TC2006 Lenguajes de Programación

Tarea 4: Programación Avanzada en Racket

(* Equipos de 3 integrantes *)

En esta tarea pondrás en práctica tus conocimientos sobre programación recursiva con estructuras de datos y funciones de orden superior en Racket. El código fuente deberá estar documentado internamente mediante comentarios que incluyan, al menos, las matrículas y nombres de los integrantes del equipo, así como la descripción de lo que calcula cada función y el significado de cada parámetro formal de las mismas ***- 10 puntos.***

SECCIÓN I: Utilizando **recursión explícita** (**SIN primitivos de orden superior**) programar las siguientes funciones (**PUEDEN** utilizar funciones auxiliares) **- 60 puntos.**

1. Implementar las siguientes funciones sobre matrices representadas en forma de listas de renglones, utilizando **recursión explícita** (sin primitivos de orden superior). Pueden usar funciones auxiliares.

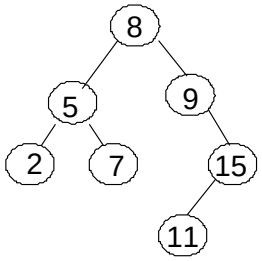
a. Implementar la función **elimina-columna** que regrese la matriz dada sin la columna especificada por su posición en la matriz. Asumir que la columna existe.
Probar con:

```
> (elimina-columna 3 '((4 0 3 1)(5 1 2 1)(6 0 1 1)))
   => '((4 0 1)(5 1 1)(6 0 1))
```

b. Implementar la función **agrega-valor** que agregue un nuevo valor a una matriz dada en la posición (renglón, columna) especificada, de la siguiente forma: si la matriz ya tiene un valor en esa posición, que lo sustituya por el valor a agregar, pero si la matriz no tiene el número de renglones y/o columnas necesarias para colocar el nuevo valor, que agregue los renglones y/o columnas mínimas necesarias para agregar el valor.
Probar con:

```
> (agrega-valor 5 '(1 2) '((1 2 3)(0 2 1)))
   => ((1 5 3)(0 2 1))
> (agrega-valor 4 '(3 5) '((1 2 3)(0 2 1)))
   => ((1 5 3 0 0)(0 2 1 0 0)(0 0 0 0 4))
```

2. Un Árbol Binario de Búsqueda (ABB) puede ser representado en Racket, por medio de una lista en el siguiente formato: (raíz subárbol-izquierdo subárbol-derecho) donde el valor de la raíz de cada (sub)árbol siempre es mayor a igual que los valores de su subárbol izquierdo y menor o igual que los valores de su subárbol derecho.
Por ejemplo, si se tiene definido el siguiente ABB, su representación en Racket sería:




```
(define ABB
    '(8  (5  (2  ()  ())
             (7  ()  ()))
         (9  ()
             (15 (11  ()  ())
                 () ))))
```

a. Implementar la función **rango** que dado un **árbol binario de búsqueda** regrese el rango de los valores contenidos en él como la lista (min max) donde min el valor más pequeño y max es el valor más grande.

Probar con:

```
> (rango ABB)              => (2 15)
> (rango '(8 () ()))       => (8 8)
```

b. Implementar la función **cuenta-nivel** que regrese la cantidad de nodos en cierto nivel de un árbol binario. La raíz se encuentra en el nivel 0.
Probar con:

```
> (cuenta-nivel 3 ABB)          => 1
> (cuenta-nivel 2 ABB)          => 3
> (cuenta-nivel 2 '(a (b (c () ()) ()) (d (e () ()) ()))  => 2
```

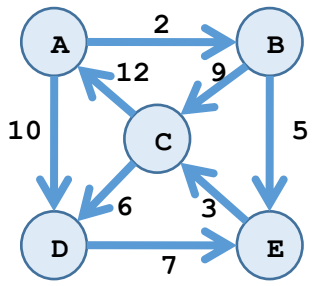
SECCIÓN II: Programar las siguientes funciones **SIN recursión explícita** y utilizando los primitivos de orden superior ***map***, ***apply*** y la forma especial ***lambda*** (**NO DEBEN** utilizar funciones auxiliares) **– 30 puntos**.

3. Un grafo dirigido y ponderado puede ser representado por medio de una lista de arcos. La lista de representación del grafo en este caso, tiene un registro por cada nodo del grafo, y este registro a su vez es una lista con el siguiente formato:

```
(nodo_x (nodo_adyacente_1 peso_arco_1)
        (nodo_adyacente_2 peso_arco_2)
          ...
        (nodo_adyacente_n peso_arco_n))
```

Donde $nodo_x$ es el identificador de un nodo origen en el grafo, $nodo\_adyacente_i$ es el identificador de un nodo destino y $peso\_arco_i$ es la ponderación del arco que une al $nodo_x$ con el $nodo\_adyacente_i$. Bajo este formato, el siguiente grafo tendría la representación en Racket que se muestra:

```
(define g
'((A (B 2) (D 10))
  (B (C 9) (E 5))
  (C (A 12) (D 6))
  (D (E 7))
  (E (C 3))
 )
```



a. Implementar la función **nodos-destino** que liste los nodos destino que tienen a N como nodo origen directo.
Probar con:

```
> (nodos-destino g 'A)      => (B D)
> (nodos-destino g 'D)      => (E)
> (nodos-destino g 'F)      => ()
```

b. Implementar la función **elimina-nodo** que recibe como argumentos un grafo y el nombre de un nodo, y regresa el grafo sin el nodo especificado (si existe). Como se puede ver en los ejemplos, también elimina los arcos dirigidos al nodo eliminado.
Probar con:

```
> (elimina-nodo g 'B)  => ((A (D 10))(C (A 12)(D 6))(D (E 7))(E (C 3)))
> (elimina-nodo g 'D)  => ((A (B 2))(B (C 9)(E 5))(C (A 12))(E (C 3))
> (elimina-nodo g 'C)  => ((A (B 2)(D 10))(B (E 5))(D (E 7))(E))
```