Université de Caen Normandie — Année 2018-2019
UFR des Sciences — seconde session

Licence Sciences et Technologies – 2ème année
MIM3C1, Algorithmique et structures de données
Session 2 – 24 juin 2019 – durée 2h

Les notes et documents de cours, TD et TP sont autorisés.
Les calculatrices et les téléphones portables sont interdits.
Le sujet fait **trois pages**.
Les trois exercices sont *indépendants* et les questions sont aussi largement indépendantes.
Attention à lire attentivement les énoncés et à bien gérer votre temps.

> Chaque candidat doit, en début d'épreuve, porter son nom dans le coin de la copie réservé à cet usage ; il le cachettera par collage après la signature de la feuille d'émargement. Sur chacune des copies intercalaires, il portera son numéro de place et numérotera chaque copie.

Point à respecter : Comme on l'a fait en cours et en TD, vous écrirez toutes les procédures demandées en **langage algorithmique**. Lorsque l'on demande une procédure itérative ou récursive, vous devez impérativement suivre la consigne. Les questions avec $*$ sont des questions qui peuvent présenter des difficultés.

Exercice 1. Listes simplement chaînées – temps indicatif 45 mn

On considère une liste d'altitudes d'une randonnée. Les altitudes sont mesurées en mètres et sont prises tous les 500 mètres. Par exemple la liste $(200, 250, 270, 230)$ que l'on appellera $R1$ signifie que la randonnée commence à 200m d'altitude, elle arrive à 250m d'altitude après 500m de randonnée, à 270m d'altitude après 1000m de randonnée et finalement se termine à 230m d'altitude après 1500m de randonnée.

Pour les questions suivantes, on supposera que les randonnées sont constituées de listes d'entiers naturels (entiers positifs ou nuls) contenant les altitudes en mètres. Pour coder ces listes, on utilise une structure *simplement chaînée* dont chaque nœud est une structure à deux attributs : un entier `valeur` et un pointeur sur nœud `suivant`.

Dans tout l'exercice, le coût d'une procédure sera le nombre de nœuds visités sur la ou les listes lors de l'exécution de celle-ci.

Question 1. a. Rappelez la définition de la structure `nœud` et du type `liste`.
b. Dessinez la structure chaînée qui représente la liste $R1$.
c. Donnez l'instruction pour accéder à la valeur 270 et l'instruction pour modifier 230 en remplaçant cette valeur par 220.

Question 2. a. Écrivez une procédure itérative `distanceRandoIt(R: liste): réel` qui retourne le nombre de kilomètres parcourus avec la randonnée R. On renvoie 0 lorsque la liste contient 0 ou 1 nœud.
b. Écrivez une procédure récursive `distanceRandoRec(R: liste): réel` de cette procédure.

Question 3. a. Écrivez une procédure itérative `sommetRando(R: liste): entier` qui retourne la plus haute altitude mesurée pour la randonnée R. On renvoie 0 lorsque la liste est vide.
b. Calculez la coût de cette procédure en fonction de k, le nombre de nœuds de R. Est-ce que ce coût dépend des altitudes contenues dans R ?

Question 4. Deux nœuds consécutifs d'une randonnée R constituent une montée lorsque le second nœud a une valeur supérieure ou égale à la valeur du premier nœud. Le dénivelé d'une montée est la différence entre les altitudes du second nœud et l'altitude du premier nœud.

a. Écrivez une procédure (itérative ou récursive) `montee(R: liste): booléen` qui renvoie `Vrai` lorsque R ne contient que des montées et `Faux` sinon. Vous renverrez `Vrai` si la liste contient 0 ou 1 nœud.
b. Quel est le coût de cette procédure lorsque R ne contient que des montées ? Dans le cas contraire, quel est le coût dans le meilleur des cas et dans le pire des cas ?

Question 5. * On appelle dénivelé cumulé positif d'une randonnée la somme des dénivelés des montées.
a. Écrivez une procédure itérative `denivele(R: liste): entier` qui retourne le dénivelé cumulé positif de R. Vous renverrez 0 si la liste contient 0 ou 1 nœud.
b. Donnez la trace de l'exécution de cette procédure sur $R1$ en précisant bien toutes les étapes.

Exercice 2. Arbres binaires et Arbres binaires de recherche – temps indicatif 40 mn

On souhaite utiliser un arbre binaire pour stocker les ordinateurs éventuellement disponibles dans un magasin.

On stocke pour chaque nœud deux valeurs : le prix et la disponibilité.

On utilise pour cela la structure de nœud suivante :

```
structure noeud{
     prix : entier
     disponible : booléen
     gauche : pointeur sur noeud
     droit : pointeur sur noeud
}
type arbreBinaire = pointeur sur noeud
```

Question 1. Écrivez une procédure récursive `disponiblesRec(A: arbreBinaire)` qui affiche selon l'ordre préfixe le prix des ordinateurs disponibles.

Question 2. * Écrivez une procédure itérative `disponiblesIt(A: arbreBinaire)` qui affiche selon l'ordre préfixe le prix des ordinateurs disponibles. Vous expliquerez la méthode utilisée.

Question 3. Écrivez une procédure `plusCher(A: arbreBinaire): entier` qui retourne le prix de l'ordinateur le plus cher.

Question 4. On souhaite maintenant que l'arbre binaire soit un ABR où l'attribut utilisé pour effectuer les comparaisons est le prix. On ajoute l'instruction

```
type ABR = pointeur sur noeud
```

a. Rappelez la condition sur les nœuds pour qu'un arbre binaire soit un ABR.
b. Dessinez un ABR de hauteur 2 avec les paires (prix,disponible) suivantes :
(500, `Faux`), (700, `Vrai`), (800, `Vrai`), (1000, `Vrai`), (1300, `Vrai`), (2000, `Faux`).

Question 5. Proposez une procédure affichant les ordinateurs disponibles par ordre croissant.

Question 6. a. Écrivez une procédure `plusCherABR(A: ABR): entier` qui retourne le prix de l'ordinateur le plus cher.
b. Comparez cette procédure avec celle de la question 3.
On note n le nombre de nœuds de l'ABR A et h sa hauteur. Donnez le coût maximal de l'algorithme (nombre de comparaisons) en fonction de h. Pour $h = 3$, donnez un ABR où $n = h + 1$ et un autre ABR où $n = 2^{h+1} - 1$.
Expliquez pourquoi, pour un nombre de nœuds fixé, nous devons construire un ABR avec la hauteur la plus petite possible.

Exercice 3. Tries ou arbres préfixes – temps indicatif 35 mn

Question 1. On considère l'ensemble de mots $\mathcal{E} =$ { mante, main, manuel, lac, lait, manteau, lactose, manne, manie }. Dessinez la forêt mémorisant cet ensemble de mots.

Question 2. On définit la structure NoeudF suivante

```
structure NoeudF{
   valeur : caractère
   enfant : pointeur sur noeudF
   fratrie : pointeur sur noeudF
   present : booléen
}
type trie = pointeur sur NoeudF
```

Dessinez $T1$ le trie obtenu avec cette stucture pour l'ensemble de mots $\mathcal{E}$.
Rappelez le rôle de chacun des attributs de `nœudF` pour un trie.
Comment vérifie-t-on qu'un nœud est une feuille de la forêt ?

Question 3. Redonnez la définition du degré d'un nœud d'un trie et le degré d'un trie.
Quel est le degré maximum d'un trie avec la structure NoeudF. Donnez le degré de $T1$.

Question 4. Exécutez la procédure suivante sur $T1$ et expliquez ce qu'elle retourne.

```
mystere (T : trie, k : entier) : entier
   si T = None alors
      retourner 0
   si T->fratrie = None alors
      retourner max(k, mystere(a->enfant, 1))
   retourner max(mystere(a->enfant,1), mystere(a->fratrie, k+1))

mystere(T1,1)
```

Question 5. * On dit qu'un mot est présent dans le trie lorsqu'il est formé par les caractères rencontrés sur le chemin allant de la racine jusqu'à un nœud dont l'attribut `present` prend la valeur `Vrai`. Un mot préfixe du trie est un mot présent dans le trie tel qu'il existe un autre mot présent dans le trie commençant par ce mot. Écrivez une procédure `motPrefixe(T: trie, m: chaîne de caractères): booléen` qui vérifie si m est un mot préfixe de T.

Question 6. Écrivez une procédure `plusPetit(T: trie): chaîne de caractère` qui affiche le plus petit mot présent dans T (selon l'ordre lexicographique) qui n'est pas un mot préfixe de T.

Question 7. Écrivez une procédure `plusGrand(T: trie): chaîne de caractère` qui affiche le plus grand mot présent dans T (selon l'ordre lexicographique) qui n'est pas un mot préfixe de T.