EA3
Examen du mercredi 21 juin 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  quesaco(T : tableau de n entiers):
2      if( T vide ):
3        return (-,-)
4      i <- 1
5      a <- 1
6      b <- 0
7      while( i < T.length){
8        if( T[i] == T[b] )
9          a <- a+1
10       else if( T[i] > T[b] ){
11         b <- i
12         a <- 1
13       }
14       i <- i+1
15     }
16     return (a, b)
```

1. Exécuter l'algorithme `quesaco` sur le tableau `T={6, 4, 6, 7, 4, 6, 7, 7, 4}` en donnant à chaque étape les valeurs de `i`, `a` et `b`.
   Attention, un résultat non détaillé ne sera pas lu !
2. Soit `T` un tableau d'entiers. On donne la propriété $\mathcal{P}_i$ suivante :
   `T[b]` est le maximum de `T[0..i-1]` et `a` est le nombre d'occurences de `T[b]` dans `T[0..i-1]`.
   Vous allez montrer que $\mathcal{P}_i$ est un invariant de la boucle `while` de l'algorithme `quesaco`.
   a. Montrer que $\mathcal{P}_1$ est vraie à l'entrée dans la boucle `while`.
   b. En supposant que $\mathcal{P}_i$ est vraie au début de la boucle `while`, montrer qu'alors $\mathcal{P}_{i+1}$ est vraie après une exécution de la boucle.
   c. En déduire que $\mathcal{P}_i$ est un invariant de la boucle `while` de l'algorithme `quesaco` et prouver ce que fait l'algorithme `quesaco`.

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).

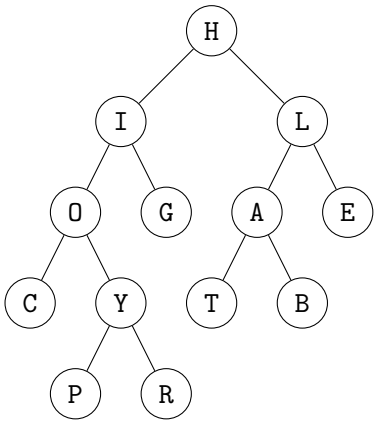
Pour représenter un arbre binaire, on considère ici :

- 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. Écrire une fonction `parcours(Arbre A, entier positif k)` qui effectue un parcours en largeur et affiche les `k` premières valeurs des **noeuds internes** de `A` dans l'ordre du parcours. Si `A` contient moins de `k` noeuds internes, l'algorithme affiche tous les noeuds internes.

2. Exécuter votre algorithme sur l'arbre `B` suivant pour `k=5` en détaillant les étapes (notamment l'évolution des structures de données utilisées dans l'algorithme) :



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

a. préfixe

b. suffixe

c. infixe

4. Écrire une fonction `profondeur(Arbre A, lettre l)` qui retourne la profondeur du noeud de `A` étiqueté par `l`. On suppose ici que les valeurs des noeuds sont toutes distinctes. S'il n'y a pas de noeud d'étiquette `l` dans `A`, la fonction retourne la valeur '-'. Par exemple sur l'arbre `B`, la profondeur du noeud étiqueté `'A'` est 2.

5. Exécutez votre algorithme sur l'arbre `B` et la lettre `l='Y'` en détaillant les étapes.

Exercice 3 : tableau de listes (6 points)

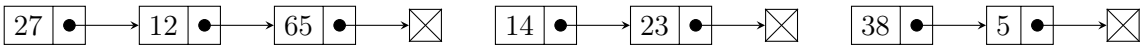
Dans cet exercice, lorsque vous écrirez un algorithme, vous avez le droit de définir (écrire le pseudo-code) de nouvelles fonctions auxiliaires (et surtout celles d'accès à une liste chaînée) et c'est même recommandé.

On considère la structure de donnée `Tab` représentant un tableau de `Liste`, où le type `Liste` représente une liste chaînée et est définit de la façon suivante :

- un type `Cellule` avec deux champs, un champ `val` de type entier et un champ `suivant` de type `Cellule`,
- un type `Liste` avec un champ `tete` de type `Cellule`.

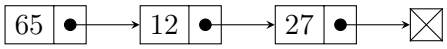
Une liste vide est une liste dont la tête est `null`.

1. Créer la structure de données `P` de type `Tab` contenant les trois listes suivantes :

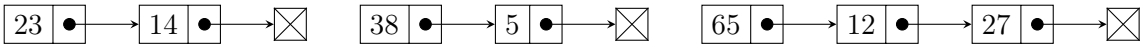


2. Écrire une fonction `eltMax(Liste L)` qui retourne l'élément de type `Cellule` de la liste `L` qui contient la plus grande valeur de la liste.

3. Écrire une fonction `maxEnTete(Liste L)` qui modifie cette liste de façon à ce que l'élément de valeur maximale de la liste se retrouve au début de la liste. `maxEnTete(Liste L)` échange les places de l'élément de valeur maximale avec l'élément placé en tête. Dans l'exemple précédent, la liste `P[0]` devient, après appel de `maxEnTete` :



4. Écrire une fonction `tri(Tab T)` qui trie dans l'ordre croissant le tableau de listes `T` suivant la valeur maximale de chaque liste de `T`. Votre algorithme doit s'inspirer d'un algorithme de tri du cours et peut modifier l'ordre des éléments dans chaque liste. Par exemple, `tri(P)`, ordonne les éléments de `P` comme suit :



5. Donner le nom de l'algorithme de tri du cours utilisé dans la question précédente ainsi que sa complexité (au pire) en temps. Cet algorithme est-il optimal pour la complexité (au pire) en temps ? Si non, citer un algorithme de tri du cours avec une meilleure complexité au pire et donner cette complexité.

Exercice 4 : tas (4 points)

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

$T_1 =$ | 25 | 12 | 9 | 5 | 8 | 3 | 10 | 2 | 1 | $\quad T_2 =$ | 28 | 15 | 7 | 11 | 9 | 3 | 6 | 10 | 5 |

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

2. Insérer la valeur 17 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 maximal dans le tas de la question précédente en utilisant l'algorithme d'extraction du maximum dans un tas maximal et en détaillant les étapes.

4. Construire le codage de Huffman pour le texte « *heterogeneite* » en détaillant à chaque étape la file de priorité et l'arbre construit. Coder ensuite le texte donné.