PROYECTO

FORMATO DE IMAGENES

Estructuras de datos, primera entrega del proyecto formato de imágenes.

Realizado por

German Velasco

Andrés Felipe Rodríguez

Eduard Cadena

Fabián Zapata Bonivento

Abril 2022

Índice

1. Diseño de TADS

A continuación, se presentan los diferentes TADS, especificados mediante una plantilla de nombre, datos y operaciones del objeto (tipo de dato abstracto) a representar. Posteriormente se realiza el diagrama de relación que permite visualizar de manera sencilla lo descrito en la primera parte, así como las relaciones entre los distintos datos abstractos.

1.1. Especificación de los TADS

1.1.1. TAD Imagen 2D Conjunto

mínimo de datos.

- Ancho de la imagen, numero entero, corresponde al ancho en pixeles de la imagen 2D.
- Largo de la imagen, numero entero, corresponde al largo en pixeles de la imagen 2D.
- Un vector de vectores de tipo Píxel. *Nota: es indispensable en el presente diseño interpretar las imágenes como un vector de vectores y no como una matriz, ya que la complejidad aumentaría.*

Comportamiento u Operaciones del objeto.

- *Imagenes_2D(),* constructor vacío para inicializar el objeto Imagen en memoria.
- *Imagenes_2D(int ancho, int largo),* constructor que inicializa el objeto con un ancho y largo que se recibe por parámetro.
- *setAncho(int ancho),* reemplaza el ancho actual por el que se recibe como parámetro.
- *setLargo(int largo),* reemplaza el largo actual por el que se recibe como parámetro.
- *getLargo(),* retorna el largo de la imagen en una variable entera.
- *getAncho(),* retorna el ancho de la imagen en una variable entera.
- *// info_Imagen();//*

- *leerArchivo3D(string nombre),* carga una imagen 3D en memoria y la retorna.
- *leerArchivo2D(string nombre),* carga una imagen 2D en memoria y la retorna.
- *frecuencias(),* la función se utiliza para organizar dentro del árbol de Hauffman los símbolos, es decir entre mayor sea la frecuencia del símbolo menor será la cantidad de caracteres que tendrá dentro de este árbol. ***Tipo de la función: std::vector<std::pair<int,int>>***
- *codificar(string nombreArchivo),* la función se utiliza para comprimir un archivo siguiendo la metodología de Hauffman
- *decodificar(),* la función se utiliza para descomprimir un archivo siguiendo la metodología de Hauffman

1.1.2. TAD Imagen 3D. Conjunto

mínimo de datos.

- Ancho de la imagen, numero entero, corresponde al ancho en pixeles de la imagen 3D.
- Largo de la imagen, numero entero, corresponde al largo en pixeles de la imagen 3D.
- Un vector de vectores de tipo Imágenes 2D, indispensable para poder realizar las proyecciones enunciadas.

Comportamiento u Operaciones del objeto.

- *Imagenes_3D(),* constructor vacío para inicializar el objeto Imagen en memoria.
- *setCantidadImagenes(int cantidadImagenes),* reemplaza la cantidad de imágenes por la que se recibe como parámetro.
- *getCantidadImagenes(),* retorna la cantidad de imágenes actual.
- *// info_Imagen3d();//*
- *leerArchivo3D(string nombre),* carga una imagen 3D en memoria y la retorna.
- *proyeccion2D(string dirección, string criterio, string nombre),* realiza la proyección 2d de un conjunto de imágenes 3d, se proyectan tanto en el eje x, y o z.

1.1.3. TAD Píxel

Conjunto mínimo de datos.

- Componente RED, numero entero, que representa el componente rojo en la escala RGB. Se representa de la siguiente forma (255,0,0).
- Componente GREEN, numero entero, que representa el componente verde en la escala RGB. Se representa de la siguiente forma (0,255,0).
- Componente BLUE, numero entero, que representa el componente azul en la escala RGB. Se representa de la siguiente forma (0,0,255).
- Componente GRIS, representa los valores numéricos de la escala de grises de la imagen.

Comportamiento u Operaciones del objeto.

- *setR(int r);* reemplaza el componente RED(0-255) por el que se recibe como parámetro.
- *setG(int g),* reemplaza el componente GREEN(0-255) por el que se recibe como parámetro.
- *setB(int b),* reemplaza el componente BLUE(0-255) por el que se recibe como parámetro.
- *setGris(int gris),* reemplaza el componente GRIS(0-255) por el que se recibe como parámetro.
- *getR(), retorna el componente RED actual.*
- *getG(), retorna el componente GREEN actual.*
- *getB(), retorna el componente BLUE actual.*
- *getGris(), retorna el componente GRIS actual.*

1.2. Diagrama de relación entre TADS

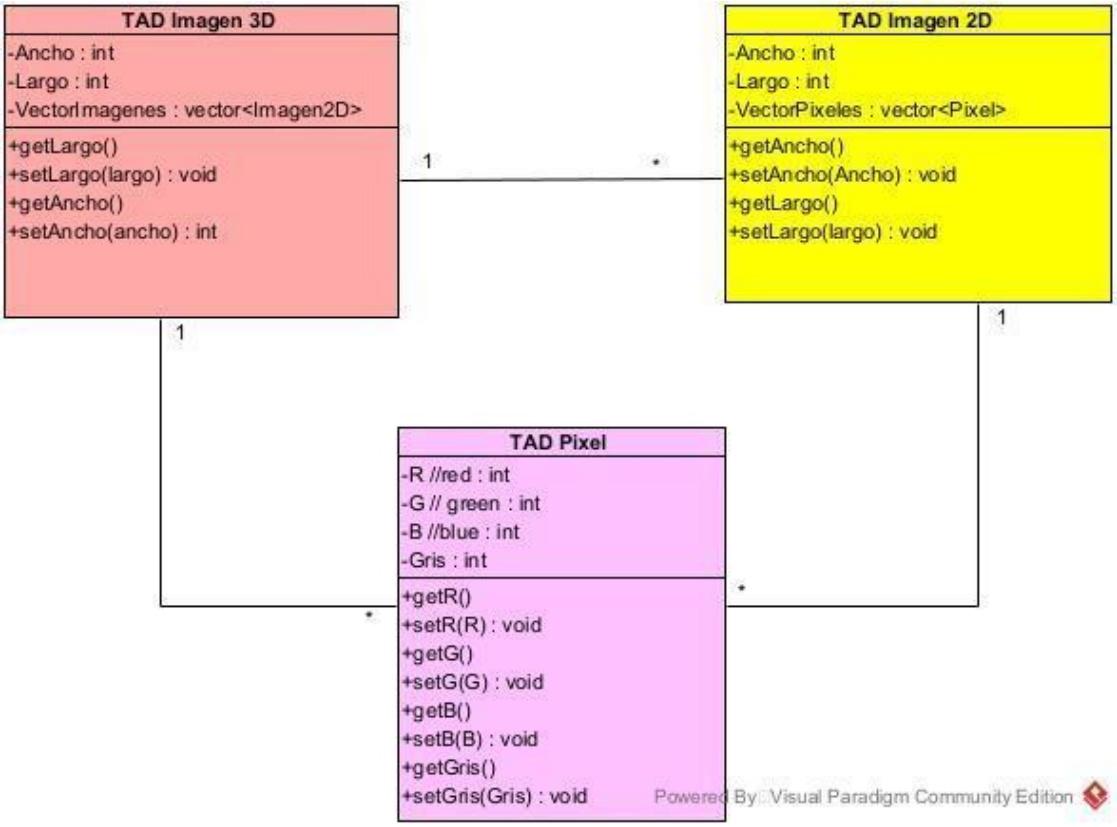

Figura 1: Diagrama de relación entre TADs

2. Funciones auxiliares

2.1 getUserInput()

getUserInput() es una función auxiliar la cual tiene como objetivo principal la recepción de los comandos digitados por el usuario.

2.2 getCommandArgs()

getcommandArgs() es una función auxiliar la cual tiene como objetivo principal devolver la lista de argumento de los comandos que intervienen en la funcionalidad del sistema.

2.3 badArgsMessage ()

badArgsMessage() es una función auxiliar la cual tiene como objetivo informar al usuario que ha ingresado un comando erróneamente por pantalla.

2.4 toBinary()

toBinary() es una función auxiliar la cual tiene como objetivo convertir a un número binario, se utiliza para “castear” y no presentar problemas al momento de enviar parámetros en las funciones nuevas implementadas.

3. Funcionamiento general de los comandos

3.1. Comando ayuda

El comando ayuda principalmente itera en la cantidad de comandos ydespliega las posibilidades que tiene el usuario para navegar en el aplicativo.

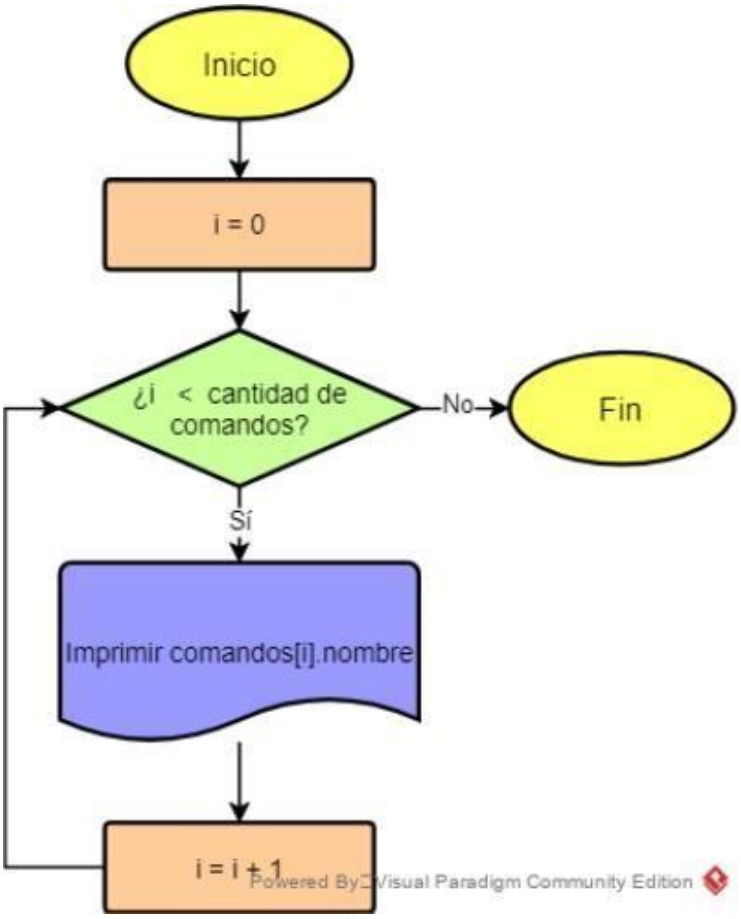

Figura 2: Diagrama de flujo comando ayuda

3.2. Comando salir

El comando se ejecuta desde el flujo principal del programa, y por ende, no tiene una especificación.

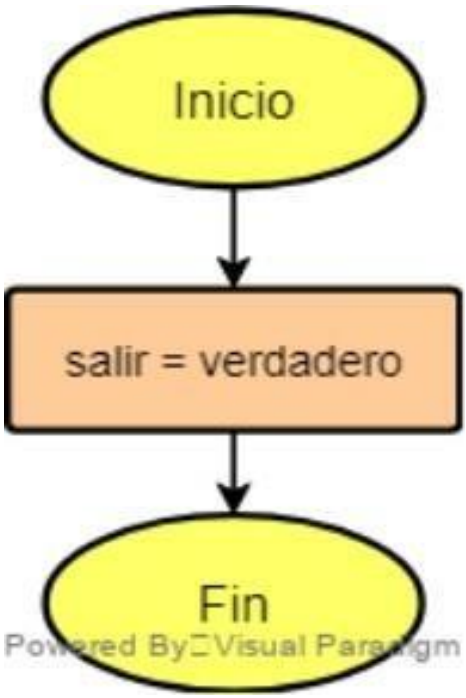

Figura 3: Diagrama de flujo comando salir

3.3. Comando cargar imagen

El comando debe cargar en memoria la imagen identificada con el nombre_imagen.pgm . Una vez cargada la información en memoria, el comandodebe mostrar el mensaje de carga satisfactoria. Si por alguna razón no es posiblecargar la imagen (nombre de archivo erróneo o no existe), el comando debe mostrar el mensaje de error.

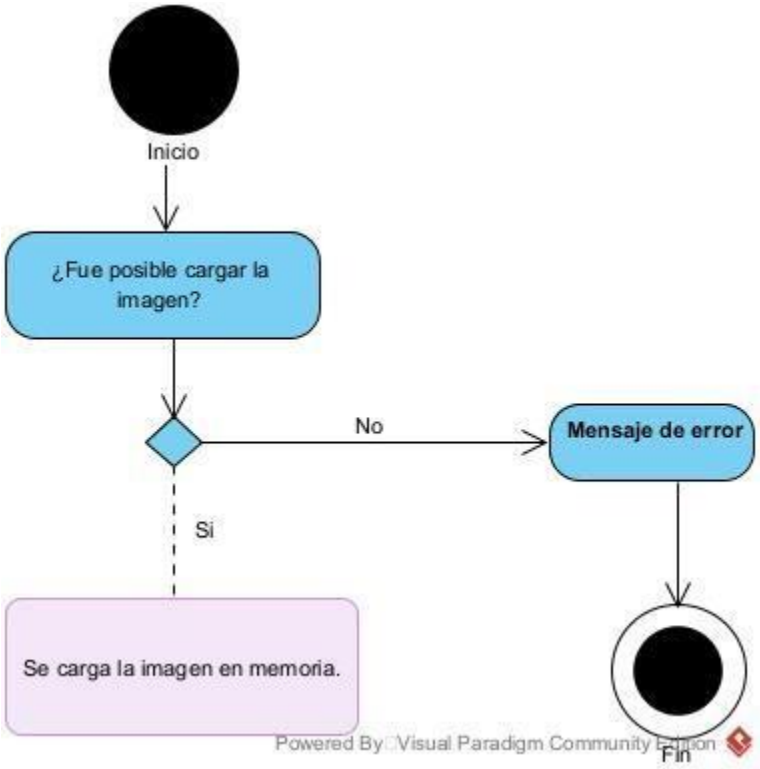

Figura 4: Diagrama de flujo comando cargar imagen.

3.4. Comando cargar volumen

El comando debe cargar en memoria la serie ordenada de imágenes identificada con el nombre_base y cuyo tamaño corresponde a $n\_im\text{-}imágenes$ (la serie podrá tenermáximo 99 imágenes). Todas las imágenes de la serie deben estar nombradas nombre_base xx.pgm, donde xx corresponde a dos dígitos de identificación de la posición de la imagen en la serie (varía en el rango 01 - n_im). Una vez cargada toda la información en memoria, el comando debe mostrar el mensaje de carga satisfactoria. Si por alguna razón no es posible cargar completamente la serie ordenada de imágenes (nombre de base erróneo, cantidad de imágenes no corresponde, error en alguna imagen), el comando debe mostrar el mensaje de error.

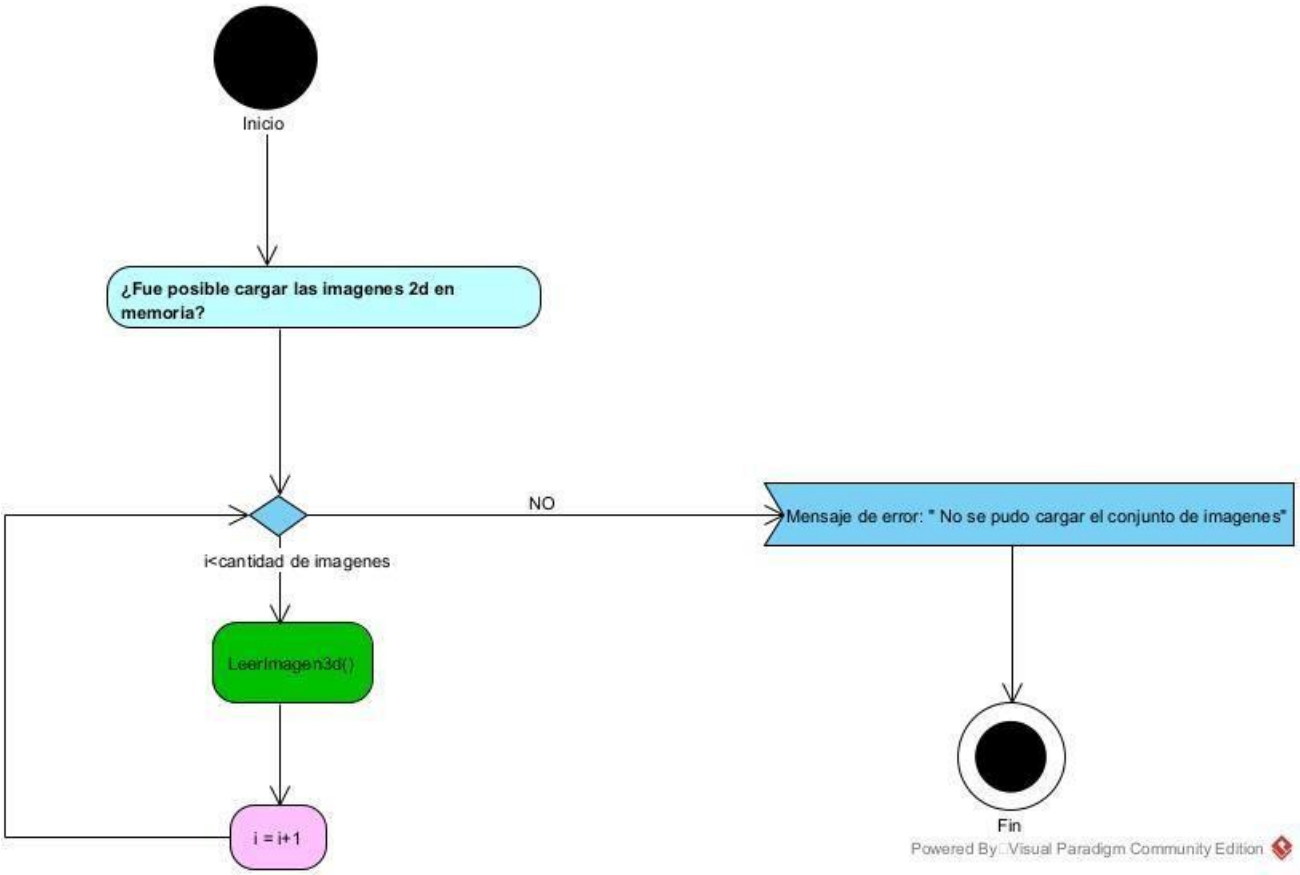

Figura 5: Diagrama de flujo comando cargar volumen

3.5. Comando info imagen

El comando debe mostrar en pantalla la información básica de la imagen actualmente cargada en memoria: nombre de archivo, ancho en pixeles y alto enpixeles. Si no se ha cargado aún una imagen en memoria, el comando debe mostrar el mensaje de error.

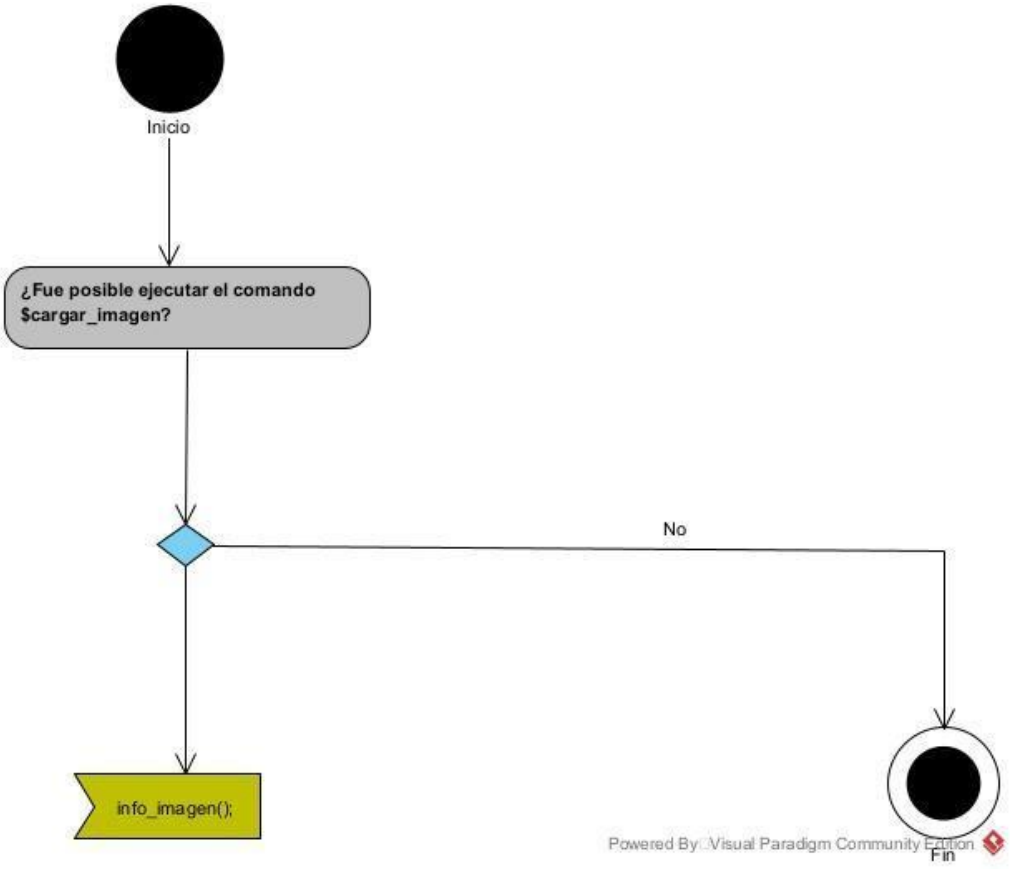

Figura 6: Diagrama de flujo comando info imagen.

3.6. Comando info volumen

El comando debe mostrar en pantalla la información básica de la serie de imágenes (volumen) cargadas actualmente en memoria: nombre base, cantidad de imágenes, ancho en pixeles y alto en pixeles. Si no se ha cargado aún un volumen en memoria, el comando debe mostrar el mensaje de error.

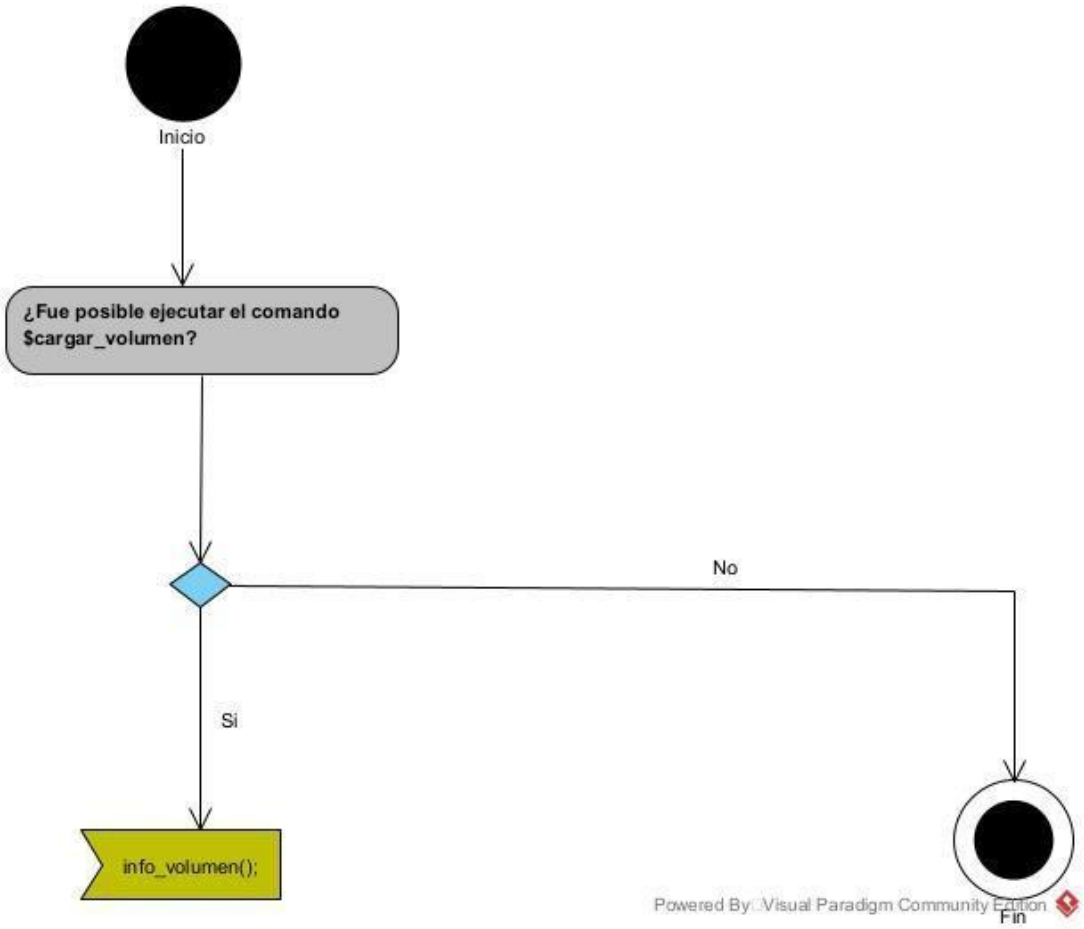

Figura 7: Diagrama de flujo comando info volumen.

3.7. Comando proyección

El comando debe tomar la serie ordenada de imágenes (ya cargada en memoria), y de acuerdo con la dirección especificada por el usuario, debe recorrer cada posición en el plano perpendicular a la dirección dada, y para cada una debe colapsar toda la información existente en la dirección dada utilizando el criterio especificado. Esto genera un sólo valor de píxel para cada posición del plano perpendicular, generando así una imagen 2D con la proyección de la información en el volumen. La dirección puede ser una entre x (en dirección de las columnas), y (en dirección de las filas) o z (en dirección de la profundidad). El criterio puede ser uno entre mínimo (el valor mínimo de intensidad), máximo (el valor máximo de intensidad), promedio (el valor promedio de intensidad) o mediana (el valor mediana de intensidad). Una vez generada la proyección, debe guardarse como imagen en formato PGM como nombre_archivo.pgm.

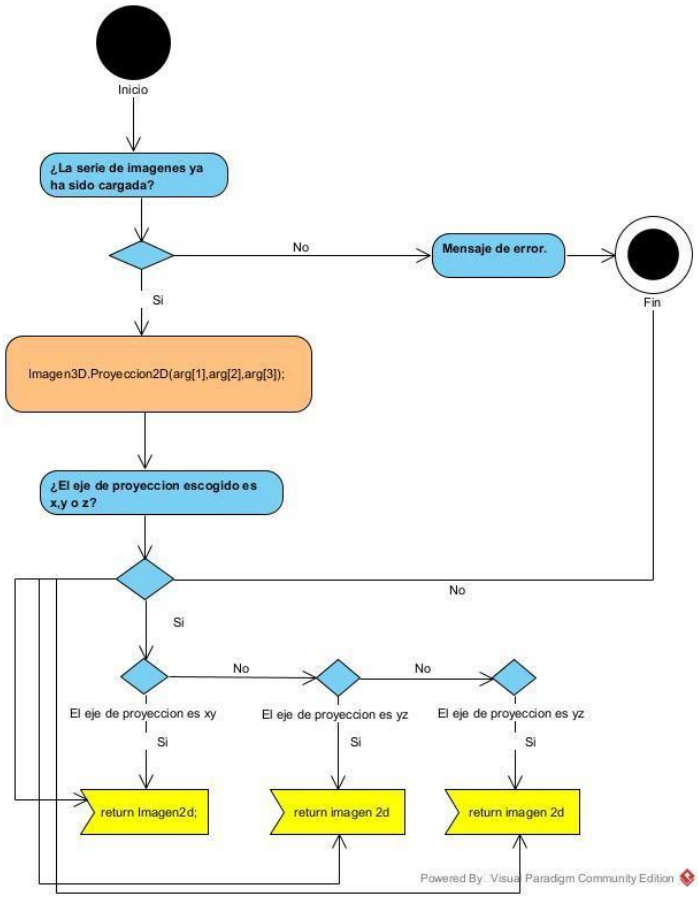

Figura 8: Diagrama de flujo comando proyección 2D

3.8. Comando codificar

Un principio muy importante alrededor de la compresión de datos es codificar cada símbolo de un mensaje usando la cantidad mínima posible de bits. Por ejemplo, si un mensaje estuviera escrito en lenguaje ASCII, el cual tiene 256 símbolos diferentes, la cantidad mínima de bits por símbolo requerida sería de 8. Otro principio esencial es que, aquellos símbolos que aparecen más frecuentemente en un mensaje, sería útil codificarlos con menos bits que aquellos menos frecuentes, de tal forma que el mensaje comprimido ocupe el menor espacio posible.

De esta manera , el comando codificar recibe como parámetro un nombre de archivo y crea un archivo binario siguiendo la configuración de bytes expuesta en el código de Hauffman.

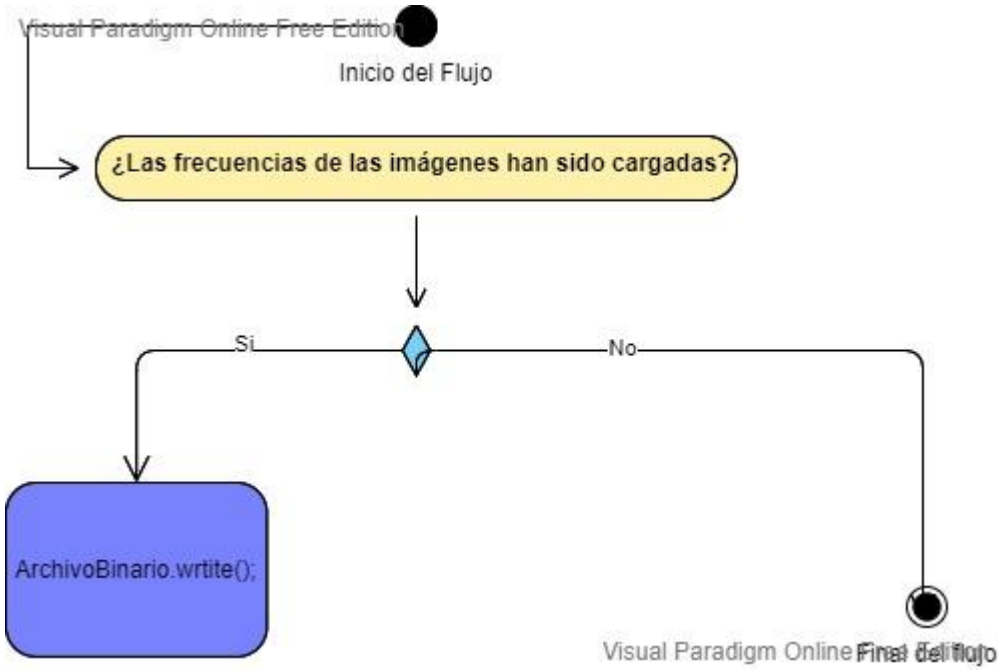

Figura 9: Diagrama de flujo comando codificar

3.9. Comando decodificar

3.9.1 Pseudo código

1. Cada vez que se lee un byte en txt, se devolvera una cadena hasta que se leen todos los bytes en txt, luego la cadena resultante es el resultado de la *codificación de Huffman*.

2. Recorre el resultado de la codificación de Huffman, restaura los píxeles originales de acuerdo con la lista de palabras de código que ha sido generada por la codificación de Huffman y genera imágenes bpm originales de acuerdo con los píxeles restaurados.

3. **comando:** decodificar_archivo nombre_archivo.huffman nombre_imagen.pgm

salida en pantalla:(proceso satisfactorio) El archivo nombre_archivo.huffman ha sido decodificado exitosamente.
(mensaje de error) El archivo nombre_archivo.huffman no ha podido ser decodificado

descripción: El comando debe cargar en memoria (en la estructura más adecuada) la información decodificación contenida en el archivo nombre_archivo.huffman y luego debe generar la correspondienteimagen decodificada en formato PGM, almacenándola en disco bajo el nombre nombre_imagen.pgm .

3.9.2 Encabezado de la función.

void Imagenes_2D::decodificar(std::string nombreHuffman, std::string nombreOutput)

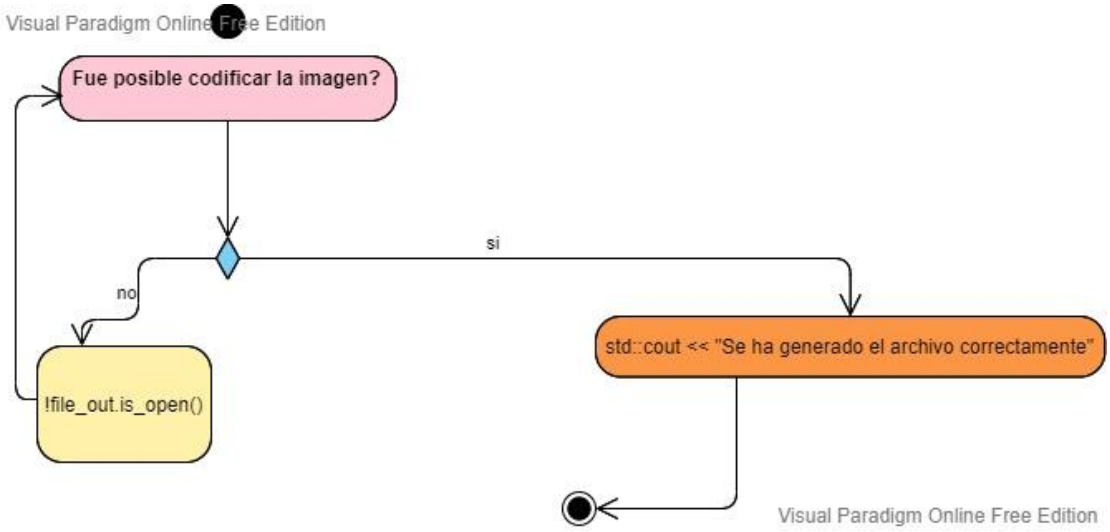

Figura 10: Diagrama de flujo comando decodificar

4. Análisis y resultados

"La codificación de Huffman, llamada así por su inventor D.A. Huffman, alcanza la cantidad mínima de redundancia posible en un conjunto fijo de códigos de longitud variable.
Esto no significa que la codificación de Huffman es un método de codificación óptimo. Significa que proporciona la mejor aproximación para los símbolos de codificación cuando se utiliza códigos de ancho fijo. El problema con el código de codificación de Huffman o Shannon-Fano es que utilizan un número entero de bits. Si la cantidad media de un carácter dado es de 2,5 bits, el código de Huffman para ese carácter debe ser 2 ó 3 bits, no 2,5. Debido a esto, la codificación de Huffman no puede considerarse un método de codificación óptima, aunque es la mejora aproximación que utilizan los códigos fijos con un número entero de bits"[1].

El código Huffman no solo es adecuado para archivos de imagen, sino que también se puede usar para archivos binarios y archivos de texto después de la fusión de símbolos. Pero todavía existen las siguientes deficiencias en la aplicación de visión: el número de datos de entrada está limitado por el tamaño de la tabla Huffman realizable; la decodificación es más complicada; es necesario conocer la distribución de frecuencia de los datos de entrada. Además, debido a la longitud desigual del código, existe un problema de coincidencia de velocidad entre la entrada y la salida. La solución es establecer un registro de búfer de cierta capacidad. La ventaja sobresaliente de su algoritmo es que la imagen comprimida no tiene distorsión y la unidad de píxel comprimida es la más cercana al valor de entropía real de la imagen, por lo que su relación de compresión debería ser la más alta. [2]

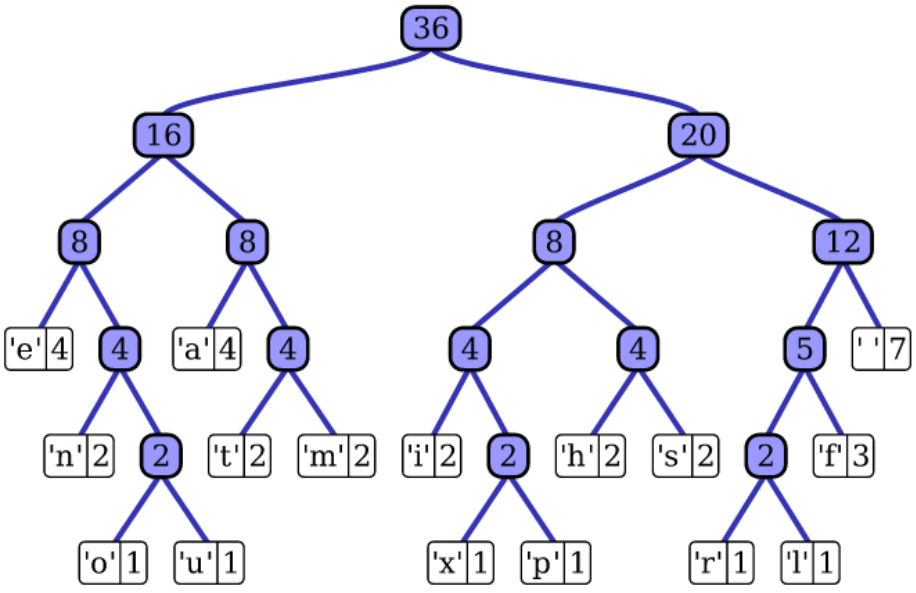

Figura 11: Diagrama codificación de HUFFMAN

5. Referencias

[1] Byte, B. y. (s/f). *Decodificando el algoritmo de Huffman*. Github.io. Recuperado el 26 de abril de 2022, de https://bitybyte.github.io/Descomprimiendo-datos-Huffman/

[2] *Revistas de la Universidad Nacional de Córdoba*. (s/f). Edu.ar. Recuperado el 26 de abril de 2022, de https://revistas.unc.edu.ar