EA3
Examen du mercredi 20 juin 2018

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. Vous trouverez en annexe quelques algorithmes.*

Conventions :

Étant donné un tableau `T`, on note :

- `T.length` la longueur du tableau `T`,
- `T[i..j]` le sous tableau de `T` des valeurs d'indices `i` à `j` inclus, et si `j<i`, `T[i..j]` dénote le tableau vide.

Les indices des tableaux vont de 0 à `T.length-1`.

Exercice 1 : Que se passe-t-il ? (7 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 trie de n entiers, g: entier, d:entier, x:entier){
2       m <- (g+d)/2
3       if x = t[m] {
4         if m = 0 { return (-1, -1) }
5         else { return (m-1, m) }
6       }
7       if x < t[m] {
8         if t[g..m-1] est vide {
9           if m = 0 { return (-1, -1) }
10          else { return (m-1, m) }
11        }
12        else { return quoi(t, g, m-1, x) }
13      }
14      if x > t[m] {
15        if t[m+1..d-1] est vide {
16          if m = (t.length - 1) { return (-1, -1) }
17          else { return (m, m+1) }
18        }
19        else { return quoi(t, m+1, d, x) }
20      }
21 }
```

1. Exécuter l'algorithme `quoi` sur le tableau `T={6,14,17,21,28,43,50,55}` pour les valeurs respectives de `x` 2, 39 et 52 en remplissant pour chaque valeur de `x` le tableau suivant :

appel de quoi	`m`	`T[m]`
quoi(T, 0, 7, x)	...	...
...	...	...

et donnant le retour de `quoi`.

2. La récursion de `quoi` est-elle terminale ? Expliquez pourquoi en deux phrases au plus.

3. On présente maintenant l'algorithme suivant :

```
1  what(t : tableau trie de n entiers, x:entier) {
2    g <- 0
3    d <- (t.length-1)
4    while g <= d {
5      m <- (g+d)/2
6      if x = t[m] {
7        d <- m-1
8        g <- m
9      }
10     if x < t[m] { d <- m-1 }
11     else { g <- m+1 }
12   }
13   if d < 0 or g > (t.length - 1) { return (-1, -1) }
14   else { return (d, g) }
15 }
```

4. Justifier la terminaison de l'algorithme `what`.

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

$$\begin{cases} \textit{T est trié par ordre croissant et} \\ x \notin \textit{T[0..g-1]} \\ x \notin \textit{T[d+1..T.length-1]}. \end{cases}$$

Vous allez montrer que $\mathcal{P}_{g,d}$ est un invariant de la boucle `while` de l'algorithme `what`.

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 `what` et prouver ce que fait l'algorithme `what`.

6. À quel algorithme, vous fait penser l'algorithme `what` ? En déduire la complexité au pire de l'algorithme `what`.

Exercice 2 : tas (3 points)

1. Écrire l'algorithme `entasser_it(t, i)`, version itérative de l'algorithme `entasser` vu en cours, qui permet d'entasser dans un tableau `t` d'entiers, l'élément de `t` d'indice `i`.

2. 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 obtenir un tas à partir du tableau. Vous présenterez cette suite d'appels de la façon suivante :

appel	tableau obtenu après l'appel
entasser(...,...)	...
...	...

a. $T_1 =$

	22	21	12	19	9	10	5	15	17	3	7	8	1	2	4	13	6	14	11

b. $T_2 =$

	24	11	23	8	20	16	15	4	6	17	5	14	12	7	13	3	1	5	2	9	10

Exercice 3 : tri fusion (4 points)

Étant donné un tableau T d'entiers, on appelle inversion toute paire (i, j) telle que $i < j$ et $T[i] > T[j]$. Par exemple, le tableau $T = [4, 1, 3, 2]$ contient les inversions $(0, 1)$, $(0, 2)$, $(0, 3)$ et $(2, 3)$.

1. Modifier l'algorithme de tri fusion vu en cours afin qu'il calcule également le nombre d'inversions contenues dans le tableau passé en paramètre.

2. Quelle est la complexité de votre algorithme ?

Exercice 4 : arbres (6 points)

Pour représenter un arbre général plan A, on utilise la bijection vue en cours pour le représenter. La représentation est donc un arbre binaire tel que chaque noeud de l'arbre représenté connait son fils gauche, son frère droit et son père. On définit donc les types suivants :

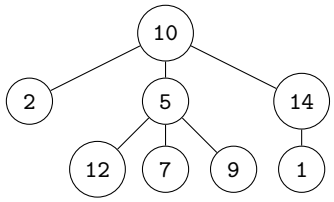
- un type `Noeud` avec quatre champs, un champ entier `cle`, trois champs `filsG`, `frereD` et `pere` de type `Noeud`,
- un type `Arbre` avec un champ `racine` de type `Noeud`.

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

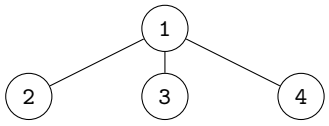
- `Noeud(c)` qui construit et retourne un `Noeud` feuille dont la clé est égale à `c` et les champs `frereD` et `pere` sont `null`,
- `Arbre(r)` qui construit et retourne un `Arbre` de racine r,
- `setFilsG(n, p)` (respectivement `setFrereD(n, p)`, `setPere(n, p)`) qui affecte au champ `filsG` (respectivement `frereD`, `pere`) du `Noeud n`, le `Noeud p`.

On pourra également supposer que l'on dispose d'une fonction `affiche(x)` qui affiche la valeur de `x`.

1. Appliquer la bijection vue en cours à l'arbre général plan A suivant :



2. En utilisant les fonctions pré-définies ci-dessus, créer la structure de données pour l'arbre général plan A suivant :



3. Étant donné un arbre général plan A, écrire un algorithme `sommes_part(B)` qui affiche les sommes partielles des clés obtenues dans un parcours en profondeur de l'arbre A, où `B` est de type `Arbre` et est la représentation de A.

Si lors du parcours en profondeur de l'arbre A, la suite ordonnée de clés est $7, 3, 9, 12, 5, 6$, alors les sommes partielles sont 7, 10, 19, 31, 36 et 42.

4. Étant donné un arbre général plan A, écrire un algorithme `parcours(B)` qui affiche les clés de A dans l'ordre d'un parcours en largeur de A, où `B` est de type `Arbre` et est la représentation de A.

Vous utiliserez les fonctions suivantes d'accès à une file `F` : `add(F,x)` qui ajoute l'élément `x` à la file `F`, `get(F)` qui retire et retourne le premier élément de la file `F`, `vide(F)` qui retourne vrai si la file est vide, faux sinon.

Annexe

```
fusion(T: tableau de n entiers, deb:entier, fin: entier){
  m <- (deb+fin)/2
  for i from 0 to m-deb{ G[i] <- T[deb+i] }
  for i from 0 to fin-m-1{ D[i] <- T[m+1+i] }
  i <- 0
  j <- 0
  for k from deb to fin{
    if G[i] <= D[j] {
      T[k] <- G[i]
      i <- i+1
    }
    else{
      T[k] <- D[j]
      j <- j+1
    }
  }
}

triFusion(T, deb, fin){
  if deb < fin {
    m <- (deb+fin)/2
    triFusion(T,deb,m)
    triFusion(T,m+1, fin)
    fusion(T, deb, fin)
  }
}
```

```
entasser(T: tableau, i: indice) {
  max <- i
  if 2i < T.length and T[2i] > T[max] {
    max <- 2i
  }
  if 2i+1 < T.length and T[2i+1] > T[max] {
    max <- 2i+1
  }
  if max != i {
    echanger(T[i], T[max])
    entasser(T, max)
  }
}
```