Travaux dirigés 3: Arbres

Rappels de cours *Soit T un arbre. Si tout élément $x \in T$ a au-plus deux fils, un* fils gauche $g(x)$ *et un* fils droit $d(x)$*, alors T est un* arbre binaire *(on note $g(x)$ ou $d(x) = NIL$ si le fils gauche ou droit n'existe pas, respectivement). Le* sous-arbre T_x de T enraciné en $x \in T$ *est l'arbre de racine x induit par tous les descendants de x dans T (si x n'existe pas, alors $T_x = \emptyset$). L'arbre T est un* arbre binaire de recherche *si pour tout $x \in T$, $y \in T_{g(x)}$ et $z \in T_{d(x)}$, on a $y \leq x \leq z$ (où $\leq$ est une relation d'ordre totale sur T).*

Exercice 1 Ecrire une procédure qui retourne l'élément minimum d'un arbre binaire de recherche, démontrer qu'elle est valide et donner sa complexité. Même question avec le maximum.

Exercice 2 Soit $T \subseteq \mathbb{N}$ un arbre binaire de recherche et $a \in \mathbb{N}$.
1) Ecrire une procédure qui teste si $a \in T$, démontrer qu'elle est valide et donner sa complexité.
2) Supposons que $a \in T$. Soit $b = \min\{b \in T : a \leq b\}$ le successeur de a dans T. Ecrire une procédure retournant b, démontrer qu'elle est valide et donner sa complexité.
3) Mêmes questions avec le prédécesseur.

Algorithme 1: PARCOURS-INFIXE(x)

si $x \neq NIL$ **alors**
    PARCOURS-INFIXE($g(x)$)
    Mettre x en queue de la liste L
    PARCOURS-INFIXE($d(x)$)
fin

Exercice 3 $\star$ Soit $L = (x_1, \ldots, x_n)$ la liste construite par un parcours infixe sur la racine d'un arbre binaire (**Algorithm 1**). Montrer que L est triée dans l'ordre croissant.

Exercice 4 Soit A un ensemble de n entiers (différents). Peut-on définir plusieurs arbres binaires de recherche sur A? Soit T un arbre à n éléments. Montrer qu'il existe une unique bijection $\phi : T \to A$ telle que T soit un arbre binaire de recherche pour l'ordre (strict) défini sur T par $x \leq y \Leftrightarrow \phi(x) \leq \phi(y)$.

Rappels de cours *Un tableau A est une suite finie de longueur[A] éléments dont le $i$ème élément est noté $A[i]$. Tout tableau A définit une structure d'arbre de racine $A[1]$ par la fonction* père $p(i) = \lfloor i/2 \rfloor$ *définie pour tout indice i de 2 à longueur[A]. Cet arbre est* binaire *car chaque élément non-feuille a alors un* fils gauche $g(i) = 2i$ *ou/et un* fils droit $d(i) = 2i + 1$*. Si l'ensemble des éléments de A est totalement ordonné et si $A[p(i)] \geq A[i]$, alors A est un* tas*. On peut réordonner tout tableau de façon à en faire un tas en appliquant* **Algorithm 2**.

Algorithme 2: CONSTRUIRE-TAS(A)

$taille[A] \leftarrow longueur[A]$
for $i \leftarrow \lfloor longueur[A]/2 \rfloor$ à (décroître) 1 **do**
  ENTASSER(A,i)
end for

Algorithme 3: ENTASSER(A,i)

$l \leftarrow 2i$
$r \leftarrow 2i + 1$
if $l < taille[A]$ et $A[l] > A[i]$ **then**
  $max \leftarrow l$
else
  $max \leftarrow i$
end if
if $r \leq taille[A]$ et $A[r] > A[max]$ **then**
  $max \leftarrow r$
end if
if $max \neq i$ **then**
  échanger $A[i]$ et $A[max]$
  ENTASSER(A,max)
end if

Exercice 5 ⋆ *Tas.*

Soit T un tas. On note n_T le nombre de sommets, h_T la hauteur, et $n_T(h)$ le nombre de sommets à hauteur h de T. (La hauteur d'un sommet est sa distance max vers une feuille, celle d'un tas ou d'un arbre est celle de sa racine.) On note T_g (resp. T_d) le sous-tas enraciné à gauche (resp. droite) de la racine de T.

1) Montrer par induction (i.e. récurrence) sur h que si T est complet (si $n_T = 2^{h_T} - 1$), alors $n_T(h) = 2^{h_T - h}$.

2) Montrer que si T_g est complet, alors $n_{T_g}(h) = 2^{h_T - h - 1}$ et $n_{T_d} \leq 2^{h_T} - 1$.

3) Montrer que si T_d est complet, alors $n_{T_d}(h) = 2^{h_T - h - 2}$ et $n_{T_g} \leq 2^{h_T} - 1$.

4) En déduire que $n_T(h) \leq \lceil \frac{n_T}{2^{h+1}} \rceil$.

5) En déduire que l'on peut contruire un tas en $O(n)$. *(Indications : Montrer qu'on entasse un élément à hauteur h en $O(h)$ (***Algorithm 3***). Pour calculer la valeur de la série $\sum_{k=0}^{\infty} kx^k$ (pour $x = 1/2$), calculer $\sum_{k=0}^{\infty} x^k$ puis dériver les deux termes de l'égalité.*

Algorithme 4: TRIER-TAS(A)

```
CONSTRUIRE-TAS(A)
for i ← longueur[A] à 2 do
    échanger A[1] et A[i]
    taille[A] ← taille[A] − 1
    ENTASSER(A,1)
end for
```

Exercice 6 Donner la complexité du tri par tas (**Algorithm 4**).