Universidad de santiago de Chile

Labor Lætitia nostra

Informe para el ramo Paradigmas de programación

Laboratorio 1: Paradigma Funcional
Editor de imágenes usando Scheme, DrRacket

Sergio Osvaldo Andres Espinoza Gonzalez

Facultad de ingeniería

Septiembre de 2022

Índice

Índice de cuadros

1. Introducción

En este informe se abordará el laboratorio número 1 de la asignatura Paradigmas de programación con el formato expuesto en el indice del mismo. A continuación, se darán breves descripciones de los dos grandes pilares a ver para encontrar la posterior resolución.

1.1. Descripción del problema

Para empezar debemos conocer el problema a resolver, el cual es la creación de un software que permita la edición de imágenes de manera similar al software de código abierto GIMP o el de pago conocido como Photoshop.

1.2. Descripción del paradigma

Para la resolución del problema planteado anteriormente, se hará uso de un lenguaje de programación llamado Racket el cual es un derivado de Scheme que se encuentra enfocado en el paradigma de programación funcional, ya teniendo en cuenta esto podría surgirnos la pregunta "¿Que es el paradigma funcional?"

1.2.1. Paradigma funcional

El paradigma funcional es aquel que, como dice su nombre, hace uso de funciones y todo lo relacionado con las mismas, es decir, funciones de orden superior, evaluación de funciones en tiempo real, currificación y más.(Ramiro, 2019)

2. Análisis del problema

Ya entendiendo por encima el problema y el paradigma a utilizar, para solucionarlo profundizaremos en el mismo, ya que, conociendo bien el problema y sus requisitos específicos llegaremos a una mejor solución.

1. TDAs: El primer desafio será diseñar los tipos de datos abstractos que nos permitan alcanzar la solución de la manera más eficiente posible, por lo que este primer requisito es más que nada de diseño de la solución.
2. Contructor de imágenes: Ya habiendo mencionado los TDAs el que principalmente se llevará nuestra atención será aquel que represente una imagen como tal, ya que, al implementarlo, debemos tener diferentes consideraciones como:
   - Tipo de imagen: En un principio se deberían tener 3 tipos de imagen correspondientes a bitmap, pixmap y hexmax y que el programa pueda reconocer de que tipo se trata; la implementación de lo antes mencionado se verá en la siguiente sección.
   - Debe ser posible también poder cambiar entre imágenes tipo pixmap y hexmap (RGB y hexadecimal).
   - ¿Está comprimida?: Es necesario saber si una imagen está o no comprimida
   - Comprimir y descomprimir: Como ya se dijo que una imagen puede estar o no comprimida entonces deben haber funciones que compriman y descompriman la imagen como tal, dependiendo de la implementación podría verse también como requisito una función que nos diga cuál es el color más usado en la imagen (Histograma).
3. Cambiar dirección o rotar: Otro requisito especifífico a abordar, es dar la posibilidad de invetir la imagen ya sea en el eje X o Y, además de poder rotar esta en 90° hacia la derecha o izquierda.
4. Recortar: También se debe dar la posibilidad de recortar la imagen para obtener una nueva imagen de un subsector de la original.
5. Invertir colores: Como el nombre lo dice, se debe ser capaz de invertir los colores independiente del tipo de imagen.

6. Ajustar canal: Debe también darse la posibilidad de ajustar el canal de una imagen con pixeles RGB-D (con profundidad).

7. Imagen a string: Debe también cubrirse la posibilidad de que la imagen pueda ser representada como un conjunto de caracteres.

8. Separar en capas: La imagen debe tener profundidad, y para esto, se deben tener varias capas entre las que le es posible separarse.

3. Diseño de la solución

Comenzando ahora con el diseño de la solución para la problemática de este semestre, se realizará la definición y especificación de los distintos TDAs para luego ver como puede ser la posible implementación de los mismos y lo que se realizará para los problemas específicos.

3.1. TDAs

Empezando con los TDAs el enfoque será que una imagen esta compuesta de pixeles los cuales son de alguno de los siguientes 3 tipos, pixbit-d, pixrgb-d o pixhex-d; haciendo los TDAs de estos 3 tipos de pixel junto con el TDA imagen podríamos lograr la mayoría de los requisitos específicos sin problema, sin embargo para un manejo de ciertos requisitos específicos como el de que se pase una imagen de color RGB a hexadecimal donde no entraría en ninguno de los TDAs de pixeles específicamente, podríamos definir un quinto TDA "pixeles"donde hayan funciones de 2 o más tipo de pixeles, o, en otro caso, hagan uso de un conjunto de pixeles del mismo tipo. Finalmente se definiría un último TDA histogram para el requisito con el mismo nombre.

3.1.1. Definición

Ya sabiendo cuales son los TDA a implementar se procederá con su definición:

- TDA image: Una imagen será aquel conjunto de pixeles con un ancho y un alto específicos que puede estar o no comprimida, por lo cual, se debe poder saber cual es el color más usado en la misma.
- TDA pixeles: Será aquel conjunto con bajo grado de especificación donde se podrán realizar operaciones con 2 o más tipos de pixeles, ó, con 2 o más pixeles del mismo tipo.
- TDA pixbit-d: Será aquel pixel con una posición específica en una imagén de 3 dimensiones con 2 posibles colores, blanco o negro.
- TDA pixrgb-d: Será aquel pixel con una posición específica en una imagén de 3 dimensiones con color de tipo rgb.
- TDA pixhex-d: Será aquel pixel con una posición específica en una imagén de 3 dimensiones con color de tipo hexadecimal.
- TDA histogram: Será el gráfico de cuantos pixeles de cada color hay contenidos en una imagén

3.1.2. Especificación

La especificación general se puede encontrar en el cuadro 1 de los anexos; si se desean tener más detalles se recomienda ver el código de cada TDA donde se específica de manera más detallada debido al poco espacio de este medio.

3.1.3. Implementación

La implementación de una imagen será con una lista que contendrá un bit (int 0 ó 1) para saber si esta comprimida o no donde 1 será comprimida y 0 no; tendrá también 2 enteros positivos que son el ancho y el alto de la misma, junto con una lista con los diferentes pixeles que tiene; por último tendrá un elemento con el color más usado en la imagen, cuyo tipo dependerá del tipo de imagen que sea (bitmap, pixmap o hexmap).

Los pixeles en sí estarán un poco más adelante, pero algo que los 3 tipos de estos tendrán en común, es que serán implementados con listas «"'¡HEAD donde el segundo y tercer elemento corresponden a la posición (x,y) del pixel y el último elemento de la lista corresponderá a la profundidad del pixel, donde, estos 3 valores mencionados, son enteros positivos.
======= donde el primer y segundo elemento corresponden a la posición (x,y) del pixel y el último elemento de la lista corresponderá a la profundidad del pixel, donde, estos 3 valores mencionados son enteros positivos.
'"»¿8e23d0c5baa17fd80f687b0fed619b5925b384f0

La gran diferencia se encontrará en los datos que hay entre medio de los valores (x,y) y depth (profundidad) los cuales corresponden al color del pixel; los TDA de pixeles quedarían de la siguiente forma:

- Pixbit-d: Este tipo de pixel tendrá un bit entre medio de tipo entero, el cual solo podrá tomar los valores 0 y 1. De esta forma un pixbit-d será una lista de la forma:

$$'(int, int, int, int) = (x \geq 0, y \geq 0, bit(0|1), depth \geq 0)$$

- Pixrgb-d: Este tipo de pixel tendrá además tendrá 3 enteros entre 0 y 255 que representarán la cantidad de cada color que tiene el pixel siendo r = rojo (red), g = verde (green) y b = azul (blue) quedandonos una lista de la forma:

$$'(int, int, int, int, int, int) = (x \geq 0, y \geq 0, 0 \leq r \leq 255, 0 \leq g \leq 255, 0 \leq b \leq 255, depth \geq 0)$$

- Pixhex-d: Este tipo de pixel tendrá un string con formato #XXXXXX donde las X pueden tomar valores de un dígito hexadecimal (entre 0 y F(15)) donde las 2 primeras X es la cantidad de rojo, las 2 de al medio son de verde y las 2 últimas son de azul; la representación quedaría de la forma:

$$'(int, int, string, int) = (x \geq 0, y \geq 0, hex, depth \geq 0)$$

Por otro lado el histograma será una lista de pares en los cuales el primer elemento será un color y el segundo la cantidad de veces que ese color se repite.

4. Consideraciones de implementación

El código del proyecto estará en la carpeta Código, donde se encontrará el main con las funciones principales del enunciado; en esta carpeta se encontrará también otra carpeta que contiene los TDAs mencionados en la sección anterior "TDAs". No se usará ninguna biblioteca externa al proyecto y el compilador usado será el incluido en DrRacket.

5. Instrucciones de uso

En un principio las funciones se harán pensando en que no es necesario que se ingresen todos los pixeles a usar dentro de una imagen, ya que, se puede rellenar de ser necesario; ni que tengan un orden en concreto al ingresarse siempre y cuando podamos saber su posición con sus coordenadas (x,y); debido a esto no hay muchas instrucciones de uso como tal, más allá de que se cumpla el formato de los ejemplos del cuadro 2, aunque claro, hay muchos más ejemplos en el archivo de pruebas.

Tener en cuenta que:

- img1: (image 2 1 (pixrgb-d 0 0 0 140 255 0) (pixbit-d 1 0 140 0 255 0))
- img2: (image 1 2 (pixbit-d 0 0 0 0) (pixbit-d 0 1 0 0))

Las imagenes de ejemplo son pequeñas para que quepan en la tabla

6. Resultados obtenidos

Los resultados de la implementación y sus requisitos especifícos fueron resumidos en el cuadro 3 junto con cuales se completaron, el puntaje de funcionamiento según la pauta dada de autoevaluación (entre 0 y 1) y la posible razón de los fallos; las pruebas realizadas fueron evaluaciones de funciones en el archivo de pruebas que se subirá a la plataforma github y uvirtual.

7. Conclusión

Partiendo bajo la premisa de que practicamente todo se puede representar con funciones, el alcance de este paradigma se vuelve casi ilimitado, pero, como bien es sabido, no tenemos la costumbre de usarlo. Debido a lo anterior, pudieron haber sido un poco lentos los avances en un principio, esto porque, a pesar de poder usar funciones en el paradigma imperativo procedural al que estamos tan acostumbrados, se suele usar variables mutables en estas, en cambio, el paradigma de programación funcional directamente no posee variables como tal, lo que puede ser una importante limitación para alguien que viene del imperativo procedural.

Finalmente destacaría que, a pesar de no tener la comodidad de usar elementos que siempre se han usado en las asignaturas anteriores de la carrera, este es un paradigma de programación muy interesante y que me divertí bastante practicandolo en DrRacket para la elaboración de este laboratorio y espero aprender más de el en el futuro.

Referencias

Ramiro, G. (2019). La programación funcional: Un poderoso paradigma. (3), 63-77. Descargado de http://revistas.ucasal.edu.ar/index.php/CI/article/view/182

Anexos

X	Image	Histogram	Pixels	Pixbit-d	Pixrgb-d	Pixhex-d
Constructor	Sí	Sí	No	Sí	Sí	Sí
Pertenencia	Sí	No	No	Sí	Sí	Sí
Selectores	Sí	No	Sí	Sí	Sí	Sí
Modificadores	No	No	Sí	Sí	Sí	Sí
Otras funciones	Sí	No	Sí	No	No	No

Cuadro 1: Especificación de TDAs

Función	Uso	Resultado esperado
2. image	(image 2 1 (pixbit-d 0 0 0 0) (pixbit-d 1 0 0 0))	'(0 2 1 (0 0 1 0) (1 0 1 0) 1)
3. bitmap?	(bitmap? (img1)) donde img1	#f
4. pixmap?	(pixmap? (img1))	#t
5. hexmap?	(hexmap? (img1))	#f
6. compressed?	(compressed? img1)	#f
7. flipH	(flipH img2)	'(0 1 2 (0 0 0 0) (0 1 0 0) 0)
8. flipV	(flipV img2)	'(0 1 2 (0 1 0 0) (0 0 0 0) 0)
9. crop	(crop img2 0 0 0 0)	'(0 1 1 (0 0 0 0) 0)
10. imgRGB->imgHex	(imgRGB->imgHex img1)	'(0 2 1 (0 0 #008CFF 0) (1 0 #8C00FF 0) #008CFF)
11. histogram	(histogram img2)	'((0 2) (1 0))
12. rotate90	(rotate90 img2)	'(0 2 1 ((1 0 0 0) (0 0 0 0)) 0)
13. compress	(compress img2)	'(1 1 2 () 0)
14. edit	Esta función se usa en conjunto con otras	N/A
15. invertColorBit	(edit invertColorBit img2)	'(0 1 2 (0 0 1 0) (0 1 1 0) 0)
16. invertColorRGB	(edit invertColorRGB img2)	'(0 2 1 (0 0 255 115 0 0) (1 0 115 255 0 0) (0 140 255))
17. adjustChannel	(edit (adjustChannel getpixrgb.r changepixrgb.r incCh) img1)	'(0 2 1 (0 0 1 141 0 0) (1 0 141 1 0 0) (1 141 0))
18. image->string	(image->string img2 pixbit->string)	"0\t\n0\t\n"
19. depthLayers	(depthLayers img2)	'((0 1 2 (0 0 0 0) (0 1 0 0) 0))
20. decompress	(decompress (compress img2))	'(0 1 2 (0 0 0 0) (0 1 0 0) 0)

Cuadro 2: Ejemplos de uso

Requisito	Completado?	Puntaje autoevaluación	Posible razón fallos
1. TDAs	Sí	1	N/A
2. image	Sí	1	N/A
3. bitmap?	Sí	1	N/A
4. pixmap?	Sí	1	N/A
5. hexmap?	Sí	1	N/A
6. compressed?	Sí	1	N/A
7. flipH	Sí	1	N/A
8. flipV	Sí	1	N/A
9. crop	Sí	1	N/A
10. imgRGB->imgHex	Sí	1	N/A
11. histogram	Sí	1	N/A
12. rotate90	Sí	1	N/A
13. compress	Sí	1	N/A
14. edit	Sí	1	N/A
15. invertColorBit	Sí	1	N/A
16. invertColorRGB	Sí	1	N/A
17. adjustChannel	Sí	1	N/A
18. image->string	Sí	1	N/A
19. depthLayers	Sí	0.25	N/A
20. decompress	Sí	0.25	N/A

Cuadro 3: Resultados obtenidos