ISIMA 1ère année — 1er février 2013

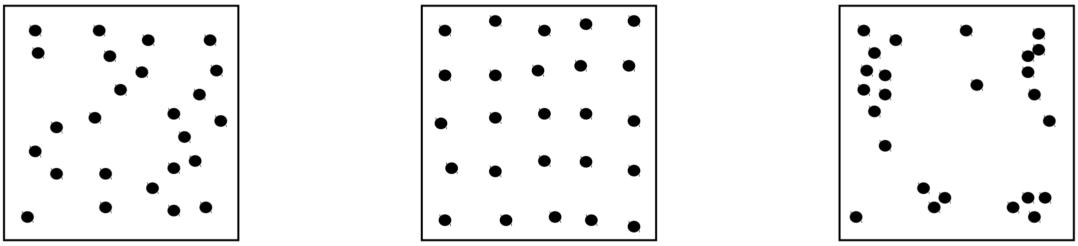
Durée : 2 heures

Documents autorisés

PROGRAMMATION FONCTIONNELLE

Exercice 1

Voici trois ensembles, E_1, E_2, E_3, de 25 points chacun :



E_1 E_2 E_3

Un seul coup d'œil suffit pour voir que l'ensemble E_2 “remplit mieux” le carré que les deux autres. Pour qu'un programme fasse le même constat il faut, en notant les points $P_1,\dots,$ P_n, qu'il cherche pour quel ensemble la valeur $\underset{i,j}{Min}$ (distance (P_i, P_j)) est maximale.

Le but de cet exercice est d'écrire une fonction `Maximin` qui fasse cela. Par exemple, si on lui passe en arguments une liste `LEP` contenant E_1, E_2 et E_3, et une fonction `d` représentant une distance d définie sur ces ensembles, alors l'évaluation de `(Maximin LEP d)` retourne la sous-liste représentant E_2. La fonction `d` est telle que l'évaluation de `(d P1 P2)` retourne la distance d (`P1`, `P2`) entre `P1` et `P2`.

1°) Ecrire, en utilisant des schémas d'application itératifs (`map` et/ou `append-map`), une fonction `listDist` ayant comme arguments un ensemble de points `EP` et une fonction `d` représentant une distance d définie sur `EP`, et telle que l'évaluation de l'expression `(listDist EP d)` retourne une liste contenant les valeurs d (P_i, P_j) pour <u>tous</u> les couples (P_i, P_j) de points de `EP`.

<u>Solution</u> :

```
(define listDist (lambda (EP d)
   (append-map (lambda (Pi) (map (lambda (Pj) (d Pi Pj)) EP)) E)) ))
```

2°) Ecrire une fonction `minPos` ayant comme arguments une liste `L` contenant au moins deux nombres positifs ou nuls, et telle que l'évaluation de l'expression `(minPos L)` retourne le minimum des valeurs strictement positives de la liste, s'il y en a, et 0 sinon.

<u>Remarque</u> : Un bonus sera attribué aux solutions qui ne parcourent la liste qu'une seule fois.

<u>Solution</u> :

```
(define minPos (lambda (L)
   (if (null? (cdr L)) 0
```

```
    (let ((AR (minPos (cdr L))))
      (if (= (car L) 0) AR
        (if (<= (car L) AR (car L) AR) ) ) ) ))
```

3°) Ecrire une fonction `eltMax` ayant comme arguments une liste non vide `L` et une fonction `f` prenant comme argument un élément de `L`, et telle que l'évaluation de l'expression `(eltMax L f)` retourne l'élément `x` de `L` pour lequel la valeur de l'évaluation `(f x)` est maximale.

<u>Remarque</u> : Un bonus sera attribué aux solutions qui ne parcourent la liste qu'une seule fois et qui, sur chaque élément `x` de `L`, n'évaluent `(f x)` qu'une seule fois.

<u>Solution</u> :

```
(define eltMax (lambda (L f) (eltmaxRT (cdr L) f (car L) (f (car L)))))
(define eltMaxRT (lambda L f eltm m)
  (if (null? L) eltm
    (let ((fcarL (f (car L))))
      (if (> fcarL m) (eltMaxRT (cdr L) f (car L) fcarL)
                      (eltMaxRT (cdr L) f eltm m) ) ) ) ))
```

4°) Ecrire la fonction `Maximin`. Cette fonction aura comme arguments :

- une liste de sous-listes, `LEP`, chaque sous-liste représentant un ensemble de points ;
- une fonction `d` représentant une distance *d* définie sur ces ensembles.

L'évaluation de `(Maximin LEP d)` retourne la liste représentant l'ensemble de points pour lequel la valeur $\underset{i,j}{Min}$ (distance (P_i, P_j)) est maximale.

<u>Solution</u> :

```
(define Maximin (lambda (LEP d)
  (eltMax LEP (lambda (EP) (minPos (listDist EP d)))) ))
```

Exercice 2

En mathématiques, une fraction continue d'un nombre réel x, notée [a_0, a_1, ...], est une expression de la forme :

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cdots}}} .$$

Elle est construite de la façon suivante : a_0 est la partie entière de x ; donc, $x - a_0$ est un nombre inférieur à 1, donc, s'il n'est pas nul, il est l'inverse d'un nombre supérieur à 1 dont la partie entière est a_1 et qui peut être lui même représenté par la fraction continue [a_1, ...]. Pour un nombre rationnel, la fraction rationnelle est toujours finie et sera notée [a_0, ... a_n].

Les fractions continues sont utilisées, entre autres, pour déterminer des approximations rationnelles d'un nombre irrationnel : par exemple, pour $x = \sqrt{2}$, comme $1 < x < 2$, $a_0 = 1$, et comme $\frac{1}{\sqrt{2}-1} = \sqrt{2}+1$, on a [$a_1, a_2, ...$] = [$1+a_0, a_1, ...$], d'où $\sqrt{2}$ = [1, 2, 2, ...].

Dans tout cet exercice, une liste ($a_0, a_1, ..., a_n$) représentera :

- soit, de manière exacte, un nombre rationnel dont la fraction continue est [$a_0, ... a_n$] ;
- soit, de manière approximative, un nombre dont la fraction continue est [$a_0, ... a_n, \ldots$].

1°) Le but de cette question est d'écrire une fonction Scheme `fcToNb` ayant comme argument une liste `FC`, et telle que l'évaluation de l'expression `(fcToNb FC)` retourne le nombre rationnel associé à la fraction continue représentée par `FC`.

a) Ecrire une fonction `fcToNb1` en appliquant une solution récursive classique (type schéma récursif). La liste `FC` représentant [$a_0, ... a_n$] sera ($a_0, a_1, ..., a_n$).

<u>Solution</u> :

```
(define fcToNb1 (lambda(FC)
   (if (null? (cdr FC))
      (car FC)
      (+ (car FC) (/ 1 (fcToNb1 (cdr FC)))) ) ))
```

b) Ecrire la fonction `fcToNb2` en appliquant une solution récursive terminale (type schéma itératif). La liste `FC` représentant [$a_0, ... a_n$] sera, pour faciliter la solution, représentée à l'envers sous la forme ($a_n, ..., a_1, a_0$).

<u>Solution</u> :

```
(define fcToNb2 (lambda(FC) (fcToNbRT (cdr FC) (car FC))))
(define fcToNbRT (lambda(FC x)
   (if (null? FC) x
       (fcToNbRT (cdr FC) (+ (car FC) (/ 1 x))) ) ))
```

2°) Ecrire une fonction Scheme `PePf` ayant comme argument un nombre réel `x` et telle que l'évaluation de l'expression `(PePf x)` retourne un couple formé respectivement des parties entières et fractionnaires de `x`. Par exemple `(PePf 1.414)` doit retourner `(1 0.414)`. La complexité temporelle de cette fonction doit être de l'ordre de la taille de la représentation binaire de `x`, c'est-à-dire de l'ordre de $\log_2$ `x`. Pour cela, il faut déduire (sauf pour les cas de base) les parties entières et fractionnaires de `x` à partir de celles de `x`/2 en tenant le raisonnement mathématique suivant : `x`/2 = Partie entière (`x`/2) + Partie fractionnaire (`x`/2), donc `x` = 2 × Partie entière (`x`/2) + 2 × Partie fractionnaire (`x`/2) ; il faut cependant faire attention au fait que 2 × Partie fractionnaire (`x`/2) peut être plus grand que 1.

<u>Solution</u> :

```
(define PePf (lambda(x)
   (if (< x 1)
      (list 0 x)
```

```
      (let* ((C (PePf (/ x 2))) (Pex/2 (car C)) (Pfx/2 (cadr C)))
         (if (>= Pfx/2 0.5)
            (list (+ 1 (* 2 Pex/2)) (- (* 2 Pfx/2) 1))
            (list (* 2 Pex/2) (* 2 Pfx/2)) ) ) ) ))
```

3°) En utilisant la fonction `PePf`, écrire une fonction Scheme `nbToFc` ayant comme arguments un nombre réel `x` et un entier `n`, et telle que l'évaluation de l'expression `(nbToFc x n)` retourne une liste ($a_0, ..., a_k$), avec $k \leq$ `n`, et telle que, si $k <$ `n`, [$a_0, ..., a_k$] est exactement la fraction continue de `x` et si $k =$ `n`, $a_0, ..., a_n$, sont les premiers termes de la fraction continue de `x`.

Solution :

```
(define nbToFc (lambda(x n)
   (if (= n 0) ()
      (let* ((C (PePf x)) (Pex (car C)) (Pfx (cadr C)))
         (if (= Pfx 0)
            (list x)
            (cons Pex (nbToFc (/ 1 Pfx) (- n 1))) ) ) ) ))
```

4°) On peut considérer que la longueur de la liste générée par la fonction `nbToFc` ne permet qu'une maîtrise implicite de la précision. Ecrire une fonction Scheme `nbToFc2` ayant comme arguments deux nombres réels `x` et `e`, et telle que l'évaluation de l'expression `(nbToFc2 x e)` retourne une liste de type ($a_k, ..., a_0$), conformément à l'ordre choisi dans la question 1°) b), telle que | [$a_0, a_1, ..., a_k$] – `x` | < `e`.

Solution :

```
(define nbToFc2 (lambda(x e)
   (let* ((C (PePf x)) (Pex (car C)) (Pfx (cadr C)))
         (if (= Pfx 0)
            (list x)
            (nbToFcRT x (/ 1 Pfx) e (list Pex)) ) ) ))
(define nbToFcRT (lambda(x r e FC)
   (let* ((C (PePf r)) (Per (car C)) (Pfr (cadr C)))
      (if (or (= Pfr 0) (< (abs (- (fcToNb2 FC) x)) e))
            FC
            (nbToFcRT x (/ 1 Pfr) e (cons Per FC)) ) ) ))
(define abs (lambda (x) (if (> x 0) x (- 0 x))))
```