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

Licence Sciences et Technologies – 2ème année
MIM3C1, Algorithmique et structures de données
19 décembre 2018, durée 2h

Les notes et documents de cours, TD et TP sont autorisés.
Le sujet fait **quatre pages**.
Les quatre 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 40 mn

On considère des listes d'entiers naturels (entiers positifs ou nuls). Pour coder ces listes, on utilise une structure *simplement chaînée* dont chaque nœud est une structure à deux champs : 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. Rappelez la définition de la structure `nœud` et du type `liste`. Dessinez la structure chaînée qui représente la liste $L1 = (11, 5, 7, 2, 9, 6, 1)$. Donnez les deux valeurs de `L1->suivant->suivant->valeur`, `L1->suivant->suivant->suivant->suivant->valeur`.

Question 2. Écrivez une procédure itérative `longueurListe(L: liste): entier` qui retourne le nombre de nœuds de la liste L.

Question 3. On souhaite vérifier qu'une liste contient au plus 10 nœuds. On propose la procédure suivante

```
auPlusDix(L : liste) : booléen
   retourner longueurListe(L) <= 10
```

Donnez le coût de cette procédure en fonction du nombre de nœuds de la liste L. En déduire la classe de complexité de cette procédure. Pourquoi cette procédure n'est pas efficace ? Proposez une procédure itérative dont le coût maximimaum ne dépend pas de la taille de la liste.

Question 4. Donnez une procédure itérative `presentListeIt(L: liste, x: entier): booléen` qui retourne VRAI si x est une valeur de L et FAUX sinon. Proposez une version récursive de cette procédure.

Question 5. On suppose que deux listes $L1$ et $L2$ codent deux ensembles E_1 et E_2 (donc chaque liste est constituée de valeurs distinctes).

Utilisez la procédure `presentListeIt(L: liste, x : entier): booléen` pour écrire une procédure `sousEnsemble1(L1, L2: liste): booléen` qui retourne VRAI lorsque toutes les valeurs de $L1$ sont dans la liste $L2$ et FAUX sinon. Donnez le coût de cette procédure en fonction de n_1, le nombre de nœuds de $L1$ et n_2, le nombre de nœuds de $L2$.

Question 6. * On suppose maintenant que les listes $L1$ et $L2$ sont triées. Donnez une procédure `sousEnsemble2(L1, L2: liste) : booléen` qui retourne VRAI lorsque toutes les valeurs de $L1$ sont dans la liste $L2$ et FAUX sinon. Donnez le coût de cette procédure en fonction de n_1, le nombre de nœuds de $L1$ et n_2, le nombre de nœuds de $L2$. Par exemple si $L1 = (2, 4, 5)$ et $L2 = (1, 2, 4, 5, 7)$ la procédure retourne VRAI et si $L1 = (2, 3, 5)$ et $L2 = (1, 2, 4, 5, 7)$ la procédure retourne FAUX.

Exercice 2. Forêts et tries – temps indicatif 40 mn

Forêts et arbres généraux

On considère ici des forêts et des arbres généraux dont les nœuds ont pour valeur des entiers.

Question 1. Expliquez comment faire pour coder une forêt ou un arbre général avec un arbre binaire. Redonnez la structure de nœud, adaptée à cette situation, que nous noterons `nœudF`. Définissez les types `arbreGeneral` et `foret`.

Question 2. Écrivez une procédure `nombreFeuilles(F: foret): entier` qui retourne le nombre de feuilles de F.

Question 3. Écrivez une procédure `affichageIntervalle(F: foret, a,b: entier)` qui affiche toutes les valeurs des nœuds comprises entre a et b au sens large.

Trie

Question 4. On considère l'ensemble de mots $\mathcal{E}$ = { demago, demain, demande, demander, demandeur, demandeuse, labo, labour, laboureur, labrador }. Montrez comment mémoriser cet ensemble de mots avec une forêt.

Codez cette forêt avec un arbre binaire. On obtient ainsi un `trie` que nous nommerons $T1$. Redonnez la structure de nœud des tries que l'on notera `nœudT` et le type `trie`. Rappelez le rôle de chacun des attributs de `nœudT`.

Question 5. Donnez une procédure `nombremots(T: trie): entier` qui retourne le nombre de mots de T. Vous ferez impérativement le parcours par ordre lexicographique décroissant. Vous donnerez le coût de la procédure en fonction du nombre de nœuds. Peut-on trouver un meilleur algorithme ? Justifiez votre réponse.

Question 6. * On suppose ici que la valeur des nœuds n'est pas un caractère mais une chaîne de caractères.
Expliquez ce que fait la procédure suivante. On supposera qu'elle s'applique à un sous-arbre non vide.
On applique cette procédure au trie $T1$ défini à la question 4. Donnez la valeur de la racine de $T1$ après avoir appliqué cette procédure.

```
contraction(T : trie): trie
   si T->enfant != None alors
      tmp: trie ; tmp = T->enfant
      tant que tmp != None et tmp->fratrie = None faire
         T->valeur = T->valeur + tmp->valeur   # + désigne la concaténation
         tmp = tmp->enfant                              # de chaînes de caractères
      T->enfant = tmp
   retourner T
```

Question 7. * On considère la procédure `elastique(T: trie):trie` suivante :

```
elastique(T: trie): trie
   si T != None alors
      T = contraction(T)
      T->enfant = elastique(T->enfant)
      T->fratrie = elastique(T->fratrie)
  retourner T
```

Expliquez ce que fait cette procédure.
Donnez le trie que vous obtenez après avoir effectué la procédure `elastique` sur le trie $T1$.

Exercice 3. Arbres binaires de recherche – temps indicatif 25 mn

On considère dans cet exercice des arbres binaires de recherche (ABR) dont les valeurs sont des entiers. On définit comme coût d'une procédure sur un ABR le nombre de comparaisons entre deux entiers.

Question 1. Redonnez la définition d'un ABR dans le cas où toutes les valeurs sont différentes.

Question 2. Dessinez un ABR ayant 7 nœuds et une hauteur 2.

Question 3. Donnez les trois opérations principales que l'on effectue sur un ABR. On souhaite que ces trois opérations aient une complexité dans le pire des cas en $\Theta(\log n)$. Pour cela, quelle contrainte doit-on avoir sur la hauteur des ABR considérés par rapport au nombre de nœuds ? Justifiez votre réponse.

Question 4. Redonnez la procédure récursive `presentABR(A: ABR, x: entier): ABR` qui retourne `None` si la valeur x n'est pas présente et sinon retourne un pointeur sur le nœud contenant la valeur x. Modifiez cette procédure afin d'obtenir une procédure récursive `cheminABR(A: ABR, x: entier): liste` qui retourne une liste chaînée contenant les valeurs des nœuds comparées avec x.

Dans quel cas cette procédure retourne-t-elle une branche complète de l'arbre ?

Exercice 4. Algorithmes de tri – temps indicatif 15mn

Question 1. Donnez les différentes classes de complexité en moyenne des algorithmes de tri que vous connaissez. Indiquez un algorithme pour chaque classe.

Question 2. * Quelle probabilité sur les tableaux à trier en entrée est proposée pour calculer la complexité en moyenne des algorithmes de tris ?
Donnez la complexité dans le pire des cas et la complexité en moyenne de QUICKSORT.
Comment est le tableau en entrée dans le pire des cas ?
Pour quelle raison la complexité en moyenne a plus d'importance ?