Tarea nº1 Lenguajes de Programación

Víctor Daniel Ríos Tapia - 18.568.652-3

Damian Andrés Reyes Cordero - 18.950.453-5

25 Septiembre 2017

1. Resumen

En el presente informe se explican los procedimientos necesarios para crear un lenguaje de programación simple, utilizando como herramienta el programa DrRacket y el lenguaje de programación Racket.

El lenguaje de programación creado debe ser utilizado para implementar una serie de funciones que nos permitan crear, editar, y manipular grafos.

2. Objetivo

Diseñar un lenguaje de programación simple mediante el uso de Racket. Implementar, con el lenguaje creado, un programa que cree, edite, y modifique grafos de al menos 2 tipos distintos.

3. Introducción

Racket es un lenguaje de programación multiparadigma de propósito general; lo que quiere decir que posee las características necesarias para poder 'ser programado' de distintas maneras y para distintos propósitos, según se requiera; como por ejemplo, para acceso a bases de datos, comunicación entre computadoras, cálculos matemáticos, comunicación entre dispositivos, etc.

Por otra parte, un grafo es un conjunto de nodos (vértices) y enlaces (aristas) que nos permiten representar y estudiar, de manera gráfica, las relaciones que existen entre distintas unidades que interactúan entre sí dentro de un sistema.

Como se señaló anteriormente, Racket puede ser utilizado para modelar comunicaciones entre objetos; y a su vez, las comunicaciones entre objetos pueden ser representadas mediante el uso de grafos; es por esto que, a continuación, se mostrará cómo crear un lenguaje de programación simple con el cual pueda implementarse una serie de funciones que nos permitirán crear y editar grafos en distintas medidas.

4. Desarrollo

Como primera instancia nos enfocaremos en el diseño base que necesitaríamos para poder crear un grafo; dejando de lado el lenguaje de programación y preocupándonos sólo en la parte lógica. Como se dijo anteriormente un grafo es una agrupación de nodos unidos mediante aristas; dicho conjunto se utiliza principalmente para mostrar de forma gráfica el funcionamiento de un sistema; y como existen distintos tipos de sistemas en nuestra vida diaria, también existen distintos tipos de grafos para utilizar. Algunos tipos de grafos son:

- Simple: Grafo que acepta una sola arista por cada 2 nodos cualesquiera. Es la definición estándar de un grafo.
- Cíclico: Consiste en un camino cerrado en el que no se repite ningún nodo, a excepción del primero, ya que este aparece dos veces, como principio y fin del camino.
- Estrella: Tipo de grafo que consiste en un nodo central rodeado de N nodos hojas, todos estos unidos al central por medio de sus respectivas aristas.
- Multigrafo: Grafo que permite más de una arista entre dos vértices. Estas aristas se llaman múltiples o lazos (loops en inglés).
- Dirigido: Son grafos en los cuales se ha añadido una orientación a las aristas, representada gráficamente por una flecha.
- Etiquetado: Grafos en los cuales se ha añadido un peso o valor a las aristas; o bien un etiquetado a los nodos.
- Aleatorio: Grafo cuyas aristas están asociadas a una probabilidad.
- Hipergrafo: Grafos en los cuales las aristas tienen más de dos extremos, es decir, las aristas son incidentes a 3 o más nodos.
- Infinito: Grafos con conjunto de nodos y aristas de cardinalidad infinita.
- Plano: Son aquellos cuyas aristas pueden representarse sin que se crucen entre ellas.
- Regular: Aquel cuyos nodos poseen el mismo grado de valencia. El grado de valencia es un valor que representa la cantidad de aristas que posee un nodo.

Hay varias formas de diseñar grafos; los cuales en su mayoría requieren del uso de vectores, matrices, listas, o cualquier sistema que nos permita almacenar datos de forma ordenada, y al cual sea posible acceder mediante el uso de algún tipo de índice.

Como nuestro lenguaje a crear debe poder manipular al menos 2 tipos distintos de grafos, nos enfocaremos en los grafos cíclicos, y los estrella; ya que estos poseen una dificultad menor al momento de plantearse; por lo tanto, comenzamos definiendo 2 funciones principales que nos permitirán diseñar dichos tipos de grafos, las cuales son:

- Función star: Cuando llamamos a ésta función y le pasamos como parámetro un número entero positivo, nos devuelve un listado de pares que representa un grafo estándar de tipo estrella; el cual está constituido de un nodo 'central' (que llamamos así para una mejor visualización al momento de usar la función) con sus 'n' nodos unidos a él, siendo 'n' el número del parámetro indicado. Si utilizamos estrella y le pasamos un 3, nos mostrará una lista de pares que dirá 'central'.1, 'central'.2, 'central'.3 respectivamente.

  Básicamente la función genera una recursión en donde se muestra por pantalla la creación de los pares sucesivamente, hasta que genera tantos pares como lo indique 'n'.

  A nivel sintáctico la función está dada por:

  $(define(starx)(if(< x1)null(cons(cons"central"x)(star(-x1)))))$

  Y se accede a ella escribiendo: (star n).

Ejemplo:

```
> (star 5)
'(("central" . 5)
  ("central" . 4)
  ("central" . 3)
  ("central" . 2)
  ("central" . 1))
```

- Función circ + circIn: Utilizando la función circ, y adjuntando como parámetro un número entero positivo, nos devuelve una lista de pares, la cual representa al grafo cíclico como una cadena de uniones continuas de todos sus nodos; dicha cantidad de nodos será igual al parámetro 'n' indicado al momento de llamar a la función.

  Dado que un grafo cíclico inicia y termina el recorrido en un mismo nodo, nuestra función circ debe interactuar con otra función, llamada 'circIn'.

  La función circIn nos permite realizar el ciclo de concatenación, tal como lo hacía la funcion star nombrada anteriormente; por otro lado, circ nos permite iniciar y finaliza el ciclo, añadiendo nuevamente el nodo inicial al final de las iteraciones; además, es la función a la que ingresa el parámetro del total de nodos a crear.

  A nivel general todos los pares creados poseen un orden de concatenación, por ejemplo, si llamamos a la función circ con un 3, nos debe entregar una lista lineal con los pares (3.1)(1.2)(2.3).

  Finalmente nuestras 2 funciones, a nivel sintáctico, están dadas por:

  $(define(circx)(cons(consx1)(reverse(circInx))))$

  $(define(circInx)(if(< x2)null(cons(cons(-x1)x)(circIn(-x1)))))$

  Y se accede a ella mediante: (circ n) (que accede internamente a circIn)

  Ejemplo:

```
> (circ 5)
'((5 . 1) (1 . 2) (2 . 3) (3 . 4) (4 . 5))
```

Una vez creadas nuestras funciones bases, llega el momento de pensar en las sub-funciones que nos permitirán 'alterar' nuestro grafo. Las 2 funciones básicas son agregar y borrar nodos de nuestros grafos; las cuales están dadas por:

- Función adde: Nos permite agregar una cantidad 'x' de nodos a un grafo estrella 'y'. Básicamente, 'adde' se une con la funcion 'star' anteriormente nombrada.

Ejemplo:

```
> (adde 4 (star 3))
'(("central" . 7)
  ("central" . 6)
  ("central" . 5)
  ("central" . 4)
  ("central" . 3)
  ("central" . 2)
  ("central" . 1))
```

- Función addc: Nos permite agregar una cantidad 'x' de nodos a un grafo cíclico 'y'. Básicamente, 'addc' se une con la funcion 'circ' anteriormente nombrada.

  Ejemplo:

```
> (addc 3 (circ 2))
'((5 . 1) (1 . 2) (2 . 3) (3 . 4) (4 . 5))
```

Recapitulando, al tener éstas 4 funciones básicas principales es posible crear grafos de tipos estrellas y cíclicos de tamaño 'n' utilizando las funciones 'star' y 'circ'; además, encadenando funciones podemos añadir a dichos grafos cualquier cantidad de nodos que se desee utilizando 'adde' y 'addc', para grafos estrellas y cíclicos respectivamente.

Por otro lado, también podemos borrar nodos, o grupos de éstos con las siguientes funciones:

- Función esStar: Recibe 2 parámetros. Su objetivo es eliminar todos los nodos desde el 1 hasta el indicado por el primer parámetro de un grafo de tipo estrella. El tamaño del grafo es el indicado por el segundo parámetro.

  Ejemplo:

```
> (esStar 3 6)
'(("central" . 6) ("central" . 5) ("central" . 4))
```

- Función esCirc: Recibe 2 parámetros. Su objetivo es eliminar todos los nodos desde el 1 hasta el indicado por el primer parámetro de un grafo de tipo cíclico. El tamaño del grafo es el indicado por el segundo parámetro.

Ejemplo:

```
> (esCirc 3 6)
'((6 . 1) (4 . 5) (5 . 6))
```

- Función enStar: Elimina el nodo indicado por el primer parámetro de un grafo de tipo estrella, cuyo tamaño total es el indicado por el segundo parámetro.

  Ejemplo:

```
> (enStar 2 4)
'((("central" . 4) ("central" . 3)) ("central" . 1))
```

- Función enCirc: Elimina el nodo indicado por el primer parámetro de un grafo de tipo cíclico, cuyo tamaño total es el indicado por el segundo parámetro.

  Ejemplo:

```
> (enCirc 2 4)
'((4 . 1) (3 . 4))
```

Con todas las funciones indicadas hasta el momento es posible generar un grafo cíclico o estrella de cualquier tamaño, además de poder borrar varios nodos al mismo tiempo, o uno si es que se desea. Por otro lado también cabe la posibilidad de agregar nodos a cualquier de los 2 tipos de grafos.

Finalmente la único que falta es crear una forma de determinar si un grafo entregado por parámetro cumple con ser estrella o cíclico; para ello, crearemos las siguientes funciones:

- Función star?: Recibe como parámetro un grafo, y entrega como retorno 'sí' o 'no'. El grafo debe haber sido diseñado previamente en consola, y debe cumplir con el diseño implementado para los grafos estrellas; vale decir, una lista de pares en donde cada par posee el mismo nodo en la parte izquierda (nodo que representa el centro del grafo).

  La función 'star?' recorre la lista término a término, verificando que todos los pares tengan el mismo nodo central. Si la condición anterior se cumple, entonces se da por hecho que el grafo sí es de tipo estrella. La función 'star?' puede ser utilizada tanto para grafos con nodos numéricos como para grafos con nodos de letras.

  A nivel sintáctico la función star? está dada por:

  $(define(star?x)(if(equal?f(equal?(cdrx)null))(if(equal?(car(carx))(car(car(cdrx))))(star?(list*(cdrx)))"No")"Sí"))$

- Función circ? + proceso: Recibe como parámetro un grafo, y entrega como retorno 'sí' o 'no'. El grafo debe haber sido diseñado previamente en consola, y debe cumplir con el diseño implementado para los grafos cíclicos; vale decir, una lista ordenada pares en donde el valor de la derecha del par, debe ser igual al valor de la izquierda del par siguiente; y en donde el valor que posea la parte

izquierda del primer par ha de ser igual que el valor de la derecha del último par.

La función 'circ?' verifica si el el ultimo par y el primero están unidos por sus partes derecha e izquierda respectivamente. De no cumplirse lo anterior la función nos retorna 'no' inmediatamente. Si se cumple la condición se utiliza la función proceso, la cual verifica que todos los nodos centrales de la lista estén unidos, calculando las igualdades de los términos derecho e izquierda de 2 pares continuos. Si esta secuencia se cumple, y dado que se cumplió la primera, se da por hecho de que la lista cumple con ser un grafo cíclico, y se retorna un 'sí'. Si alguno de los términos intermedios no tienen unión, entonces se entiende que el grafo no es continuo y por ende no es cíclico, retornando un 'no'.

A nivel sintáctico la función circ? y proceso están dadas por:

$(define(circ?x)(if(equal?t(equal?(car(carx))(cdr(lastx))))(procesox)"No"))$

$(define(procesox)(if(equal?f(equal?(cdrx)null))(if(equal?(cdr(carx))(car(car(cdrx))))(proceso(list*(cdrx)))"No")"Sí"))$

Finalmente con éstas últimas funciones es posible diseñar un grafo cíclico o estrella por consola, ya sea éste de palabras o de números, y validar si efectivamente cumplen con el diseño planteado mediante 'circ?' y 'star?', para grafos cíclicos y estrellas respectivamente.

5. Conclusión

La programación funcional es un tipo de paradigma que se basa en el uso de funciones matemáticas; las cuales pueden ser declaras con el objetivo de representar diversas instancias, y que a su vez, cada una de las instancias permita realizar ciertas acciones que cambiarán el curso del programa.

Racket es un lenguaje de programación funcional, que posee una alta versatilidad en la redacción de sus funciones, lo que a su vez permite que pueda utilizarse para diseñar lenguajes de programación.

La mayor dificultad que uno puede hallar al momento de utilizar Racket es el manejo correcto de la sintaxis y las funciones. Generar funciones y utilizar la recursividad de forma eficiente son temas que requieren de práctica y de una forma de pensar 'abstracta' la cual, usualmente, no se ve tan fortalecida, debido a la costumbre que se posee de programar de forma imperativa/secuencial.

Quizás una de las razones por la cual Racket no es tan conocido ha de deberse a lo mencionado anteriormente; la barrera de pensamiento que separa a este tipo de lenguajes de otros, produciendo una 'doble inversión', porque no basta con aprender los comandos para generar acciones, sino que, además, debes de cambiar tu forma de pensar para poder utilizarlos.

6. Bibliografía

- Lenguaje Racket e introducción a este: https://beautifulracket.com/ (visitado por última vez el 23 Septiembre a las 10:30 P.M)
- Más sobre Racket y DrRacket: https://es.slideshare.net/JoseHernandezMoya/introduccin-a-dr-racket (visitado por última vez el 23 Septiembre a las 10:30 P.M)
- Utilidad: https://practicaltypography.com/why-racket-why-lisp.html (visitado por última vez el 23 Septiembre a las 10:30 P.M)

7. Anexos

Código completo:

```
#lang racket

(define testCircSTR '(("C" . "A") ("A" . "B") ("B" . "C")))  ;Grafo de testeo, con nodos numéricos, para verificar si es cíclico
(define testCircINT '((3 . 1) (1 . 2) (2 . 3)))  ;Grafo de testeo, con nodos numéricos, para verificar si es cíclico

(define testStarSTR '(("central" . 3) ("central" . 2) ("central" . 1))) ;Grafo de testeo, con nodos alfanuméricos, para verificar si es estr
(define testStarINT '((1 . 3) (1 . 2) (1 . 1))) ;Grafo de testeo, con nodos numéricos, para verificar si es estrella

(define (star x)(if (= x 0) null (cons  (cons "central" x) (star (- x 1) )))) ;Crea un grafo estrella con 'x' cantidad de nodos.

(define (circIn x)(if (< x 2)null (cons  (cons (- x 1) x) (circIn (- x 1))))) ;Crea un grafo cíclico sin último nodo

(define (circ x) (cons (cons x 1) (reverse(circIn x)))) ;Une el nodo final e inicial de un grafo cíclico

(define (adde x y) (star (+(cdr (car y))x)));Agrega la cantidad deseada 'x' de nodos a un grafo estrella de tamaño 'y'

(define (addc x y) (circ (+(caar y) x)));Agrega la cantidad deseada 'x' de nodos a un grafo cíclico de tamaño 'y'

(define (esStar x y) (remove* (star x) (star y))) ;Elimina desde el primer nodo hasta el nodo 'x' de un grafo estrella de tamaño 'y'

(define (esCirc x y) (remove* (circ (+ x 1)) (circ y))) ;Elimina desde el primer nodo hasta el nodo 'x' de un grafo cíclico de tamaño 'y'

(define (enStar x y) (list* (remove* (star x) (star y)) (cdr(star x)))) ;Elimina el nodo seleccionado 'x' de un grafo estrella de tamaño 'y'

(define (enCirc x y) (list* (remove* (circ (+ x 1)) (circ y)) )) ;Elimina el nodo seleccionado 'x' de un grafo cíclico de tamaño 'y'

(define (star? x) (if (equal? #f (equal? (cdr x) null )) (if(equal? (car(car x)) (car(car(cdr x)))) (star? (list* (cdr x))) "No" ) "Sí" ) )

(define (circ? x) (if (equal? #t (equal? (car (car x)) (cdr (last x)))) (proceso x) "No")) ;Indica si el grafo X, indicado por parámetro, es

(define (proceso x) (if (equal? #f (equal? (cdr x) null)) (if(equal? (cdr(car x)) (car(car(cdr x)))) (proceso (list* (cdr x))) "No" ) "Sí"))
```

Link código y otras cosas en GitHub: https://github.com/damianichi/TareaLenguaje/