# Tarea 4: Programación con Listas en Scheme

Andres G. Cavazos A01195067

Gustavo Ferrufino A00812572

SECCIÓN 1: Recursividad profunda en listas de Scheme **(20 puntos)**

Para la implementación de las siguientes funciones puedes apoyarte en funciones auxiliaries que tú diseñes. En caso de hacerlo, incluir el código de estas.

1. Implementar la función recursiva tokeniza reciba una lista posiblemente imbricada y

la convierta en una lista plana que sustituya cada número por el símbolo N, cada

símbolo por el símbolo S y cada sublista por una secuencia de símbolos I <elementos>

D, donde <elementos> es una secuencia de elementos que depende del contenido de

la sublista.

Ejemplos de prueba:

> (tokeniza ‘(1 a b 2)) => (N S S N)

> (tokeniza ‘(1 (a (2 b) 5))) => (N I S I N S D N D)

> (tokeniza ‘(((1 a)))) => (I I N S D D)

(define (tokeniza l)

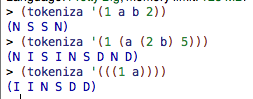
(cond [(null? l) ()]

[(number? (car l)) (cons 'N (tokeniza (cdr l)))]

[(symbol? (car l)) (cons 'S (tokeniza (cdr l)))]

[ else (append (append (cons 'I (tokeniza (car l))) (list 'D )) (tokeniza (cdr l)))]

))



2. Implementar la función recursiva subelementos que reciba una lista posiblemente

imbricada y regrese otra lista con las mismas sublistas, pero sustituyendo los

elementos por su cantidad en cada sublista.

Ejemplos de prueba:

> (subelementos ‘(1 2 3 4 5 6 7)) => (7)

> (subelementos ‘(1 (2 (3 4) 5) (6 7))) => (3 (3 (2))(2))

> (subelementos ‘(1 (2 (3 4))(5 (6 7)))) => (3 (2 (2))(2 (2)))

(define (subelementos l)

(cons (length l)

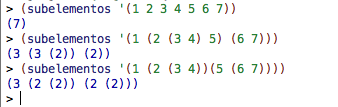
(sub l)))

(define (sub l)

(cond ((null? l) ())

((list? (car l)) (cons (cons (length (car l)) (sub (car l))) (sub (cdr l)) ))

(else (sub (cdr l)))))



SECCIÓN 2: Estructuras de datos en Scheme **(70 puntos)**

MATRICES

3. Implementar la función recursiva sucesores que reciba una matriz de símbolos que

representen un tablero del juego del gato y la marca de un jugador, y regrese una lista

de nuevos tableros que indiquen las jugadas válidas del jugador.

Ejemplos de prueba:

> (sucesores ‘((X v X)(v O v)(X v O)) ‘O)

=> (((X O X)(v O v)(X v O)) ((X v X)(O O v)(X v O))

((X v X)(v O O)(X v O)) ((X v X)(v O v)(X O O)))

> (sucesores ‘((X v X)(O O v)(X O O)) ‘X)

=> (((X X X)(O O v)(X O O)) ((X v X)(O O X)(X O O)))

(define sucesores

(lambda (mat symbol)

(sucesoresAux '() mat '() symbol)

)

)

(define sucesoresAux

(lambda (i f lista simbolo)

(cond((null? f) lista)

(else (sucesoresAux

(append i (list (car f)))

(cdr f)

(append lista (intercambio i (cdr f) (convertir (car f) simbolo) '()))

simbolo)))

)

)

(define convertir

(lambda (ren simbolo)

(convertirAux ren '() simbolo '())

)

)

(define intercambio

(lambda (i f ren lista)

(cond((null? ren) lista)

(else (intercambio

i

f

(cdr ren)

(append lista (list (append i (list (car ren)) f)))))

)

)

)

(define convertirAux

(lambda (i f simbolo lista)

(cond((null? i) lista)

((equal? 'v (car i))

(convertirAux

(cdr i)

(append f (list (car i)))

simbolo

(append lista (list (append f (list simbolo) (cdr i))))))

(else (convertirAux

(cdr i)

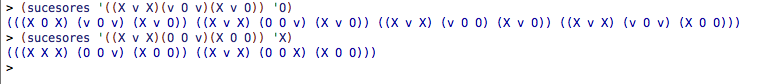
(append f (list (car i)))

simbolo

lista)))

)

)



4. Implementar el predicado recursivo ganador? que reciba una matriz de símbolos que

representen un tablero del juego del gato (ordenado por renglón) y una marca de

jugador, y determine si este jugador ha ganado el juego. Recordar que para ganar se

deben tener 3 marcas en línea, horizontal, vertical o diagonalmente.

Ejemplos de prueba:

> (ganador? ‘((X X v)(v O O)(X v O)) ‘O) => #f

> (ganador? ‘((X v X)(O O O)(X v O)) ‘O) => #t

> (ganador? ‘((X X O)(v O O)(X v O)) ‘O) => #t

> (ganador? ‘((X X O)(v O O)(O v X)) ‘O) => #t

(define ganador?

(lambda (lista jug)

(cond

((null? lista) #f)

((renglones lista jug) #t)

((renglones (apply map list lista) jug) #t)

((diagonal lista jug) #t)

(else (diagonal (reverse (apply map list lista)) jug)))

)

)

(define renglon

(lambda (ren simbolo)

(if (null? ren)

#f

(if (equal? (car ren) simbolo)

(if (null? (cdr ren))

#t

(renglon (cdr ren) simbolo))

#f)

)

)

)

(define diagonal

(lambda (mat simbolo)

(if (null? mat)

#f

(if (equal? (caar mat) simbolo)

(if (null? (cdr mat))

#t

(diagonal (map cdr (cdr mat)) simbolo))

#f)

)

)

)

(define renglones

(lambda (lista simbolo)

(if (null? lista)

#f

(if (renglon (car lista) simbolo)

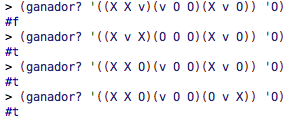
#t

(renglones (cdr lista) simbolo))

)

)

)



TABLAS

Utilizando la siguiente base de datos de equipos de futbol con registros almacenados

mediante una lista con formato (eq jg je jp gf gc), donde eq = nombre de equipo, jg =

juegos ganados, je = juegos empatados, jp = juegos perdidos, gf = goles a favor y gc

goles en contra:

**(define ligaMX ‘(( America 6 2 1 15 6 )( Atlas 5 3 1 13 8 )**

**( CruzAzul 2 3 4 7 10 )( Guadalajara 2 3 3 6 9 )**

**( Jaguares 3 4 2 13 12 )( Leon 3 0 6 15 16 )**

**( LeonesNegros 1 3 5 3 10 )( Monterrey 5 1 2 10 5 )**

**( Morelia 0 3 6 8 21 )( Pachuca 4 1 4 11 11 )**

**( Puebla 2 4 3 7 11 )( Queretaro 4 2 3 13 10 )**

**( SantosLaguna 4 3 2 12 9 )( Tijuana 2 5 2 11 9 )**

**( Toluca 5 2 2 11 8 )( UANL 3 4 2 14 11 )**

**( UNAM 3 2 4 12 12 )( Veracruz 1 5 3 5 8 )))**

5. Implementar la función recursiva estadistica que a partir de la base de datos de

equipos de futbol regrese una lista con los nombres de los equipos, su # de juegos

jugados, la diferencia de goles y el # de puntos de cada uno, ordenada por # de

puntos y en caso de empate en puntos, por diferencia de goles. Recordar que el

número de puntos se calcula como 3 puntos por cada juego ganado y 1 por cada juego

empatado.

Ejemplos de prueba:

**> (estadistica ligaMX**) => (( 1 America 9 9 20 )( 2 Atlas 9 5 18 )

( 3 Toluca 9 3 17 )( 4 Monterrey 8 5 16 )

( 5 SantosLaguna 9 3 15 )( 6 Queretaro 9 3 14 )

( 7 UANL 9 3 13 )( 8 Jaguares 9 1 13 )

( 9 Pachuca 9 0 13 )( 10 Tijuana 9 2 11 )

( 11 UNAM 9 0 11 )( 12 Puebla 9 -4 10 )

( 13 Leon 9 -1 9 )( 14 CruzAzul 9 -3 9 )

( 15 Guadalajara 8 -3 9 )( 16 Veracruz 9 -3 8 )

( 17 LeonesNegros 9 -7 6 )( 18 Morelia 9 -13 3 ))

(define estadistica

(lambda (lista)

(statsCallFuncs lista '() 1)))

(define statsCallFuncs ; Enumera Tabla

(lambda (lista listaAux pos)

(if (null? lista)

listaAux

(statsCallFuncs (remve lista (order lista)) (append listaAux (list (append (list pos) (formt (order lista))))) (+ pos 1)))))

(define order

(lambda (lista)

(statsFunc lista '())))

(define statsFunc

(lambda (lista b)

(cond

((null? lista) b)

((null? b) (statsFunc (rest lista) (first lista)))

((< (points b) (points (first lista))) (statsFunc (rest lista) (first lista)))

((and (= (points b) (points (first lista))) (< (diffGoals b) (diffGoals (first lista)))) (statsFunc (rest lista) (first lista)))

((and (= (points b) (points (first lista))) (< (diffGoals b) (diffGoals (first lista)))) (statsFunc (rest lista) (first lista)))

(else (statsFunc (rest lista) b)))))

(define formt

(lambda (team)

(list (first team) (+ (cadr team) (caddr team) (cadddr team)) (- (first (cddddr team)) (cadr (cddddr team))) (points team)))); suma partidos jugados y resta diferencia de goles

(define remve

(lambda (lista object)

(if (null? lista)

'()

(if (equal? (first lista) object)

(rest lista)

(cons (first lista) (remve (rest lista) object))))))

(define points

(lambda (team)

(if (null? team)

0

(+ (\* (cadr team) 3) (caddr team)))))

(define diffGoals

(lambda (team)

(- (first (cddddr team)) (cadr (cddddr team)))

))



ÁRBOLES BINARIOS

Un Árbol Binario (AB) puede ser representado en Scheme, por medio de una lista en el

siguiente formato: (raíz subárbol-izquierdo subárbol-derecho). Por ejemplo, si se tiene

definido el siguiente AB: Su representación en Scheme sería:

**(define AB ‘(8 (5 (2 () ())**

**(7 () ()))**

**(9 ()**

**(15 (11 () ())**

**() ))))**

6. Implementar la función recursiva nivel que a partir de un árbol binario regrese el

mismo árbol, pero donde el valor de cada nodo represente el nivel en el que se

encuentra dicho nodo. La raíz del árbol se encuentra en el primer nivel.

Ejemplos de prueba:

**> (nivel ‘(1 (2 ()())(3 ()())))** => (1 (2 ()())(2 ()()))

**> (nivel AB)** => (1 (2 (3 ()())(3 ()()))(2 ()(3 (4 ()())()))))

(define (nivel arbol)

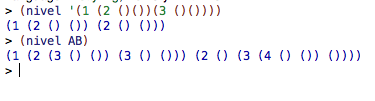
(nivelAux arbol 1))

(define (nivelAux arbol n)

(if (null? arbol) '()

(append (list n (nivelAux (cadr arbol) (+ n 1)))

(list (nivelAux (caddr arbol) (+ n 1))))))



7. Implementar la función recursiva acumulado que a partir de un árbol regrese el

mismo árbol, pero donde el valor de cada nodo represente la suma de todos los nodos

del subárbol del cual ese nodo es la raíz.

Ejemplos de prueba:

> (acumulado ‘(1 (2 ()())(3 ()()))) => (6 (2 ()())(3 ()()))

> (acumulado AB) => (57 (14 (2 ()())(7 ()()))(35 ()(26 (11 ()())()))))

(define (acumulado arbol)

(if (null? arbol)

'()

(append (list (contador arbol)(acumulado (cadr arbol)))

(list (acumulado (caddr arbol))))))

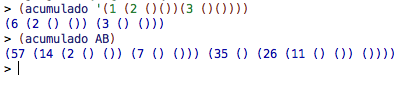
(define (contador arbol)

(if (null? arbol)

0

(+ (+ (car arbol)(contador (cadr arbol)))

(contador (caddr arbol)))))



GRAFOS

Un grafo dirigido y ponderado puede ser representado en Scheme por medio de

una matriz de adyacencias. La matriz de representación del grafo en este caso, es

una lista con una sublista por cada nodo del grafo, que contiene el nombre del nodo y un

valor por cada nodo del grafo. El valor asociado con un nodo hacia el cual tiene un arco

será el peso del mismo, mientras que el valor asociado con un nodo hacia el cual no tiene

arco, será 0. Se debe establecer un orden en los nodos para determinar el número de

renglón que ocupará en la matriz.

8. Implementar la función recursiva costo-ruta que verifique si una secuencia de nodos

es una ruta posible sobre un grafo, en cuyo caso regrese su costo; de lo contrario

regresar el símbolo no-ruta. Una ruta es posible si existe una secuencia de arcos en el

grafo que la describan a partir de su primer nodo.

Ejemplos de prueba:

> (ruta g ‘(A E D)) => no-ruta

> (ruta g ‘(A D E)) => 17

> (ruta g ‘(A B C D E)) => 24

> (ruta g ‘(A B C A B C A)) => 46

(define (ruta grafo lista)

(if (= (rutaAux grafo lista) 0)

'no-ruta

(rutaAux grafo lista)

))

(define (rutaAux grafo lista)

(if (null? (cdr lista))

0

(+(sacarValor (cdr (sacarRenglon grafo (car lista))) (valorLetra (cadr lista)))(rutaAux grafo (cdr lista)))

))

(define (sacarValor lista num)

(if (null? lista)

0

(if(= num 0)

(car lista)

(sacarValor (cdr lista)(- num 1)))))

(define (sacarRenglon grafo letra1)

(if (null? grafo)

'()

(if(equal? letra1 (caar grafo))

(car grafo)

(sacarRenglon (cdr grafo) letra1))))

(define (valorLetra letra)

(cond ((equal? letra 'A)0)

((equal? letra 'B)1)

((equal? letra 'C)2)

((equal? letra 'D)3)

((equal? letra 'E)4)

((equal? letra 'F)5)

((equal? letra 'G)6)

((equal? letra 'H)7)

((equal? letra 'I)8)

((equal? letra 'J)9)

((equal? letra 'K)10)

((equal? letra 'L)11)

((equal? letra 'M)12)

((equal? letra 'N)13)

((equal? letra 'O)14)

((equal? letra 'P)15)

((equal? letra 'Q)16)

((equal? letra 'R)17)

((equal? letra 'S)18)

((equal? letra 'T)19)

((equal? letra 'U)20)

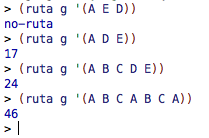
((equal? letra 'V)21)

((equal? letra 'X)22)

((equal? letra 'Y)23)

((equal? letra 'Z)24)

(else 0)))



9. Implementar el predicado recursivo tour? que existe un tour completo en un grafo a

partir de un nodo. Un tour es una ruta que incluye a todos los nodos del grafo.

Ejemplos de prueba:

> (tour? g) => #t

> (tour? ‘((X 0 5)(Y 0 2))) => #f