EA3
Examen du jeudi 12 janvier 2017

Durée : 3 heures
Une feuille A4 de notes manuscrites autorisée
Appareils électroniques éteints et rangés

Préliminaires : *Ce sujet est constitué de 4 exercices qui peuvent être traités dans l'ordre de votre choix. Le barème est donné à titre indicatif. Il est conseillé de lire l'intégralité du sujet avant de commencer. Il est bien entendu préférable de ne faire qu'une partie du sujet correctement plutôt que de tout bâcler.*

Exercice 1 : Que se passe-t-il ? (4 points)

```
1  quoi(T : tableau de n entiers):
2      i <- 0
3      j <- T.length-1
4      while( i < j ){
5        a <- i
6        b <- j
7        for( k = i à j par pas de 1){
8          if( T[k] < T[a] )
9            a <- k
10         else if ( T[k] > T[b] )
11           b <- k
12       }
13       if( a=j and b=i)
14         échanger T[i] et T[j]
15       else if( a=j ){
16         échanger T[a] et T[i]
17         échanger T[b] et T[j]
18       }
19       else{
20         échanger T[b] et T[j]
21         échanger T[a] et T[i]
22       }
23       i <- i+1
24       j <- j-1
25     }
```

1. Exécuter l'algorithme `quoi` sur le tableau `T={2, 6, 7, 5, 3, 1, 4}` en décrivant chaque étape.
   Attention, un résultat non détaillé ne sera pas lu !
2. Que fait la boucle `for` de l'algorithme `quoi` (ligne 7 à 12) ?
3. Soit `T` un tableau d'entiers et `n` sa longueur. On donne la propriété $\mathcal{P}_{i,j}$ suivante :
   `T[0..i-1]` (resp. `T[j+1..n-1]`) est un tableau trié par ordre croissant et contenant les plus petites (resp. grandes) valeurs de `T`.
   Vous allez montrer que $\mathcal{P}_{i,j}$ est un invariant de la boucle `while` de l'algorithme `quoi`.

- **a.** Montrer que $\mathcal{P}_{0,n-1}$ est vraie à l'entrée dans la boucle `while`.
- **b.** En supposant que $\mathcal{P}_{i,j}$ est vraie au début de la boucle `while`, montrer qu'elle reste vraie après une exécution de la boucle.
- **c.** En déduire que $\mathcal{P}_{i,j}$ est un invariant de la boucle `while` de l'algorithme `quoi` et prouver ce que fait l'algorithme `quoi`.

4. La complexité en temps de cet algorithme est quadratique en la longueur du tableau passé en paramètre. Est-ce que cet algorithme est optimal ? Si non, citer un algorithme avec une meilleure complexité donnant le même résultat que `quoi` en rappelant sa complexité.

Exercice 2 : parcours d'arbres binaires (6 points)

Dans cet exercice, lorsque vous écrirez un algorithme, vous avez le droit de définir (écrire le pseudo-code) de nouvelles fonctions auxiliaires.

On pourra supposer que l'on dispose de la fonction `affiche(n)` qui affiche la valeur du nœud `n`, et des fonctions usuelles d'accès à une file (`set`, qui insère en queue et `get`, qui retire l'élément en tête et le retourne).

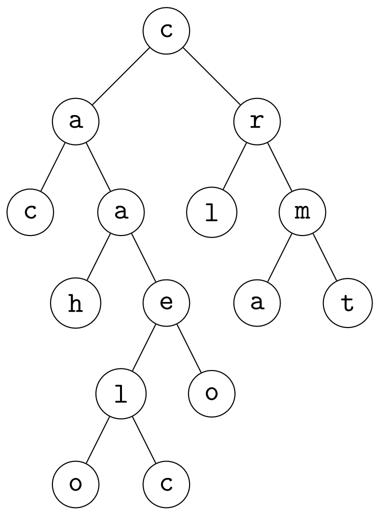
Pour représenter un arbre binaire, on considère ici :

- un type `Noeud` avec trois champs, un champ caractère `val`, deux champs `G` et `D` de type `Noeud`,
- un type `Arbre` avec un champ `racine` de type `Noeud`.

Un arbre vide est un arbre dont le champ `racine` est `null` et une feuille a ses deux champs `G` et `D` à `null`.

1. Étant donné un arbre binaire A, écrire un algorithme qui effectue un parcours en profondeur et affiche les valeurs des feuilles de cet arbre A dans l'ordre du parcours.

2. Exécuter votre algorithme sur l'arbre B suivant en détaillant les étapes :



3. Donner le mot obtenu lors d'un parcours infixe de l'arbre B ci-dessus (ici on prendra en compte tous les nœuds).

4. Donner le mot obtenu lors d'un parcours en largeur de l'arbre B (ici on prendra en compte tous les nœuds).

5. En modifiant l'algorithme de parcours en largeur vu en cours, écrire un algorithme qui affiche les valeurs des nœuds d'un arbre binaire A à profondeur k, où k est un entier positif ou nul.

Exercice 3 : arbres d'arité quelconque (6 points)

Dans cet exercice, lorsque vous écrirez un algorithme, vous avez le droit de définir (écrire le pseudo-code) de nouvelles fonctions auxiliaires (et surtout celles d'accès à une liste chaînée) et c'est même recommandé.

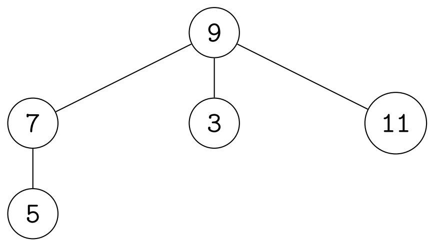
On veut coder des arbres plans dont les noeuds sont d'arité quelconque, c'est-à-dire que chaque nœud interne peut avoir un nombre quelconque d'enfants.

On définit une structure de données `Node` comportant deux champs, un champ entier `val` et un champ `kids`. Le champ `kids` est une liste simplement chaînée de `Node` et représente la liste d'enfants ordonnée du plus à gauche au plus à droite, c'est-à-dire que la tête de la liste pointe sur l'enfant le plus à gauche. Une feuille est alors un `Node` dont le champ `kids` est une liste vide. Un arbre est défini par une structure de donnée `Tree` avec un champ `root` de type `Node` qui représente la racine de l'arbre.

Pour représenter une liste simplement chaînée de `Node`, on considère ici :
- un type `Cellule` avec deux champs, un champ `val` de type `Node` et un champ `suivant` de type `Cellule`,
- un type `Liste` avec un champ `tete` de type `Cellule`.

Une liste vide est une liste dont la tête est `null`.

1. Créer la structure de données correspondant à l'arbre suivant :



2. Soit l'algorithme suivant :

```
algo_arbre(r: Node):
    affiche(r)
    for( x in r.kids )
      algo_arbre(x)
```

Que fait cet algorithme ?

3. Écrire une fonction `contient(Tree: a, Node: n)` qui détermine si l'arbre `a` contient le nœud `n`.

4. Écrire une fonction `insertion(Node m, Node r, Node n)` qui insère le nœud `m` comme dernier enfant du nœud `n` (si celui-ci appartient à l'arbre enraciné en `r`).

5. Écrire une fonction `suppression_aux(Node n, Node r)` qui supprime de l'arbre enraciné en `r` le nœud `n` (s'il s'y trouve) ainsi que tous ses descendants. On suppose ici que `r` et `n` ne désignent pas le même nœud.

6. Écrire une fonction `suppression(Node n, Tree a)` qui supprime de `a` le nœud `n` (si celui-ci appartient à `a`) ainsi que tous ses descendants. Attention au cas où `n` est la racine de `a`.

Exercice 4 : tas (4 points)

1. Pour chaque tableau, dire s'il représente un tas minimum. Dans le cas positif, donner le tas sous forme d'arbre et dans le cas négatif, expliquer pourquoi :

$T_1 =$

3	12	7	14	19	8	11	18	16

$T_2 =$

2	9	5	7	11	10	6	8	14

Dans la suite, vous pourrez utiliser la représentation d'un tas sous forme de tableau ou d'arbre.

2. Insérer la valeur 4 dans le tas de la question précédente en utilisant l'algorithme d'insertion dans un tas et en détaillant les étapes.
3. Extraire l'élément minimal dans le tas de la question précédente en utilisant l'algorithme d'extraction du minimum dans un tas minimum et en détaillant les étapes.
4. Construire le codage de Huffman pour le texte « *sempiternellement* » en détaillant à chaque étape la file de priorité et l'arbre construit. Coder ensuite le texte donné.