Simulation mutuelle : Arbres binaires comptage et parcours

Objectifs

À la fin de cette séance, vous devriez être capable de :

— parcourir un arbre binaire dans tous les sens ;

Exercice 1 : Réflexes

On considère des arbres binaires dont les noeuds ont 0, 1 ou 2 fils.
On note h, n et f respectivement la hauteur, le nombre de noeuds et le nombre de feuilles d'un arbre.

Un arbre de hauteur h est dit *équilibré* si :

— toutes ses feuilles sont aux niveaux h ou $h-1$ (si $h \geq 1$),

— s'il y a des noeuds unaires (des noeuds ayant exactement 1 fils), ils sont au niveau $h-1$ (si $h \geq 1$).

Dans le cas d'un arbre équilibré, quel est l'ordre de grandeur (justifiez) :

1. de h en fonction de n ?
2. de n en fonction de h ?
3. de f en fonction de n ?

Correction de l'exercice 1

Tous les nœuds unaires étant au niveau $h-1$, tous les niveaux $\leq h-1$ sont complets, ce qui fait déjà :

$$1 + 2 + \ldots + 2^{h-1} = 2^h - 1$$

nœuds.

Ensuite, au niveau h il y a entre une feuille (on ne peut pas faire moins sinon l'arbre n'est pas de hauteur h) et 2^h feuilles (cas du niveau h complet).

D'où $2^h \leq n \leq 2^{h+1} - 1$. On en déduit facilement les ordres de grandeur suivants :

1. $n = \Theta(2^h)$
2. $h = \Theta(\log n)$
3. Enfin il y a entre 1 et 2 feuilles par nœud de niveau $h-1$, d'où $2^{h-1} \leq f \leq 2^h$. À cause des ordres de grandeur précédents cela donne :
   $f = \Theta(n)$ (les feuilles pèsent aussi lourd que le reste de l'arbre).

Exercice 2 : Échauffement

Écrivez 3 fonctions qui calculent respectivement la hauteur, le nombre de noeuds et le nombre de feuilles d'un arbre binaire.
Pour écrire ces fonctions, vous avez besoin de manipuler un type abstrait `arbre`. Vous préciserez les primitives d'accès nécessaires.

Exercice 3 : Feuille haute

Pour un arbre binaire A, écrivez une fonction qui calcule la hauteur de la feuille la plus haute (la feuille la plus proche de la racine)

1. par un parcours en profondeur d'abord récursif,
2. par un parcours en largeur d'abord.
3. Donnez les principes permettant la preuve (de correction totale) de ces deux algorithmes (récursif et itératif).

Correction de l'exercice 3

FeuilleHaute(a)
if *a est une feuille* **then**
1. Renvoyer 0

else
Renvoyer 1 + minimum(FeuilleHaute(FilsGauche(a)), FeuilleHaute(FilsDroit(a)))

2. Dans un parcours en largeur, la première feuille rencontrée est la moins profonde.

FeuilleHauteLargeur(a)
F = FileVide de couples (noeud, entier)
F = Enfiler (a,0) dans F
$x = a$
$h = 0$
while *x n'est pas une feuille* **do**
  (x,h) = Tête(F)
  F = Défiler(F)
  if *x n'est pas vide* **then**
    F = Enfiler (FilsGauche(x), h+1) dans F
    F = Enfiler (FilsDroit(x), h+1) dans F

Renvoyer h

3. a) arrêt :
   - version récursive : les appels récursifs sont faits sur des noeuds dont la distance aux feuilles décroit strictement (de 1). On arrivera donc forcément à une feuille et à l'arrêt de la récursivité. (car les arbres dont on parle sont finis).
   - version itérative : c'est un parcours par niveau de l'arbre qui se termine lorsqu'on rencontre la première feuille.

   b) correction :
   - version récursive : la procédure est correcte dans le cas de base (avec la convention qu'un arbre réduit à une feuille est de hauteur 0). En supposant que les appels récursifs sont corrects (ils renvoient la hauteur de la feuille la plus haute de chacun des sous-arbres de a), alors le calcul $1 + min(\dots,\dots)$ donne bien la hauteur de la feuille la plus haute de a.
   - version itérative : on enfile les couples (noeud, hauteur du noeud) par hauteur croissante. La première feuille défilée est la plus haute, et on récupère sa hauteur.

Exercice 4 : Feuilles à la profondeur p

Pour un arbre binaire A et une profondeur p, écrivez une fonction qui calcule le nombre de feuilles à la profondeur p

1. par un parcours en profondeur d'abord récursif,
2. par un parcours en largeur d'abord.

Pour chacun de ces deux algorithmes (récursif et itératif), que pouvez vous dire de son coût ? Y-a-t-il un algorithme moins coûteux que l'autre ?

Correction de l'exercice 4

1.
```
NbFeuillesProf(a, p)
if p = 0 then
    if a est une feuille then
        Renvoyer 1
    else
        Renvoyer 0
else
    if a est vide then
        Renvoyer 0
    else
        Renvoyer NbFeuillesProf(FilsGauche(a), p-1) + NbFeuillesProf(FilsGauche(a), p-1)
```

2.
```
NbFeuillesProfLargeur( a, p )
F = FileVide de couples (noeud, entier)
F = Enfiler (a,p) dans F
n = 0
while F n'est pas vide do
    (x,p) = Tête(F)
    F = Défiler(F)
    if p = 0 then
        n=n+1
    if x n'est pas vide then
        F = Enfiler (FilsGauche(x), p-1) dans F
        F = Enfiler (FilsDroit(x), p-1) dans F
Renvoyer n
```