ISIMA 1ère année

3 septembre 2004
Durée : 2 heures
Documents autorisés

PROGRAMMATION FONCTIONNELLE

Exercice :
On appelle voisinage d'un élément `x` d'une liste `L` la sous-liste de `L` contenant, au plus, les trois éléments suivants, dans cet ordre : l'élément précédant `x` dans `L`, s'il existe ; l'élément `x` lui-même ; et l'élément suivant `x` dans `L`, s'il existe.
Ecrire une fonction `V` ayant comme arguments une liste `L` et un symbole `x`, telle que l'évaluation de l'expression `(V L x)` retourne le voisinage de `x` dans `L` si `x` est élément de `L`, et retourne la liste vide sinon.
Exemple :

- l'évaluation de `(V '(1 2 3 4) 1)` retourne `(1 2)` ;
- l'évaluation de `(V '(1 2 3 4) 2)` retourne `(1 2 3)` ;
- l'évaluation de `(V '(1 2 3 4) 4)` retourne `(3 4)` ;
- l'évaluation de `(V '(1 2 3 4) 5)` retourne `()` ;
- l'évaluation de `(V '(1) 1)` retourne `'(1)` ;

Solution :
```
(define V (lambda (L x)
   (if (and L (cdr L))
      (if (equal? x (car L))
         (list x (cadr L))
         (if (equal? x (cadr L))
            (if (cddr L) (list (car L) x (caddr L)) L)
            (V (cdr L) x) ) )
      (if (and L (equal? x (car L))) L ()) ) ))
```

Problème :
On considère, dans un espace métrique, un ensemble de points P_1, P_2, …, P_n, dont les distances deux à deux sont consignées dans une "matrice des distances" représentée par une liste de n sous-listes contenant chacune n valeurs réelles positives ou nulles telles que la $j^{ème}$ valeur de la $i^{ème}$ sous-liste représente la distance entre P_j et P_i. Etant donnée une valeur `dmin`, on peut construire un graphe dont les sommets sont les points P_1, P_2, …, P_n, et tel qu'il existe un arc entre deux points si leur distance est inférieure ou égale à `dmin`.
Le but de ce problème est d'écrire une fonction `CC` ayant comme argument une liste `MD` représentant une matrice des distances et un nombre réel `dmin`, telle que l'évaluation de l'expression `(CC MD dmin)` retourne la liste des composantes connexes du graphe défini ci-dessus, chaque composante connexe étant représentée par la liste des indices des points qui la compose.
Exemple : $n = 5$, P_1, …, P_5 sont alignés sur un axe, et d'abscisses respectives 1, 2, 3, 5 et 6, et `dmin` = 1,5.
L'évaluation de
```
(CC '((0 1 2 4 5)
      (1 0 1 3 4)
      (2 1 0 2 3)
```

```
      (4 3 2 0 1)
      (5 4 3 1 0))
```
1.5) doit retourner `((5 4) (3 2 1))` ou toute autre configuration équivalente représentant les composantes connexes du graphe reliant, parmi les points P_1 à P_5, ceux qui sont à une distance inférieure à 1,5.

1°) Ecrire une fonction `LI` ayant comme arguments une liste `L` et un prédicat `P`, telle que l'évaluation de `(LI L P)` retourne la liste des indices dans `L` (dans un ordre quelconque) des éléments de `L` satisfaisant le prédicat `P`.

Par exemple l'évaluation de `(LI '(a 6 b 9) integer?)` doit retourner `(4 2)`.

Solution :

```
(define LI (lambda (L P) (LI2 L P () 1)))
(define LI2 (lambda (L P LI i)
   (if L
      (if (P (car L)) (LI2 (cdr L) P (cons i LI) (+ i 1))
                      (LI2 (cdr L) P LI (+ i 1)) )
      LI ) ))
```

2°) Ecrire, en utilisant un schéma d'application itératif, une fonction `MI` ayant comme arguments une matrice de distances `MD` et un nombre réel `dmin`, telle que l'évaluation de `(MI MD dmin)` retourne une liste de sous-listes, la $i^{ème}$ sous-liste contenant les indices des points dont la distance à P_i est inférieure à `dmin`.

Par exemple l'évaluation de

```
(MI '((0 1 2 4 5)
      (1 0 1 3 4)
      (2 1 0 2 3)
      (4 3 2 0 1)
      (5 4 3 1 0))
    1.5)
```

doit retourner `((2 1) (3 2 1) (3 2) (5 4) (5 4))`.

Solution :

```
(define MI (lambda (MD dmin)
   (map (lambda (L) (LI L (lambda (x) (< x dmin)))) MD) ))
```

3°) Ecrire la fonction `CC`.

Solution :

```
(define CC (lambda (MD dmin) (CC2 (MI MD dmin))))
(define CC2 (lambda (MI)
   (if MI
      (insere (car MI) (CC2 (cdr MI)))
      () ) ))
(define insere (lambda (L M)
   (if M
      (if (intersection L (car M))
         (cons (union L (car M)) (cdr M))
         (cons (car M) (insere L (cdr M))) )
      (list L) ) ))
(define intersection (lambda (L1 L2)
   (if (and L1 L2)
      (let ((I (intersection (cdr L1) L2)))
```

```
         (if (appartient? (car L1) L2) (cons (car L1) I) I) )
      () ) ))
(define union (lambda (L1 L2)
    (if (and L1 L2)
      (let ((U (union (cdr L1) L2)))
         (if (appartient? (car L1) U) U (cons (car L1) U)) )
      () ) ))
```