EA3
Examen du jeudi 10 janvier 2019

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 : listes chaînées (4 points)

Pour représenter une liste chaînée, on considère ici :
- un type `Cellule` composé de deux champs, un champ caractère `clé` et un champ `suiv` de type `Cellule`,
- un type `Liste` composé d'un champ `tête` de type `Cellule`.

Une liste vide est une liste dont le champ `tête` est `null`. On suppose que l'on dispose des fonctions suivantes :
- `Cellule(a)` qui construit et retourne une `Cellule` dont la clé est égale à `a`,
- `Liste(c)` qui construit et retourne une `Liste` dont la tête est la `Cellule c`.

1. Écrire un algorithme en place `inverse_liste(L : liste chainée)` qui inverse une liste chaînée en n'effectuant qu'un seul parcours de la liste. Vous avez le droit d'ajouter un champ au type `Cellule`.
2. Écrire un algorithme `insere_tri(L : liste chainée triée, x : clé)` qui insère la clé `x` à la bonne place dans la liste chaînée triée `L` en n'effectuant, au pire, qu'un seul parcours de la liste.
   Note : le tri s'effectue ici dans l'ordre croissant.

Exercice 2 : pleins de données (4 points)

Votre machine reçoit des données de type `DATA` définit par les champs suivants :
- `titre` qui est une chaîne de caractères,
- `auteurs` qui est une chaîne de caractères,
- `vol` qui est un entier,
- `act` qui est une chaîne de caractères.

On souhaite pouvoir comparer les données deux à deux. On suppose que l'on dispose d'une fonction `lexico(ch1, ch2)` qui compare deux chaînes de caractères et retourne -1 si `ch1<ch2`, 0 si `ch1=ch2`, 1 si `ch1>ch2`. Soit `d1` et `d2` deux données. On dit que `d1<d2` si :
- `d1.titre` est avant `d2.titre` pour l'ordre lexicographique,
- ou `d1.titre` est égal à `d2.titre` et `d1.auteur` est avant `d2.auteur` pour l'ordre lexicographique,
- ou `d1.titre` est égal à `d2.titre` et `d1.auteur` est égal à `d2.auteur` pour l'ordre lexicographique et `d1.vol<d2.vol`.

Si les champs `titre`, `auteur` et `vol` sont égaux, alors `d1=d2`.
On suppose qu'à un instant donné, on ne stocke pas plus de `MAX` données sur votre ordinateur.
Lorsqu'une donnée arrive, son champ `act` peut prendre deux valeurs `REC` ou `EFF`. Si la valeur est `REC`, la donnée doit être stockée sur votre ordinateur s'il y a strictement moins de `MAX` données stockées, sinon la donnée à ernregistrer est ignorée. Si la valeur est `EFF`, elle doit être effacée. Quelles structure de données et algorithmes choisissez-vous afin d'effectuer les opérations `REC` et `EFF` en maximum $log_2(n)$ opérations de comparaisons où n est le nombre de données enregistrées sur votre ordinateur à un instant donné ? Les algorithmes qui ne sont pas du cours devront être écrits, ceux du cours pourront juste être cités.

L2 Informatique Année 2018-2019

Exercice 3 : Que se passe-t-il ? (10 points)

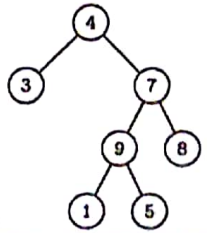
Pour représenter un arbre binaire, on considère ici :
- un type `Noeud` avec quatre champs, un champ entier `clé`, deux champs `G` et `D` de type `Noeud`, et un champ entier `a`,
- un type `Arbre` avec un champ `racine` de type `Noeud`.

Un arbre vide est un arbre dont le champ `racine` est `null` et une feuille a ses deux champs `G` et `D` à `null`. 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`,
- `Arbre(r)` qui construit et retourne un `Arbre` de racine `r`, où `r` est un `Noeud`.

Soit l'algorithme suivant :

```
1  f(s : Noeud) :
2    s.a <- -1
3    if s.G != null {
4      f(s.G)
5      if s.a < s.G.a { s.a <- s.G.a }
6    }
7    if s.D != null {
8      f(s.D)
9      if s.a < s.D.a { s.a <- s.D.a }
10   }
11   s.a <- s.a + 1
```

1. Exécuter l'algorithme `f` sur le nœud racine de l'arbre suivant :



en remplissant le tableau suivant :

appel	s.clé	s.a ligne 6	s.a ligne 10	s.a ligne 12
f(④)	...	...	...	...
⋮	⋮	⋮	⋮	⋮

2. La récursion de l'algorithme `f` est-elle terminale ? Si oui justifier pourquoi, si non dire quelles variables sont stockées sur la pile d'exécution au moment de l'appel récursif.
3. a. Sur quel parcours d'arbre s'appuie l'algorithme ?
   b. En déduire la complexité en nombre de comparaisons en fonction du nombre n de nœuds de l'arbre de `f`.

4. Donner l'affichage obtenu lors d'un parcours suffixe de l'arbre de la question 1 si l'algorithme `visite(s)` affiche la clé du nœud `s`.
5. Faire de même pour le parcours en largeur.
6. Le but de cette question est de montrer la correction de l'algorithme `f`. Soit `s` un nœud d'un arbre binaire.
   a. Montrer que `f(s)` termine quelque soit le nœud `s`. Vous pourrez faire une preuve par induction sur la hauteur du nœud s.
   b. Si `s` est une feuille de l'arbre, quelle est la valeur de `s.a`? Et si `s` est un nœud interne de l'arbre?
   c. En déduire ce que fait `f(s)` en justifiant. Vous pourrez faire une preuve par induction sur la hauteur du nœud s.
   À quelle notion définie en cours correspond `s.a`?
7. Écrire un algorithme `aff_hauteur(a : Arbre, k : entier positif)` qui affiche les clés des noeuds de hauteur k.

Exercice 4 : tas (4 points)

1. On démarre avec un tas maximum `T` vide, puis on fait les opérations suivantes sur le tas : `insertion(T,4)`, `insertion(T,7)`, `insertion(T,3)`, `insertion(T,8)`, `insertion(T,2)`, `suppressi` où `insertion` et `suppression` sont les opérations sur les tas maximum décrites en cours.
   Pour chaque opération, donner le tableau `T` juste avant le premier échange de cases de `T`, la suite d'échanges effectués (un échange est un couple d'indices correspondant aux indices des deux cases de `T` à échanger), puis le tableau obtenu :

opération	T avant	suite d'échanges	T après
`insertion(T,4)`	...	...	...
⋮	⋮	⋮	⋮

2. On suppose maintenant qu'on a diminué la clé du nœud d'indice `i` dans `T`. D'après le cours, donner l'appel qui permet de reconstruire un tas maximum.
3. Écrire un algorithme `augmente_cle(T, i, x)` qui augmente la valeur de `T[i]` de la valeur `x` positive et ensuite modifie `T` par une suite d'échanges de cases, de façon à ce que `T` retrouve sa structure de tas maximum.