Université de Caen Normandie — Année 2017-2018
UFR des Sciences — 2ème session

Licence Sciences et Technologies – 2ème année
MIM3C1, Algorithmes et structures de données
Décembre 2017, durée 2h

Les notes et documents de cours, TD et TP sont autorisés.
Le sujet fait **4 pages**.
Les 5 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 les procédures demandées en **langage algorithmique** (pseudo-code).

Exercice A. Listes simplement chaînées (durée indicative : 35 minutes)

Dans cet exercice, on considère des *listes d'entiers* supposés positifs ou nuls. Comme on le fait habituellement, on représente une telle liste par une structure *simplement chaînée* dont chaque nœud est une structure à deux champs : `val` un entier et `suiv` un pointeur sur nœud.

Question 1. a. Dessinez la structure chaînée qui représente la liste $(3, 7, 3, 4)$ qu'on appellera `L`. N'oubliez pas le pointeur qui donne accès au premier nœud de la liste.

b. Sur cet exemple, précisez les trois valeurs `L->suiv->val`, `L->suiv->suiv->suiv->val` et `L->suiv->suiv->suiv->suiv`.

Question 2. Ecrivez une procédure `ConstruitListe(x: entier, L: liste): liste` qui retourne une liste dont le premier élément est `x` et dont la suite de la liste est `L`. Par exemple, pour `x = 7` et `L = (3,7,3,4)`, la procédure retourne la liste `(7,3,7,3,4)`. Quelle est la complexité de votre procédure ?

Question 3. a. Ecrivez une procédure *récursive* `Produit(L: liste): entier` qui retourne le produit des éléments de la liste `L`. Par exemple, le produit des éléments de la liste $(5, 2, 6, 1)$ est $5 \times 2 \times 6 \times 1 = 60$. Si `L` est la liste vide, on convient que le produit de ses éléments est `1`.

b. Ecrivez la même procédure sous forme *itérative*. En voici l'entête :
`ProduitIteratif(L: liste): entier`.

Question 4. a. Ecrivez une procédure *récursive* `KièmeElement(L: liste, k: entier): entier` qui retourne le `k`-ième élément de la liste `L`, s'il existe, et retourne `-1` sinon.
Par exemple, pour la liste `L = (5,2,6,1)`, la procédure retourne son premier élément, donc `5`, si `k = 1` ; la procédure retourne `-1` si `k = 6` puisque `L` n'a pas de 6ème élément.
Indication : Considérez 3 cas : 1) cas `L` vide ou `k < 1`, et sinon : 2) cas `k = 1` ; 3) cas `k > 1`.

b. Donnez la complexité de votre procédure en fonction de `k` et/ou de la longueur `n` de la liste `L`. Justifiez votre réponse.

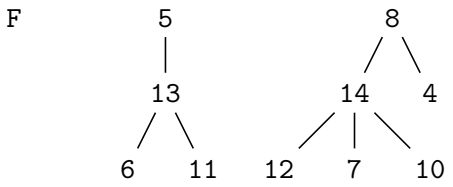
Question 5. Ecrivez une procédure *récursive*
`InsereAvant(L: liste, x: entier, y:entier): liste` qui insère dans une liste `L` un élément `x` *avant* un élément `y` et retourne la liste ainsi modifiée. La procédure insère `x` *juste avant la*

première occurrence de `y`. Par exemple, si `L = (7,3,7,3,4)` et si `x = 5` et `y = 3`, alors la procédure retourne la liste modifiée `(7,5,3,7,3,4)`. Si `y` n'est pas présent dans `L`, par exemple si `y = 8`, alors la procédure retourne `L` sans modification.
Indication : Considérez 3 cas : 1) `L` vide, et sinon : 2) `L->val = y` ; 3) `L->val` différent de `y`.

Exercice B. Forêts (durée indicative : 30 minutes)

Question 6. Pour la forêt `F` ci-dessous formée de 2 arbres (arbres généraux), donnez la liste des valeurs des nœuds :

- **a.** en ordre *préfixe*,
- **b.** en ordre *postfixe* (on dit aussi ordre *suffixe*),
- **c.** dans l'ordre du *parcours en largeur*. (*Rappel :* Le parcours en largeur d'une forêt `F` consiste à parcourir d'abord les nœuds de niveau 0, donc les racines de la forêt, puis ceux de niveau 1, etc.) Vous préciserez la liste des nœuds de niveau 0 de `F`, la liste des nœuds de niveau 1, etc.



Question 7.
- **a.** Rappelez la définition de la représentation chaînée *fils-frère* d'un arbre général ou d'une forêt. On supposera que les valeurs des nœuds sont des entiers.
- **b.** Dessinez la représentation fils-frère de la forêt `F` donnée ci-dessus.

Dans toute la suite de l'exercice, les forêts sont supposées être représentées sous la forme fils-frère.

Question 8. Ecrivez une procédure `AffichePrefixe(F: forêt)` qui affiche les valeurs des nœuds d'une forêt `F` en ordre *préfixe*.

Question 9.
- **a.** On rappelle qu'un nœud d'une forêt est *interne* s'il a au moins un fils. Donnez la liste des *nœuds internes* de la forêt représentée ci-dessus.
- **b.** Complétez les trois parties notées `.....` de la procédure *récursive* donnée ci-dessous qui affiche les nœuds internes d'une forêt `F` *non vide* quelconque. Justifiez vos réponses.

```
AfficheNoeudsInternes(F: forêt) # F est supposée non vide
    si F->fils != None alors
        afficher(......);  ......
    si F->frère != None alors ......
```

- **c.** Combien de fois chaque pointeur de la représentation fils-frère de la forêt est-il examiné par la procédure ? Si `n` est le nombre de nœuds de la forêt combien de tests de pointeur (le pointeur est-il différent de `None` ?) la procédure fait-elle exactement ? Justifiez vos réponses.

Exercice C. Arbres binaires de recherche (durée indicative : 20 minutes)

On considère dans cet exercice des arbres binaires de recherche (ABR) dont les valeurs sont des entiers.

Question 10. a. Dessinez un ABR de hauteur 2 et contenant les sept valeurs `1,3,6,7,9,12,14`.

b. Définissez ce qu'est un ABR (définition du cours). *Rappel :* C'est une définition *récursive*.

c. Quel parcours (préfixe, infixe, postfixe/suffixe, parcours en largeur ?) d'un ABR doit on effectuer pour afficher ses valeurs en ordre croissant ? Justifiez votre réponse.

Question 11. a. Dans un ABR où se trouve le nœud de valeur minimale ? Dessinez un exemple d'ABR dont le nœud de valeur minimale n'est pas une feuille.

b. Ecrivez une procédure *récursive* `SupprimeMinABR(A: ABR): (entier,ABR)` qui, pour un ABR `A` supposé *non vide*, retourne le couple formé de la valeur minimale de l'ABR `A` et de l'ABR où on a supprimé cette valeur minimale.
Indication : Considérez 2 cas : 1) `A->gauche = None` ; 2) `A->gauche` différent de `None`.

c. Donnez la complexité en temps (dans le pire des cas) de cette procédure en fonction de `n`, le nombre de nœuds, et/ou de `h`, la hauteur de l'arbre. Justifiez votre réponse.

Exercice D. Tri d'un tableau (durée indicative : 15 minutes)

On rappelle l'algorithme de tri-sélection (ou tri par sélection) d'un tableau `T[0..n-1]` (tableau de `n` éléments, entiers par exemple), vu en TP et en TD.

```
Donnée: un tableau d'entiers T[0..n-1]
variables utilisées: i, imin, j: entiers
pour i variant de 0 à n-2 faire
    imin=i
    pour j variant de i+1 à n-1 faire
        si T[j]<T[imin] alors imin=j
    (T[i],T[imin])= (T[imin],T[i]) # on échange les éléments T[i] et T[imin]
```

Question 12. Rappelez (en quelques mots) pourquoi cet algorithme trie correctement le tableau `T[0..n-1]` en ordre croissant. *Indication :* Rappelez d'abord quelle est la valeur de `imin` après l'exécution des instructions :

```
imin=i
pour j variant de i+1 à n-1 faire
    si T[j]<T[imin] alors imin=j
```

Question 13. Combien l'algorithme fait-il de comparaisons d'éléments du tableau (nombre de tests : `si T[j]<T[imin]...`) ? Vous donnerez ce nombre en fonction de `n`.

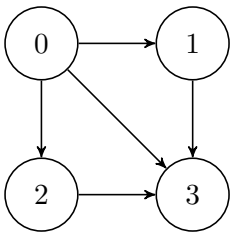
Question 14. a. Par quel nombre environ le nombre de comparaisons est-il multiplié quand, au lieu de trier un tableau de `n` éléments, on trie un tableau de `10n` éléments, toujours par le tri-sélection ? Le tri-sélection est-il efficace pour trier un tableau de grande taille ? Justifiez votre réponse.

b. Donnez un exemple de tri plus efficace que le tri-sélection, pour un tableau de grande taille, et précisez la complexité de ce tri (sans donner de justification).

TOURNEZ LA PAGE SVP →

Exercice E. Graphe (durée indicative : 15 minutes)

Question 15. *Etude d'un exemple :* Soit le graphe orienté $\mathtt{G} = (\mathtt{S}, \mathtt{A})$ à $\mathtt{n} = 4$ sommets où $\mathtt{S} = \{0, 1, 2, 3\}$ et $\mathtt{A} = \{(0, 1), (0, 2), (0, 3), (1, 3), (2, 3)\}$ représenté ci-dessous :



a. Dessinez la représentation de ce graphe par tableau de listes de successeurs.

b. Donnez la matrice d'adjacence de ce graphe.

Question 16. *Cas général :*

a. Donnez la taille de la représentation d'un graphe orienté par tableau de listes de successeurs en fonction de son nombre `n` de sommets et de son nombre `p` d'arcs.

b. Comparez la taille de la représentation d'un graphe orienté par son tableau de listes de successeurs avec la taille de sa matrice d'adjacence quand son nombre `n` de sommets est grand (par exemple, $\mathtt{n} = 10^6$) alors que son nombre `p` d'arcs est inférieur à `10n`.