2I003 Arbres binaires 7

Exercice 1 – Nombres de nœuds d'un arbre binaire

Soit T un arbre binaire. On note $h(T)$ sa hauteur, $n(T)$ son nombre de nœuds, $f(T)$ son nombre de feuilles, $n_1(T)$ son nombre de nœuds à 1 fils, $n_2(T)$ son nombre de nœuds à 2 fils, $ni(T)$ son nombre de nœuds internes et $a(T)$ son nombre d'arêtes.

Question 1

1. Dessiner un arbre binaire T_0 tel que $h(T_0) = 4$, $f(T_0) = 3$, $n_1(T_0) = 2$ et $n_2(T_0) = 2$.
2. Montrer par induction structurelle que $h(T) \leq n(T) \leq 2^{h(T)} - 1$, pour tout arbre binaire T.
3. Montrer par induction structurelle que $f(T) = n_2(T) + 1$, pour tout arbre binaire non vide T.
   En déduire que $n(T) = n_1(T) + 2n_2(T) + 1$ et que $f(T) \leq ni(T) + 1$, pour tout arbre binaire non vide T.
4. Montrer par induction structurelle que $n(T) = a(T) + 1$, pour tout arbre binaire non vide T.

Exercice 2 – Complexité de fonctions définies sur les arbres

Question 1

1. Pour un arbre binaire T de taille n, montrer que :
   (a) la complexité de `ABtaille(T)` est en $\Theta(n)$
   (b) la complexité de `ABhauteur(T)` est en $\Theta(n)$.
2. Pour deux arbres binaires T_1 et T_2 de tailles n_1 et n_2, montrer que la complexité de `ABegal(T1, T2)` :
   (a) est en $\Theta(1)$ dans le meilleur cas,
   (b) est en $\Theta(\min(n_1, n_2))$ dans le pire cas.

Exercice 3 – Parcours d'un arbre binaire

Question 1

1. Dessiner un arbre binaire dont le parcours préfixe est $(5, 3, 7, 2, 8, 1, 6, 9, 4)$. La solution est-elle unique ?
2. Dessiner un arbre binaire dont le parcours infixe est $(3, 7, 5, 1, 8, 6, 2, 4, 9)$. La solution est-elle unique ?
3. Dessiner un arbre binaire dont le parcours préfixe est $(5, 3, 7, 2, 8, 1, 6, 9, 4)$ et le parcours infixe est $(3, 7, 5, 1, 8, 6, 2, 4, 9)$. La solution est-elle unique ?
4. Existe-t-il un arbre binaire dont le parcours préfixe est $(5, 3, 7, 2, 8, 1, 6, 9, 4)$ et le parcours infixe est $(3, 7, 5, 4, 9, 2, 1, 8, 6)$?

Question 2

On suppose que P et I sont des listes telles que :

(pi1) les éléments de P sont deux à deux distincts,

(pi2) P et I ont la même longueur et sont composées des mêmes éléments.

1. Montrer que, s'il existe un arbre binaire dont le parcours préfixe est P et le parcours infixe est I alors cet arbre est unique.
2. Le résultat précédent reste-il vrai si l'on ne suppose pas que les éléments de P sont deux à deux distincts ?

Exercice 4 – Ombre d'un arbre

L'*ombre* d'un arbre binaire T est l'arbre obtenu en remplaçant chaque étiquette de T par $\bullet$.
En voici la définition inductive :

$$\text{ombre}(T) = \begin{cases} \emptyset & \text{si } T = \emptyset \\ (\bullet, \text{ombre}(G), \text{ombre}(D)) & \text{si } T = (x, G, D) \end{cases}$$

Question 1

1. Montrer que, pour tout arbre binaire T, l'ombre de T a même taille et même hauteur que T.
2. Si T est un arbre binaire de taille n, quel est le parcours préfixe de l'ombre de T ? son parcours infixe ? son parcours suffixe ?
3. Soit T_1 et T_2 des arbres binaires. Montrer que les propriétés suivantes sont équivalentes :
   (i) T_1 et T_2 sont égaux
   (ii) T_1 et T_2 ont même ombre et même parcours préfixe
   (iii) T_1 et T_2 ont même ombre et même parcours infixe
   (iv) T_1 et T_2 ont même ombre et même parcours suffixe.
4. Soit E un ensemble ayant p éléments. Calculer le nombre d'arbres binaires étiquetés sur E qui ont la même ombre et la même taille n.

Exercice 5 – Recherche dans un arbre binaire

On rappelle la définition de la fonction de recherche vue en cours :

```
def ABcherche(x, T):
    if estABvide(T):
        return False
    if x == T.clef:
        return True
    if ABcherche(x, T.gauche):
        return True
    return ABcherche(x, T.droit)
```

Question 1

Montrer que `ABcherche(x,T)` se termine et renvoie la valeur `True` si x est dans T et la valeur `False` sinon.

Question 2

Calculer la complexité de `ABcherche(x, T)` dans le meilleur cas et dans le pire cas.

Exercice 6 – Arbres stricts

Un *arbre binaire strict* est un arbre binaire non vide dans lequel tout nœud a 0 ou 2 fils.

Question 1
Soit T arbre binaire strict T. Montrer que $f(T) = ni(T) + 1$. En déduire que $n(T) = 2ni(T) + 1 = 2f(T) - 1$. Peut-on construire un arbre strict de taille 100 ?

Question 2
Soit T arbre binaire tel que $f(T) = ni(T) + 1$, Montrer que T est strict.
Voici une définition inductive des arbres stricts.
Un arbre binaire strict T étiqueté sur un ensemble E est :
- soit une feuille $(x, \emptyset, \emptyset)$,
- soit un triplet (x, G, D) où $x \in E$ et G, D sont des arbres binaires stricts étiquetés sur E.

Question 3
Soit T un arbre binaire strict étiqueté sur $\{0, 1\}$, tel que tout nœud interne a pour étiquette 0 et toute feuille a pour étiquette 1. Montrer que le parcours infixe de T est une liste L commençant par 1, se terminant par 1 et composée alternativement de 1 et de 0.

Exercice 7 – Peignes

Un *peigne gauche* est un arbre binaire non vide dont tous les sous-arbres droits sont réduits à des feuilles.
En voici une définition inductive.
Un peigne gauche T étiqueté sur un ensemble E est :
- soit une feuille $(x, \emptyset, \emptyset)$,
- soit un triplet $(x, G, (y, \emptyset, \emptyset))$ où $x \in E$, $y \in E$ et G peigne gauche étiqueté sur E.

Question 1
Montrer que tout peigne gauche est un arbre strict.

Question 2
Montrer que, pour tout peigne gauche T :
1. $f(T) = ni(T) + 1$ et $n(T) = 2ni(T) + 1$
2. $f(T) = h(T)$.

Question 3
Soit T un peigne gauche étiqueté sur $\{0, 1\}$, tel que tout nœud interne a pour étiquette 0 et toute feuille a pour étiquette 1. Montrer que le parcours préfixe de T est une liste L composée de $h(T) - 1$ éléments égaux à 0 suivis de $h(T)$ éléments égaux à 1.

Question 4
Donner la définition d'un *peigne droit* et reprendre les questions précédentes.

Question 5
Un *peigne* T étiqueté sur un ensemble E est :
- soit une feuille $(x, \emptyset, \emptyset)$,
- soit un triplet $(x, G, (y, \emptyset, \emptyset))$ où $x \in E$, $y \in E$ et G peigne étiqueté sur E
- soit un triplet $(x, (y, \emptyset, \emptyset), D)$ où $x \in E$, $y \in E$ et D peigne étiqueté sur E.

Tout peigne est-il soit un peigne gauche soit un peigne droit ? Justifier la réponse.

Exercice 8 – Arbres binaires d'expressions

Dans tout l'exercice les opérateurs arithmétiques $+, *, -, /$ sont binaires.
Les expressions arithmétiques bien formées (préfixes, resp. suffixes) sont obtenues par parcours (préfixes, resp. suffixes) d'arbres binaires d'expressions.

Question 1

En utilisant les résultats de l'exercice 6 dire pourquoi :
— l'expression $* \ + \ / \ 9 \ 2 \ 7 \ + \ 5$, n'est pas une expression préfixe bien formée
— l'expression $5 \ 3 \ 9 \ * \ + 3 \ 9 *$ n'est pas une expression suffixe bien formée.
On rappelle la définition de la fonction d'évaluation d'un arbre binaire d'expression :

```
def ABevaluation(T):
  if estABfeuille(T):
    return T.clef
  elif T.clef == "+":
    return ABevaluation(T.gauche) + ABevaluation(T.droit)
  elif T.clef == "-":
    return ABevaluation(T.gauche) - ABevaluation(T.droit)
  elif T.clef == "*":
    return ABevaluation(T.gauche) * ABevaluation(T.droit)
  else :
    return ABevaluation(T.gauche) / ABevaluation(T.droit)
```

Question 2

Montrer que, pour tout arbre d'expression T, `ABevaluation(T)` effectue exactement $ni(T)$ opérations. En déduire que la complexité de `ABevaluation(T)` est en $\Theta(n)$, où n est la taille de T.
On rappelle le théorème énoncé en cours :
Théorème – Une liste L de longueur n est une *expression préfixe bien formée* ssi

(a) le nombre de valeurs numériques de L est égal au nombre d'opérateurs de L augmenté de 1
(b) pour tout $i < n - 1$, le nombre de valeurs numériques de $L[0..i]$ est inférieur ou égal au nombre d'opérateurs de $L[0..i]$.

Question 3

Pour chacune des listes suivantes, dire si elle est une expression préfixe bien formée, en justifiant la réponse :
— $L_0 = [' +', 3, ' *', 2, ' -', ' *', 4, 5, 6]$
— $L_1 = [' +', 3, ' *', 2, 4, ' -', ' *', 5, 6]$
— $L_2 = [' +', 3, ' *', ' -', ' *', 4, 5, 6]$
— $L_3 = [' +', 3, ' *', ' -', ' *', 4, 2, 5, 6]$.

Question 4

Prouver le théorème.
On rappelle les définitions des fonctions qui permettent de transformer une expression préfixe bien formée en un arbre binaire d'expression :

```
def operandesPref(L):
  cpt_operateurs = 0
  cpt_valeurs = 0
  i = 1
  while cpt_operateurs >= cpt_valeurs:
    if estOperateur(L[i]):
      cpt_operateurs = cpt_operateurs + 1
    else:
      cpt_valeurs = cpt_valeurs + 1
    i = i + 1
  return (L[1:i], L[i:])

def prefVersAB(L):
  x = L[0]
```

```
  if estNombre(x):
    return ABfeuille(x)
  (L1, L2) = operandesPref(L)
  return AB(x, prefVersAB(L1), prefVersAB(L2))
```

Question 5
Dessiner l'arbre des appels récursifs effectués par `prefVersAB(L0)` avec $L0 = [' +', 3, ' *', 2, ' -', ' *', 4, 5, 6]$.

Question 6
Montrer que le parcours préfixe de `prefVersAB(L)` est égal à `L` pour toute expression préfixe bien formée `L`.

Question 7
Adapter le théorème et les questions précédentes au cas des expressions suffixes bien formées.

Exercice 9 – Des expressions arithmétiques vers les arbres

Dans cet exercice les expressions sont supposées bien formées.

Question 1
Représenter les expressions postfixées suivantes sous forme d'arbres binaires :

$$5\ 3\ 9\ *\ +\ 3\ 9\ *\ -$$

$$3\ 2\ +\ 3\ *\ +\ 9\ 3\ /\ 5\ *\ -$$

La fonction ci-dessous permet de transformer une expression arithmétique postfixée bien formée stockée dans un tableau de taille n en un arbre d'expression, en utilisant une pile.

```
def postVersAB_ite(tab):
  n = len(tab) ; P = [] ; i = 0
  while i<n:
    if estNombre(tab[i]):
      P.append(ABfeuille(tab[i]))
    else:
      x = P.pop() ; y = P.pop()
      P.append(AB(tab[i],y,x))
    i = i+1
  return P.pop()
```

La fonction `estNombre` teste si son argument est un nombre.

Question 2
Montrer l'évolution de la pile pour l'expression $5\ 3\ 9\ *\ +\ 5\ +\ 3\ 9\ *\ -$.

Question 3
Quelle est la complexité de `postVersAB_ite` ?
On considère maintenant des expressions arithmétiques sous forme préfixe.
Voici le schéma de l'algorithme qui construit l'arbre d'une expression préfixe bien formée :

— ou bien on lit un opérateur et alors on l'empile
— ou bien on lit un nombre x et alors

1. on crée l'arbre binaire réduit à la feuille de contenu x et on l'empile
2. `tant que` l'on est dans la configuration suivante :

le sommet (premier élément) de la pile est un arbre le deuxième élément de la pile est un arbre le troisième élément de la pile est un opérateur

on dépile les trois premiers éléments pour en faire un arbre et on empile cet arbre
`fin tant que`

Remarque : la pile peut contenir des opérateurs ou des arbres.
Remarque : l'expression étant supposée bien formée, il est inutile de tester si le troisième élément de la pile est un opérateur lorsque les deux premiers éléments de la pile sont des arbres.
D'où la fonction :

```
def prefVersAB_Ite(tab):
  n = len(tab) ; P = [] ; i = 0
  while i<n:
    if estOperateur(tab[i]):
      P.append(tab[i])
    else:
      x = ABfeuille(tab[i])
      encore = True
      while encore:
        y = P.pop()
        if estOperateur(y):
          P.append(y) ; P.append(x)
          encore = False
        else:
          z = P.pop()
          P.append(AB(z,y,x))
          x = P.pop()
          if P == []:
            encore = False
    i = i+1
  return x
```

Question 4
Montrer l'évolution de la pile pour l'expression : $* + * 2\ 3\ 4 + 5\ 6$.

Question 5
Quelle est la complexité de `prefVersAB_ite` ?