**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](#_Toc118231351)

[**1.1** **DESCRIPCIÓN DEL PROBLEMA** 3](#_Toc118231352)

[**1.2** **DESCRIPCIÓN DEL PARADIGMA** 3](#_Toc118231353)

[2. DESARROLLO 4](#_Toc118231354)

[**2.1** **ANALISIS DEL PROBLEMA** 4](#_Toc118231355)

[**2.2** **DISEÑO DE LA SOLUCIÓN** 5](#_Toc118231356)

[**2.3** **ASPECTOS DE IMPLEMENTACIÓN** 6](#_Toc118231357)

[**2.3.1** **EJEMPLOS DE USO** 6](#_Toc118231358)

[**2.3.2** **RESULTADOS ESPERADOS** 7](#_Toc118231359)

[**2.3.3 POSIBLES ERRORES** 7](#_Toc118231360)

[**2.4** **Resultados y autoevaluación** 7](#_Toc118231361)

[**2.4.1** **RESULTADOS** 7](#_Toc118231362)

[**2.4.2 AUTOEVALUACIÓN** 7](#_Toc118231363)

[3. CONCLUSIÓN 7](#_Toc118231364)

[4. BIBLIOGRAFÍA Y REFERENCIAS 8](#_Toc118231365)

[5. ANEXOS 8](#_Toc118231366)

# **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 sus pixeles tiene asociado su color por un bit, negro como 0 y blanco como 1. 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, 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: Proceso que consiste en encontrar una asignación de variables de tal modo que los términos sean equivalentes.
* Backtracking automático: Estrategia de búsqueda de soluciones con vuelta atrás, utilizada para encontrar soluciones a una consulta.
* Estructuras de datos basadas en árboles: Los tipos de datos compuestos internamente se implementan como árboles, un ejemplo son las listas cuya estructura es un árbol binario.

Además de contar con otros conceptos como son:

* **Átomo:** Son aquellas cosas sobre las que basa el conocimiento que queremos expresar. Se escriben en minúsculas o entre “” si se desea que su primera palabra sea mayúscula o contenga espacio.
* **Predicado:** Son las cosas que queremos decir. Es un conjunto de hechos o reglas que tienen un mismo nombre y la misma cantidad de términos relacionados. Los resultados o variables van en mayúsculas.
* **Hechos:** Tipo de clausura que definen una relación entre uno o más términos, el nombre de la relación empieza en minúscula y siempre son verdad.
* **Regla:** Tipo de clausura que define una relación entre uno o más términos. Su veracidad depende de la conjunción de objetivos que tenga en su cuerpo mediante el operador lógico AND.
* **Consultas:** Es una pregunta que coincide con un hecho o regla de la base de conocimientos (en nombre y aridad). Busca automáticamente si existe un hecho o regla que unifique la consulta realizada respondiendo true o false. Cabe señalar que responder false indica que no existe un hecho o regla que satisfaga la consulta en la base de conocimientos.

Al momento de desarrollar los predicados hay que evitar el problema de la negación del mundo cerrado que se puede generar al negar un hecho o predicado, ya que si no se utiliza de buena manera la negación puede ocurrir que la consulta responda true al ingresar un termino que no se haya contemplado en la base de conocimientos.

# **DESARROLLO**

## **ANALISIS DEL PROBLEMA**

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

* **image:** Una imagen como una lista con ancho, largo (int) y lista de pixeles.
* **pixrgb:** Un píxel pixmap-d como una lista con dos coordenadas x e y, colores RGB y profundidad (int).
* **pixrgb\_comprimido:** Un píxel comprimido pixmap-d como una lista con dos coordenadas x e y (int), colores RGB (string) y profundidad (int).
* **pixbit:** Un píxel bitmap-d como una lista con dos coordenadas x e y, bit y profundidad (int).
* **pixbit\_comprimido:** Un píxel comprimido bitmap-d como una lista con dos coordenadas x e y (int), una lista con -1 y bit (int), y la profundidad (int).
* **pixhex:** Un píxel hexmap-d como una lista con dos coordenadas x e y (int), color hexadecimal (string) y la profundidad (int)
* **pixhex\_comprimido:** Un píxel comprimido hexmap-d como una lista con dos coordenadas x e y (int), color hexadecimal (list int) y profundidad (int).

Además de las siguientes operaciones pertenecientes al TDA image:

* **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 y lo convierte en una nueva imagen.
* **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) desarrollado 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**

Por otro lado, se distinguen algunos casos particulares para soluciones de algunos requerimientos funcionales utilizando compilador SWI-Prolog como lo son:

**Predicado imageToHistogram**: Para poder crear el histograma se desarrollaron predicados auxiliares para lograr hacer los pasos de extraer color, contar color y eliminar el color de la lista de pixeles de la imagen. Con base a lo anterior se fue formando la lista con la cantidad y el color, parando el ciclo cuando la lista de pixeles este vacía. Ver la Figura N°1, Figura N°2 y Figura N°3 en ANEXOS para poder ver la salida de histogram con imágenes Pixmap-d, Bitmap-d y Hexmap-d. P.8

**Predicado imageCompress:** Para poder comprimir se pensó en una forma de cambiar la información del píxel más repetido con otro dato distinto de modo que se pueda descomprimir con la función decompress más tarde. Para el caso de una imagen Bitmap-d se eligió reemplazar el valor del bit más repetido por una lista con -1 y su bit, así para volver a la forma original bastó con comprobar la lista y reemplazar a la forma original en consecuencia. En el caso de una imagen Hexmap-d se eligió reemplazar el píxel repetido por una lista con tres números, cada número representando en valor color rojo, verde y azul del string, con esto para volver a la imagen original se podría recuperar la lista y con base a ella crear el string original para reemplazarlo donde corresponde. Finalmente, para comprimir una imagen Pixmap-d se optó por reemplazar el valor entero del espectro RGB por su equivalente a string hexadecimal y así, para volver a la forma original, bastaría con leer el string de cada color y transformarlo a número antes de colocarlo donde se necesite. Ver la Figura N°4, Figura N°5 y Figura N°6 en ANEXOS para poder ver la salida de compress con imágenes Pixmap-d, Bitmap-d y Hexmap-d. P.9

**Predicado imageFlipH:** Para implementar flipH se creó una predicado flipH\_formato que modifica la coordenada <y> de cada píxel de modo que parezca que la imagen se invirtió horizontalmente utilizando recursión natural. Las nuevas coordenadas son ordenadas utilizando sort() antes de ser ingresado en la imagen. Cabe señalar que en imageFlipV y imageRotate90 se implementó un enfoque de solución similar con variaciones en los cambios de coordenadas (x, y). Ver la Figura N°7 en ANEXOS para poder ver la salida de flipH\_formato en una imagen 3x4. P.10

## **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 otros predicados:

* **TDA image:** Corresponde a una imagen. Representación: (int x int x list). Ver Tabla N°1 en ANEXOS para ver su implementación, P.12.
* **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.13.
* **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.13.
* **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.14.
* **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.14.
* **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.14.
* **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.14.

Ver la Figura N°8 en ANEXOS para poder ver en que archivos se importan otros archivos. P.10.

### **EJEMPLOS DE USO**

Para comenzar, se debe verificar que se tengan todos los archivos TDAs y el archivo pruebas\_21090869\_CastroVenegas en una misma carpeta, con ello se podrán probar las consultas luego de compilar el archivo de pruebas con los ejemplos de cada una de las operaciones como script de pruebas. Se recomienda antes de consultar utilizar el comando “set\_prolog\_flag(answer\_write\_options,[max\_depth(0)]).” Para poder ver todos los pixeles de la imagen y no su forma medio visible o en defecto presionar la letra W luego de hacer la consulta. Por otra parte, hay que asegurarse que los argumentos de los predicados estén bien ingresados. Por ejemplo, se podrá aplicar imageInvertColorRGB a un píxel pixmap-d, pero si se desea aplicar a una imagen se podría modificar un píxel de está utilizando el predicado imageChangePixel, pero para ello necesita ingresar como entrada la imagen original, el píxel a modificar y una variable en donde colocar la imagen resultante (en ese orden).

Ver Figura 9 en ANEXOS para ver ejemplos claros de ejecución de algunos predicados. P.11

### **RESULTADOS ESPERADOS**

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

### **2.3.3 POSIBLES ERRORES**

Si los argumentos en la consulta no se ingresan incorrectamente, no debería de haber problemas en obtener los resultados esperados para cada requerimiento funcional solicitado.

## **Resultados y autoevaluación**

### **RESULTADOS**

Tras consultar en consola, se confirmo que los predicados funcionan correctamente y se obtienen los resultados esperados. Se logró crear cada uno de los predicados obligatorios y opcionales verificando distintos casos.

### **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 SWISH (Prolog online) SWIP-Prolog en un comienzo, también resulto desafiante lograr filtrar elementos de una lista correctamente sin crear variables anónimas y lograr que un predicado unificará como se esperaba. Por otra parte, no hubo complicaciones en cuanto fue el uso de GitHup y Prolog a la hora de desarrollar el trabajo, señalando un logró en comprender conceptos como hechos, reglas y recursividad, etc. Finalmente, en comparación con el paradigma funcional desarrollado en el laboratorio uno, se observaron menos líneas de código y menos problemas a la hora de pensar como desarrollar una idea. A diferencia de Scheme en Prolog podrías utilizar un predicado de más de una manera y no solo una, ejemplo de esto sería la posibilidad de utilizar image como constructor, selector y modificador a la vez, algo que no se hizo finalmente porque se optó por separarlos, también como se podrían utilizar variables fue más fácil a la hora de recuperar datos para un predicado. Finalmente, se puede decir que la experiencia de este segundo laboratorio junto con el primero pueda servir de apoyo para próximo.

# **BIBLIOGRAFÍA Y REFERENCIAS**

Flores, V. (2022). “Proyecto Semestral de Laboratorio”. Paradigmas de Programación. Enunciado de Proyecto Online. Recuperado de: <https://docs.google.com/document/d/14XEMPsuHicTIK7Hv3OuvW3wlG-SE1Ms9sJq9xwVLJSc/edit>

Flores, V. (2022). “4 - P. Lógico”. Paradigmas de Programación. Material de clases Online. Recuperado de: <https://uvirtual.usach.cl/moodle/course/view.php?id=10036&section=16>

Autores Desconocidos. (2022). “Swi-Prolog Documentation’’. Swi-Prolog. Documentación Online. Recuperado de: <https://www.swi-prolog.org/pldoc/index.html>

Davis, C. Esteban, J (2022). “Tutorial de Prolog”. SWISH. Documentación Online. Recuperado de: <https://swish.swi-prolog.org/p/Tutorial%20de%20prolog.swinb>

# **ANEXOS**

**Texto, Carta

Descripción generada automáticamenteFigura N°1: Consulta imageToHistogram sobre una imagen 2x3 Pixmap-d**

**Figura N°2: Consulta imageToHistogram sobre una imagen 2x2 Hexmap-d**

Texto, Carta

Descripción generada automáticamente

**Figura N°3: Consulta imageToHistogram sobre una imagen 3x3 Bitmap-d**

Texto, Carta

Descripción generada automáticamente

**Texto, Carta

Descripción generada automáticamenteFigura N°4: Consulta imageCompress a una imagen 3x3 Pixmap-d**

**Figura N°5: Consulta imageCompress a una imagen 2x2 Hexmap-d**

Texto, Carta

Descripción generada automáticamente

**Figura N°6: Consulta imageCompress a una imagen 2x2 Bitmap-d**

Texto, Carta

Descripción generada automáticamente

**Figura N°7: Como cambia el formato de una imagen 3x4 con flipH\_formato**

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**Figura N°8: Diagrama de importación de TDAs**

Diagrama

Descripción generada automáticamente

**Figura 9: Ejemplos de uso de predicados**

Nota: En el script de pruebas se encontrarán ejemplos de consultas. Las siguientes figuras muestran algunos de estos en la consola.

Crear una imagen con image. rotar una imagen con imageRotate90 y obtener el string de una imagen con imageToString en una misma consulta.

Texto, Carta

Descripción generada automáticamente

Crear una imagen con image , modificar el color de un pixrgb por su color opuesto con imageInvertColorRGB y cambiar un píxel de la imagen con imageChangePixel en una misma consulta.

Texto

Descripción generada automáticamente

Crear una imagen con image y convertir una imagen Pixmap-d a Hexmap-d con imageRGBToHex en una misma consulta.

Texto

Descripción generada automáticamente

**Tabla N°1: Predicador del TDA Image**

|  |  |  |
| --- | --- | --- |
| **Tipo** | **Nombre** | **Descripción** |
| Constructor y modificador | image | Crea una image pixmap-d, hexmap-d o bitmap-d |
| Pertenencia | imageIsBitmap | Verifica si la imagen es Bitmap-d |
| Pertenencia | imageIsHexmap | Verifica si la imagen es Hexmap-d |
| Pertenencia | imageIsPixmap | Verifica si la imagen es Pixmap-d |
| Pertenencia | imageIsCompress | Verifica si la imagen se comprimió o no |
| Pertenencia | esImage | Verifica si es una imagen pixmap-d, hexmap-d o bitmap-d |
| Selector | obtPixelesImage | Obtiene la lista de pixeles de una imagen |
| Selector | obtCoordImage | Obtiene el ancho (x) y el largo (y) de una imagen |
| Otros predicados | cambiarCoordXY | Modifica las coordenadas (x,y) de un píxel |
| Otros predicados | flipH\_formato | Modifica los pixeles para que se inviertan horizontalmente |
| Modificador | imageFlipH | Modifica la imagen para invertirla horizontalmente |
| Otros predicados | flipV\_formato | Modifica los pixeles para que se inviertan verticalmente |
| Modificador | imageFlipV | Modifica la imagen para invertirla verticalmente |
| Otros predicados | rangoXY | Verifica si un píxel esta dentro del rango definido en imageCrop |
| Otros predicados | crop\_filtro | Modifica los pixeles para eliminar aquellos que no están dentro del rango definido en imageCrop en eliminarElemento |
| Otros predicados | crop\_formato | Modifica las coordenadasde los pixeles restantes de crop\_filtro para la nueva imagen |
| Otros predicados | mayor | Obtiene el mayor entre dos números |
| Otros predicados | menor | Obtiene el menor entre dos números |
| Modificador | imageCrop | Recorta la imagen dentro de un rango establecido creando otra imagen |
| Otros predicados | numeroString | Entrega un string hexadecimal equivalente al número |
| Otros predicados | rgbHex | Transforma un número con unidad y decena a string |
| Otros predicados | listRGB\_HEX | Convierte una lista de RGB a un string hexadecimal |
| Otros predicados | stringRGB | Convierte un pixrgb-d a pixhex-d |
| Otros predicados | formatoRGB\_Hex | Modifica los pixeles pixrgb-d a pixhex-d |
| Modificador | imageRGBToHex | Convierte una imagen Pixmap-d a Hexmap-d |
| Otros predicados | imageInvertColorRGB | Modifica el color de un pixrgb-d por su color opuesto |
| Otros predicados | igualColor | Verifica si dos píxeles tienen mismo color RGB, bit o igual string |
| Otros predicados | eliminarElemento | Elimina los pixeles que cumplen la condición para ser eliminados |
| Otros predicados | listaEliminarColor | Modifica los pixeles para eliminar aquellos que tengan el color definido en eliminarElemento |
| Otros predicados | contarColor | Cuenta las veces que un color aparece en los pixeles de la imagen |
| Otros predicados | colorPixel | Obtiene el color de un píxel |
| Otros predicados | histograma\_formato | Crea el histograma con base en los pixeles de la imagen |
| Otros predicados | imageToHistogram | Obtiene el histograma de una imagen |
| Otros predicados | rotate90\_formato | Modifica los pixeles para rotar 90° a la derecha |
| Modificador | imageRotate90 | Modifica una imagen para rotar 90° a la derecha |
| Otros predicados | stringNumero | Entrega un numero equivalente al string hexadecimal |
| Otros predicados | hexRGB | Convierte un string hexadecimal de dos letras a número |
| Otros predicados | listHEX\_RGB | Convierte un string hexadecimal en una lista con valores RGB |
| Otros predicados | comprimirPixel | Comprime un píxel |
| Otros predicados | comprimirPixeles | Comprime la lista de pixeles según corresponda |
| Otros predicados | pixelMasRepetido | Obtiene el píxel más repetido del histograma |
| Modificador | imageCompress | Comprime el píxel más repetido de una imagen |
| Otros predicados | igualCoordXY | Verifica si dos pixeles tiene misma coordenada (x,y) |
| Otros predicados | changePixel | Reemplaza el nuevo píxel de los pixeles |
| Modificador | imageChangePixel | Reemplaza un píxel de la imagen |
| Otros predicados | imageString\_formato | Crea la cadena de string de la imagen |
| Otros predicados | imageToString | Entrega el string de la imagen |
| Otros predicados | obtenerProfundidad | Obtiene la profundidad de un píxel |
| Otros predicados | listaEliminarProfundidad | Modifica los pixeles para eliminar los que son iguales a la profundidad definida en eliminarElemento |
| Otros predicados | listaProfundidad | Obtiene todas las profundidades de los pixeles |
| Otros predicados | reemplazarPixelBlanco | Modifica el píxel para que tenga color blanco y tenga la misma profundidad |
| Otros predicados | pixelesIgualProfundidad | Crea nuevos pixeles, lo que tengan misma profundidad mantiene color y los que no cambian en reemplazarPixelBlanco |
| Otros predicados | profundidad\_formato | Crea la lista de imágenes separados por profundidad |
| Otros predicados | imageDepthLayers | Obtiene la lista de imágenes separados por profundidad |
| Otros predicados | pixelDescomprimido | Descomprime un píxel comprimido |
| Otros predicados | esPixel | Verifica si es un píxel pixrgb-d, pixhex-d o pixbit-d |
| Otros predicados | pixelesDescomprimidos | Descomprime todos los pixeles comprimidos según corresponda |
| Modificador | imageDecompress | Descomprime una imagen comprimida |

**Tabla N°2: Predicados del TDA Pixbit**

|  |  |  |
| --- | --- | --- |
| **Tipo** | **Nombre** | **Descripción** |
| Constructor y modificador | pixbit | Crea un píxel pixbit-d |
| Pertenencia | esBitmap | Verifica si la lista de pixeles es pixbit-d |
| Selector | obtColorPixbit | Obtiene el color bit de un pixbit-d |
| Selector | obtCoordPixbit | Obtiene las coordenadas (x,y) de un pixbit-d |
| Selector | obtProfundidadPixbit | Obtiene la profundidad de un pixbit-d |

**Tabla N°3: Predicados del TDA Pixbit\_comprimido**

|  |  |  |
| --- | --- | --- |
| **Tipo** | **Nombre** | **Descripción** |
| Constructor y modificador | pixbit\_comprimido | Crea un píxel pixbit-d comprimido |
| Pertenencia | esBitmapComprimido | Verifica si la lista de pixeles es pixbit-d comprimido |
| Selector | obtColorPixbitC | Obtiene lista con color de un pixbit-d comprimido |
| Selector | obtCoordPixbitC | Obtiene las coordenadas (x,y) de un pixbit-d comprimido |
| Selector | obtProfundidadPixbitC | Obtiene la profundidad de un pixbit-d comprimido |

**Tabla N°4: Predicados del TDA Pixhex**

|  |  |  |
| --- | --- | --- |
| **Tipo** | **Nombre** | **Descripción** |
| Constructor y modificador | pixhex | Crea un píxel pixhex-d |
| Pertenencia | esHexmap | Verifica si la lista de pixeles es pixhex-d |
| Selector | obtColorPixhex | Obtiene el color string de un pixhex-d |
| Selector | obtCoordPixhex | Obtiene las coordenadas (x,y) de un pixhex-d |
| Selector | obtProfundidadPixhex | Obtiene la profundidad de un pixhex-d |

**Tabla N°5: Predicados del TDA Pixhex\_comprimido**

|  |  |  |
| --- | --- | --- |
| **Tipo** | **Nombre** | **Descripción** |
| Constructor y modificador | pixhex\_comprimido | Crea un píxel pixhex-d comprimido |
| Pertenencia | esHexmapComprimido | Verifica si la lista de pixeles es pixhex-d comprimido |
| Selector | obtColorPixhexC | Obtiene lista con color de un pixhex-d comprimido |
| Selector | obtCoordPixhexC | Obtiene las coordenadas (x,y) de un pixhex-d comprimido |
| Selector | obtProfundidadPixhexC | Obtiene la profundidad de un pixhex-d comprimido |

**Tabla N°6: Predicados del TDA Pixrgb**

|  |  |  |
| --- | --- | --- |
| **Tipo** | **Nombre** | **Descripción** |
| Constructor y modificador | pixrgb | Crea un píxel pixrgb-d |
| Pertenencia | esPixmap | Verifica si la lista de pixeles es pixrgb-d |
| Selector | obtColorPixrgb | Obtiene el color RGB de un pixrgb-d |
| Selector | obtCoordPixrgb | Obtiene las coordenadas (x,y) de un pixrgb-d |
| Selector | obtProfundidadPixrgb | Obtiene la profundidad de un pixrgb-d |
| Otros predicados | colorProfundidadPixrgb | Crea un string con el color RGB y la profundidad de un pixrgb-d |

**Tabla N°7: Predicados del TDA Pixrgb\_comprimido**

|  |  |  |
| --- | --- | --- |
| **Tipo** | **Nombre** | **Descripción** |
| Constructor y modificador | Pixrgb\_comprimido | Crea un píxel pixrgb-d comprimido |
| Pertenencia | esPixmapComprimido | Verifica si la lista de pixeles tiene un pixrgb-d comprimido |
| Selector | ObtColorPixrgbC | Obtiene el color RGB de un pixrgb-d comprimido |
| Selector | ObtCoordPixrgbC | Obtiene las coordenadas (x,y) de un pixrgb-d comprimido |
| Selector | obtProfundidadPixrgbC | Obtiene la profundidad de un pixrgb-d comprimido |