**INFORME DE LABORATORIO 2: SIMULACIÓN DE UN PROGRAMA PARA REALIZAR UN TRATAMIENTO DE IMÁGENES SIMPLIFICADO**



Nombre: Aracely Castro V.

Profesor: Roberto Gonzales I.

Asignatura: Paradigmas de Programación (2/2022)

Índice

[1. INTRODUCCIÓN 3](#_Toc115040750)

[**1.1** **DESCRIPCIÓN DEL PROBLEMA** 3](#_Toc115040751)

[**1.2** **DESCRIPCIÓN DEL PARADIGMA** 3](#_Toc115040752)

[2. DESARROLLO 4](#_Toc115040753)

[**2.1** **ANALISIS DEL PROBLEMA** 4](#_Toc115040754)

[**2.2** **DISEÑO DE LA SOLUCIÓN** 5](#_Toc115040755)

[**2.3** **ASPECTOS DE IMPLEMENTACIÓN** 6](#_Toc115040756)

[**2.3.1 EJEMPLOS DE USO** 6](#_Toc115040757)

[**2.3.2 RESULTADOS ESPERADOS** 6](#_Toc115040758)

[**2.3.3 POSIBLES ERRORES** 6](#_Toc115040759)

[**2.4** **Resultados y autoevaluación** 6](#_Toc115040760)

[**2.4.1** **RESULTADOS** 6](#_Toc115040761)

[**2.4.2 AUTOEVALUACIÓN** 6](#_Toc115040762)

[3. CONCLUSIÓN 7](#_Toc115040763)

[4. BIBLIOGRAFÍA Y REFERENCIAS 7](#_Toc115040764)

[5. ANEXOS 7](#_Toc115040765)

# **INTRODUCCIÓN**

Con el objetivo de aplicar conceptos del paradigma de programación lógico usando el lenguaje de programación Prolog en la resolución de un tratamiento de imágenes simple, a continuación, se dará detalle sobre el proceso de desarrollo de la solución al problema a través del compilador SWI-Prolog versión 8.4.3. El informe constará de una breve introducción de cómo surgió el problema, una descripción del paradigma utilizado, el análisis del problema y como fue el diseño de las soluciones para algunas funciones, sus aspectos de implementación, las instrucciones necesarias para compilar el archivo con el script de pruebas junto con ejemplos de algunas funciones, los resultados y la autoevaluación. Finalmente, se dará una conclusión respecto a todo lo anterior.

## **DESCRIPCIÓN DEL PROBLEMA**

Al igual que en el laboratorio uno se busca desarrollar la simulación de un programa para el tratamiento de imágenes de forma simplificada como GIMP y Adobe Photoshop. Este simulador permitiría crear imágenes con distintos formatos de pixeles sobre los cuales se pueden aplicar distintas operaciones como, por ejemplo, recortar una imagen, invertir una imagen, obtener histograma de la imagen, entre otros. Para implementar esto, se deben tener en cuenta que:

Una imagen Pixmap-d es una imagen donde cada uno de sus pixeles contiene información de espacio de colores y profundidad (R)ed, (G)reen, (B)lue y (D)epth. Cada color cubre valores entre 0 y 255 formando en conjunto un color del espectro RGB. La profundidad en cambio ofrece información más detallada de la imagen como espacio tridimensional.

Una imagen Bitmaps-d es una imagen donde, en este caso, cada uno de sus pixeles tiene asociado solo un color, negro (1) o blanco (0) representando si hay tinta o no, cada píxel también tendría una profundidad asociada.

Una imagen Hexmap-d es similar a una imagen Pixmap-d con la diferencia que los valores RGB son representados en hexadecimal.

## **DESCRIPCIÓN DEL PARADIGMA**

El paradigma lógico forma parte de la familia de los paradigmas declarativos, su ventaja esta en facilitar pensar en la solución al problema en vez de los detalles procedurales que llevan a su solución. Prolog funciona declarando una base de conocimientos con hechos y reglas en donde se pueden hacer consultas sobre estas. El paradigma lógico cuenta con tres mecanismos básicos que son:

Unificación:

Backtracking automático:

Estructuras de datos basadas en árboles:

Además de contar con otros conceptos como son:

Átomo:

Predicado:

Clausuras:

# **DESARROLLO**

## **ANALISIS DEL PROBLEMA**

Se identifica que para hacer un tratamiento de imágenes simples en Prolog hay que considerar los siguientes elementos:

* **imagen:** Lista con ancho y largo de enteros y lista de pixeles.
* **pixrgb:** Lista con dos coordenadas x e y, colores RGB entero y profundidad.
* **pixrgb\_comprimido:** Lista con dos coordenadas x e y, colores RGB string y profundidad.
* **pixbit:** Lista con dos coordenadas x e y, bit y profundidad.
* **pixbit\_comprimido:** Lista con dos coordenadas x e y, una lista con -1 y bit, y la profundidad.
* **pixhex:** Lista con dos coordenadas x e y, un string y la profundidad
* **pixhex\_comprimido:** Lista con dos coordenadas x e y, color hexadecimal string y profundidad.

Además de las siguientes operaciones:

* **image** (Constructor): Crea una imagen bitmap, hexmap o pixmap.
* **imageIsBitmap, imageIsHexmap, imageIsPixmap, imageIsCompress** (Pertenencia): Verifica si la imagen es bitmap, hexmap, pixmap o si fue comprimida respectivamente.
* **imageFlipH, imageFlipV** (Modificador): invierte los pixeles de una imagen horizontal o verticalmente respectivamente.
* **imageCrop** (Modificador): Recorta una imagen a partir de un cuadrante definido por cuatro puntos.
* **imageRGBToHex** (Modificador): Convierte una imagen pixmap a hexmap.
* **imageToHistogram** (Otras funciones): Muestra un histograma de una imagen.
* **imageRotate90** (Modificador): Rota los pixeles de una imagen 90° a la derecha.
* **imageCompress** (Modificador): Comprime el color más frecuente de una imagen.
* **imageChangePixel** (Modificador): Reemplaza un píxel de una imagen por otro nuevo.
* **imageToString** (Otras funciones): Entrega una cadena string de la imagen.
* **imageDepthLayers** (Otras funciones): Entrega una lista de imágenes separados por profundidad.
* **imageToDecompress** (Modificador): Descomprime una imagen comprimida.

Se pide que cada TDA (Tipo de Dato Abstracto) se implemente con lo necesario para cumplir con los requerimientos funcionales y que en el archivo principal se contenga las funciones con un script de pruebas para probar los predicados.

## **DISEÑO DE LA SOLUCIÓN**

ImageToString

imageCompress

imageRotate90

## **ASPECTOS DE IMPLEMENTACIÓN**

El compilador utilizado fue Prolog versión 8.4.3, no se utilizaron bibliotecas externas de ningún tipo, esto para pensar las soluciones a los requerimientos funcionales usando solamente el paradigma lógico y los predicados básicos de Prolog. Mientras se desarrollaban los requerimientos funcionales, se crearon los siguientes TDAs cuya estructura respecto el orden de representación, constructor, pertenencia, selectores, modificadores y otras funciones:

* **TDA image:** Corresponde a una imagen. Representación: (int x int x list). Ver Tabla N°1 en ANEXOS para ver su implementación, P.XX.
* **TDA pixbit:** Corresponde a un píxel bitmap. Representación: (int x int x bit ([0|1]) x int). Ver Tabla N°2 en ANEXOS para ver su implementación, P.XX.
* **TDA pixbit\_comprimido:** Corresponde a un píxel bitmap comprimido. Representación: (int X int X (-1, bit([0|1])) x int). Ver tabla N°3 en ANEXOS para ver su implementación, P.XX.
* **TDA pixhex:** Corresponde a un píxel hexmap. Representación: (int x int x string x int). Ver Tabla N°4 en ANEXOS para ver su implementación, P.XX.
* **TDA pixhex\_comprimido:** Corresponde a un píxel hexmap comprimido. Representación: (int x int x (int X int X int) x int). Ver Tabla N°5 en ANEXOS para ver su implementación, P.XX.
* **TDA pixrgb:** Corresponde a un píxel pixmap. Representación: (int x int x int x int x int x int). Ver Tabla N°6 en ANEXOS para ver su implementación, P.XX.
* **TDA pixrgb\_comprimido:** Corresponde a un píxel pixmap. Representación: (int x int x string x string x string x int). Ver Tabla N°7 en ANEXOS para ver su implementación, P.XX.

Ver la Figura N°8 en ANEXOS para poder ver en que archivos se importan otros archivos. P.XX.

### **EJEMPLOS DE USO**

Para comenzar, se debe verificar que se tengan todos los archivos TDAs en una misma carpeta, de lo contrario el archivo pruebas\_21090869\_CastroVenegas no se ejecutará al compilar el programa con “Run” tras compilar. Luego de ejecutado se podrán ver los ejemplos de cada una de las operaciones en el archivo como script de pruebas. Una observación importante es que si se desea crear una nueva imagen para probar las funciones, esta no debe tener el mismo nombre que una de las imágenes ya definidas y que las dimensiones y el número de píxeles en la imagen deben de ser correctos, todo esto para que no surja un error al compilar o al aplicar un predicado a la imagen. También asegurarse que los argumentos de los predicados estén bien ingresados. Por ejemplo, se podrá aplicar la función imageInvertColorRGB a un píxel pixmap, pero si se desea aplicar a una imagen puede modificar un píxel de esta utilizando el predicado imageChangePixel, pero para ello necesita ingresar como entrada la imagen original, el pixel a modificar y una variable en donde colocar la imagen resultante (en ese orden).

Ver Figura X en ANEXOS para ver ejemplos claros de ejecución de algunas funciones. P.XX

### **RESULTADOS ESPERADOS**

Se espera que se haya hecho un simulador de tratamiento de imágenes simple donde cada predicado no cause errores y compile el archivo con el script de pruebas correctamente.

### **2.3.3 POSIBLES ERRORES**

## **Resultados y autoevaluación**

### **RESULTADOS**

### **2.4.2 AUTOEVALUACIÓN**

La Autoevaluación se realiza de la siguiente forma: 0: No realizado – 0.25: Funciona 25% de las veces – 0.5: Funciona 50% de las veces 0.75: Funciona 75% de las veces – 1: Funciona 100% de las veces. Para ver la tabla de Autoevaluación, ver la Tabla XX del anexo XX

Tras mejorar la implementación de los requerimientos funcionales no se encontró errores, por lo que se considera de que funcionan el 100% de las veces

# **CONCLUSIÓN**

Tras haber trabajado en implementar los requerimientos funcionales para un simulador de tratamiento de imágenes simple, ser puede decir que se cumplió el objetivo de aplicar conceptos del paradigma lógico usando el lenguaje de programación Prolog. Las complicaciones que hubo durante el desarrollo de las funciones fueron el tener que mover las funciones hechas en SWIP-Prolog de internet a SWIP-Prolog compilador

# **BIBLIOGRAFÍA Y REFERENCIAS**

# **ANEXOS**