ISIMA 1ère année

3 septembre 2009
Durée : 2 heures
Documents autorisés

PROGRAMMATION FONCTIONNELLE

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 pour le graphe défini ci-dessus, la liste des sous ensembles de points tels que pour chacun d'eux ils ne soient pas à une distance supérieure à dmin d'un autre point du sous ensemble (composante connexe). Chaque composante connexe est 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 `((4 5) (1 2 3))` 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 *(lister n)* qui permette d'obtenir la liste dans l'ordre des entiers de 1 à n (vous pourrez utiliser la fonction renverse si nécessaire)
Exemple : (lister 5) retourne (1 2 3 4 5)

Solution :

```
define (listerl n)
 (if (< n  1) () (cons n (listerl (- n 1)))))
;Value: listerl

;Value: listerl
(define (lister n)
 (reverse (listerl n)))
```

2°) On souhaite écrire 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)`.*

La solution envisagée est d'utiliser parallèlement à la liste L, la liste des entiers de 1 à 'la longueur de L' et de parcourir ces 2 listes simultanément.
Ainsi lorsque un terme de L vérifie une propriété P, il est simple de retourner l'indice correspondant représentant sa place dans la liste.

2-1°) Vous écrirez, en **utilisant une fonction d'application vue en cours**, une fonction auxiliaire *(LI2 L LINDICE P)* qui parcourt parallèlement les 2 listes L et LINDICE et retourne les valeurs de LINDICE de mêmes positions que les objets de L vérifiant la propriété P.
Exemple : `(LI2 '(a 6 b 9) '(1 2 3 4) integer?)` retourne `(2 4)`

Solution :

```
(define (LI2  L LN P)
  (append-map (lambda (L1 L2)
           (if (P L1) (list L2) ()))
           L LN))
```

2-2°) Ecrire la fonction LI

Solution :

```
(define (LI L P)
  (LI2 L (lister (length L)) P))
```

3°) Ecrire, **en utilisant une fonction d'application vue en cours,** 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 `((1 2) (1 2 3) (2 3) (4 5) (4 5))`.

Solution :

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

4°) Ecrire **à l'aide d'un schéma de réduction vu en cours que vous donnerez,** une fonction (union L1 L2) ou L1 et L2 sont des listes d'entiers représentant des ensembles et qui retourne une liste représentant l'union ensembliste de L1 et L2

Exemple : (union '(1 2 6) '(1 2 3))) retourne (6 1 2 3).

Solution :

```
(define (union L1 L2)
(SR L1 L2 (lambda (tt rr)
          (if (member tt rr) rr
            (cons tt rr)))))

(define (SR L B C)
  (if (null? L)
      B
      (C (car L) (SR (cdr L) B C))))
;Value: sr
```

5°) Ecrire une fonction (lunion LL) ou LL est une liste de listes, qui retourne une liste contenant l'union ensembliste de toutes les sous listes de LL

Exemple : (lunion '((1 2 6) (1 2 3) (2 3))) retourne '(6 1 2 3).

Solution :

```
(define (unionl L)
  (if (null? L) ()
      (union (car L) (unionl (cdr L)))))
```

6°) Ecrire en **utilisant une fonction d'application vue en cours** une fonction (LM X LL) ou X est un objet et LL une liste de sous liste contenant des objets de type X telle que LM retourne la liste des sous listes de LL contenant X.

Exemple : `(LM 2 '((1 2) (1 2 3) (2 3) (4 5) (4 5)))` *retourne* `((1 2) (1 2 3) (3 2))`

Solution :

```
(define (LM X LL)
  (append-map (lambda (Y)
            (if (member X Y)
                (list Y)
                ()))
            LL))
```

7°) Ecrire une fonction (LML L LL) ou L est une liste d'entiers et LL une liste de sous-listes d'entiers telle que (LML L LL) retourne la liste des sous listes de LL contenant au moins un objet de L)

Exemple : `(LML '(1 6) '((1 2) (1 2 3) (2 6) (2 3) (4 5) (4 5)))` *retourne* `((1 2) (1 2 3) (2 6))`

Solution :

```
(define (LML L LL)
  (append-map (lambda (X)
            (LM X LL))
            L))
```

8°) Ecrire une fonction (PGC L1 LL) permettant de retourner la plus grande composante connexe contenant les points représentés par L1 (Vous pouvez considérer que (equal? L1 L2) retourne vrai si L1 et L2 représentent les mêmes ensembles)
Exemple : (PGC '(1 6) '((1 2) (1 2 3) (2 3) (2 6) (4 5) (4 5))) retourne (6 1 2 3)

Solution :
```
(define (PGC1 L  LL)
  (unionl (LML L LL)))

(define (PGC L LL)
  (let ((PG1 (PGC1 L LL)))
    (if (equal? PG1 L)
      PG1
      (PGC PG1 LL))))
```

9°)Ecrire une fonction (SupprCC C1 LL) retournant la liste des sous-listes de LL n'apparaissant pas dans C1. Il suffit pour cela de réécrire la liste LL en ne gardant pas les sous-listes dont un élément apparaît dans C1. **Vous utiliserez une fonction d'application et aussi la fonction member.**
Exemple : (SupprCC '(1 2 3) '((1 2) (1 2 3) (2 3) (4 5) (4 5))) retourne ((4 5) (4 5))

Solution :
```
(define (SupprCC  C L)
  (append-map (lambda (x)
            (if (member (car x) C) () (list x)))
            L))
```

10°)Ecrire CC

Solution :
```
(define (CC MD dmin)
  (let ((CC2 (MI MD dmin)))
    (epure CC2)))

(define (epure LL)
  (if (null? LL) ()
      (let* ((PG1 (PGC (car LL) LL))
           (NL (SupprCC PG1 LL)))
      (cons PG1 (epure (SupprCC PG1 NL))))))
```