UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENERÍA INFORMÁTICA

INFORME DE LABORATORIO: IMPLEMENTACIÓN DE UN SOFTWARE DE MANIPULACIÓN DE IMÁGENES EN DR. RACKET

DIEGO TORO BARRENECHEA
20.807.369-9

Paradigmas de Programación (13310-A1)
Profesor Roberto Gonzales I.

Índice

1. INTRODUCCIÓN **3**

1.1 DESCRIPCIÓN DEL PROBLEMA 3
1.2 DESCRIPCIÓN DEL PARADIGMA 3
1.3 OBJETIVOS 3

2. DESARROLLO **3**

2.1 ANÁLISIS DEL PROBLEMA 3
2.2 DISEÑO DE LA SOLUCIÓN 4
2.2.1 TIPOS DE DATOS CREADOS 4
2.2.1.1 TDA Image *4*
2.2.1.2 TDA pixbit-d *4*
2.2.1.3 TDA pixhex-d *5*
2.2.1.4 TDA pixrgb-d *5*
2.2.1.5 TDA histogram *5*
2.2.2 FUNCIONES OBLIGATORIAS 5
2.2.3 ASPECTOS DE IMPLEMENTACIÓN 6
2.3 INSTRUCCIONES DE USO 7
2.4 POSIBLES ERRORES 7
2.5 RESULTADOS ESPERADOS 7

3. CONCLUSION **7**

3.2 RESULTADOS OBTENIDOS 7
3.3 AUTOEVALUACIÓN 8
3.4 PALABRAS FINALES 8

4. BIBLIOGRAFÍA **9**

5. ANEXO **10**

1. Introducción

El presente informe se encuentra asociado al laboratorio 1 del ramo Paradigmas de Programación, en este laboratorio se trabaja con el paradigma funcional sobre el lenguaje de programación Scheme, siendo utilizado Dr. Racket como compilador de este mismo. Durante la extensión de este curso se presentará el problema a través de una breve descripción y veremos cual fue la forma de abordar este problema a través del análisis del problema, diseño de la solución y aspectos de la implementación, se explicará como poder usar el programa, que resultados obtuvimos y como fue nuestra autoevaluación para finalizar con la conclusión.

1.1 Descripción del problema

Se solicita crear un software de manipulación de imágenes bajo un enfoque simplificado, algo similar a lo que hacen programas como Photoshop o GIMP pero de una forma mucho más sencilla y haciendo uso del paradigma funcional en este laboratorio. El programa nos permite crear imágenes como un tipo de dato abstracto, ingresándoles ancho, alto y especificándole todos los pixeles que se necesiten para formar esa imagen. A las imágenes se le pueden aplicar modificaciones como rotarla en 90 grados, comprimirla, invertir sus colores y un montón de otras más.

1.2 Descripción del paradigma

El paradigma funcional tiene como unidad básica de abstracción y programación a las funciones, además utiliza la programación declarativa, lo que significa que responde a la pregunta de ¿Qué? y no del ¿Cómo? el cual es la aproximación común de los lenguajes de programación más populares ya que usualmente se debe especificar la forma de llegar al resultado, mientras que aquí solo nos importa el resultado más no como llegamos a este. Esto se traduce en un alto nivel de abstracción.

Para lograr la programación declarativa el programa se expresa con propiedades y reglas que deben cumplirse en lugar de instrucciones.

Cabe mencionar que Scheme no es un lenguaje puro de programación funcional ya que tiene aspectos de programación imperativa que han sido aplicados a lo largo del proyecto.

1.3 Objetivos

El objetivo principal de esta experiencia es aprender el paradigma funcional y las bases de este al poder aplicarlo a través de la programación, esto nos permitirá tener otro enfoque diferente al que se programa comúnmente, una nueva mirada más abstracta para abordar los diferentes problemas que se nos presenten. Otro objetivo corresponde a aprender a programar en Scheme a través de su compilador Dr. Racket. Por último se espera un buen entendimiento del problema a tratar ya que será algo que abordaremos durante tres laboratorios.

2. Desarrollo

2.1 Análisis del problema

El problema que se nos presenta es diseñar el software para manipular imágenes según lo indicado, para eso debemos de crear las estructuras de datos adecuadas para cumplir con todas las funciones requeridas que la mayoría son de la manipulación de la misma imagen, para eso nuestra estructura principal será *image* que estará compuesta por un ancho, un alto y una lista de pixeles,

estos últimos también corresponderán a un TDA según el tipo de representación que tengan, en este caso pueden ser *pixbit* (el color está representado por un *bit*), *pixhex* (el color está representado por un *hex*) o *pixrgb* (el color está representado por valores *RGB*). Las funciones solicitadas son las siguientes:

- ***bitmap?***: revisa si la imagen ingresada está compuesta por pixbit-d.
- ***pixmap?***: revisa si la imagen ingresada está compuesta por pixrgb-d
- ***hexmap?***: revisa si la imagen ingresada está compuesta por pixhex-d
- ***compressed?***: revisa si la imagen esta comprimida bajo la función *compress*
- ***flipH:*** voltea horizontalmente la imagen
- ***flipV:*** voltea verticalmente la imagen
- ***crop:*** recorta la imagen que se encuentren dentro del cuadro especificado en los puntos ingresados.
- ***imgRBG->imgHex:*** transforma una imagen constituida por pixrgb-d a pixhex-d
- ***histogram:*** devuelve las frecuencia con la que aparece cada color.
- ***rotate90:*** rota en 90º en sentido de las manecillas del reloj
- ***compress:*** elimina de la imagen los pixeles con el color que más se repite
- ***edit:*** aplica una función ingresada sobre la imagen
- ***invertColorBit:*** invierte los valores de los *bits* (de 0 pasa a 1 y viceversa).
- ***invertColorRGB:*** invierte los valores de los canales *r*, *g* y *b* (de 255 a 0, de 100 a 155, etc…)
- ***adjustChannel***: modifica los valores del canal especificado (*r, g* o *b)* según la función ingresada.
- ***image->string***: convierte la imagen en una representación a través de string.
- ***depthLayers***: devuelve todas las capas según la cantidad de profundidad que tengan los pixeles
- ***decompress***: permite descomprimir las imágenes que fueron comprimidas con *compress*.

2.2 Diseño de la solución

Para esta solución fueron empleados 5 tipos de datos distintas que en apoyo a las nativas otorgadas por Scheme nos permiten trabajar de forma correcta. Estas 5 estructuras son ***image, pixbit-d, pixrgb-d, pixhex-d*** e ***histogram***. Estos TDA cuentan con constructores, selectores, modificadores y otras funciones las cuales nos dan ciertas facilidades para la manipulación y creación de funciones.

2.2.1 Tipos de datos creados

2.2.1.1 TDA Image

Representación: Una lista que contiene el ancho, el alto y una lista de pixeles (del mismo tipo)
Constructor: Se le debe entregar el ancho, el alto y (*ancho x alto*) pixeles.
Funciones de pertenencia: Revisar Tabla Nº1 en Anexo

2.2.1.2 TDA pixbit-d

Representación: Una lista que contiene un número que representa su posición en el eje x, otro para la posición en el eje y, su color en *bits* (0|1) y la profundidad.
Constructor: Se le debe entregar su posición en el eje x, eje y, un bit y la profundidad

Funciones de pertenencia: Revisar Tabla Nº2 en Anexo

2.2.1.3 TDA pixhex-d

Representación: Una lista que contiene un número que representa su posición en el eje x, otro para la posición en el eje y, su color en valor hexadecimal y la profundidad.
Constructor: Se le debe entregar su posición en el eje x, eje y, un valor hexadecimal y la profundidad
Funciones de pertenencia: Revisar Tabla Nº3 en Anexo

2.2.1.4 TDA pixrgb-d

Representación: Una lista que contiene un número que representa su posición en el eje x, otro para la posición en el eje y, su color en valor RGB, siendo cada canal un número por separado y la profundidad.
Constructor: Se le debe entregar su posición en el eje x, eje y, valor del canal r, valor del canal g, valor del canal b y la profundidad
Funciones de pertenencia: Revisar Tabla Nº4 en Anexo

2.2.1.5 TDA histogram

Representación: Una lista que contiene uno o más pares según la cantidad de colores que cuente la imagen, estos pares son de la forma (color . veces que se repite)
Constructor: Se le debe entregar una imagen.
Funciones de pertenencia: Revisar Tabla Nº5 en Anexo

2.2.2 Funciones obligatorias

Nota: Todas las recursiones aplicadas en este programa son de cola, por ende, en el documento se referirá simplemente a recursión ya que es el único tipo aplicado.

- bitmap?: de forma recursiva se revisa cada pixel, que tenga la cantidad de elementos adecuadas (4) y que el tercer valor corresponda a un bit con la función *bit?*.
- pixmap?: de forma recursiva se revisa cada pixel, que tenga la cantidad de elementos adecuadas (6) y que el tercer, cuarto y quinto valor correspondan al tipo *rgb?*, o sea entre 0 y 255.
- hexmap?: de forma recursiva se revisa cada pixel, que tenga la cantidad de elementos adecuadas (4) y que el tercer valor corresponda a un hexadecimal con la función *hex?*. Esta función revisa que el string sea de 6 caracteres y los caracteres sean [0, 9] o de la A hasta la F.
- compressed?: se revisa que la lista de pixeles sea concordante con la cantidad de pixeles que debiesen haber según el ancho y alto de la imagen.
- flipH: de forma recursiva se invierten los valores en el eje X usando el ancho para restárselo.
- flipV: de forma recursiva se invierten los valores en el eje Y usando el alto para restárselo.
- crop: de forma recursiva se revisa si los pixeles se encuentran dentro de los márgenes definidos por (x_1, y_1) *e* (x_2, y_2)

- imgRGB->imgHex: de forma recursiva pasa a través de todos los pixeles y le aplica la función *RGB->Hex* para convertirlo a hexadecimal.
- rotate90: a través de la recursión y usando la formula para realizar una rotación respecto al origen en un plano cartesiano los rota, después se traslada para que queden en el primer cuadrante.
- compress: usando los datos entregados por el histograma busca los pixeles que tengan el color más repetido y los elimina.
- edit: un llamado a la función entregada y se le pasa la imagen.
- invertColorBit: a cada bit se le resta uno y le aplica el valor absoluto.
- invertColorRGB: a cada canal RGB se le resta 255 y le aplica valor absoluto.
- adjustChannel: según las funciones pasadas para elegir, modificar un canal y cual es la función que se le quiere aplicar se hace sobre la imagen.
- incCh: toma el valor del canal y lo aumenta en 1.
- img->string: llama a la función correspondiente al tipo de imagen entregada
- pixbit->string: ordena los pixeles con la función *sort-pixels* la cual se encuentra especificada en el anexo debido a su importancia, luego recursivamente extrae el bit y los concatena en una cadena haciendo uso de *\n* y *\t*.
- pixhex->string: con el apoyo de *sort-pixels* recursivamente extrae el valor hexadecimal y los concatena en una cadena haciendo uso de *\n* y *\t*.
- pixrgb->string: haciendo uso de *imgRGB->imgHex* se convierte a imagen hexadecimal para poder aplicar la función de arriba y ahorrar código.

depthLayer y *decompress* no pudieron ser implementadas en ese programa.

La principal herramienta utilizada fue la recursividad, ya que era algo que se encontraba creado y se podía reciclar el código, el enfoque para la recursividad de cola es usar una función envolvente principal, y que esta llame por primera vez a la recursión, y desde ahí se siga llamando hasta terminar. Otras funciones que fueron un gran aporte fueron *map* y *sort*, especialmente la última en funciones como *compress* y *image->string* ya que se le puede aplicar una *#:key* para ordenar según ese valor.

2.2.3 Aspectos de Implementación

La estructura del programa consiste en 5 archivos del tipo TDA donde están los constructores, funciones de pertenencia, modificadores y otras funciones que permiten apoyar la creación de las funciones obligatorias. Las funciones obligatorias se encuentran en el archivo *main*, debido a que así se solicitó y permite una mejor revisión, existe una excepción en ciertas funciones que no pueden estar en el archivo *main* debido a que son usadas en otros TDA que al importar *main* lanza un error, igualmente estas funciones particulares quedan especificadas donde se encuentran. No se usaron bibliotecas y el compilador fue Dr. Racket.

Un aspecto importante a mencionar es que por restricciones del constructor del TDA *image*, hay veces que no se puede usar (principalmente en las recursiones) ya que el constructor debe ser ingresado de la forma (*width, height pixel1 pixel2 pixel3 ...)* mientas que con la recursión resulta *(width height '(pixel1 pixel2 pixel3 ...))*.

2.3 Instrucciones de uso

Para correr el programa basta con ejecutar el archivo *main* en Dr. Racket y correrlo, desde ahí se puede hacer uso de todas las funciones contenidas en este programa.

Para la creación de una imagen se puede usar el siguiente código:

(define bit-image (image 2 2 (pixbit-d 1 1 0 0) (pixbit-d 1 2 1 0) (pixbit-d 2 1 0 1) (pixbit-d 2 2 1 1)))

Y a partir de ese se pueden usar todas las funciones, como
(bitmap? img) (flipH img) (rotate90 img)
etc…

2.4 Posibles errores

Dentro de los posibles errores se encuentra la función *adjustChannel*, debido a que no logré implementarla de la forma solicitada que es *(edit (adjustChannel getR setR incCh) img1)* y tiene que ser utilizada como *(edit (adjustChannel getR setR incCh img1) img1)).*
La función *img->string* devuelve la imagen como si se viera en el primer cuadrante de un plano cartesiano, no como esta representada en los ejemplos del laboratorio que es una forma espejo desde el primer al segundo cuadrante. No todas las funciones cuentan con validaciones por lo que es posible ingresar datos con el tipo de dato equivocado y que estos produzcan un error en la función y no arroje un error previo por parte del software (y no del compilador).

Otro error se encuentra en la función *crop* y *compress*, ya que estas funciones **modifican la cantidad de pixeles** de la imagen, y al usar los datos del ancho y alto para muchas operaciones de otras funciones estas pueden resultar erróneas.

2.5 Resultados esperados

Se espera lograr una ejecución perfecta de todas las funciones solicitadas bajo los ejemplos entregados por el profesor y los que use para la evaluación, además de haber entendido y aplicado el paradigma funcional junto al lenguaje de programación Scheme.

3. Conclusion

3.2 Resultados obtenidos

Se lograron realizar la mayoría de las funciones requeridas a excepción de las dos últimas, aunque las funciones no realizan todas las validaciones necesarias de los datos ingresados se vela que el usuario ingrese los valores indicados. A excepción de un par de errores y pequeñas modificaciones señaladas en el punto 2.4 está todo bien.

Para revisar que todo esté bien se hicieron múltiples pruebas con códigos propios y códigos entregados por el profesor logrando que todo estuviera bien. El único problema es después de usar *crop* y *compress* aplicar otras funciones sobre estas, ya que como se menciona en el punto 2.4 los cálculos pueden (y suelen) resultar erróneos.

3.3 Autoevaluación

La autoevaluación va entre 0 y 1, representando el porcentaje de veces que funciona dicha implementación.
Para ver la tabla de autoevaluación, ver Tabla Nº6 en el Anexo.
Debido a que todas las funciones pueden ser aplicadas sobre imágenes que han sido modificadas por *crop* o *compress* han sido asignadas con un máximo de 0.75, ya que no pueden funcionar en todos los casos.

3.4 Palabras finales

El proyecto ya se encunetra terminado, queda un sabor agrío por no haber podido realizar todas las funciones y que las funciones *crop* y *compress* arruinen la efectividad de todas las demás. Feliz de poder aprender un nuevo paradigma y lenguaje de programación, el que me acompañara y me permitirá darle un nuevo enfoque a los problemas que me toque enfrentar en un futuro no tan lejano.

El hecho de trabajar con un paradigma tan distinto a lo que se está acostumbrado fue una de las grandes complicaciones de este trabajo, entrenando nuevamente el cerebro para trabajar con este paradigma, pero a medida que se acercaban las ultimas funciones las ideas estaban mucho más pulidas y necesitaban menos correcciones una vez eran plasmadas en código.

4. Bibliografía

Chacon, S. & Straub, B. (2006, 26 agosto). Pro Git. Apress.

Flatt, M., Findler, R. B. & PLT. (s. f.). The Racket Guide. Recuperado 26 de septiembre de 2022, de https://docs.racket-lang.org/guide/index.html

Gonzales, R. (s. f.-a). 2022_02 Laboratorio 1. Google Docs. Recuperado 26 de septiembre de 2022, de https://docs.google.com/document/d/1hUAooKwBj3TWv-yuzBZtNbuaC8iNkzOZdbLpD8P9B8c/edit#

Gonzales, R. (s. f.-b). 2022_02 Laboratorio (General). Google Docs. Recuperado 26 de septiembre de 2022, de https://docs.google.com/document/d/1D6g3S3vLC-zziOsSprLIBPypkd89s2QEkJs1F_kbHm4/edit

Learn racket in Y Minutes. (s. f.). Recuperado 26 de septiembre de 2022, de https://learnxinyminutes.com/docs/es-es/racket-es/

Racket Documentation. (s. f.). Recuperado 26 de septiembre de 2022, de https://docs.racket-lang.org/index.html

5. Anexo

Tipo de función	Nombre	Descripción
Pertenencia	image?	Revisa si corresponde a un TDA image
Selector	getWidth	Extrae el ancho de la imagen
Selector	getHeight	Extrae el alto de la imagen
Selector	getPixels	Extrae los pixeles de la imagen
Otras funciones	sort-pixels	Crea una lista por cada fila de pixeles (eje Y) ordenados de menor a mayor respecto al eje X
Otras funciones	recursion-sort-pixels	Apoyo a función sort-pixels
Otras funciones	recursion-bit->str	Apoyo a función pixbit->string
Otras funciones	recursion-hex->str	Apoyo a función pixhex->string
Otras funciones	recursion-fliph	Apoyo a función flipH
Otras funciones	recursion-flipv	Apoyo a función flipV
Otras funciones	recursion-crop	Apoyo a función crop
Otras funciones	RGB->hex	Convierte valores RGB a valor hexadecimal
Otras funciones	recursion-rgb->hex	Apoyo a función RGB->hex
Otras funciones	recursion-rotate90	Apoyo a función rotate90
Otras funciones	recursion-compress	Apoyo a función compress
Otras funciones	recursion-compress-rgb	Apoyo a función compress

Tabla 1. Funciones del TDA image

Tipo de función	Nombre	Descripción
Pertenencia	bit?	Revisa si el valor es un bit (0 o 1)
Pertenencia	bitmap?	Revisa si la imagen esta compuesta por pixbit-d
Selector	getBit	Extrae el valor del bit de un pixbit-d
Otras funciones	recursion-bit	Apoyo a función bitmap?

Tabla 2. Funciones del TDA pixbit

Tipo de función	Nombre	Descripción
Pertenencia	hex?	Revisa si el string es un valor hexadecimal
Pertenencia	hexmap?	Revisa si la imagen esta compuesta por pixhex-d
Selector	getHex	Extrae el valor hexadecimal de un pixhex-d
Otras funciones	recursion-hex	Apoyo a función hexmap?

Tabla 3. Funciones del TDA pixhex

Tipo de función	Nombre	Descripción
Pertenencia	rgb?	Revisa si el numero esta dentro del rango RGB (0, 255)
Pertenencia	pixmap?	Revisa si la imagen esta compuesta por pixrgb-d
Selector	getR	Extrae el valor del canal R de un pixrgb-d
Selector	getG	Extrae el valor del canal G de un pixrgb-d
Selector	getB	Extrae el valor del canal B de un pixrgb-d
Modificadores	setR	Modifica el valor del canal R de un pixrgb-d
Modificadores	setG	Modifica el valor del canal G de un pixrgb-d
Modificadores	setB	Modifica el valor del canal B de un pixrgb-d

Otras funciones	recursion-rgb	Apoyo a función pixmap?

Tabla 4. Funciones del TDA pixrgb

Tipo de función	**Nombre**	**Descripción**
Otras funciones	eq-rgb?	Compara dos listas de valores RGB para ver si es el mismo color

Tabla 5. Funciones del TDA histogram

Funciones	**Evaluación**
TDAs	1
bitmap?	1
pixmap?	1
hexmap?	1
compressed?	1
flipH	1
flipV	1
crop	1
imgRGB->imgHex	1
histogram	1
rotate90	1
copmress	1
edit	0.75
invertColorBit	1
invertColorRGB	1
adjustChannel	0.25
image->string	1
depthLayers	0
decompress	0

Tabla 6. Autoevaluación

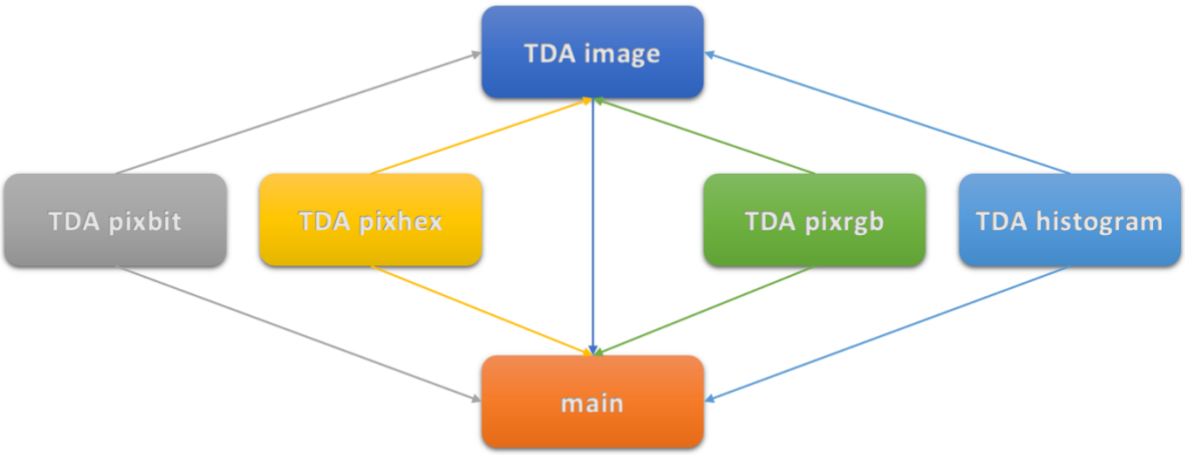

Figura 1. Dependencias de los TDA y archivos.