Université Polytechnique HAUTS-DE-FRANCE

Licence 3 Informatique

Année 2018-2019

Programmation fonctionnelle - Code : S5IFPRF

Responsable : S. PIECHOWIAK

Matériel : Documents interdit – Calculatrices, ordinateurs et téléphones interdits

Exercices : Evaluation de fonctions ...

Que donne l'évaluation de chacune des requêtes suivantes :

```
1) (list '(1 2 3) '(4 5 6) '(7 8 9))
2) (cons '() '())
3) (cons (list '())'())
4) (cons (cons '() (cons '() (cons '()'() ))) '())
```

5) On définit f1 et f2 par :

```
(define f1
    (if (zero? N)
        #t
        (f2 (- N 1))))
```

```
(define f2
    (if (zero? N)
        #f
        (f1 (- N 1))))
```

Que valent : `(f1 5)` et `(f2 5)` ?

6) On définit la fonction mystere1 :

```
(define mystere1
  (lambda (N)
        (or (f1 N) (f2 N))))
```

Que vaut `(mystere1 5)` ?

7) Que vaut `(azerty '(1 2 8 10 12 20) 0 1)`

avec :

```
(define azerty
  (lambda (L V K)
    (if (null? L)
        (/ V K)
        (if (< 10 (car L))
            (azerty (cdr L) (+ (* 3 (car L)) V) (+ K 2))
            (azerty (cdr L) (+ (* 2 (car L)) V) (+ K 1))))))
```

La fonction map permet d'appliquer une fonction aux valeurs passées en argument et de récupérer la liste des résultats.

La syntaxe est : $(map\ f\ l_1\ l_2\ \dots\ l_n\ )$ où les $l_i\ = (l_i^1\ l_i^2\ \dots\ l_i^m)$ sont des listes de même taille et f est une fonction d'arité n.

Ainsi, $\left(map\ f(l_1^1\ l_1^2\ \dots\ l_1^m)(l_2^1\ l_2^2\ \dots\ l_2^m)\dots(l_n^1\ l_n^2\ \dots\ l_n^m)\right)$

renvoie $(f\ l_1^1 \dots\ l_n^1)(f\ l_1^2 \dots\ l_n^2)\dots(f\ l_1^m \dots\ l_n^m))$

Exemples :

```
(map + '(1 2 3) '(4 5 6) '(7 8 9)) = ((+ 1 4 7) (+ 2 5 8) (+ 3 6 9)
) = (12 15 18)
```

```
(map factorielle '(1 2 3)
```

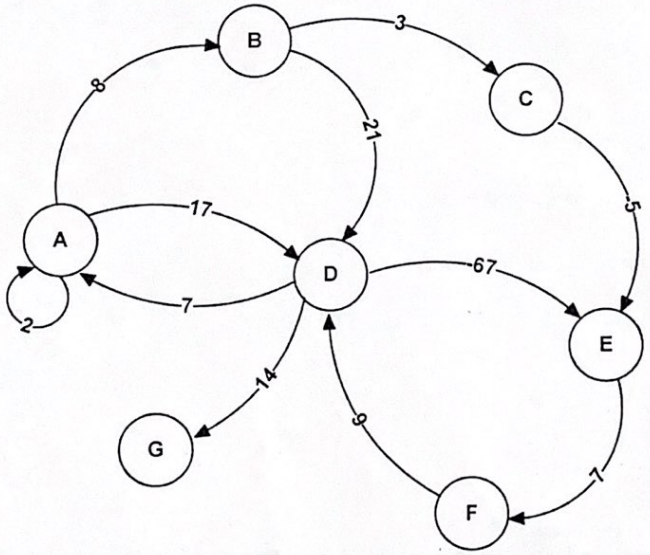
Que donne l'évaluation de chacune des requêtes suivantes :

```
8)  (map list '(1 2 3) '(4 5 6) '(7 8 9))
9)  (map + '(1 2 3) '(4 5 6) '(7 8 9))
10) (map list '(1 2 3))
11) (map (lambda (Y) (cons 0 Y)) '((1) (2) (3)))
12) (map (lambda (Y) (cdr Y)) '((1 2) (3 4) (5 6)))
13) (map (lambda (Y) (cons (car Y) (cdr Y))) '((1 2) (3 4) (5 6)))
```

PROBLEME : Sur les graphes ...

Un graphe orienté est défini par un couple G = (ND, AR) dans lequel ND est un ensemble de nœuds auxquels sont associées des valeurs et AR est un ensemble d'arc orientés. Chaque arc est défini par un triplet (D, V, A) dans lequel D est un nœud du graphe (appelé nœud de départ) V est une valeur associée à l'arc (appelée valuation ou pondération de l'arc) et A est un nœud du graphe (appelé nœud d'arrivée).

Voici un exemple de graphe orienté :



Pour simplifier le problème on suppose qu'on confond le nœud et sa valeur et que toutes les valeurs sont différentes. Dans l'exemple le nœud A a pour valeur A, le nœud B a pour valeur B etc.

On représente un tel graphe de la manière suivante :

```
( (A ((8 B) (17 D) (2 A)))        ; la liste des arcs valués
  (B ((3 C) (21 D))
  (C ((-5 E))
    ...
)
```

(ND , AR)

depart valeur arrivé

AR : (D , V , A)

Définir les fonctions Scheme suivantes :

- **(estUnNoeud? G N)** qui renvoie #t si le nœud N est présent dans le graphe G et renvoie #f sinon
- **(estUnArc? G A)** qui renvoie #t si l'arc A est présent dans le graphe G et renvoie #f sinon
- **(successeursDirects G N)** qui renvoie la liste de tous les successeurs directement connectés (par un arc) au nœud N
- **(successeurs G N)** qui renvoie la liste de tous les successeurs directement ou indirectement connectés au nœud N
- **(renverse G)** qui renvoie un graphe possédant les mêmes nœuds que G mais dont les arcs sont inversés
- **(estUnChemin? G N1 N2 ARCS)** qui renvoie #t si la liste des arcs présents dans la liste ARCS est bien un chemin entre les noeuds N1 et N2.
- **(longueur G chemin)** qui renvoie la longueur du chemin. Le chemin est donné sous la forme d'une liste de triplets (N1 V N2). La longueur représente la somme des valuations de ces arcs.

Barème indicatif :

exercice (12 pts) : 1 pt par requête

problème : (10 pts) 1 + 1 + 1 + 2 + 2 + 2 + 1

Université Polytechnique HAUTS-DE-FRANCE

Licence 3 Informatique

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES HAUTS-DE-FRANCE

Année 2019-2020
Programmation fonctionnelle - Code : S5IFPRF
Responsable : S. PIECHOWIAK

Matériel : Documents interdit – Calculatrices, ordinateurs et téléphones interdits

Barème indicatif : Exo. (9 pts) : 1 pt par requête / problème 1: (4 pts) / problème 2: (10 pts) 1 + 1 + 1 + 2 + 2 + 2 + 1

Exercices : Evaluation de fonctions ...

Que donne l'évaluation de chacune des requêtes suivantes :

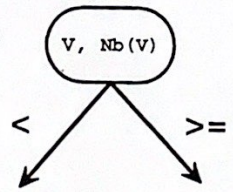
```
1) (list '(1 2 3) '(4 5 6) '(7 8 9))
2) (cons '() '())
3) (cons (list '())'())
4) (cons (list '()) (list '()))
5) (cons (cons '() (cons '() (cons '()'()))) '())
```

Donner une expression en Scheme/Racket dont l'évaluation vaut :

```
6)  ((()))
7)  (()(())())
8)  (1 (2 (3)))
9)  (((1) 2) 3)
```

PROBLEME 1 : Sur les arbres ...

On souhaite représenter les arbres binaires de manière plus compacte que celle vue en cours, en TD ou en TP. Pour cela, à chaque nœud est associé non seulement une valeur mais également le nombre d'occurrences de cette valeur. Ainsi chaque valeur n'est représentée que par un seul nœud. Cette représentation est surtout intéressante si le nombre de valeurs différentes est « important ». Par contre si les valeurs sont toutes différentes, cette représentation n'est pas adaptée.



On appellera un arbre binaire compact, un arbre ainsi représenté dans la suite du problème.
En supposant que les valeurs sont ajoutées suivant l'ordre de la liste, dessiner l'arbre qui sera obtenu avec

la liste de valeurs : **(1 1 4 6 8 4 6 7 7 7 7 9 0 3 5 4 1 0 0)**. Dessiner également l'arbre binaire ordonné obtenu avec la structure simple vue en cours (un nœud par valeur).
Chaque nœud d'un arbre binaire compact sera représenté par un quadruplet **(V Nb G D)** dans lequel **V** est la valeur associée au nœud, **Nb** est le nombre d'occurrence de cette valeur, **G** est le sous arbre gauche (c'est un arbre binaire compact) et **D** est le sous arbre gauche (c'est un arbre binaire compact). L'arbre vide est représenté par la liste vide **( )** .

1. Donner une fonction **nbFeuilles** qui renvoie le nombre de feuilles de l'arbre binaire compact passé en argument.

(nbFeuilles A) => nombre de feuilles de l'arbre binaire compact A

2. Donner une fonction **nbNoeuds** qui renvoie le nombre de noeuds de l'arbre binaire compact passé en argument.

(nbNoeuds A) => nombre de noeuds de l'arbre binaire compact A

3. Donner une fonction **nbValeurs** qui renvoie le nombre de valeurs de l'arbre binaire compact passé en argument (et qui prend en compte les nombres d'occurrence de chaque valeur)

(nbValeurs A) => nombre de valeurs de l'arbre binaire compact A

4. Donner une fonction **formeExpansee** qui renvoie la liste obtenue en parcourant l'arbre binaire compact passé en argument en répétant autant de fois chaque valeur que le nombre d'occurrences indique. Par exemple **(formeExpansee '(4 3 (2 1 ( ) ( )) (7 2 ( ) ( ))))** renverra la liste **(2 4 4 4 7 7)**

(formeExpansee A) => forme expansée de A

5. Donner une fonction **ajoutUn** qui ajoute une seule occurrence d'une valeur dans un arbre binaire compact. Cette fonction ne doit pas modifier l'arbre initial mais construit un nouvel arbre identique à l'arbre initial auquel on a ajouté une valeur.

(ajoutUn A V) => arbre binaire compact augmenté de la valeur V

6. Donner une fonction **ajoutPlusieurs** qui ajoute plusieurs occurrences d'une valeur dans un arbre binaire compact. Cette fonction ne doit pas modifier l'arbre initial mais construit un nouvel arbre identique à l'arbre initial auquel on a ajouté une valeur.

(ajoutPlusieurs A V N) => arbre binaire compact augmenté de N fois la valeur V

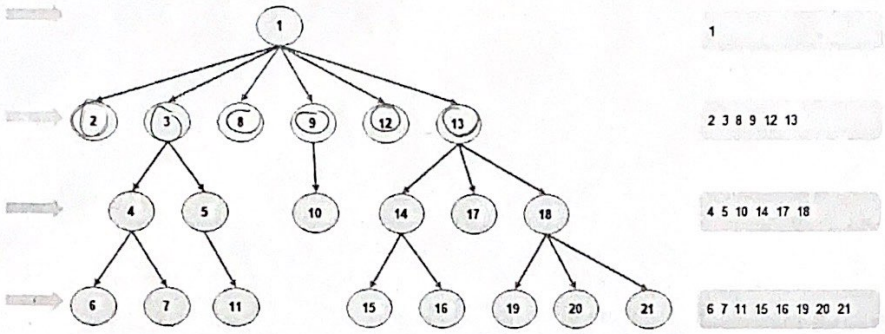
7. Donner une fonction **detruireUn** qui détruit une occurrence d'une valeur dans un arbre binaire compact. Cette fonction ne doit pas modier l'arbre initial mais construit un nouvel arbre identique à l'arbre initial auquel on a retiré une valeur.

(detruireUn A V) => arbre binaire compact diminué d'une occurrence de V

PROBLEME 2 : Sur les parcours d'arbres ...

On souhaite parcourir un arbre selon le parcours dit « en largeur d'abord ». Ce parcours consiste à parcourir toutes les feuilles d'un même niveau de l'arbre avant de parcourir le niveau suivant. Le premier niveau est celui de la racine et le dernier niveau est celui des feuilles de plus bas niveau.

Le schéma suivant donne un exemple.



Définir une fonction qui renvoie la liste des valeurs présentes dans un arbre donné en parcourant cet arbre en largeur d'abord. Pour cela on utilisera une file. Pour rappel dans une file on extrait les éléments par la tête et on ajoute par la queue.

Cette file est initialisée avec le nœud racine. A chaque fois qu'on extrait la tête de la file on ajoute tous ses fils directs par la queue.