Logotipo  Descripción generada automáticamente con confianza mediaUna caricatura de una persona  Descripción generada automáticamente con confianza bajaUniversidad de Santiago de Chile

Facultad de Ingeniería

Departamento de Ingeniería Informática

Laboratorio 2: Editor de imágenes con Prolog

Catalina Jofré León.

Profesor: Miguel Truffa.

Fecha: 2 de noviembre de 2022.

TABLA DE CONTENIDOS

[1- INTRODUCCIÓN 3](#_Toc118121223)

[2- SOBRE EL PARADIGMA 3](#_Toc118121224)

[3- DESCRIPCION DEL PROBLEMA 3](#_Toc118121225)

[4- ANALISIS DEL PROBLEMA 4](#_Toc118121226)

[5- DISEÑO DE LA SOLUCIÓN 5](#_Toc118121227)

[5.1 CREACION DE LOS TDA. 5](#_Toc118121228)

[5.1.1 TDA IMAGE 5](#_Toc118121229)

[5.1.2 TDA PIXBIT 5](#_Toc118121230)

[5.1.3 TDA PIXHEX 5](#_Toc118121231)

[5.1.4 TDA PIXRGB 5](#_Toc118121232)

[6- ASPECTOS DE IMPLEMENTACION 6](#_Toc118121233)

[7- INSTRUCCIONES DE USO 6](#_Toc118121234)

[8- RESULTADOS OBTENIDOS 6](#_Toc118121235)

[9- CONCLUSIONES 7](#_Toc118121236)

[10- BIBLIOGRAFIA Y REFERENCIAS 8](#_Toc118121237)

[11- ANEXOS 9](#_Toc118121238)

[Tabla de funciones. 9](#_Toc118121239)

[Manual de uso 10](#_Toc118121240)

# INTRODUCCIÓN

En el presente informe se explicará detalladamente el segundo laboratorio del semestre de la materia de Paradigmas de Programación, además de la solución planteada e implementada, y su paradigma asociado. Este proyecto trata de la aplicación del paradigma lógico en el lenguaje de programación Prolog, utilizando la plataforma online o la aplicación de escritorio, ambas obtenidas de la pagina web “https://www.swi-prolog.org”, para la creación de un programa de edición de imágenes que permite diversas operaciones en estas; por esto mismo es necesario una buena comprensión de la problemática y de un adecuado uso de las herramientas entregadas, tales como paradigma lógico, abstracciones del problema, el uso de Tipos de Dato Abstracto o TDA por sus siglas en ingles y el uso de recursión.

El desarrollo de este informe consta de una breve explicación sobre el paradigma lógico, para luego tener el primer acercamiento al proyecto con una contextualización del problema a resolver, junto con un análisis de este y para comprender razonamiento utilizado para la solución y sus fundamentos, todo esto enlazado en la siguiente parte donde detalladamente se explicará la implementación aplicada y como el paradigma fue fundamental en la abstracción del problema; teniendo lista la solución se encuentra un instructivo de uso del programa además de los resultados y finalizando con una conclusión que representa un análisis total de la realización del objetivo y una comparación con el paradigma del laboratorio anterior.

Como principal objetivo de este trabajo se encuentra:

* Aplicar conceptos del paradigma de programación lógico usando el lenguaje de programación Prolog en la resolución de un problema acotado, el cual es la realización de un programa tipo GIMP simplificado.

# SOBRE EL PARADIGMA

# DESCRIPCION DEL PROBLEMA

Como se menciono anteriormente, el principal objetivo y problema a solucionar es realizar un programa de edición de imágenes, como lo son GIMP y Adobe Photoshop, pero con un enfoque simplificado, este programa debe realizar diversas operaciones en una imagen como rotar, invertir, recortar, comprimir, histograma y cambiar color, entre otras. Este proyecto se concentrará principalmente en el manejo de imágenes del tipo “RGB-D”, es decir, imágenes con colores y profundidad, las letras hacen referencia a “RED GREEN BLUE DEPTH”, siendo esta ultima la que contiene la información de la profundidad en un espacio tridimensional, con los datos de estos 4 elementos más otros dos que son el alto y ancho se puede conformar una imagen completamente. Se tienen tres tipos de imágenes a considerar, las cuales son:

* **Bitmap-d:** Imágenes en blanco y negro, cuya unidad principal es el pixbit compuesto por lo que puede ser un 1 que llevado a los colores es blanco o el 0 que corresponde al negro.
* **Hexmap-d:** Imágenes de colores pero con representación única de 6 valores hexadecimales, su unidad principal es el pixhex, un ejemplo de representación hexadecimal es “#FF00FF” que es fucsia.
* **Pixmap-d:** Al igual que el conjunto de pixeles anterior, el pixmap-d corresponde a una imagen con colores, pero con la diferencia de que estos colores se representan de la forma RGB o “Rojo”, “Verde” y “Azul”, en un pixbit, unidad principal de pixmap, un ejemplo de color escrito en RGB es el color fucsia (255 0 255).

Teniendo en consideración el ancho de la imagen más el alto y estos tres tipos de pixeles se puede trabajar completamente en la modificación de una imagen aplicando las diversas operaciones mencionadas.

# ANALISIS DEL PROBLEMA

Principalmente el problema trata de la manipulación de imágenes aplicando las mismas operaciones en todas sin importar el tipo de imagen que sea, por lo que las implementaciones deben ser universales o tener claramente diferenciadas las condiciones para aplicar la operación según el tipo de imagen, cabe recalcar que cuando se habla de “imagen” se habla de la información que contiene una, alto, ancho y los pixeles, es decir una representación simplificada de una imagen, por lo mismo todas las operaciones se realizaran directamente en la consola de Dr. Racket, usando solo la biblioteca interna y sin hacer uso de variables. Antes de pensar en las intervenciones a realizar en fotografía, hay que conocer como es la representación en la plataforma de Scheme de los principales aspectos de esta:

* **Imagen:** Lista compuesta de tres elementos principales, ancho y alto de la imagen, y una lista de pixeles, todo esto representado de la forma “(Width(int) Height(int) [pixbit-d|pixrgb-d|pixhex-d])”, esta construcción es posible solo si los pixeles están implementados:
  + **Pixbit-d:** Lista constituida de cuatro números enteros presentados como “(pixbit-d= x(int) y(int) bit([0|1]) depth(int))”, lo que demostrado en un ejemplo seria ‘(0 2 1 12), esto significa un pixel ubicado en la posición (0,2) de color blanco y de profundidad 12.
  + **Pixhex-d:** Lista formada por cuatro valores donde tres son enteros y el que esta ubicado en la tercera posición corresponde a un string, la representación dada en Scheme es la siguiente, “(pixhex-d=x(int) y(int) hex(String) depth(int))”, un ejemplo de esto sería, ‘(3 5 “#FFFF00” 34), es decir, un pixel ubicado en la posición (3,5) de color amarillo con profundidad 34.
  + **Pixrgb-d:** Construido en Scheme como una lista de seis elementos enteros, mostrado como “(pixrgb-d=x(int) y(int) R(C) G(C) B(C) depth(int))”, donde los elementos ubicados en la posición tres, cuatro y cinco corresponden a los colores rojo, verde y azul respectivamente y la “C” que los acompaña significa el intervalo numérico que puede tomar este valor, que va desde 0 a 255, como ejemplo se tiene ‘(3 6 255 0 255 65), un pixbit de posición (3,6) de color fucsia y profundidad 65.

Con esto claro, se puede proceder con las operaciones y la realización de los TDA que se deben realizar para cumplir a cabalidad con el laboratorio:

* Image: Constructor de la imagen, con ancho, alto y lista de pixeles.
* imageIsBitmap: Función que ve si una imagen es bitmap-d.
* imageIsHexmap: Función que ve si una imagen es hexmap-d.
* imageIsPixmap: Función que ve si una imagen es pixmap-d.
* imageIsCompressed: Función que ve si una imagen esta comprimida.
* imageFlipH: Permite invertir la imagen horizontalmente.
* imageFlipV: Permite invertir la imagen verticalmente.
* imageCrop: Corta la imagen en un cuadrante definido.
* imageRGBToHex: Transformar la imagen desde una Pixmap a una Hexmap.
* imageToHistogram: Analiza la cantidad de veces que se repite cada color en la imagen.
* imageRotate90: Rota la imagen 90° a la derecha.
* imageCompress: Comprime la imagen eliminando los pixeles que tienen el color más frecuente.
* imageChangePixel: Predicado que permite cambiar un pixel de la imagen a elección.
* imageInvertColorRGB: Cambia el color de la imagen RGB por el color simétricamente opuesto de este.
* imageToSstring: Transforma una imagen a una representación tipo string.
* imageDepthLayers: Permite la separación de capas de profundidad, dando de resultado una lista de imágenes de diversas profundidades cuyas imágenes se sustituyen los pixeles de otro nivel por color blanco.
* imageDecompress: Descomprime una imagen comprimida.

# DISEÑO DE LA SOLUCIÓN

## 5.1 CREACION DE LOS TDA.

Comenzando con el proceso de la solución lo primero que se debe realizar es la creación de los Tipos de Datos Abstractos utilizar para las operaciones solicitadas, cuya estructura es la siguiente, Representación, Constructores, Funciones de Pertenencia, Selectores, Modificadores y otras funciones asociadas al TDA.

### 5.1.1 TDA IMAGE

TDA fundamental del laboratorio, pues la mayoría de las operaciones son aplicadas en las imágenes de esta implementación. Su representación es una lista formada por ancho y alto y otra lista de pixeles, lo que reproduce una imagen; el constructor recibe ancho, alto y la lista con las funciones para crear pixeles, y así formar una lista de salida que contiene estos elementos para su futuro uso. Además cuenta con las otras estructuras del TDA.

### 5.1.2 TDA PIXBIT

El TDA PIXBIT representa la formación de pixeles en una lista que pueden tomar valores entre 1 y 0, además incluye la posición de cada pixel, representado con las letras "x" e "y" y la profundidad de estos, es la unidad fundamental de una imagen llamada bitmap-d, su constructor produce de una lista que recibe las posiciones x e y, un bit y la profundidad. Cuenta también con las otras estructuras de un TDA.

### 5.1.3 TDA PIXHEX

### 5.1.4 TDA PIXRGB

# ASPECTOS DE IMPLEMENTACION

# INSTRUCCIONES DE USO

# RESULTADOS OBTENIDOS

# CONCLUSIONES

# BIBLIOGRAFIA Y REFERENCIAS

# ANEXOS

## Tabla de funciones.

|  |  |  |
| --- | --- | --- |
| Predicado | Grado de alcance | Resultados |
| Image | 100% | Cada una de las pruebas resulto satisfactoria y se pudo entregar el resultado correcto. |
| imageIsBitmap | 100% | Cada una de las pruebas resulto satisfactoria y se pudo entregar el resultado correcto. |
| imageIsHexmap | 100% | Cada una de las pruebas resulto satisfactoria y se pudo entregar el resultado correcto. |
| imageIsPixmap | 100% | Cada una de las pruebas resulto satisfactoria y se pudo entregar el resultado correcto. |
| imageIsCompressed | 100% | Cada una de las pruebas resulto satisfactoria y se pudo entregar el resultado correcto. |
| imageFlipH | 100% | Cada una de las pruebas resulto satisfactoria y se pudo entregar el resultado correcto. |
| imageFlipV | 100% | Cada una de las pruebas resulto satisfactoria y se pudo entregar el resultado correcto. |
| imageCrop: | 100% | Cada una de las pruebas resulto satisfactoria y se pudo entregar el resultado correcto. |
| imageRGBToHex | 100% | Cada una de las pruebas resulto satisfactoria y se pudo entregar el resultado correcto. |
| imageToHistogram | 100% | Cada una de las pruebas resulto satisfactoria y se pudo entregar el resultado correcto. |
| imageRotate90 | 100% | Cada una de las pruebas resulto satisfactoria y se pudo entregar el resultado correcto. |
| imageCompress | 100% | Cada una de las pruebas resulto satisfactoria y se pudo entregar el resultado correcto. |
| imageChangePixel | 100% | Cada una de las pruebas resulto satisfactoria y se pudo entregar el resultado correcto. |
| imageInvertColorRGB | 100% | Cada una de las pruebas resulto satisfactoria y se pudo entregar el resultado correcto. |
| imageToSstring | 100% | Cada una de las pruebas resulto satisfactoria y se pudo entregar el resultado correcto. |
| imageDepthLayers | 95% | Los resultados están parcialmente correctos, ya que falta eliminar algunos pixeles repetidos. |
| imageDecompress | 100% | Cada una de las pruebas resulto satisfactoria y se pudo entregar el resultado correcto. |

## Manual de uso

Ver que los archivos estén todos en la misma carpeta. Abrir el archivo de pruebas y hacer clic en run, que esta en la esquina superior derecha.