EA3
Examen du jeudi 11 janvier 2018

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

Préliminaires : *Ce sujet est constitué de 3 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 ? (8 points)

Dans cet exercice, lorqu'on parle de tri, on suppose que le tri donne des éléments dans l'ordre croissant.

```
1  quoi(t : tableau trié de n entiers, v: entier){
2      g <- 0
3      d <- n-1
4      while g <= d {
5        m <- (g+d)/2
6        if v < t[m]{
7          d <- m-1
8        }
9        else{ g <- m+1 }
10     }
11
12     if g > n-1 { return - }
13     return g
14 }
```

1. Exécuter l'algorithme `quoi` sur le tableau `T={2,3,12,17,24,67}` pour les valeurs respectives de `v` 3, 32 et 75 en remplissant pour chaque valeur de `v` le tableau suivant :

g	d	T[m]
...	...	...

et donnant le retour de `quoi`.

2. Soit `T` un tableau d'entiers et `n` sa longueur. On donne la propriété $\mathcal{P}_{g,d}$ suivante :

$$\begin{cases} \texttt{T est trié par ordre croissant et} \\ \texttt{v} \geqslant \texttt{t[g-1]} \text{ si } \texttt{0} < \texttt{g} \\ \texttt{v} < \texttt{t[d+1]} \text{ si } \texttt{d} < \texttt{n-1}. \end{cases}$$

Vous allez montrer que $\mathcal{P}_{g,d}$ 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 $\mathcal{P}_{g,d}$ est vraie au début de la boucle `while`, montrer qu'elle reste vraie après une exécution de la boucle.

c. En déduire que $\mathcal{P}_{g,d}$ est un invariant de la boucle `while` de l'algorithme `quoi` et prouver ce que fait l'algorithme `quoi`.

3. A quel algorithme, vous fait penser l'algorithme `quoi` ? En déduire la complexité au pire de l'algorithme `quoi`.
4. En vous inspirant du tri par insertion, écrire un algorithme de tri qui utilise l'algorithme `quoi`.
5. En vous appuyant sur la complexité du tri par insertion et celle de l'algorithme `quoi`, expliciter la complexité de votre algorithme de tri.

Exercice 2 : tas (3 points)

Pour chaque tableau, dire s'il représente un tas maximum. Dans le cas positif, donner le tas sous forme d'arbre et dans le cas négatif, expliquer pourquoi et donner la suite d'appels à `entasser` nécessaire pour transformer le tableau en un tas. Vous présenterez cette suite d'appels de la façon suivante :

appel	tableau obtenu après l'appel
...	...

1. $T_1 =$

22	15	11	17	16	9	10	14	18	12	13	6	8	1	3	5	2	4	7

2. $T_2 =$

19	15	11	13	14	8	10	7	6	12	9	5	1	4	2	3

Exercice 3 : arbres (Partie I : 4,5 points, Partie II : 4,5 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 des fonctions :

- `push(p: pile, val: valeur)` et `pop(p: pile)` pour respectivement ajouter une valeur à une pile, et retirer et retourner le sommet de la pile,
- `add(f: file, val: valeur)` et `get(f: file)` pour respectivement ajouter une valeur à une file, et retirer et retourner la tête de la file.

Partie I. Arbres binaires plans

Pour représenter un arbre binaire, on considère ici :

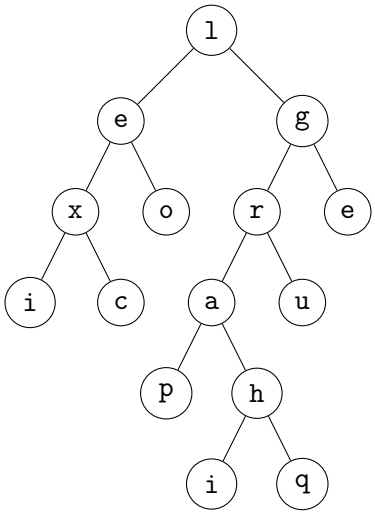
- un type `NoeudB` avec quatre champs, un champ caractère `clé`, deux champs `G` et `D` de type `NoeudB`, et un champ entier `p`,
- un type `ArbreB` avec un champ `racine` de type `NoeudB`.

Un arbre vide est un arbre dont le champ `racine` est `null` et une feuille a ses deux champs `G` et `D` à `null`. On suppose également que l'on dispose des fonctions suivantes :

- `NoeudB(c)` qui construit et retourne un `NoeudB` feuille dont la clé est égale à `c`,
- `ArbreB(r)` qui construit et retourne un `ArbreB` de racine r.

1. Étant donné un arbre binaire A, écrire un algorithme `majP` qui effectue un parcours préfixe et met à jour le champ `p` de chaque nœud n de la façon suivante :
   - si n est la racine de A, alors $n.p = 0$,
   - sinon, si n est un enfant gauche de parent π, $n.p = \pi.p + 1$,
   - sinon, n est un enfant droit de parent π et $n.p = \pi.p$.
2. Étant donné un arbre binaire A, écrire un algorithme `sommeP` qui utilise le parcours en largeur d'un arbre pour retourner la somme des champs `p` de ses nœuds.

Soit l'arbre B suivant :



3. Donner le mot obtenu lors d'un parcours infixe de l'arbre B ci-dessus en supposant que l'action faite lors de la visite d'un nœud est l'affichage de la clé de ce nœud.

4. Donner le mot obtenu lors d'un parcours suffixe de l'arbre B en supposant que l'action faite lors de la visite d'un nœud est l'affichage de la clé de ce nœud.

5. Dessiner l'arbre général plan obtenu en appliquant à l'arbre B la bijection vue en cours.

Partie II. Arbres généraux plans

Pour représenter un arbre général, on considère ici :
- un type `NoeudG` avec deux champs, un champ caractère `clé` et un champ `enfants` qui est une liste chaînée de `Cellule` dont les valeurs sont des `NoeudG`,
- un type `ArbreG` avec un champ `racine` de type `NoeudG`.

Un arbre vide est un arbre dont le champ `racine` est `null` et une feuille a sa liste `enfants` vide.
Enfin pour représenter une liste chaînée, on considère ici :
- un type `Cellule` avec un champ `val` de type `NoeudG` et un champ `succ` de type `Cellule`,
- un type `Liste` avec un champ `tête` de type `Cellule`.

Une liste vide est une liste dont le champ `Cellule` est `null`.

6. Écrire un algorithme `frere2fils` qui étant donné un `NoeudB r` feuille et une liste chaînée `l`, modifie `r` de la façon suivante :
- `r.G.clé = l.tête.val.clé`
- `r.G.D`$^{i-1}$`.clé = e`$_i$`.val.clé` $\forall 1 < i \leqslant k$ avec `l=[e`$_1$`,...,e`$_k$`]` et `l.tête = e`$_1$ (`D`$^{i-1}$ $= \underbrace{D \cdot \ldots \cdot D}_{i-1 \text{ fois}}$)

7. Écrire un algorithme `gen2bin` qui étant donné un arbre général construit et retourne l'arbre binaire obtenu en appliquant la bijection vue en cours.

8. En utilisant les algorithmes des questions 2 et 7, écrire un algorithme `sommeProf` qui retourne la somme des profondeurs des nœuds d'un arbre général plan. Vous expliquerez pourquoi vous pouvez utiliser ici l'algorithme de la question 2.