Examen écrit de structures de données

G. Falquet, C.-L. Mottaz Jiang, semestre d'été 2002

NOM : ______________

PRÉNOM: ______________

Tous les documents sont autorisés
Durée : trois heures
Ecrivez vos réponses directement sur l'énoncé

Instructions pour l'écriture des algorithmes

Ecrire les algorithmes en *Java* ou en *pseudo-code*

- L'algorithme doit être précis
- Pas d'ambiguïtés
- Ne pas supposer que le lecteur est intelligent

Le pseudo-code doit obligatoirement être structuré selon les règles ci-dessous.

Instructions

Séquence	Instruction ; instruction ; …
Condition	si (condition) instruction; … sinon instruction; …
Itération	tant que (condition) instruction; … OU pour v de initial à final instruction; …
Affectation	var = expression OU var ← expression
Appel de fonctions ou de méthodes	z = f(x) objet . méthode (paramètres)

Vous ne pouvez utiliser que les *méthodes* et *fonctions*
- mentionnées dans l'énoncé
- usuelles (min, max, abs, racine, etc.)
- que vous avez vous-même définies

Variables

Déclarer les types des *variables*, par exemple : entier x, Ensemble y

Les *types* utilisables sont :
- les types prédéfinis (entier, flottant, chaîne, booléen)
- les tableaux (de types prédéfinis ou d'objets)
  entier[ ] tab1, Route[ ] réseau
- les classes fournies dans l'énoncé (cf annexes)

Commentaires

Expliquez vos algorithmes, soit par un texte (bref), soit en insérant des *commentaires*. Par exemple :

```
// On cherche l'élément supérieur à Z
Tant que x[i] <= z { i = i + 1 }
E = x[i]
// Ensuite on copie la partie du tableau qui …
```

Bien distinguer *commentaires* et *instruction* (// ou /* … */ ou remarque …)

NOM : ________________

Nombres entiers (1 pt)

a) Supposons que le type ENTIER soit représenté sur 16 bits, en complément à 2.

- Quel est le plus grand entier positif représentable ?

- Si X est ce plus grand entier, quel sera le résultat de l'opération X+1 ? (on suppose que c'est l'arithmétique modulaire habituelle qui est utilisée)

b) Supposons maintenant qu'on ne connaisse pas le nombre de bits utilisés pour représenter le type ENTIER.
Ecrivez un algorithme qui détermine quel est ce nombre de bits (utilisez ce que vous avez fait dans la première partie de cette question). L'algorithme doit être efficace : si n est le nombre de bits, il doit prendre un temps de l'ordre de n et non pas de 2^n.

Algorithmes (1 pt)

La suite de Fibonacci est composée des nombres F_1, F_2, F_3, ... obéissant à la règle
$F_1 = 1$
$F_2 = 1$
si $i > 2$, $F_i = F_{i-2} + F_{i-1}$

Le début de la suite est donc :

1, 1, 2, 3, 5, 8, 13, 21, 34, ...

Pour calculer le $n^{ème}$ nombre de la suite on décide d'utiliser l'algorithme récursif suivant :

```
entier fonction F(n : entier)  {
   si (n = 1 ou n = 2) retourne 1
   sinon retourne F(n-2) + F(n-1)
}
```

a) Expliquez pourquoi cet algorithme est très inefficace.

b) Ecrivez un algorithme beaucoup plus efficace pour calculer F(n), par exemple en utilisant l'idée de la programmation dynamique (sous une forme très simple) .

NOM : ________________

Fonctions (Maps) (1 pt)

Ecrivez la méthode `union(Fonction f)` qui réalise l'union de deux fonctions.
On suppose que les deux fonctions ont des clés de type `T` et des valeurs de type `ensemble d'éléments de type T`

Exemple
si

```
f1 =   {  a → { ab, ad, af } ,
          b → { bo, bu, br } ,
          d → { do, de } }
```

et

```
f2 =   {  d → { du, di, do } ,
          c → { ce, ci, ca } ,
          a → { ar, af } }
```

alors, après l'instruction `f1.union(f2),`

```
f1 =   {  a → { ab, ad, af, ar } ,
          b → { bo, bu, br } ,
          d → { do, de, du, di } ,
          c → { ce, ci, ca } }
```

Remarques

- Le type Fonction est donné en annexe. Vous pouvez également utiliser le type Map de Java.
- Attention ! Ne pas confondre avec la méthode `public void putAll(Map t)` des classes HashMap et TreeMap de Java. Avec putAll, si une clé de `f2` existe déjà dans `f1`, la valeur associée à cette clé dans `f1` sera "écrasée" par celle qui est associée à la clé dans `f2`. Ici, si une clé de `f2` existe déjà dans `f1`, on va faire l'union de leurs valeurs.

NOM : _________________

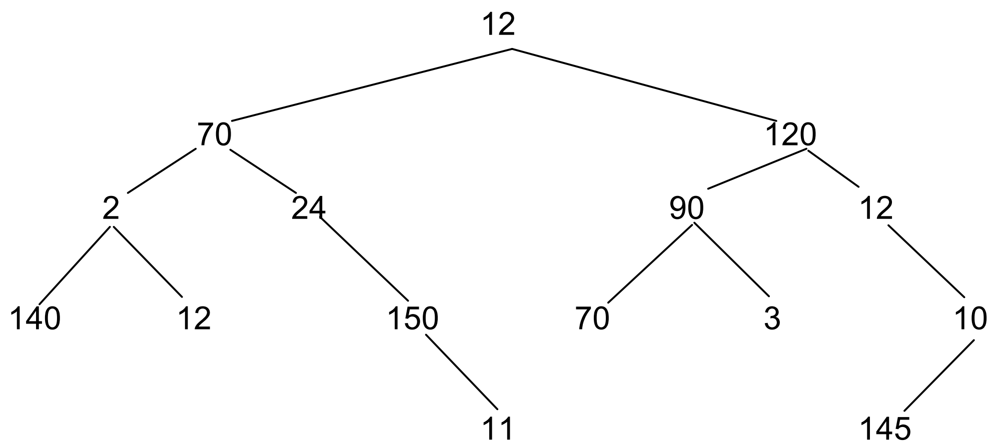
Arbres (1.5 pt)

On considère un arbre binaire (de nombres entiers) qui est implémenté à l'aide de la structure de données suivante :

```
classe Arbre
     // variables d'instance
     int valeur ;   // valeur du nœud racine
     Arbre gauche ; // sous-arbre de gauche
     Arbre droite ; // sous-arbre de droite
```

a) Ecrivez la méthode `int plusGrandNiveau(int valeurRecherchée, int niveauCourant)` qui va chercher le niveau le plus élevé de `valeurRecherchée` dans l'arbre.
Remarque : la racine de l'arbre correspond au niveau 0.

Exemple
Si on a l'arbre ci-dessous, `plusGrandNiveau(12, 0)` doit rendre 3.



NOM : _________________

Arbres (suite)

b) Ecrivez une méthode `Liste parentPlusPetit(int valCourante, Liste l)` qui construit la liste des valeurs de nœuds, dont le parent a une valeur plus petite.

Pour l'arbre ci-dessus, `parentPlusPetit(0, new Liste)`
doit rendre `[70, 140, 12, 150, 120, 145]`

Annexe : définition des types

Type **Liste** d'éléments de type T (un type quelconque)

résultat	opération	paramètres	effet
Liste	new Liste		crée une liste vide
Liste	insérer	int i, T elem	insère elem à la position i
Liste	remplacer	int i, T elem	remplace le ie élément par elem
Liste	supprimer	int i	supprime l'élément à la position i
T	element	int i	élément se trouvant à la position i
int	taille		taille de la liste (nombre d'éléments)
boolean	estVide		teste si la liste est vide (longueur=0)

Type **Ensemble** d'éléments de type T

résultat	opération	paramètres	effet
Ensemble	new Ensemble		crée un ensemble vide
boolean	ajoute	T elem	ajoute elem à l'ensemble; retourne vrai si elem a été ajouté et faux si elem était déjà dans l'ensemble
	ajouteTout	Ensemble e	ajoute tous les éléments de e
boolean	retire	T elem	retire l'élément de l'ensemble; retourne vrai si elem a été retiré et faux si elem n'était pas dans l'ensemble
int	taille		cardinalité de l'ensemble (nombre d'éléments)
boolean	estVide		teste si l'ensemble est vide
boolean	appartient	T elem	teste si elem appartient à l'ensemble

Type **Fonction** (Map)

résultat	opération	paramètres	effet
Fonction	new Fonction		crée une fonction vide
	lier	TC c, TV v	ajoute la paire ($c \to v$) à la fonction
TV	delier	TC c	supprime la paire dont la clé est c, retourne l'ancienne valeur de c
int	taille		cardinalité de la fonction (nombre de paires)
boolean	estVide		teste si la fonction est vide
boolean	estLie	TC c	teste si la clé c existe dans la fonction
TV	valeur	TC c	retourne la valeur associée à la clé c
Ensemble	clés		retourne l'ensemble des clés de la fonction
Liste	valeurs		retourne une liste des valeurs de la fonction

Type **Itérateur** sur une collection (Ensemble, Liste) d'éléments de type T

résultat	opération	paramètres	effet
Iterateur	new Iterateur	Collection c	crée un nouvel itérateur sur la collection c
T	suivant		fournit l'élément suivant de la collection
boolean	encore		retourne vrai s'il reste des éléments à voir