Universidad Nacional Autónoma de México

Facultad de Ciencias

Lenguajes de Programación

Tarea 5: Recursión

Johann Ramón Gordillo Guzmán
418046090

José Jhovan Gallardo Valdéz
310192815

Tarea presentada como parte del curso de **Lenguajes de Programación** impartido por la profesora **M.I. Karla Ramírez Pulido**.

15 de Abril del 2020

Link al código fuente: https://github.com/JohannGordillo/

1. Preguntas

1. Define la función recursiva ocurrencias-elementos que toma como argumentos dos listas y devuelve una lista de parejas, en donde cada pareja contiene en su parte izquierda un elemento de la segunda lista y en su parte derecha el número de veces que aprarece dicho elemento en la primera lista. Por ejemplo:

   > (ocurrencias-elementos '(1 3 6 2 4 7 3 9 7) '(5 2 3))
   '((5 . 0) (2 . 1) (3 . 2))

   Respuesta.

   Primero definimos una función recursiva auxiliar **count-ocurrences** que devuelve el número de veces que aparece un elemento en una lista. Con ayuda de ésta, definimos la función **ocurrencias-elementos** haciendo uso de la recursión. El código en Racket es el siguiente:

```
#lang plai

;; =====================================
;; >> Autor: Johann Gordillo.
;; >> Email: jgordillo@ciencias.unam.mx
;; >> Fecha: 11/04/2020
;; =====================================

;; Devuelve las ocurrencias de un elemento en una lista.
(define (count-ocurrences l e)
  (cond
    [(empty? l)  ;; Caso Base.
     0]
    [(equal? (car l) e)
     (add1 (count-ocurrences (cdr l) e))]
    [else
     (count-ocurrences (cdr l) e)]))

;; Devuelve una lista de parejas, en donde cada pareja
;; contiene en su parte izquierda un elemento de la
;; segunda lista y en su parte derecha el número de
;; veces que aparece dicho elemento en la primera lista.
(define (ocurrencias-elementos l1 l2)
  (cond
    [(empty? l2)  ;; Caso Base.
     '()]
    [else
     (let ([head (car l2)])
       (cons
        (cons
         head
         (count-ocurrences l1 head))
        (ocurrencias-elementos l1 (cdr l2))))]))
```

```
Welcome to DrRacket, version 7.5 [3m].
Language: plai, with debugging; memory limit: 128 MB.
> (ocurrencias-elementos '(1 3 6 2 4 7 3 9 7) '(5 2 3))
'((5 . 0) (2 . 1) (3 . 2))
> (ocurrencias-elementos '(1 3 6 2 4 7 3 9 7) '())
'()
>
```

2. Muestra los registros de activación generados por la función definida en el Ejercicio 1 con la llamada (ocurrencias-elementos '(1 2 3) '(1 2)).

Respuesta.

Sea **body-ocurrencias-elementos**:

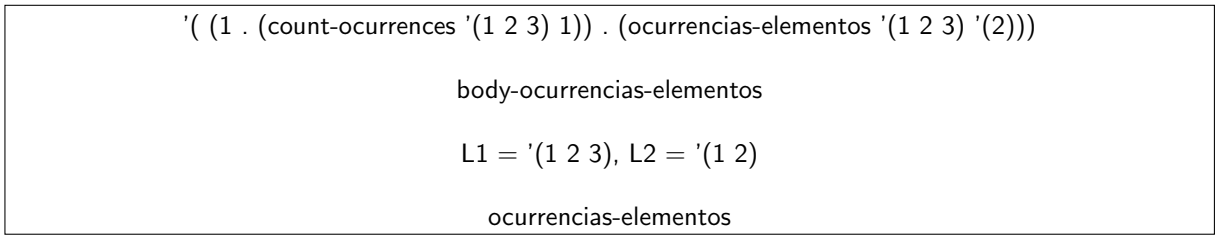
```
(cond
   [(empty? L2) '()]
   [else
      (let ([head (car L2)])
         (cons
            (cons head (count-ocurrences L1 head))
            (ocurrencias-elementos L1 (cdr L2))))]))
```

Sea **body-count-ocurrences**:

```
(cond
   [(empty? l) 0]
   [(equal? (car l) e)
      (add1 (count-ocurrences (cdr l) e))]
   [else
      (count-ocurrences (cdr l) e)]))
```

Nota: Para *count-ocurrences* se tomará como que la salida es inmediatamente el número de veces que un elemento está en una lista dada, sin mostrarse completo el procedimiento recursivo en los registros de activación. Sin embargo, en el ejercicio 4 se muestra el proceso de manera exhaustiva. Esto con el fin de no hacer tan largo el documento y siendo ambos ejercicios análogos.

Ingresa (ocurrencias-elementos '(1 2 3) '(1 2)):



Ingresa (count-ocurrences '(1 2 3) 1):

1 body-count-ocurrences L1 = '(1 2 3), e = 1 count-ocurrences

'((1 . (count-ocurrences '(1 2 3) 1)) . (ocurrencias-elementos '(1 2 3) '(2))) body-ocurrencias-elementos L1 = '(1 2 3), L2 = '(1 2) ocurrencias-elementos

Sale (count-ocurrences '(1 2 3) 1):

'((1 . 1) . (ocurrencias-elementos '(1 2 3) '(2))) body-ocurrencias-elementos L1 = '(1 2 3), L2 = '(1 2) ocurrencias-elementos

Entra (ocurrencias-elementos '(1 2 3) '(2)):

'((2 . (count-ocurrences '(1 2 3) 2)) . (ocurrencias-elementos '(1 2 3) '())) body-ocurrencias-elementos L1 = '(1 2 3), L2 = '(2) ocurrencias-elementos

'((1 . 1) . (ocurrencias-elementos '(1 2 3) '(2))) body-ocurrencias-elementos L1 = '(1 2 3), L2 = '(1 2) ocurrencias-elementos

Entra (count-ocurrences '(1 2 3) 2):

1 body-count-ocurrences L1 = '(1 2 3), e = 2 count-ocurrences

'((2 . (count-ocurrences '(1 2 3) 2)) . (ocurrencias-elementos '(1 2 3) '())) body-ocurrencias-elementos L1 = '(1 2 3), L2 = '(2) ocurrencias-elementos

'((1 . 1) . (ocurrencias-elementos '(1 2 3) '(2))) body-ocurrencias-elementos L1 = '(1 2 3), L2 = '(1 2) ocurrencias-elementos

Sale (count-ocurrences '(1 2 3) 2):

'((2 . 1) . (ocurrencias-elementos '(1 2 3) '())) body-ocurrencias-elementos L1 = '(1 2 3), L2 = '(2) ocurrencias-elementos

'((1 . 1) . (ocurrencias-elementos '(1 2 3) '(2))) body-ocurrencias-elementos L1 = '(1 2 3), L2 = '(1 2) ocurrencias-elementos

Entra (ocurrencias-elementos '(1 2 3) '()):

'()
body-ocurrencias-elementos
L1 = '(1 2 3), L2 = '()
ocurrencias-elementos

'((2 . 1) . (ocurrencias-elementos '(1 2 3) '()))
body-ocurrencias-elementos
L1 = '(1 2 3), L2 = '(2)
ocurrencias-elementos

'((1 . 1) . (ocurrencias-elementos '(1 2 3) '(2)))
body-ocurrencias-elementos
L1 = '(1 2 3), L2 = '(1 2)
ocurrencias-elementos

Sale (ocurrencias-elementos '(1 2 3) '()):

'((2 . 1) . '())
body-ocurrencias-elementos
L1 = '(1 2 3), L2 = '(2)
ocurrencias-elementos

'((1 . 1) . (ocurrencias-elementos '(1 2 3) '(2)))
body-ocurrencias-elementos
L1 = '(1 2 3), L2 = '(1 2)
ocurrencias-elementos

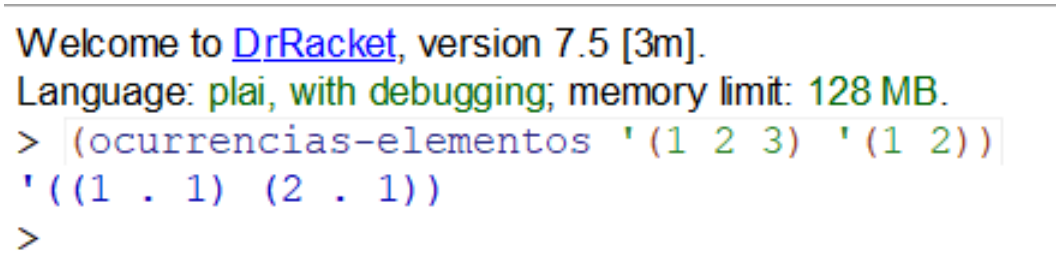
Sale (ocurrencias-elementos '(1 2 3) '(2)):

'((1 . 1) . '((2 . 1) . '()))
body-ocurrencias-elementos
L1 = '(1 2 3), L2 = '(1 2)
ocurrencias-elementos

Sale (ocurrencias-elementos '(1 2 3) '(1 2)):

'((1 . 1) (2 . 1))

Podemos comprobar en RACKET:



3. Optimiza la función definida en el Ejercicio 1 usando recursión de cola. Debes transformar todas las funciones auxiliares que utilices.

Respuesta.

El código en RACKET es el siguiente:

```
#lang plai

;; ======================================
;; >> Autor: Johann Gordillo.
;; >> Email: jgordillo@ciencias.unam.mx
;; >> Fecha: 11/04/2020
;; ======================================

;; Devuelve las ocurrencias de un elemento en una lista.
(define (count-ocurrences l e)
  (count-ocurrences-tail l e 0))

;; Función auxiliar para implementar recursión de cola.
(define (count-ocurrences-tail l e acc)
  (cond
    [(empty? l) acc]
    [(equal? (car l) e)
     (count-ocurrences-tail (cdr l) e (add1 acc))]
    [else
     (count-ocurrences-tail (cdr l) e acc)]))

;; Devuelve una lista de parejas, en donde cada pareja
;; contiene en su parte izquierda un elemento de la
;; segunda lista y en su parte derecha el número de
;; veces que aparece dicho elemento en la primera lista.
(define (ocurrencias-elementos l1 l2)
  (ocurrencias-elementos-tail l1 l2 '()))

;; Función auxiliar para implementar recursión de cola.
(define (ocurrencias-elementos-tail l1 l2 acc)
  (cond
    [(empty? l2) acc]
    [else
     (let [(head (car l2))]
       (ocurrencias-elementos-tail
        l1
        (cdr l2)
        (append acc
                (list (cons
                       head
                       (count-ocurrences l1 head))))))]))
```

4. Muestra los registros de activación generados por la función definida en el Ejercicio 3 con la llamada (ocurrencias-elementos '(1 2 3) '(1 2)).

Respuesta.

Sea **body-ocurrencias-elementos-tail**:

```
(cond
    [(empty? l2) acc]
    [else
        (let [(head (car l2))]
            (ocurrencias-elementos-tail
                l1
                (cdr l2)
                (append acc
                    (list (cons
                        head
                        (count-ocurrences l1 head))))))]))
```

Sea **body-count-ocurrences-tail**:

```
(cond
    [(empty? l) acc]
    [(equal? (car l) e)
        (count-ocurrences-tail (cdr l) e (add1 acc))]
    [else
        (count-ocurrences-tail (cdr l) e acc)]))
```

Ingresa (ocurrencias-elementos '(1 2 3) '(1 2)):

(ocurrencias-elementos-tail '(1 2 3) '(1 2) '())
(ocurrencias-elementos-tail L1 L2 '())
L1 = '(1 2 3), L2 = '(1 2)
ocurrencias-elementos

Sale (ocurrencias-elementos '(1 2 3) '(1 2)) y entra (ocurrencias-elementos-tail '(1 2 3) '(1 2) '()):

(ocurrencias-elementos-tail '(1 2 3) '(2) (append '() (list (cons 1 (count-ocurrences '(1 2 3) 1)))))
body-ocurrencias-elementos-tail
L1 = '(1 2 3), L2 = '(1 2), acc = '()
ocurrencias-elementos-tail

Entra (count-ocurrences '(1 2 3) 1):

(count-ocurrences-tail '(1 2 3) 1 0)
(count-ocurrences-tail L1 e 0)
L1 = '(1 2 3), e = 1
count-ocurrences

(ocurrencias-elementos-tail '(1 2 3) '(2) (append '() (list (cons 1 (count-ocurrences '(1 2 3) 1)))))
body-ocurrencias-elementos-tail
L1 = '(1 2 3), L2 = '(1 2), acc = '()
ocurrencias-elementos-tail

Sale (count-ocurrences '(1 2 3) 1) y entra (count-ocurrences-tail '(1 2 3) 1 0):

(count-ocurrences-tail '(2 3) 1 1)
body-count-ocurrences-tail
L = '(1 2 3), e = 1, acc = 0
count-ocurrences-tail

(ocurrencias-elementos-tail '(1 2 3) '(2) (append '() (list (cons 1 (count-ocurrences '(1 2 3) 1)))))
body-ocurrencias-elementos-tail
L1 = '(1 2 3), L2 = '(1 2), acc = '()
ocurrencias-elementos-tail

Sale (count-ocurrences-tail '(1 2 3) 1 0) y entra (count-ocurrences-tail '(2 3) 1 1):

(count-ocurrences-tail '(3) 1 1)
body-count-ocurrences-tail
L = '(2 3), e = 1, acc = 1
count-ocurrences-tail

(ocurrencias-elementos-tail '(1 2 3) '(2) (append '() (list (cons 1 (count-ocurrences '(1 2 3) 1)))))
body-ocurrencias-elementos-tail
L1 = '(1 2 3), L2 = '(1 2), acc = '()
ocurrencias-elementos-tail

Sale (count-ocurrences-tail '(2 3) 1 1) y entra (count-ocurrences-tail '(3) 1 1):

(count-ocurrences-tail '() 1 1)
body-count-ocurrences-tail
L = '(3), e = 1, acc = 1
count-ocurrences-tail

(ocurrencias-elementos-tail '(1 2 3) '(2) (append '() (list (cons 1 (count-ocurrences '(1 2 3) 1)))))
body-ocurrencias-elementos-tail
L1 = '(1 2 3), L2 = '(1 2), acc = '()
ocurrencias-elementos-tail

Sale (count-ocurrences-tail '(3) 1 1) y entra (count-ocurrences-tail '() 1 1):

1 body-count-ocurrences-tail L = '(), e = 1, acc = 1 count-ocurrences-tail

(ocurrencias-elementos-tail '(1 2 3) '(2) (append '() (list (cons 1 (count-ocurrences '(1 2 3) 1))))) body-ocurrencias-elementos-tail L1 = '(1 2 3), L2 = '(1 2), acc = '() ocurrencias-elementos-tail

Sale (count-ocurrences-tail '() 1 1):

(ocurrencias-elementos-tail '(1 2 3) '(2) '((1 . 1))) body-ocurrencias-elementos-tail L1 = '(1 2 3), L2 = '(1 2), acc = '() ocurrencias-elementos-tail

Sale (ocurrencias-elementos-tail '(1 2 3) '(1 2) '()) y entra (ocurrencias-elementos-tail '(1 2 3) '(2) '((1 . 1))):

(ocurrencias-elementos-tail '(1 2 3) '() (append '((1 . 1)) (list (cons 2 (count-ocurrences '(1 2 3) 2))))) body-ocurrencias-elementos-tail L1 = '(1 2 3), L2 = '(2), acc = '((1 . 1)) ocurrencias-elementos-tail

Entra (count-ocurrences '(1 2 3) 2):

(count-ocurrences-tail '(1 2 3) 2 0) (count-ocurrences-tail L e 0) L = '(1 2 3), e = 2 count-ocurrences

(ocurrencias-elementos-tail '(1 2 3) '() (append '((1 . 1)) (list (cons 2 (count-ocurrences '(1 2 3) 2))))) body-ocurrencias-elementos-tail L1 = '(1 2 3), L2 = '(2), acc = '((1 . 1)) ocurrencias-elementos-tail

Sale (count-ocurrences '(1 2 3) 2) y entra (count-ocurrences-tail '(1 2 3) 2 0):

(count-ocurrences-tail '(2 3) 2 0) body-count-ocurrences-tail L = '(1 2 3), e = 2, acc = 0 count-ocurrences-tail

(ocurrencias-elementos-tail '(1 2 3) '() (append '((1 . 1)) (list (cons 2 (count-ocurrences '(1 2 3) 2))))) body-ocurrencias-elementos-tail L1 = '(1 2 3), L2 = '(2), acc = '((1 . 1)) ocurrencias-elementos-tail

Sale (count-ocurrences-tail '(1 2 3) 2 0) y entra (count-ocurrences-tail '(2 3) 2 0):

(count-ocurrences-tail '(3) 2 1)
body-count-ocurrences-tail
L = '(2 3), e = 2, acc = 0
count-ocurrences-tail

(ocurrencias-elementos-tail '(1 2 3) '() (append '((1 . 1)) (list (cons 2 (count-ocurrences '(1 2 3) 2)))))
body-ocurrencias-elementos-tail
L1 = '(1 2 3), L2 = '(2), acc = '((1 . 1))
ocurrencias-elementos-tail

Sale (count-ocurrences-tail '(2 3) 2 0) y entra (count-ocurrences-tail '(3) 2 1):

(count-ocurrences-tail '() 2 1)
body-count-ocurrences-tail
L = '(3), e = 2, acc = 1
count-ocurrences-tail

(ocurrencias-elementos-tail '(1 2 3) '() (append '((1 . 1)) (list (cons 2 (count-ocurrences '(1 2 3) 2)))))
body-ocurrencias-elementos-tail
L1 = '(1 2 3), L2 = '(2), acc = '((1 . 1))
ocurrencias-elementos-tail

Sale (count-ocurrences-tail '(3) 2 1) y entra (count-ocurrences-tail '() 2 1):

1
body-count-ocurrences-tail
L = '(), e = 2, acc = 1
count-ocurrences-tail

(ocurrencias-elementos-tail '(1 2 3) '() (append '((1 . 1)) (list (cons 2 (count-ocurrences '(1 2 3) 2)))))
body-ocurrencias-elementos-tail
L1 = '(1 2 3), L2 = '(2), acc = '((1 . 1))
ocurrencias-elementos-tail

Sale (count-ocurrences-tail '() 2 1):

(ocurrencias-elementos-tail '(1 2 3) '() '((1 . 1) (2 . 1)))
body-ocurrencias-elementos-tail
L1 = '(1 2 3), L2 = '(2), acc = '((1 . 1))
ocurrencias-elementos-tail

Sale (ocurrencias-elementos-tail '(1 2 3) '(2) '((1 . 1))) y entra (ocurrencias-elementos-tail '(1 2 3) '() '((1 . 1) (2 . 1))):

'((1 . 1) (2 . 1))
body-ocurrencias-elementos-tail
L1 = '(1 2 3), L2 = '(), acc = '((1 . 1) (2 . 1))
ocurrencias-elementos-tail

Sale (ocurrencias-elementos-tail '(1 2 3) '() '((1 . 1) (2 . 1))):

'((1 . 1) (2 . 1))

5. Para implementar recursión en el lenguaje RCFWAE fue necesario añadir ambientes recursivos con el apoyo de cajas. Este intérprete hace uso de alcance estático. ¿Por qué no sería necesario agregar este tipo de ambientes al usar alcance dinámico?

   Respuesta.

   Recordemos que utilizando alcance dinámico el alcance de las variables se determina en tiempo de compilación, mientras que utiizando alcance estático es determinado en tiempo de ejecución. Además, la implementación de cerraduras (closures) no es necesaria al usar alcance dinámico.

   Cuando se usa alcance dinámico, las búsquedas en el ambiente de evaluación se realizan a partir del tope del mismo (viéndolo como una pila). Así, no se presenta el problema de que un id en el cuerpo de una función con el mismo id (recursión) es libre y por ello no es necesario usar cajas para implementar ambientes recursivos. La recursión se da de manera natural al utilizar este tipo de alcance.

   Al usar alcance estático sí se peresenta este problema y es por ello que es necesario utilizar ambientes recursivos. Por ejemplo, al ejecutar lo siguiente en Racket se presenta el mencionado problema:

```
(let ([fact (λ(n))
    (if (zero? n) 1 (* n (fact (sub1 n)))))])
        (fact 5))
```

   Dado que Racket implementa alcance estático. Sin embargo, es posible implementar recursión en el lenguaje como se vio en las clases y en las notas.

2. Bibliografía

- Ramírez, K. (2020).
  Notas del curso de Lenguajes de Programación.
  Facultad de Ciencias - UNAM
  Ciudad de México, México.

- Krishnamurthi, S. (2017).
  Programming Languages: Application and Interpretation.
  Estados Unidos de América.