Visualizador de imágenes

Douglas Santiago Diaz Rodriguez

Estructuras de datos
Pontificia Universidad Javeriana

1. Introducción

Este documento describe el diseño detallado del Sistema de Procesamiento de Imágenes desarrollado en C++. Su objetivo es presentar de manera clara y precisa la arquitectura global del sistema, así como las especificaciones de sus componentes principales y auxiliares. Se incluyen descripciones de entradas, salidas, precondiciones y postcondiciones de cada operación, junto con la especificación de los Tipos Abstractos de Datos (TADs) utilizados y esquemáticos que ilustran el funcionamiento de los algoritmos de compresión Huffman y segmentación por semillas mediante Dijkstra multifuente. Este documento servirá como guía para el desarrollo, mantenimiento y extensión futura del sistema.

2. Visión general del sistema

El Sistema de Procesamiento de Imágenes es una aplicación de consola desarrollada en C++ que permite realizar operaciones de carga, visualización y procesamiento de imágenes en formato PGM, así como procesamiento de volúmenes 3D y segmentación basada en semillas. Está compuesto por los siguientes módulos principales:

- **Interfaz de Línea de Comandos (CLI)**: gestiona la interacción con el usuario, parsea comandos y sincroniza la ejecución de las operaciones.
- **Gestor de Imágenes (PGM Handler)**: carga y guarda imágenes en formato P2, gestiona la matriz de píxeles y proporciona funcionalidades de información básica.
- **Compresor Huffman**: implementa la construcción del árbol de Huffman, codificación y decodificación de imágenes.
- **Volumetría**: carga múltiples cortes PGM numerados como un volumen 3D y genera proyecciones 2D (máximo, mínimo, promedio, mediana).
- **Segmentación por Semillas (Dijkstra Multifuente)**: aplica un algoritmo de caminos mínimos multifuente para segmentar regiones de la imagen a partir de puntos semilla.
- **Sistema Central (ImageProcessingSystem)**: orquesta los módulos anteriores, manteniendo el estado de la imagen o volumen cargado.

Cada módulo se comunica mediante estructuras de datos definidas (TADs) y aprovecha contenedores estándar (vectores, colas de prioridad). La arquitectura facilita la extensibilidad y el mantenimiento al separar claramente las responsabilidades de cada componente. Posteriormente se detallará la especificación de entradas, salidas, condiciones y pseudocódigo de cada operación.

3. Procedimiento principal

3.1 Descripción El método start() de ImageProcessingSystem implementa el ciclo principal de la aplicación, gestionando la interacción con el usuario a través de la CLI. Itera continuamente, leyendo comandos, delegando la ejecución a las operaciones auxiliares y mostrando resultados hasta que se recibe el comando salir.

3.2 Entradas

- Comandos y parámetros leídos desde la entrada estándar (std::cin).

3.3 Salidas

- Mensajes informativos, de error y de confirmación impresos en la salida estándar (std::cout).
- Estado interno modificado (imagen o volumen cargado en memoria).
- Archivos de salida generados en disco (imágenes .pgm, archivos .huf) según el comando.

3.4 Precondiciones y Postcondiciones

- **Precondiciones:**
  - El objeto ImageProcessingSystem debe existir e inicializarse correctamente.
  - El entorno de ejecución debe contar con permisos de lectura/escritura en el directorio de trabajo.
- **Postcondiciones:**
  - Después de cada comando, el estado interno del sistema refleja la operación ejecutada.
  - Al recibir salir, se libera memoria y finaliza la aplicación.

4. Operaciones auxiliares

Para cada comando se detallan: propósito, parámetros, resultados, precondiciones, postcondiciones, descripción y manejo de errores.

4.1 ayuda

- **Propósito:** Mostrar al usuario la lista de comandos disponibles.
- **Parámetros:** Ninguno.
- **Resultados:** Impresión de la guía de uso.
- **Precondiciones:** Ninguna.
- **Postcondiciones:** Ningún cambio en el estado interno.
- **Descripción:** Escribe en consola cada línea de ayuda predefinida.
- **Manejo de Errores:** No aplica.

4.2 cargar_imagen <archivo.pgm>

- **Propósito:** Cargar en memoria una imagen PGM en formato P2.
- **Parámetros:** archivo.pgm (ruta al fichero).
- **Resultados:** Población de imageData, width, height, maxVal.
- **Precondiciones:** El fichero debe existir y ser PGM P2.
- **Postcondiciones:** Imagen cargada en memoria.

- **Descripción:** Abre el fichero, lee cabecera "P2", dimensiones y valores de píxel.
- **Manejo de Errores:** Si no existe o formato inválido, imprime mensaje de error y no modifica el estado.

4.3 info_imagen
- **Propósito:** Mostrar información básica de la imagen cargada.
- **Parámetros:** Ninguno.
- **Resultados:** Dimensiones y valor máximo impresos en consola.
- **Precondiciones:** Imagen cargada (imageData no vacío).
- **Postcondiciones:** Estado interno sin cambios.
- **Descripción:** Consulta variables width, height, maxVal.
- **Manejo de Errores:** Si no hay imagen, imprime "No hay una imagen cargada".

4.4 codificar_imagen <salida.huf>
- **Propósito:** Comprimir la imagen con Huffman.
- **Parámetros:** salida.huf (archivo de destino).
- **Resultados:** Archivo binario con codificación y árbol guardado.
- **Precondiciones:** Imagen cargada.
- **Postcondiciones:** Archivo .huf creado.
- **Descripción:** Cuenta frecuencias, construye árbol Huffman, genera códigos y escribe datos.
- **Manejo de Errores:** Si falla apertura de archivo, imprime error.

4.5 decodificar_archivo <entrada.huf> <salida.pgm>
- **Propósito:** Descomprimir archivo Huffman a PGM.
- **Parámetros:** entrada.huf, salida.pgm.
- **Resultados:** Imagen PGM generada.
- **Precondiciones:** Archivo .huf válido existente.
- **Postcondiciones:** Archivo .pgm creado.
- **Descripción:** Lee árbol, reconstruye secuencia de píxeles y escribe en P2.
- **Manejo de Errores:** Mensajes si no existe o formato incorrecto.

4.6 cargar_volumen <base> <n>
- **Propósito:** Cargar un conjunto de PGM numerados como volumen 3D.
- **Parámetros:** base (prefijo), n (número de cortes).
- **Resultados:** Estructura tridimensional en memoria.
- **Precondiciones:** Todos los archivos baseXX.pgm deben existir.
- **Postcondiciones:** Volumen cargado.
- **Descripción:** Lee secuencialmente base01.pgm a base0n.pgm.
- **Manejo de Errores:** Si falta algún fichero, imprime error y aborta carga.

4.7 proyeccion2D <eje> <tipo> <salida.pgm>
- **Propósito:** Generar proyección 2D de un volumen.
- **Parámetros:** eje (x|y|z), tipo (max|min|prom|med), salida.pgm.
- **Resultados:** Imagen PGM de proyección.
- **Precondiciones:** Volumen cargado.
- **Postcondiciones:** Archivo PGM creado.
- **Descripción:** Recorre volumen, aplica función de reducción sobre eje y escribe PGM.
- **Manejo de Errores:** Comando inválido o volumen no cargado.

4.8 segmentar <salida.pgm> <x1> <y1> <label1> [...]

- **Propósito:** Segmentar la imagen por semillas usando Dijkstra multifuente.
- **Parámetros:** salida.pgm, pares (x, y, label) de 1 a 5 semillas.
- **Resultados:** Archivo PGM con etiquetas de región.
- **Precondiciones:** Imagen cargada y 1–5 semillas válidas.
- **Postcondiciones:** Archivo de segmentación generado.
- **Descripción:** Inicializa distancias, ejecuta Dijkstra multifuente y asigna etiquetas.
- **Manejo de Errores:** Si no hay imagen, semillas inválidas o escritura falla, imprime mensaje.

4.9 salir

- **Propósito:** Finalizar la aplicación.
- **Parámetros:** Ninguno.
- **Resultados:** Terminación del programa.
- **Precondiciones:** Ninguna.
- **Postcondiciones:** Liberación de recursos y salida.
- **Descripción:** Rompe el bucle principal y retorna de start().
- **Manejo de Errores:** No aplica.

5. Especificación de los TADs

TAD HuffmanNode Datos mínimos:

• value, entero, símbolo (valor de píxel) del nodo (0..maxVal).

• frequency, entero sin signo, número de ocurrencias del símbolo.

• left, puntero a HuffmanNode, subárbol izquierdo.

• right, puntero a HuffmanNode, subárbol derecho.

Operaciones:

• CrearNodo(value: int, freq: unsigned) → HuffmanNode* // constructor de hoja

• CrearInterno(left: HuffmanNode*, right: HuffmanNode*) → HuffmanNode* // constructor de nodo interno

• GetValue() → int

• GetFrequency() → unsigned

• SetLeft(node: HuffmanNode*)

• SetRight(node: HuffmanNode*)

TAD NodeComparator Datos mínimos:

• operator(), función de comparación que devuelve true si a.frequency > b.frequency.

Operaciones:

• Compare(a: HuffmanNode*, b: HuffmanNode*) → bool

TAD Matrix2D Datos mínimos:

• rows, entero, número de filas.

• cols, entero, número de columnas.

• data, vector<vector>, contenedor de elementos.

Operaciones:

• Init(rows: int, cols: int) → Matrix2D // inicializa con valores por defecto

• Get(i: int, j: int) → T& // acceso a elemento (precondición: índices válidos)

• Set(i: int, j: int, value: T) // asigna valor (precondición: índices válidos)

TAD Matrix3D Datos mínimos:

• layers, entero, número de capas.

• rows, entero, número de filas por capa.

• cols, entero, número de columnas por capa.

• data3D, vector<vector<vector>>, contenedor tridimensional.

Operaciones:

• Init(layers: int, rows: int, cols: int) → Matrix3D

• Get(k: int, i: int, j: int) → T& // acceso a elemento (precondición: índices válidos)

• Set(k: int, i: int, j: int, value: T)

TAD PriorityQueueState Datos mínimos:

• dist, entero largo, distancia acumulada desde semilla.

• y, entero, coordenada fila del píxel.

• x, entero, coordenada columna del píxel.

• label, entero, etiqueta de la semilla.

Operaciones:

• Push(state: State) // inserta un estado en la cola

• Top() → State // devuelve el estado con menor dist (precondición: no vacía)

• Pop() // elimina el estado superior (precondición: no vacía)

6. Conclusiones

En este documento se ha presentado el diseño completo del Sistema de Procesamiento de Imágenes, describiendo su arquitectura modular, el procedimiento principal, las operaciones auxiliares y la especificación detallada de los TADs empleados. Los esquemáticos y pseudocódigos incluidos facilitan la comprensión y sirven de guía para la implementación y mantenimiento.
Como resultados clave:

- Se valida un ciclo de interacción robusto mediante CLI, que soporta carga, compresión, decodificación, volumetría y segmentación.
- Los TADs diseñados aseguran encapsulación y claridad: HuffmanNode y NodeComparator soportan la compresión; Matrix2D y Matrix3D gestionan datos de píxel; PriorityQueueState posibilita la segmentación por Dijkstra multifuente.
- La separación de responsabilidades permite extensiones sin afectar al núcleo, cumpliendo principios de diseño SOLID.

Extensiones Futuras:

- Interfaz Gráfica (GUI): Incorporar una capa visual para selección de imágenes y visualización de resultados en tiempo real.
- Formatos Adicionales: Soportar formatos como PNG, JPEG o RAW mediante librerías externas.
- Aceleración por GPU: Paralelizar la compresión y segmentación usando CUDA u OpenCL para mejorar el rendimiento en volúmenes grandes.
- Algoritmos Avanzados: Añadir métodos de segmentación como watershed o clustering, y filtros avanzados (e.g., filtros de suavizado, detección de bordes).
- Persistencia y Logs: Implementar registro de operaciones y configuración de proyectos mediante archivos de script o base de datos.

- API de Servicios: Exponer funcionalidades como servicios web RESTful para integración en flujos de trabajo automatizados.