Année 2019-2020

Portail Mathématiques, Informatique
Licence deuxième année (L2 Info, L2 maths, L2 MIASHS)

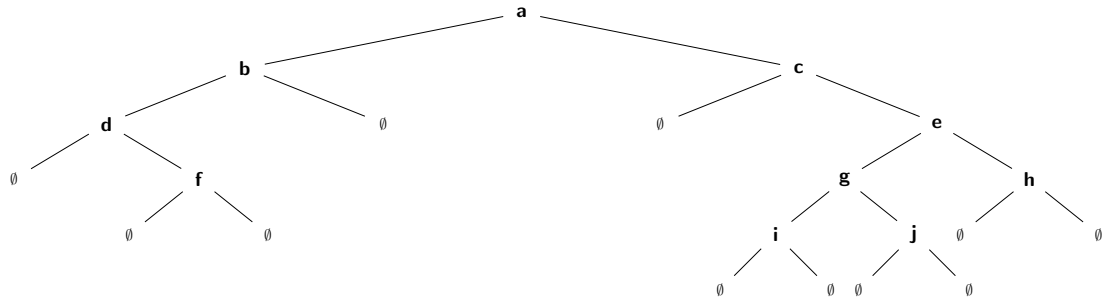
Algorithmique et structures de donnéesTD 5 et 6 – Arbres binaires

Exercice 1. **Parcours des nœuds d'un arbre binaire**

Question 1. Donnez l'ordre de visite des nœuds de l'arbre A de la figure 1.
On considérera les quatre parcours suivants :

1. parcours préfixe,
2. parcours infixe,
3. parcours postfixe (ou suffixe),
4. parcours en largeur.

FIGURE 1 – Exemple d'arbre binaire A



On rappelle la structure **nœud** chaînée d'un arbre binaire :

Structure nœud {
valeur : entier # ou caractère, ou chaîne de caractères...
gauche : pointeur sur nœud
droit : pointeur sur nœud
}

Comme on l'a vu en cours, un arbre binaire est représenté par un *pointeur sur sa racine* :
type arbreBinaire = pointeur sur nœud

Question 2. Dessinez la représentation chaînée de l'arbre A de la figure 1 avec cette structure.

Question 3. Pour chacun des quatre ordres de parcours rappelés à la question 1, écrivez une procédure qui prend en argument un arbre binaire A quelconque et affiche tous ses nœuds dans cet ordre.

Question 4. Donnez une procédure récursive qui renvoie le nombre de nœuds d'un arbre binaire.

Modifiez cette procédure pour retourner le nombre de nœuds internes.

Question 5. On considère la procédure récursive suivante qui calcule le nombre de feuilles d'un arbre binaire

```
nombreFeuilles1(A : arbreBinaire) : entier
   si A = None alors
      retourner 0
   sinon
      si A->gauche = None et A->droit = None alors
         retourner 1
      sinon retourner nombreFeuilles1(A->gauche) + nombreFeuilles1(A->droit)
```

Calculez le nombre d'appels récursifs de cette procédure en fonction du nombre de nœuds internes.

Question 6. Écrivez une procédure récursive **nombreFeuilles2(A : arbreBinaire) : entier** qui prend en argument un arbre binaire A, *supposé non vide*, et retourne son nombre de feuilles.

Indication : Pour la procédure **nombreFeuilles2**, distinguez les quatre cas suivants :

a) l'arbre est réduit à un seul nœud, c'est donc une feuille
b) le sous-arbre gauche est vide mais pas le sous-arbre droit
c) le sous-arbre droit est vide mais pas le sous-arbre gauche
d) les deux sous-arbres gauche et droit ne sont pas vides

Vous essaierez de minimiser le nombre de tests.
Quel est ce nombre minimal de tests ?

Question 7. Calculez le nombre d'appels récursifs de la procédure **nombreFeuilles2** en fonction du nombre de nœuds.

Indication : il faut regarder combien de fois un nœud est appelé.

On admettra qu'un abre localement complet A vérifie

$$N(A) = 2N_i(A) + 1.$$

Montrez que les deux procédures nécessitent (presque) le même nombre d'appels récursifs sur les arbres localement complets.

Proposez d'autres arbres binaires pour lesquels nous n'avons pas du tout le même nombre d'appels.

Question 8. Écrivez une procédure **copieArbre(A : arbreBinaire) : arbreBinaire** qui prend en argument un arbre binaire A et retourne une *copie* de celui-ci, c'est-à-dire, un arbre identique à A mais qui n'a aucun nœud commun avec A.

Question 9. * Écrivez une procédure *itérative* **affichePréfixeItératif(A : arbreBinaire)** qui affiche les nœuds d'un arbre binaire A dans l'ordre préfixe.
Indication : Vous utiliserez une *pile*. Celle-ci mémorisera les nœuds en attente de traitement (plus précisément, des pointeurs sur ces nœuds). On initialise la pile avec la racine de l'arbre et ensuite on dépile le nœud placé au sommet de la pile et on empile ses fils gauche et droit s'ils existent. Vous ferez attention d'empiler les fils gauche et droit dans le bon ordre.

Avant d'écrire la procédure, prenez le temps d'exécuter l'algorithme sur un arbre contenant quelques nœuds.

Exercice 2. **Profondeur des nœuds et longueur de cheminement**

Question 1. Rappelez ce qu'est la *profondeur* (ou *niveau*) d'un nœud dans un arbre. Quelle est la profondeur de la racine ? Donnez, pour l'arbre de la figure 1, les nœuds de niveau 0, ceux de niveau 1, ceux de niveau 2, ceux de niveau 3, ceux de niveau 4, enfin, ceux de niveau 5.

Question 2. * Écrivez une procédure récursive **afficheNiveau(A : arbreBinaire, k : entier)** qui affiche tous les nœuds de profondeur/niveau k d'un arbre binaire A.
Indication : k est la longueur du chemin à parcourir pour atteindre un nœud de profonfeur k. Lorsque k vaut 0 c'est que nous sommes sur un nœud de profondeur k. Comment évolue la longueur du chemin lorsque nous allons dans les sous-arbres gauche et droit ?

Question 3. La **longueur de cheminement** d'un arbre binaire est la somme des profondeurs de tous les nœuds de cet arbre. Calculez la longueur de cheminement de l'arbre A de la figure 1.

Question 4. * Écrivez une procédure **LC(A : arbreBinaire) : entier** qui prend en argument un arbre et retourne sa longueur de cheminement. Vous pourrez utiliser une procédure auxiliaire **LCaux(B : arbreBinaire, p : entier) : entier** qui calcule pour un sous-arbre B de A la somme des profondeurs à partir de la racine de B qui est de profondeur p dans A.

Question 5. La **longueur de cheminement externe** d'un arbre est la somme des profondeurs de toutes les feuilles de l'arbre. Autrement dit, c'est la somme des longueurs de toutes les branches de l'arbre.
Calculez la longueur de cheminement externe de l'arbre A de la figure 1.

Question 6. * Écrivez une procédure récursive **LCE(A : arbreBinaire) : entier** qui retourne la longueur de cheminement externe d'un arbre binaire A. Comme précédemment, vous pouvez utiliser une procédure auxiliaire **LCEaux(B : arbreBinaire, p : entier) : entier** qui calcule pour un sous-arbre B de A la somme des profondeurs des feuilles à partir de la racine de B qui est de profondeur p dans A.

Calculez le nombre de sommes et le nombre d'appels récursifs en fonction du nombre de nœuds internes.

Question 7. ** Écrivez une procédure itérative **LCEIt(A : arbreBinaire) : entier** qui retourne la longueur de cheminement externe d'un arbre binaire A.

Indication : la procédure doit effectuer un *parcours en largeur* de l'arbre A à l'aide d'une *file*. En parcourant (par le parcours en largeur) chaque noeud, en fait un pointeur sur ce nœud, on mettra dans la file le couple formé par ce pointeur et la profondeur/niveau du nœud dans A. Pour calculer la longueur de cheminement externe, on utilisera un compteur initialisé à 0 et, à chaque fois qu'on rencontrera une feuille, on additionnera à ce compteur la profondeur de cette feuille dans l'arbre.

Exercice 3. **Expression arithmétique**Une *expression arithmétique* (avec les opérateurs $+$ et $*$) est une chaîne de caractères définie par le schéma d'induction suivant :

(a) une lettre de l'alphabet $\mathcal{A} = \{a, \ldots, z\}$ est une expression arithmétique ;

(b) si E_1 et E_2 sont des expressions arithmétiques, alors $(E_1 + E_2)$ et $(E_1 * E_2)$ sont des expressions arithmétiques.

Une expression arithmétique peut être représentée par un arbre binaire non vide dont chaque nœud interne est un opérateur $+$ ou $*$ et chaque feuille est une lettre de $\mathcal{A}$.

Cependant, l'arbre représentant une expression arithmétique possède une structure particulière : chaque nœud qui n'est pas une feuille possède *à la fois* un fils gauche et un fils droit, il s'agit donc d'*arbre binaire localement complet*.

Question 1. Dessinez l'arbre de l'expression arithmétique $(((a * c) + (d * b)) * (b + a))$.

Question 2. Écrivez une procédure récursive **valeurExp(E : arbreBinaire, V : fonction) : entier** qui prend en argument un arbre binaire E représentant une expression arithmétique et une fonction de valuation $V : \mathcal{A} \to \mathbb{N}$ et qui retourne la valeur de l'expression arithmétique pour V (valeur que l'on notera $V(E)$).

Question 3. * Écrivez une procédure récursive **arbreExp() : arbreBinaire** qui, à partir d'une expression arithmétique lue au clavier, caractère par caractère, retourne l'arbre de l'expression. On pourra, au choix, faire un contrôle d'erreur ou supposer que l'expression lue au clavier est toujours une expression arithmétique correcte.