Examen de méthodes de programmation, génie logiciel

mercredi 21 janvier 2004 - 3H

(tous les documents de cours, TD et TP sont autorisés)

De manière générale, n'oubliez pas de commencer par présenter ce que vous faites, les difficultés rencontrées puis de proposer un code qui implémente les solutions envisagées.

1- Logique (3pts)

Codez les phrases suivantes au moyen de quantificateurs :

a) tous les éléments du tableau T1 qui sont à des positions paires apparaissent dans le tableau T2 ;

b) le tableau T1 ne contient pas plus de 3 valeurs entières multiples de 3p ;

2- Étude d'un programme et codification en SML (7 pts)

On dispose d'un vecteur trié x contenant des entiers et d'un entier t. On exécute le code ci dessous. Que fait ce code ? Justifiez votre réponse en remplaçant par exemple, les ... par les valeurs appropriées.

```
l = -1; u = n;
while l+1 != u
   /* invariant : x[l] < ... && x[u] >= ... && l < ... */
   m = (l + u) / 2
   if x[m] < t
      l = m
   else
      u = m
/* assert : l+1 = ... && x[l] < ... && x[u] >= ... */
p = u
if p >= n || x[p] != t
   p = -1
```

Écrivez une fonction *récursive* SML qui prend en paramètre un tableau d'entiers, sa taille, un entier et deux indices et qui implémente le code précédent. Cette fonction retourne la valeur de p. On rappelle que la fonction sub du module Array permet de récupérer la valeur rangée à une position donnée. Le profil de l'opération est sub : 'a array * int -> 'a. De plus les tableaux SML sont indicés à partir de 0. N'oubliez pas de justifier ce que vous faites. Exemple de profil de la fonction : `fun foo(a:int array,n:int,t:int,l:int,u:int)=....`

Voici également un exemple d'utilisation du module Array :

```
open Array;
(* déclaration d'un tableau de 12 entiers initialisé à 0 *)
val a = array(12,0:int);
(* on initialise le tableau, case par cas *)
update(a,0,0);update(a,1,2);update(a,2,4);update(a,3,6);update(a,4,8);
update(a,5,10);update(a,6,12);update(a,7,14);update(a,8,16);update(a,9,16);
update(a,10,18);update(a,11,20);

sub(a,1); (* on retourne l'élément en position 1 : 2 *)
```

3- Programmation Python (10 pts)

3.1- Exemple de classe Python (2 pts)

Soit la classe suivante :

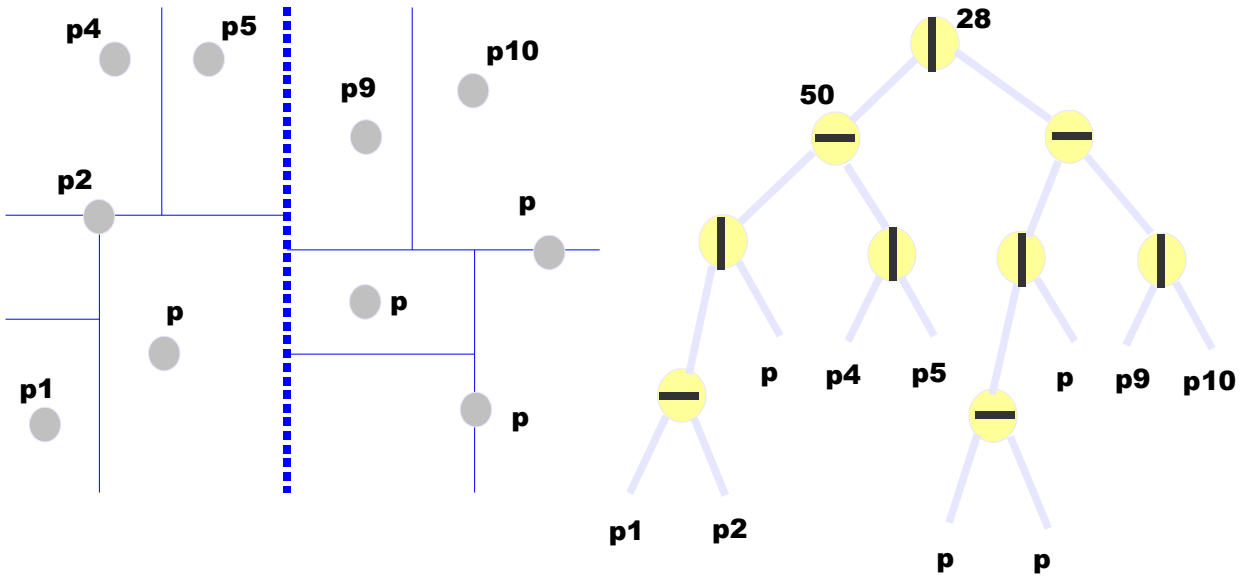
```
class Tree:
    def __init__(self,data1=None,data2=None,left=None,middle=None,right=None):
        self.data1 = data1
        self.data2 = data2
        self.left  = left
        self.middle  = middle
        self.right = right

    def __str__(self):
        print str(self.data1)+' '+str(self.data2)
        self.left.__str__()
        self.middle.__str__()
        self.right.__str__()
        return "\n"

    def A(self):
        root    = Tree(10,20)
        left    = Tree(5,6)
        middle  = Tree(11,18)
        right   = Tree(25,40)
        root.left   = left
        root.middle = middle
        root.right  = right
        return root

    def B(self,x):
      if (x==self.data1) or (x==self.data2):
            return 1
      else:
            if (self == None):
                return 0
            elif (x < self.data1):
                if (self.left != None):
                    return self.left.B(x)
                else: return 0
            elif (x < self.data2):
                if (self.middle != None):
                    return self.middle.B(x)
                else: return 0
            else:
                if (self.left != None):
                    return self.right.B(x)
                else: return 0
```

Ecrire les lignes de code permettant de créer une instance (initialisée à « l'objet vide ») de cette classe, puis d'appeler la méthode A. Dessinez graphiquement l'objet construit par invocation de la méthode A. Quel sera le résultat d'exécution après l'appel à la méthode B sur l'objet construit par l'invocation de la méthode A et avec 17 comme argument ?

3.2- Arbre 2D (8 pts)

On s'intéresse aux structures arborescentes appelées “Arbre-2D” qui permettent de stocker en mémoire des listes de points dans le plan. Pour ces arbres, il y a deux sortes de noeuds internes et on stocke les points aux feuilles. En fait, chaque noeud interne possède deux valeurs : un sens de coupure (vertical ou horizontal) et une valeur de coupure (soit une coordonnée en x soit une coordonnée en y). Pour

construire l'arbre de la figure qui suit à partir des points p1=(5,6), p2=(10,50), p4=(12,75), p3=(20,25), p5=(28,75), p6=(40,30), p9=(41,60), p10=(50,70), p7=(51,6), p8=(60,45) on procède comme suit :



La droite verticale en pointillé est la première ligne de coupure et elle permet de construire la racine de l'arbre. Tous les points à gauche de cette ligne ont des coordonnées en x inférieures ou égales à la position en x de cette droite en pointillé et tous les points à droite ont une coordonnée en x supérieure. On construit la racine avec pour valeur de coupure 28 et pour sens de coupure <<H>>. Puis on cherche à construire les fils de la racine. On considère la ligne de coupure horizontale (qui passe par p2 ici) pour le sous arbre gauche. On crée un noeud étiqueté par la coordonnée en y de p2 : 50. Tous les points en dessous de p2 se situent dans le sous arbre gauche de ce nouveau noeud, tous les points au dessus sont dans le sous arbre droit de ce nouveau noeud. On recommence la coupure jusqu'à n'avoir que des ensembles de points ne contenant qu'un seul point, dans ce cas le point est inséré.

Remarque : les noeuds internes permettent de se diriger dans un sous-arbre en fonction de la valeur de coupure qui y est rangée. La valeur de coupure est soit une coordonnée en x ou en y d'un point. Il y a alternance du sens de coupure d'un niveau de l'arbre au suivant.

Première question : écrire une classe Python pour ce type d'arbre. On distinguera les noeuds internes, les feuilles.

Deuxième question : écrire une fonction membre de la classe qui prend en paramètre un arbre et qui liste tous les points présents dans l'arbre. Cette fonction retourne une liste de points.

Troisième question : écrire une fonction membre de la classe qui recherche si un point est présent dans l'arbre. Cette fonction renvoie true ou false selon que l'on trouve ou pas.

Quatrième question : écrire une fonction membre de la classe qui, à partir d'une liste de points triés selon les x, insère les points dans l'arbre.

Cinquième Question : écrire une fonction membre de la classe qui à partir de deux points donne la hauteur du noeud père commun à ces deux noeuds. (La hauteur de la racine est 0). Par exemple, la hauteur retournée pour les points p2 et p4 est 1. Discutez des problèmes qui se posent et des solutions avant de les coder.

PROPOSITIONS DE CORRECTIONS

1- Logique (3pts)

Pas de difficulté.

2- Étude d'un programme et codification en SML (7 pts)

Le code proposé effectue la recherche de l'élément t entre les bornes l et u du tableau x. La méthode utilisée est la méthode dicotomique : on élimine la moitié des candidats à chaque test. On demande bien sûr que le tableau soit trié (ici en ordre croissant). On fait aussi l'hypothèse que x[-1]<t et que x[n] >= t. L'idée qui est codée est la suivante : à tout instant, on fait en sorte que les indices l et u « encadrent » la position de t ; on utilise l'invariant I : x[l]<t<=x[u] et l<u, ce qui donne :

```
l = -1; u = n;
while l+1 != u
   /* invariant : x[l] < t && x[u] >= t && l < u */
   m = (l + u) / 2
   if x[m] < t
      l = m
   else
      u = m
/* assert : l+1 = u && x[l] < l && x[u] >= t */
p = u
if p >= n || x[p] != t
   p = -1
```

Sous les hypothèses que nous avons prises, les initialisations du début (ligne 1) permettent de vérifier l'invariant I. L'invariant est maintenu après l'exécution soit de la branche *if* soit de la branche *else* du corps de boucle.

Lorsqu'on termine, on sait que, si t est présent alors sa première occurence est en position u (c'est l'affectation *u = m*) qui permet de garantir la propriété de « première occurence ». Les deux affectations qui suivent la boucle permettent de renvoyer -1 si l'élément n'est pas présent ou bien l'indice de t. Avec ce code nous avons donc un moyen de vérifier si un élément recherché est présent ou pas.

Le code SML qui implémente l'idée précédente par une fonction récursive est alors le suivant :

```
open Array;

val a = array(12,0:int);

update(a,0,0);
update(a,1,2);
update(a,2,4);
update(a,3,6);
update(a,4,8);
update(a,5,10);
update(a,6,12);
update(a,7,14);
update(a,8,16);
update(a,9,16);
```

```
update(a,10,18);
update(a,11,20);

fun dico(a:int array,n:int,t:int,l:int,u:int)=
     if (l+1 = u) then
          if (u >= n) orelse (sub(a,u) <> t) then ~1 else u
     else let val m = (l+u) div 2 in
                  if sub(a,m) < t then
                     dico(a,n,t,m,u)
                  else
                     dico(a,n,t,l,m)
             end;

val res = dico(a,length(a),2,~1,length(a));
print ("2 est en position " ^ Int.toString(res) ^ "\n");
val res = dico(a,length(a),4,~1,length(a));
print ("4 est en position " ^ Int.toString(res) ^ "\n");
val res = dico(a,length(a),6,~1,length(a));
print ("6 est en position " ^ Int.toString(res) ^ "\n");
val res = dico(a,length(a),8,~1,length(a));
print ("8 est en position " ^ Int.toString(res) ^ "\n");
val res = dico(a,length(a),10,~1,length(a));
print ("10 est en position " ^ Int.toString(res) ^ "\n");
val res = dico(a,length(a),11,~1,length(a));
print ("11 est en position " ^ Int.toString(res) ^ "\n");
val res = dico(a,length(a),14,~1,length(a));
print ("14 est en position " ^ Int.toString(res) ^ "\n");
val res = dico(a,length(a),16,~1,length(a));
print ("16 est en position " ^ Int.toString(res) ^ "\n");
val res = dico(a,length(a),18,~1,length(a));
print ("18 est en position " ^ Int.toString(res) ^ "\n");
val res = dico(a,length(a),20,~1,length(a));
print ("20 est en position " ^ Int.toString(res) ^ "\n");
val res = dico(a,length(a),22,~1,length(a));
print ("22 est en position " ^ Int.toString(res) ^ "\n");
val res = dico(a,length(a),12,~1,length(a));
print ("12 est en position " ^ Int.toString(res) ^ "\n");

a;

(*

Rsultats d'excution :

2 est en position 1
val it = () : unit
val res = 2 : int
4 est en position 2
val it = () : unit
val res = 3 : int
6 est en position 3
val it = () : unit
val res = 4 : int
8 est en position 4
val it = () : unit
val res = 5 : int
```

```
10 est en position 5
val it = () : unit
val res = ~1 : int
11 est en position ~1
val it = () : unit
val res = 7 : int
14 est en position 7
val it = () : unit
val res = 8 : int
16 est en position 8
val it = () : unit
val res = 10 : int
18 est en position 10
val it = () : unit
val res = 11 : int
20 est en position 11
val it = () : unit
val res = ~1 : int
22 est en position ~1
val it = () : unit
val res = 6 : int
12 est en position 6
val it = () : unit
val it = [|0,2,4,6,8,10,12,14,16,16,18,20|] : int array
*)
```

3- Programmation Python (10 pts)

3.1- Exemple de classe Python

```
class Tree:
    def __init__(self,data1=None,data2=None,left=None,middle=None,right=None):
        self.data1 = data1
        self.data2 = data2
        self.left  = left
        self.middle  = middle
        self.right = right

    def __str__(self):
        print str(self.data1)+' '+str(self.data2)
        self.left.__str__()
        self.middle.__str__()
        self.right.__str__()
        return "\n"

    def A(self):
        root    = Tree(10,20)
        left    = Tree(5,6)
        middle  = Tree(11,18)
        right   = Tree(25,40)
        root.left   = left
        root.middle = middle
        root.right  = right
        return root

    def B(self,x):
      if (x==self.data1) or (x==self.data2):
            return 1
      else:
            if (self == None):
                return 0
```

```
            elif (x < self.data1):
                if (self.left != None):
                    return self.left.B(x)
                else: return 0
            elif (x < self.data2):
                if (self.middle != None):
                    return self.middle.B(x)
                else: return 0
            else:
                if (self.left != None):
                    return self.right.B(x)
                else: return 0

t = Tree()
t1 = t.A()

print t1.B(17)
print t1.B(18)
print t1
```

Résultat d'exécution :

```
[christophe@localhost licenc]$ python tree1.py
0           => l'élément 17 n'est pas présent dans l'arbre
1           => l'élément 18 est présent
10 20      \
5 6        |
11 18      | => affichage des valeurs dans les noeuds
25 40      /
```

3.2- Arbre 2D

Voici une proposition de classe. Explicitez toutes les étapes des constructions algorithmiques.

```
class _2Dtree:
    """ Classe arbre 2D : un noeud contient data=(x,y) avec
        x un sens de coupure et y la valeur de coupure. Les variables
        left et right contiennent les fils gauche et droit du noeud """
    def __init__(self, data=None, left=None, right=None):
        self.data = data
        self.left  = left
        self.right = right

    def __str__(self):
        return str(self.data)

    def print_leaf(self, tree):
        """ affiche les feuilles de l'arbre et retourne la liste
            des feuilles """
        if tree is None: return []
        if(tree.left == None and tree.right==None):
            print tree.data
            return [tree.data]
        else:
            return self.print_leaf(tree.left) + self.print_leaf(tree.right)

    def print_tree(self, tree):
        if tree is None: return 0
        self.print_tree(tree.left)
        print tree.data,
        self.print_tree(tree.right)

    def print_postorder(self, tree):
        if tree is None: return 0
```

```
    self.print_postorder(tree.left)
    self.print_postorder(tree.right)
    print tree.data,

def print_preorder(self, tree):
    if tree is None: return 0
    print tree.data,
    self.print_preorder(tree.left)
    self.print_preorder(tree.right)

def search(self,(x,y)):
    """ Recherche si un point est dans l'arbre """
  if None==self:
        return 0
  else:
        #print self.data, self.left, self.right
        if (self.left != None) and (self.right != None):
                if(self.data[0] == "V"):
                    if (x<=self.data[1]):
                        return self.left.search((x,y))
                    else:
                        return self.right.search((x,y))
                else:
                    if(y<=self.data[1]):
                       return  self.left.search((x,y))
                    else:
                       return  self.right.search((x,y))
        elif (self.left == None) and (self.right == None):
              return (self.data[0] == x) and (self.data[1] == y)
        elif (self.left == None):
              return self.right.search((x,y))
        else:
              return self.left.search((x,y))

def commun(self,(x,y),(z,t),hauteur):
    """ Calcul de la hauteur du pere commun aux deux feuilles (x,y) et
        (z,t). Les points sont présents dans l'arbre """
  if None==self:
        return -1
  else:
        # on teste si on est sur une feuille
            if (self.left == None) and (self.right == None):
                if (((self.data[0] == x) and (self.data[1] == y)) or
                    ((self.data[0] == z) and (self.data[1] == t))):
                       return hauteur
                else: return -1
        else:
              h_left  = self.left.commun((x,y),(z,t),hauteur+1)
              h_right = self.right.commun((x,y),(z,t),hauteur+1)
                    #print h_left, h_right
                    if h_left == -1 and h_right == -1:
                        return -1
                    elif h_left == -1:
                        return self.right.commun((x,y),(z,t),hauteur+1)
                    elif h_right == -1:
                        return self.left.commun((x,y),(z,t),hauteur+1)
                    else:
                        return hauteur

def split(self,l):
    """ On decoupe la liste l en 2 parties, l'une contenant
        les éléments de l aux positions paires, l'autre contenant
        les éléments de l aux positions impaires """
    if(l==[]): return ([],[])
    elif l.__len__() == 1:
        return (l,[])
    else:
        res = self.split(l[2:])
```

```
        # print res
        if(res[0] == []) and (res[1] == []):
            return (l[0:1],l[1:2])
        elif(res[0] == []):
             res[1].insert(0,l[1])
             res = (l[0:1],res[1])
             return res
        elif(res[1] == []):
            res[0].insert(0,l[0])
            res = (res[0],l[1:2])
            return res
        else:
             res[0].insert(0,l[0])
             res[1].insert(0,l[1])
             return res

def mergeY(self,M,N):
    """ Fusionne des listes de tuples (x,y) triées selon les y. Ex:
    >>> t.mergeY([(1,1), (1, 4), (1, 6)], [(1, 2), (1, 5)])
    [(1, 1), (1, 2), (1, 4), (1, 5), (1, 6)]
    """
    if(M == []): return N
    elif (N == []): return M
    else:
        if(M[0][1] < N[0][1]):
            res = self.mergeY(M[1:],N)
            res.insert(0,M[0])
            return res
        else:
            res = self.mergeY(M,N[1:])
            res.insert(0,N[0])
            return res

def mergeX(self,M,N):
    """ Fusionne des listes de tuples (x,y) triées selon les x. Ex:
     >>> t.mergeX([(1,9),(4,9),(6,9)],[(2,9),(3,9),(7,9)])
     [(1, 9), (2, 9), (3, 9), (4, 9), (6, 9), (7, 9)]
    """
    if(M == []): return N
    elif (N == []): return M
    else:
        if(M[0][0] < N[0][0]):
            res = self.mergeX(M[1:],N)
            res.insert(0,M[0])
            return res
        else:
            res = self.mergeX(M,N[1:])
            res.insert(0,N[0])
            return res

def mergeSort(self,L,sens):
    if(L == []):
        return []
    elif L.__len__() == 1:
        return L
    else:
        res = self.split(L)
        res1 = self.mergeSort(res[0],sens)
        res2 = self.mergeSort(res[1],sens)
        # print res, res1, res2
        if(sens == "V"):
            return self.mergeX(res1,res2)
        else:
            return self.mergeY(res1,res2)

def build_tree(self,L,sens):
    long  = len(L)
    left  = L[0:(long+1)/2]
```

```
        right = L[(long+1)/2:]
        if(long < 1):
            return _2Dtree()
        elif long == 1:
            return _2Dtree(left[0])
        else:
            # print long, left, right
            if(sens == "V"):
                return _2Dtree((sens,left[left.__len__()-1:][0][0]),
                          self.build_tree(self.mergeSort(left,"H"),"H"),
                          self.build_tree(self.mergeSort(right,"H"),"H"))
            else:
                return _2Dtree((sens,left[left.__len__()-1:][0][1]),
                          self.build_tree(self.mergeSort(left,"V"),"V"),
                          self.build_tree(self.mergeSort(right,"V"),"V"))

#
# Début du programme principal
#

liste_point    =    [(5,6),(10,50),(12,75),(20,25),(28,75),(40,30),(41,60),(50,70),
(51,6),(60,45)]
t = _2Dtree()

MonArbre = t.build_tree(liste_point,"V")

ListeFeuille = MonArbre.print_leaf(MonArbre)
print ListeFeuille

if MonArbre.search((20,25)):
    print "(20,25) a été trouvé"
else: print "(20,25) pas trouvé"

if MonArbre.search((51,2)):
    print "(51,2) a été trouvé"
else: print "(51,2) pas trouvé"

H = MonArbre.commun((5,6),(28,75),0)
print "Hauteur ancetre commun à (5,6) et (28,75) = ",H

H = MonArbre.commun((10,50),(50,70),0)
print "Hauteur ancetre commun à (10,50) et (50,70) = ",H

H = MonArbre.commun((41,60),(50,70),0)
print "Hauteur ancetre commun à (41,60) et (50,70) = ",H

H = MonArbre.commun((5,6),(10,50),0)
print "Hauteur ancetre commun à (5,6) et (10,50) = ",H

"""
>>> execfile("arbre2D.py")
(5, 6)
(10, 50)
(20, 25)
(12, 75)
(28, 75)
(51, 6)
(40, 30)
(60, 45)
(41, 60)
(50, 70)
[(5, 6), (10, 50), (20, 25), (12, 75), (28, 75), (51, 6), (40, 30), (60, 45), (41,
60), (50, 70)]
(20,25) a été trouvé
(51,2) pas trouvé
Hauteur ancetre commun à (5,6) et (28,75) =  1
Hauteur ancetre commun à (10,50) et (50,70) =  0
Hauteur ancetre commun à (41,60) et (50,70) =  2
```

```
Hauteur ancetre commun à (5,6) et (10,50) =  3
>>>
"""
```