils gauche vide et, dans ce cas, ajoute le nœud x à A comme fils gauche de y.

Exercice 2. *Algorithmes sur les arbres*

1. Écrire un algorithme qui affiche les valeurs de toutes les feuilles d'un arbre binaire A.
2. Écrire un algorithme qui calcule la hauteur d'un arbre binaire A.
3. Écrire un algorithme qui prend en entrée un arbre binaire A et un entier k et affiche les valeurs des nœuds de A qui sont supérieures ou égales à k.

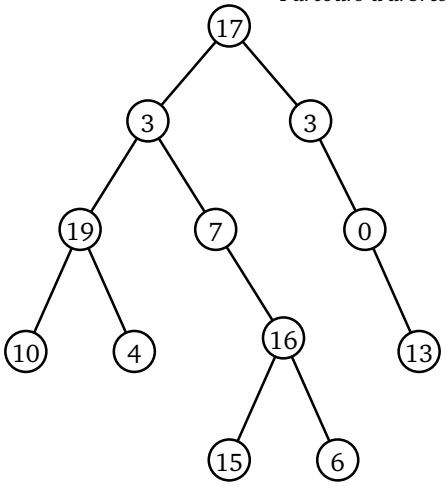
Exercice 3. *Parcours d'arbres*

Dans le cours, on a décrit le parcours infixe d'arbre. Deux autres types de parcours d'arbre existent :

— Dans un parcours *préfixe*, sont affichées récursivement la valeur du nœud courant, puis la valeur des nœuds du sous-arbre gauche, et enfin celle des nœuds du sous-arbre droit.

— Dans un parcours *postfixe*, sont affichées récursivement la valeur des nœuds du sous-arbre gauche, puis celle des nœuds du sous-arbre droit et enfin celle du nœud courant.

1. Écrire les algorithmes correspondants aux parcours préfixe et postfixe.
2. Appliquer ces algorithmes sur l'arbre ci-contre.



Exercice 4. *Rechercher*

✎ Donner une version récursive de l'algorithme de recherche d'un nœud de valeur k dans un arbre binaire.

Exercice 5. *Plus proche ancêtre*

Dans un arbre binaire A, un nœud x est un *ancêtre* d'un nœud y s'il existe une suite $x_0, \ldots, x_k$ avec $x_0 = y$, $x_k = x$ et $x_{i+1} = \text{père}(x_i)$ pour $i = 0, \ldots, k-1$. Pour deux nœuds u et v de A, le *plus proche ancêtre commun* (PPAC) de u et v est l'ancêtre de u et de v qui a la plus grande hauteur.

1. Dans l'arbre binaire représenté dans l'exercice 3, quels sont les ancêtres du nœud de valeur 4 ? Quels sont les ancêtres communs aux nœuds de valeur 19 et 15 ? Et leur plus proche ancêtre ?
2. Deux nœuds u et v d'un arbre binaire A ont-ils toujours un ancêtre commun ?
3. Écrire un algorithme qui étant donnés deux nœuds x et y d'un arbre binaire A décide si x est un ancêtre de y.

4. En déduire un algorithme qui calcule le plus proche ancêtre commun à deux nœuds u et v de A. Évaluer la complexité de votre algorithme.
5. Si ce n'est pas le cas, proposer un algorithme de complexité linéaire. *On pourra se servir d'une pile.*

Arbres binaires de recherche

Exercice 6. *Dessins d'ABR*

✎ Dessiner des arbres binaires de recherche de hauteur 6, 3 et 2 dont les nœuds ont le même ensemble de valeurs : $\{2, 4, 6, 8, 9, 11, 13\}$. Dans chaque cas, donner l'ordre d'insertion qui produit l'arbre dessiné.

Exercice 7. *Algorithmes sur les ABR*

1. Écrire un algorithme qui prend en entrée un arbre binaire de recherche A et un entier k et affiche les valeurs de nœuds de A qui sont supérieures ou égales à k. *Minimiser le nombre de tests effectués.*
2. On a vu en cours qu'un parcours infixe d'un ABR à n nœuds affichait les valeurs des nœuds dans l'ordre croissant en temps $O(n)$. Peut-il exister un algorithme qui, à partir de n valeurs, construirait un ABR contenant ces n valeurs et fonctionnant en temps $O(n)$?

Exercice 8. *Successeur*

Le but de l'exercice est d'écrire de manière détaillée la preuve de validité de l'algorithme SUCCESSEUR vu en cours. On se donne A un arbre binaire de recherche et on suppose que toutes les valeurs sont distinctes deux-à-deux.

1. Soit x un nœud de A et A_x le sous-arbre de A enraciné en x. Soient aussi y et z deux nœuds de A_x avec $\text{val}(y) < \text{val}(z)$. Montrer que pour tout nœud t de $A \backslash A_x$, soit $\text{val}(t) < \text{val}(y)$, soit $\text{val}(t) > \text{val}(z)$. *On pourra considérer un plus proche ancêtre commun de x et t.*
2. En déduire que si un nœud x de A a un fils droit, alors le successeur de x est le nœud de valeur minimale de $\text{saD}(x)$.
3. On considère maintenant un nœud x sans fils droit.
   (a) Supposons d'abord que tout ancêtre de x est fils droit de son père. Montrer alors que x est le nœud de valeur maximale dans A.

   On se place maintenant dans le cas contraire, x a donc un ancêtre qui est le fils gauche de son père. On note q un tel ancêtre de hauteur maximale dans A, et p son père. On note A_p et A_q les sous-arbres de A enracinés en p et q respectivement.
   (b) Montrer que x a la valeur maximale parmi les nœuds de A_q.
   (c) À l'aide de la question 1, en déduire que p est le successeur de x.

Tas

Exercice 9.

1. Le tableau $[25, 18, 13, 7, 12, 10, 6, 7, 9]$ est-il un tas ? *Dessiner l'arbre quasi-complet représenté par le tableau, puis indiquer si cet arbre binaire est un tas.*
2. Monter que si on utilise la réprentation en tableau pour un tas à n éléments alors les feuilles du tas sont exactement les nœuds ayant pour numéro d'ordre $\lfloor n/2 \rfloor, \lfloor n/2 \rfloor + 1, \ldots, n-1$.
3. Donner un exemple d'arbre binaire quasi-complet à n nœuds dont l'entassement sur un nœud bien choisi demande $\Omega(\log n)$ appels récursifs (on veut un tel exemple pour n quelconque).

Exercice 10. *Créer un tas*

1. En se basant sur l'algorithme TRITAS, écrire un algorithme CRÉERTAS qui prend en entrée un tableau T et réordonne ses éléments pour que T représente un tas.

2. Montrer que la complexité de CRÉERTAS est $O(n \log n)$.
3. Exécuter l'algorithme CRÉERTAS sur le tableau $[1, 2, 3, 4, 5, 6, 7]$.
4. Montrer que sur tout tableau trié par ordre décroissant, CRÉERTAS ne fait aucun entassement.
5. *(bonus)* On veut montrer que CRÉERTAS a une complexité $O(n)$.
   (a) Pourquoi cela ne contredit-il pas la question 2 ?
   (b) Montrer que pour $h < \log n$, CRÉERTAS appelle ENTASSER $\lceil n/2^{h+1} \rceil$ fois sur des tas de hauteur h.
   (c) En déduire que la complexité de CRÉERTAS est $O\left(\sum_{h=0}^{\lfloor \log n \rfloor} nh/2^h\right)$.
   (d) Montrer que $\sum_{h=0}^{+\infty} h/2^h = 2$.
   (e) En déduire le résultat.

Exercice 11. *Implémenter une file de priorité*

Donner le pseudo-code des primitives nécessaires pour une file de priorité afin
1. d'obtenir l'élément le plus prioritaire,
2. de retirer cet élément,
3. de baisser la priorité d'un élément (au pire jusqu'à 0),
4. d'augmenter cette priorité,
5. d'insérer un nouvel élément, et
6. de supprimer un élément.

Indiquer pour chacune de ses primitives sa complexité. On pourra appeler des fonctions existantes sur les tas.

Exercice 12. *Fusion !*

On dispose de k listes $L_1, \ldots, L_k$ triées en ordre croissant. Pour $i = 1, \ldots, k$ on a accès au dernier élément de la liste L_i par FIN(L_i) et à sa valeur par VAL(FIN(L_i)) ; on peut tester si la liste est vide par ESTVIDE(L_i) et on peut supprimer le dernier élément de L_i par SUPFIN(L_i). Toutes ces opérations s'effectuent en temps $O(1)$. Enfin, le nombre total d'éléments dans ces k listes est noté n. On veut fusionner ces k listes en une seule liste L triée par ordre croissant.
1. Comment implémenter cette fusion en un temps $O(nk)$?
2. Comment améliorer l'implémentation précédente pour avoir une complexité en $O(n \log k)$. *On pourra utiliser un tas.*

Exercice 13. *Tri par tas en place*

On souhaite implémenter le tri par tas en place, c'est-à-dire que l'algorithme modifie le tableau passé en entrée et ne crée pas de nouveau tableau.
1. Quelle étape du tri par tas nécessite d'être modifiée ?

On modifie légèrement la représentation d'un tas, en ajoutant une *taille* n : le tableau qui représente le tas peut être de taille quelconque ($\geq n$), et le tas est représenté par les n premières cases du tableau. Par exemple, la donnée du tableau $[13, 12, 6, 8, 4, 5, 14, 15, 17]$ et de la taille 6 représente le tas contenant les six éléments 13, 12, 6, 8, 4 et 5, les trois derniers éléments étant *ignorés*. *On suppose dans la suite qu'on peut accéder à la taille d'un tas T par la fonction* taille(T)*, et à la taille du tableau par* $n(T)$*. Dans notre exemple,* taille(T) $= 6$ *et* $n(T) = 9$.
2. En s'inspirant de l'algorithme SUPPRIMER, écrire un algorithme SUPPRIMERMAX qui supprime l'élément maximum d'un tas et le met à la place libre du tableau. *Par exemple, si on applique* SUPPRIMERMAX *au tas de l'exemple, le tableau T devient* $[12, 8, 6, 4, 13, 14, 15, 17]$.
3. En déduire un alorithme TRIENPLACE qui trie un tableau en place.
4. Quelle est la complexité de l'algorithme obtenu ?