

Universidad de Santiago de Chile
Labor Lætitia nostra



Informe para el ramo Paradigmas de programación

Laboratorio 2: Paradigma Lógico
Editor de imágenes usando Prolog

Sergio Osvaldo Andres Espinoza Gonzalez
Facultad de ingeniería

Noviembre de 2022

Índice

1. Introducción	2
1.1. Descripción del problema	2
1.2. Descripción del paradigma	2
1.2.1. Paradigma de programación lógica	2
2. Análisis del problema	2
3. Diseño de la solución	3
3.1. TDAs	3
3.1.1. Definición	3
3.1.2. Especificación	3
3.1.3. Implementación	4
4. Consideraciones de implementación	4
5. Instrucciones de uso	4
6. Resultados obtenidos	4
7. Conclusión	4
Referencias	5

Índice de cuadros

1. Especificación de TDAs	6
2. Ejemplos de uso	6
3. Resultados obtenidos	7

1. Introducción

En este informe se abordará el laboratorio número 2 de la asignatura Paradigmas de programación con el formato expuesto en el índice 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 Prolog el cual se encuentra enfocado en el paradigma de programación lógica, ya teniendo en cuenta esto podría surgirnos la pregunta "¿Que es el paradigma lógico?"

1.2.1. Paradigma de programación lógica

El paradigma lógico es aquel donde se hace uso de la lógica que subyace a un problema para su posterior resolución, esto a través de un conjunto de fórmulas lógicas que suelen verse, al llevar esto a la programación, como predicados o cláusulas de Horn (Taltavull y Antoni, 2015). Lo anterior convierte la resolución de un problema en la búsqueda de la lógica detrás del mismo, y no en una lista de pasos a seguir como acostumbramos en otros paradigmas; esto a través de una base de conocimientos con hechos y reglas, haciendo uso también de la unificación.

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 desafío 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 que estará en la sección 3.
2. Constructor 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 hexmap 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 específico a abordar, es dar la posibilidad de invertir la imagen ya sea en el eje X o Y, además de poder rotar esta en 90° hacia la derecha.

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 de imágenes RGB.
6. Imagen a string: Debe también cubrirse la posibilidad de que la imagen pueda ser representada como un conjunto de caracteres.
7. 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 que 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 imagen de 3 dimensiones con 2 posibles colores, blanco o negro.
- TDA pixrgb-d: Será aquel pixel con una posición específica en una imagen de 3 dimensiones con color de tipo rgb.
- TDA pixhex-d: Será aquel pixel con una posición específica en una imagen de 3 dimensiones con color de tipo hexadecimal.
- TDA histogram: Será el gráfico de cuantos pixeles de cada color hay contenidos en una imagen

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 especifica 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 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 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.

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 una lista de 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 listas de 2 elementos, en estas listas de 2 la cabeza será un color y el segundo elemento 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 la base de conocimientos que nos permitirá cumplir los requisitos principales que se exigen en el 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 interpretador usado será "swipl".

5. Instrucciones de uso

6. Resultados obtenidos

Los resultados de la implementación y sus requisitos específicos 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 consultas a la base de conocimientos en el archivo de pruebas que se subirá a la plataforma github y uvirtual.

7. Conclusión

Referencias

Taltavull, C., y Antoni, J. (2015). Programación lógica. , 10–12. Descargado de <http://hdl.handle.net/2445/64643>

Anexos

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

Cuadro 1: Especificación de TDAs

Función	Uso	Resultado esperado
2. image	0	0
3. bitmap?	0	0
4. pixmap?	0	0
5. hexmap?	0	0
6. compressed?	0	0
7. flipH	0	0
8. flipV	0	0
9. crop	0	0
10. imgRGB->imgHex	0	0
11. histogram	0	0
12. rotate90	0	0
13. compress	0	0
14. changePixel	0	0
15. invertColorRGB	0	0
16. image->string	0	0
17. depthLayers	0	0
18. decompress	0	0

Cuadro 2: Ejemplos de uso

Requisito	Completado?	Puntaje autoevaluación	Posible razón fallos
1. TDAs	No	0	N/A
2. image	No	0	N/A
3. bitmap?	No	0	N/A
4. pixmap?	No	0	N/A
5. hexmap?	No	0	N/A
6. compressed?	No	0	N/A
7. flipH	No	0	N/A
8. flipV	No	0	N/A
9. crop	No	0	N/A
10. imgRGB->imgHex	No	0	N/A
11. histogram	No	0	N/A
12. rotate90	No	0	N/A
13. compress	No	0	N/A
14. changePixel	No	0	N/A
15. invertColorRGB	No	0	N/A
16. image->string	No	0	N/A
17. depthLayers	No	0	N/A
18. decompress	No	0	N/A

Cuadro 3: Resultados obtenidos