Universidad de Santiago de Chile

Facultad de Ingeniería Informática

Paradigmas de Programación

Profesor Roberto González

Laboratorio 1
Paradigma Funcional

Nicolás Aguilera

Septiembre 2022

Índice

1. Introducción

El siguiente informe corresponde al desarrollo del Laboratorio 1 del curso *Paradigmas de Programación*, el cual consiste en la aplicación del *Paradigma Funcional* de programación a la resolución de un problema específico mediante el uso del lenguaje *Scheme* y el compilador *DrRacket*.

1.1. Descripción del problema

El problema consiste en la creación de funciones para el tratamiento imágenes RGB-D, similar a herramientas como GIMP y Adobe Photoshop, pero de forma simplificada. Para esto, se considera que las imagenes tienen un alto y un ancho (width x height) y que están compuestas por tres tipos de pixeles diferentes, pudiendo formar tres tipos de imagenes diferentes:

- **bitmap-d:** El valor que pueden tomar sus pixeles es de 0 o 1, indicando que el pixel se encuentra apagado o encendido (negro o blanco).
- **pixmap-d:** Los pixeles tienen tres canales de colores (R, G y B) los cuales toman valores de 0 a 255.
- **hexmap-d:** Los pixeles expresan sus canales RGB en formato hexadecimal.

La posición de cada pixel dentro de la imagen y la profundidad corresponden a valores enteros entre cero y el máximo, dependiendo de las dimensiones de la imagen y la profundidad de esta.

1.2. Descripción del paradigma

El paradigma funcional se enfoca en la aplicación de funciones, basado en el **cálculo lambda**, sin la existencia de variables tal y como se conoce en paradigmas como el *imperativo*, típico de lenguajes como *C* y *Python*. Esto facilita el formalismo y la expresión de funciones. A continuación se presentan algunos de los conceptos más importantes del paradigma:

- **Funciones:** Las cuales tienen un *dominio* y un *recorrido*, de manera similar a las funciones matemáticas. Las funciones son procedimientos que pueden entregar un valor númerico, una lista (o par), y en el caso en particular de este paradigma, otra función que también tiene un dominio y un recorrido.
- **Currificación:** Es una forma de convertir una función que recibe n parámetros en $n-1$ funciones de un solo parámetro. Muy útil para ocultar aspectos de implementación al usuario del programa.
- **Pares y listas:** En el caso de Scheme, los pares son la unidad base de estructuras más complejas como lo son las listas. Por otro lado, tanto pares y listas pueden ser conformadas por tipos de datos heterogéneos como podrían ser enteros, strings o TDAs.

- **Recursividad:** A diferencia de paradigmas como el Imperativo, la inexistencia de **ciclos** para recorrer listas obliga el uso de la recursividad en sus diferentes formas: natural, de cola, arbórea, entre otras, aunque tiene otro tipo de usos.

2. Desarrollo

2.1. Análisis del problema

El foco principal del problema es el tratamiento de imágenes mediante funciones utilizando la programación funcional. Por lo tanto, es necesario hacer mencionar y describir los elementos que conforman el problema y así orientar una solución hacia la construcción e interacción de estos elementos:

- **Imagen:** Unidad fundamental del problema. Está constituida por un alto y ancho, así como como pixeles, los cuales deben responder a las dimensiones de la imagen (alto x ancho).
- **Pixel:** Las imágenes están conformadas por sub-unidades llamadas *pixeles*, los cuales pueden ser de diferentes tipos: pixbit-d, pixrgb-d y pixhex-d, conformando los bitmap-d, pixmap-d y hexmap-d, respectivamente. Los pixeles a su vez, tienen una posición dentro de la imagen representada por coordendas x e y, un color (especificado anteriormente en *Descripción del problema*) y una profundidad.
- **Color:** Cada pixel tiene un color el cual es representado por un valor entero (o varios en el caso del pixrgb-d) o una cadena de caracteres (en hexadecimal) (en el caso de los pixhex-d).
- **Profunidad:** Además, los pixeles tienen una profundidad, que representa la "lejanía.[a] la que se encuentra ese pixel, desde el plano de la pantalla u hoja. Esta característica ayuda a la representación tridimensional de una imagen.

2.2. Diseño de la solución

2.2.1. TDAs implementados

Tal y como se mencionó anteriormente, la solución para el tratamiento de imágenes recae en una buena abstracción y representación los TDAs que son parte del problema, de tal manera que la implementación de la solución sea lo más sencilla posible y no caiga en redundancias. Esto significa que cada TDA debe estar compuesto por *Constructores*, *Funciones de Pertenencia*, *Selectores* y *Modificadores* cumpliendo cada uno sus roles específicos a la hora de implementar un TDA.

Todo la implementación de los TDAs está sujeta a una representación basada en pares y/o listas, por lo que fue importante considerar de qué manera se agrupaban los parámetros que conforman cada TDA. A continuación se describen los TDAs considerados para la solución del problema:

- **TDA image:** Su representación está dada por una lista que contiene los siguientes elementos: el **ancho** (width), el **alto** (height), el **color comprimido** de la imagen (el que será nulo si la imagen no está comprimida), una **lista de pixeles** que componen la imagen y por último una lista de **pixeles comprimidos** que contiene información reducida de los pixeles que no están en la imagen y será utilizada para su posterior recuperación (descompresión).
- **TDA Pixel:** Su representación está dada por una lista que contiene los siguientes elementos: una lista de dos elementos (**posición** x e y del pixel en la imagen), **color** del pixel (representación binaria, canales RGB y representación RGB hexadecimal) y **profundidad** del pixel.

Debido a que todos los tipos de pixeles contienen los mismos parámetros (posición, color y profundidad), se decidió considerar el **TDA Pixel** como TDA padre para la representación de los distintos tipos de pixeles. Esto debido a la redundancia de repetir algunos *Selectores*, y *Modificadores* como para obtener la posición de un pixel o su profundidad. Por lo tanto, dentro de los *Constructores* del TDA Pixel se consideraron tres funciones diferentes, una para cada tipo de pixel (pixbit-d, pixrgb-d y pixhex-d), al igual que en los *Modificadores*, donde se encuentran funciones que modifican el color de un tipo específico de pixel. Esto entrega una ventaja al momento de implementar *Selectores* y *Modificadores* en común para los tres tipos de imagen, como funciones que rotan o voltean una imagen. Para un detalle más exhaustivo de los TDAs y sus funciones, revisar los Cuadros 1, 2 y 3 del *Anexo*.

2.2.2. Algoritmos y técnicas empleadas

Tal y como se especificó en el material entregado para el desarrollo del proyecto, todos los TDAs y algoritmos asociados a la implementación de funciones se basó en pares y listas, por lo que los subproblemas que requerían de modificaciones de listas, utilizaron la *recursión de cola* debido a la ventaja computacional que entrega frente a la recursión natural al construir el resultado paso a paso, y dada también la naturaleza del paradigma y del lenguaje de programación esta opción era la mejor ya que la mayoría de los subproblemas consistían en modificaciones y construcciones de listas a partir de otras listas.

2.3. Aspectos de implementación

2.3.1. Estructura del proyecto

Se utilizaron archivos diferentes para ambos TDAs implementados: **TDAImage.rkt**[1] y **TDAPixel.rkt**[2]. Además se tiene un tercer archivo, el cual es un script de prueba que muestra y ejemplifica el uso de todas las funciones requeridas para la implementación de la solución. Este archivo tiene el nombre de **prueba_19527704_AguileraGonzalez.rkt**.

[1]Realmente llamado: **TDAImage_19527704_AguileraGonzalez.rkt**
[2]Realmente llamado: **TDAPixel_19527704_AguileraGonzalez.rkt**

2.3.2. Compilador y bibliotecas utilizadas

El lenguaje de programación utilizado es *Scheme* y el compilador es *DrRacket* en su versión 8.6, del cual solo se utilizó su biblioteca de funciones nativas, como parte de las exigencias del Laboratorio.

2.4. Instrucciones de uso

Para hacer uso de la implementación propuesta, se deben tener los tres archivos mencionados anteriormente y ejecutar el script de prueba de las funciones. En caso de no utilizar este script, se deben utilizar los *Constructores* del TDA Image y el TDA Pixel para construir una imagen. Luego, para tratar la imagen creada, se le pueden aplicar cualquiera de las funciones especificadas en los *Requerimientos Funcionales* y que se encuentran en el Cuadro 1 del *Anexo*. Algunos ejemplos de estas funciones aplicadas a imagenes tipo *bitmap* y *pixmap* pueden encontrarse en las Figuras 1 y 2 de la Sección 2 del *Anexo*.

2.4.1. Resultados esperados

Se espera que el usuario pueda crear y modificar imagenes mediante las funciones de tratamiento: rotar, voltear, comprimir e incluso mostrar los pixeles en pantalla (en formato hexadecimal), entre otras, visualizando los resultados en la terminal del compilador según la representación de los TDAs especificados con anterioridad.

2.4.2. Posibles errores

Dado que se asumió que el usuario utilizaría las funciones de forma adecuada y por lo tanto no introducirá valores fuera de los límites o tipos de datos inadecuados, puede que al hacer esto, las imágenes que cree el usuario y las modificaciones que le haga, provoquen errores que serán mostrados en la terminal del compilador.

2.5. Resultados y autoevaluación

En general, los resultados obtenidos son los esperados, logrando la creación y manipulación de imágenes a través de la aplicación de funciones usando la programación funcional.

Los criterios y resultados de la *Autoevaluación* de requisitos funcionales pueden encontrarse en los Cuadros 4 y 5 del Anexo.

2.5.1. Funciones no completadas

La única función que no se pudo completar fue **adjustChannel**, debido a la necesidad de implementarse como función de orden superior y crear funciones selectoras y modificadoras específicas para la utilización de esta.

3. Conclusión

Luego del desarrollo del Laboratorio, se puede concluir que se cumplió con los requisitos generales y particulares del proyecto. En primer lugar, crear un programa para el tratamiento de imágenes utilizando la programación funcional a través del lenguaje de programación *Scheme* y el compilador *DrRacket*. También se cumplió con la creación de TDAs acorde a los requisitos funcionales del proyecto, logrando la abstracción y representación de cada tipo de dato, en particular, para el TDA Image y TDA Pixel implementados en la solución.

En cuanto a las dificultades para utilizar el paradigma funcional, se encuentran la costumbre de utilizar lenguajes de programación basados en el paradigma imperativo, tales como *C* y *Python*, por lo que la transición a este paradigma prescindiendo de ciclos y variables fue un reto. Naturalmente está también el tiempo invertido en aprender los conceptos que rodean el paradigma (cálculo lambda, recursividad y currificación, entre otros) y el lenguaje de programación en particular y su notación poco común. Finalmente, la etapa de diseño de la solución

4. Bibliografía y referencias

1. Dybvig, R. K. (2009). *The Scheme programming language*. Mit Press. Recuperado de: `https://www.scheme.com/tspl4/`

2. Flatt M. & Findler R. (2022). *The Racket Guide*. Recuperado de: `https://docs.racket-lang.org/guide/`

3. González R. (2022). *Paradigmas de Programación: Proyecto semestral de laboratorio*. Recuperado de: Enunciado General del Proyecto Semestral

4. González R. (2022). *Paradigmas de Programación: Proyecto semestral de laboratorio. Laboratorio 1*. Recuperado de: Enunciado Laboratorio 1

5. Anexo

5.1. Sección 1

TDA Image - Requerimientos		
Nombre	**Tipo de función**	**Descripción**
image	Constructor	Crea una imagen de tipo bitmap-d, pixmap-d o hexmap-d
bitmap?	Pertenencia	Determina si una es tipo bitmap-d
pixmap?	Pertenencia	Determina si una es tipo pixmap-d
hexmap?	Pertenencia	Determina si una es tipo hexmap-d
compressed?	Pertenencia	Determina si una imagen está comprimida
flipH	Modificador	Voltea una imagen de forma horizontal
flipV	Modificador	Voltea una imagen de forma vertical
crop	Modificador	Recorta una imagen dado un cuadrante determinado por dos puntos (x_1, y_1) y (x_2, y_2)
rotate90	Modificador	Rota una imagen 90 grados en sentido antihorario
imgRGB->imgHex	Modificador	Transforma una imagen tipo pixmap-d en una tipo hexmap-d
compress	Modificador	Comprime una imagen eliminando los pixeles con el color más frecuente
invertColorBit	Modificador	Cambia el valor de los pixbit-d de la imagen al valor opuesto
invertColorRGB	Modificador	Cambia el valor de los canales RGB al valor simétricamente opuesto
decompress	Modificador	Descomprime una imagen, recuperando los pixeles previamente eliminados junto a su información
edit	Otras	Función de orden superior que permite aplicar funciones especiales a las imágenes
image->string	Otras	Muestra una imagen con sus pixeles representados como string
depthLayers	Otras	Crea una lista de imagenes que contienen pixeles de la misma profundidad
histogram	Otras	Entrega una lista de los colores que componen la imagen junto a su frecuencia

Cuadro 1: TDA Image y sus requerimientos funcionales.

TDA Image - Funciones adicionales		
Nombre	**Tipo de función**	**Descripción**
insideCrop?	Pertenencia	Verifica si un pixel de la imagen se encuentra dentro del cuadrante de corte
getWidth	Selector	Entrega el ancho de la imagen
getHeight	Selector	Entrega el alto de la imagen
getCompColor	Selector	Entrega el color comprimido de la imagen
getPixels	Selector	Entrega la lista de pixeles que conforman la imagen
getCompPixels	Selector	Entrega la lista de pixeles comprimidos
sortImage	Modificador	Ordena los pixeles de una imagen según su posición x y luego su posición y
imgColor	Otras	Crea una lista con los colores que contienen los pixeles de la imagen, sin repetir
colorFreq	Otras	Función auxiliar que crea una lista de colores de una imagen y su frecuencia
setPixels	Otras	Aplica una función modificadora a la lista de pixeles de una imagen
mostFreqColor	Otras	Entrega el color más frecuente de la imagen
compressPixels	Otras	Retorna una lista con los pixeles comprimidos
pixbit->string	Otras	Retorna la imagen bitmap-d sin modificaciones si es de ese tipo
pixrgb->string	Otras	Retorna la imagen pixmap-d cambiando la representación de canales RGB de entero a hexadecimal
pixhex->string	Otras	Retorna la imagen hexmap-d si la imagen es de ese tipo
imgDepths	Otras	Entrega una lista con las profundidades que tiene la imagen, sin repetir

Cuadro 2: Funciones adicionales del TDA Image.

TDA Pixel		
Nombre	**Tipo de función**	**Descripción**
pixbit-d	Constructor	Crea un pixel tipo pixbit-d
pixrgb-d	Constructor	Crea un pixel tipo pixrgb-d
pixhex-d	Constructor	Crea un pixel tipo pixhex-d
pixbit?	Pertenencia	Verifica si un pixel es de tipo pixbit-d
pixrgb?	Pertenencia	Verifica si un pixel es de tipo pixrgb-d
pixhex?	Pertenencia	Verifica si un pixel es de tipo pixhex-d
getPosX	Selector	Entrega la posición x del pixel
getPosY	Selector	Entrega la posición y del pixel
getBit	Selector	Entrega el valor del bit de un pixbit-d
rgbChannel	Selector	Entrega los valores de los canales RGB de un pixrgb-d
redChannel	Selector	Entrega el valor del canal R de un pixrgb-d
greenChannel	Selector	Entrega el valor del canal G de un pixrgb-d
blueChannel	Selector	Entrega el valor del canal B de un pixrgb-d
getHex	Selector	Entrega el color hexadecimal de un pixhex-d
getColor	Selector	Entrega el color de cualquier tipo de pixel
getDepth	Selector	Entrega el valor de la profundidad del pixel
setPosX	Modificador	Cambia el valor de coordenada x de un pixel
setPosY	Modificador	Cambia el valor de coordenada y de un pixel
intStr	Otras	Convierte los valores de canales RGB en su interpretación hexadecimal
intHex	Otras	Transforma el canal RGB a su interpretación hexadecimal

Cuadro 3: Funciones que conforman el TDA Pixel.

Evaluación	**Descripción**
0	No se cumple o no funciona
0.25	Falla la mayoría de las veces (funciona el 25 % de las veces)
0.5	Funcionamiento irregular (50 % de las veces)
0.75	Funcionamiento con problemas menores (75 % de las veces funciona)
1	Se cumple o funciona todo el tiempo

Cuadro 4: Evaluación y descripción utilizada para la Autoevaluación.

Función	Evaluación
TDAs	1
image	1
bitmap?	1
pixmap?	1
hexmap?	1
compressed?	1
flipH	1
crop	0.75
imgRGB->imgHex	1
histogram	1
rotate90	1
compress	1
edit	0.75
invertColorBit	1
invertColorRGB	1
adjustChannel	0
image->string	0.5
depthLayers	1
decompress	1

Cuadro 5: Requerimientos funcionales y su evaluación.

5.2. Sección 2

```
172 ;Creamos una imagen bitmap-d de 4x2 con 8 pixeles tipo pixbit-d
173 (define img1 (image 4 2
174                     (pixbit-d 0 0 1 2)
175                     (pixbit-d 1 0 1 4)
176                     (pixbit-d 2 0 1 2)
177                     (pixbit-d 3 0 0 2)
178                     (pixbit-d 0 1 0 4)
179                     (pixbit-d 1 1 0 2)
180                     (pixbit-d 2 1 0 1)
181                     (pixbit-d 3 1 1 0)))
182
183 img1
184
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
'(4 2 () (((0 0) "1" 2) ((1 0) "1" 4) ((2 0) "1" 2) ((3 0) "0" 2) ((0 1) "0" 4) ((1 1) "0" 2) ((2 1) "0" 1) ((3 1) "1" 0)) ())
```

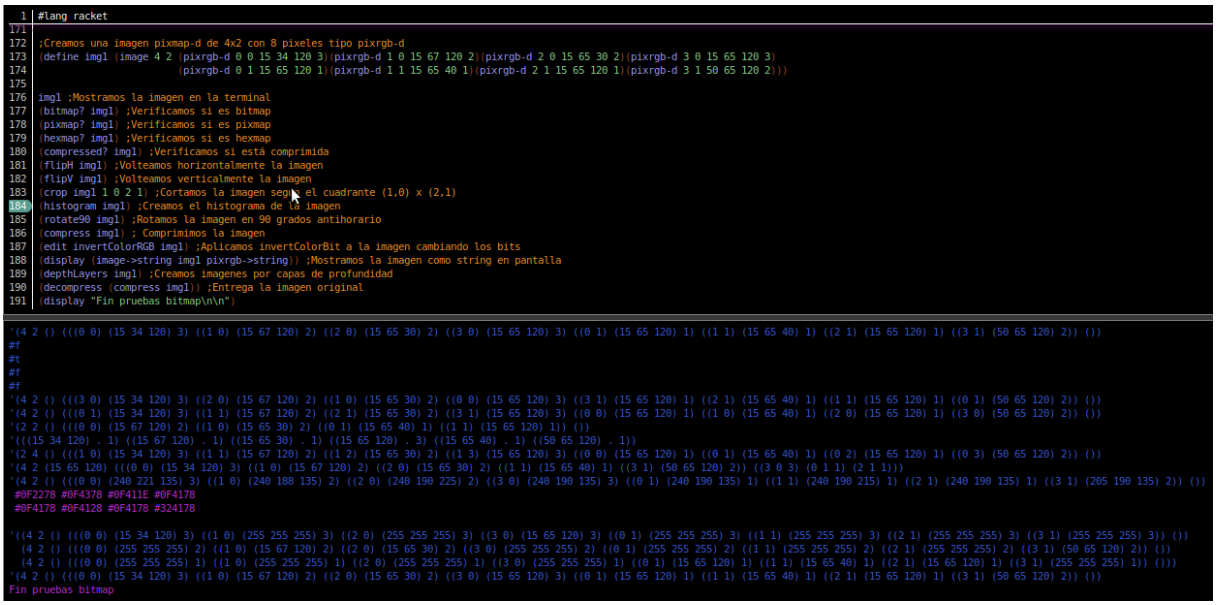
Figura 1: Creación de una imagen pixmap.

```
1  #lang racket
171
172 ;Creamos una imagen bitmap-d de 4x2 con 8 pixeles tipo pixbit-d
173 (define img1 (image 4 2 (pixbit-d 0 0 1 2) (pixbit-d 1 0 1 4) (pixbit-d 2 0 1 2) (pixbit-d 3 0 0 2)
174                         (pixbit-d 0 1 0 4) (pixbit-d 1 1 0 2) (pixbit-d 2 1 0 1) (pixbit-d 3 1 0 0)))
175
176 img1 ;Mostramos la imagen en la terminal
177 (bitmap? img1) ;Verificamos si es bitmap
178 (pixmap? img1) ;Verificamos si es pixmap
179 (hexmap? img1) ;Verificamos si es hexmap
180 (compressed? img1) ;Verificamos si está comprimida
181 (flipH img1) ;Volteamos horizontalmente la imagen
182 (flipV img1) ;Volteamos verticalmente la imagen
183 (crop img1 1 0 2 1) ;Cortamos la imagen según el cuadrante (1,0) x (2,1)
184 (histogram img1) ;Creamos el histograma de la imagen
185 (rotate90 img1) ;Rotamos la imagen en 90 grados antihorario
186 (compress img1) ; Comprimimos la imagen
187 (edit invertColorBit img1) ;Aplicamos invertColorBit a la imagen cambiando los bits
188 (display (image->string img1 pixbit->string)) ;Mostramos la imagen como string en pantalla
189 (depthLayers img1) ;Creamos imagenes por capas de profundidad
190 (decompress (compress img1)) ;Entrega la imagen original
191 (display "Fin pruebas bitmap\n\n")
```

```
'(4 2 () (((0 0) "1" 2) ((1 0) "1" 4) ((2 0) "1" 2) ((3 0) "0" 2) ((0 1) "0" 4) ((1 1) "0" 2) ((2 1) "0" 1) ((3 1) "0" 0)) ())
#t
#f
#f
#f
'(4 2 () (((3 0) "1" 2) ((2 0) "1" 4) ((1 0) "1" 2) ((0 0) "0" 2) ((3 1) "0" 4) ((2 1) "0" 2) ((1 1) "0" 1) ((0 1) "0" 0)) ())
'(4 2 () (((0 1) "1" 2) ((1 1) "1" 4) ((2 1) "1" 2) ((3 1) "0" 2) ((0 0) "0" 4) ((1 0) "0" 2) ((2 0) "0" 1) ((3 0) "0" 0)) ())
'(2 2 () (((0 0) "1" 4) ((1 0) "1" 2) ((0 1) "0" 2) ((1 1) "0" 1)) ())
'(("1" . 3) ("0" . 5))
'(2 4 () (((1 0) "1" 2) ((1 1) "1" 4) ((1 2) "1" 2) ((1 3) "0" 2) ((0 0) "0" 4) ((0 1) "0" 2) ((0 2) "0" 1) ((0 3) "0" 0)) ())
'(4 2 "0" (((0 0) "1" 2) ((1 0) "1" 4) ((2 0) "1" 2)) ((3 0 2) (0 1 4) (1 1 2) (2 1 1) (3 1 0)))
'(4 2 () (((0 0) "0" 2) ((1 0) "0" 4) ((2 0) "0" 2) ((3 0) "1" 2) ((0 1) "1" 4) ((1 1) "1" 2) ((2 1) "1" 1) ((3 1) "1" 0)) ())
 1 1 1 0
 0 0 0 0

'((4 2 () (((0 0) "1" 2) ((1 0) "1" 2) ((2 0) "1" 2) ((3 0) "0" 2) ((0 1) "1" 2) ((1 1) "0" 2) ((2 1) "1" 2) ((3 1) "1" 2)) ())
  (4 2 () (((0 0) "1" 4) ((1 0) "1" 4) ((2 0) "1" 4) ((3 0) "1" 4) ((0 1) "0" 4) ((1 1) "1" 4) ((2 1) "1" 4) ((3 1) "1" 4)) ())
  (4 2 () (((0 0) "1" 1) ((1 0) "1" 1) ((2 0) "1" 1) ((3 0) "1" 1) ((0 1) "1" 1) ((1 1) "1" 1) ((2 1) "0" 1) ((3 1) "1" 1)) ())
  (4 2 () (((0 0) "1" 0) ((1 0) "1" 0) ((2 0) "1" 0) ((3 0) "1" 0) ((0 1) "1" 0) ((1 1) "1" 0) ((2 1) "1" 0) ((3 1) "0" 0)) ()))
'(4 2 () (((0 0) "1" 2) ((1 0) "1" 4) ((2 0) "1" 2) ((3 0) "0" 2) ((0 1) "0" 4) ((1 1) "0" 2) ((2 1) "0" 1) ((3 1) "0" 0)) ())
Fin pruebas bitmap
```

Figura 2: Script de prueba para una imagen tipo bitmap.



```
#lang racket

;Creamos una imagen pixmap-d de 4x2 con 8 pixeles tipo pixrgb-d
(define img1 (image 4 2 (pixrgb-d 0 0 15 34 120 3)(pixrgb-d 1 0 15 67 120 2)(pixrgb-d 2 0 15 65 30 2)(pixrgb-d 3 0 15 65 120 3)
                        (pixrgb-d 0 1 15 65 120 1)(pixrgb-d 1 1 15 65 40 1)(pixrgb-d 2 1 15 65 120 1)(pixrgb-d 3 1 50 65 120 2)))

img1 ;Mostramos la imagen en la terminal
(bitmap? img1) ;Verificamos si es bitmap
(pixmap? img1) ;Verificamos si es pixmap
(hexmap? img1) ;Verificamos si es hexmap
(compressed? img1) ;Verificamos si está comprimida
(flipH img1) ;Volteamos horizontalmente la imagen
(flipV img1) ;Volteamos verticalmente la imagen
(crop img1 1 0 2 1) ;Cortamos la imagen según el cuadrante (1,0) x (2,1)
(histogram img1) ;Creamos el histograma de la imagen
(rotate90 img1) ;Rotamos la imagen en 90 grados antihorario
(compress img1) ; Comprimimos la imagen
(edit invertColorRGB img1) ;Aplicamos invertColorBit a la imagen cambiando los bits
(display (image->string img1 pixrgb->string)) ;Mostramos la imagen como string en pantalla
(depthLayers img1) ;Creamos imagenes por capas de profundidad
(decompress (compress img1)) ;Entrega la imagen original
(display "Fin pruebas bitmap\n\n")
```

```
'(4 2 () (((0 0) (15 34 120) 3) ((1 0) (15 67 120) 2) ((2 0) (15 65 30) 2) ((3 0) (15 65 120) 3) ((0 1) (15 65 120) 1) ((1 1) (15 65 40) 1) ((2 1) (15 65 120) 1) ((3 1) (50 65 120) 2)) ())
#f
#t
#f
#f
'(4 2 () (((3 0) (15 34 120) 3) ((2 0) (15 67 120) 2) ((1 0) (15 65 30) 2) ((0 0) (15 65 120) 3) ((3 1) (15 65 120) 1) ((2 1) (15 65 40) 1) ((1 1) (15 65 120) 1) ((0 1) (50 65 120) 2)) ())
'(4 2 () (((0 1) (15 34 120) 3) ((1 1) (15 67 120) 2) ((2 1) (15 65 30) 2) ((3 1) (15 65 120) 3) ((0 0) (15 65 120) 1) ((1 0) (15 65 40) 1) ((2 0) (15 65 120) 1) ((3 0) (50 65 120) 2)) ())
'(2 2 () (((0 0) (15 67 120) 2) ((1 0) (15 65 30) 2) ((0 1) (15 65 40) 1) ((1 1) (15 65 120) 1)) ())
'(((15 34 120) . 1) ((15 67 120) . 1) ((15 65 30) . 1) ((15 65 120) . 3) ((15 65 40) . 1) ((50 65 120) . 1))
'(2 4 () (((1 0) (15 34 120) 3) ((1 1) (15 67 120) 2) ((1 2) (15 65 30) 2) ((1 3) (15 65 120) 3) ((0 0) (15 65 120) 1) ((0 1) (15 65 40) 1) ((0 2) (15 65 120) 1) ((0 3) (50 65 120) 2)) ())
'(4 2 (15 65 120) (((0 0) (15 34 120) 3) ((1 0) (15 67 120) 2) ((2 0) (15 65 30) 2) ((1 1) (15 65 40) 1) ((3 1) (50 65 120) 2)) ((3 0 3) (0 1 1) (2 1 1)))
'(4 2 () (((0 0) (240 221 135) 3) ((1 0) (240 188 135) 2) ((2 0) (240 190 225) 2) ((3 0) (240 190 135) 3) ((0 1) (240 190 135) 1) ((1 1) (240 190 215) 1) ((2 1) (240 190 135) 1) ((3 1) (205 190 135) 2)) ())
 #0F2278 #0F4378 #0F411E #0F4178
 #0F4178 #0F4128 #0F4178 #324178

'((4 2 () (((0 0) (15 34 120) 3) ((1 0) (255 255 255) 3) ((2 0) (255 255 255) 3) ((3 0) (15 65 120) 3) ((0 1) (255 255 255) 3) ((1 1) (255 255 255) 3) ((2 1) (255 255 255) 3) ((3 1) (255 255 255) 3)) ())
  (4 2 () (((0 0) (255 255 255) 2) ((1 0) (15 67 120) 2) ((2 0) (15 65 30) 2) ((3 0) (255 255 255) 2) ((0 1) (255 255 255) 2) ((1 1) (255 255 255) 2) ((2 1) (255 255 255) 2) ((3 1) (50 65 120) 2)) ())
  (4 2 () (((0 0) (255 255 255) 1) ((1 0) (255 255 255) 1) ((2 0) (255 255 255) 1) ((3 0) (255 255 255) 1) ((0 1) (15 65 120) 1) ((1 1) (15 65 40) 1) ((2 1) (15 65 120) 1) ((3 1) (255 255 255) 1)) ()))
'(4 2 () (((0 0) (15 34 120) 3) ((1 0) (15 67 120) 2) ((2 0) (15 65 30) 2) ((3 0) (15 65 120) 3) ((0 1) (15 65 120) 1) ((1 1) (15 65 40) 1) ((2 1) (15 65 120) 1) ((3 1) (50 65 120) 2)) ())
Fin pruebas bitmap
```

Figura 3: Script de prueba para una imagen tipo pixmap.