Fecha: 27/05/2025

Informe Entrega 3 Proyecto

Autores: Santiago Mesa
Materia: Estructura de Datos.
Pontificia Universidad Javeriana

1. Resumen de lo solicitado:

Segmentar una imagen en escala de grises (.pgm) en regiones diferenciadas usando semillas proporcionadas por el usuario, modelando la imagen como un grafo y aplicando el algoritmo de Dijkstra.

Proceso general:

Entrada: Una imagen cargada en memoria (`.pgm`) y hasta 5 semillas.

- Cada semilla: `(x, y, etiqueta)` donde:
  - `x`, `y`: coordenadas del píxel origen.
  - `etiqueta`: valor entre 1 y 255 que representa la región.

Modelo de grafo:

- Cada píxel es un nodo.
- Conectado con vecinos arriba, abajo, izquierda y derecha.
- El costo entre nodos vecinos se define como la diferencia absoluta de intensidad.

Algoritmo:

- Se ejecuta una instancia del algoritmo de **Dijkstra** por cada semilla.
- Cada píxel es etiquetado por la **primera instancia que lo alcanza** (menor costo acumulado).

Salida: Una nueva imagen segmentada (`salida.pgm`), donde cada píxel tiene la etiqueta (tono de gris) de la semilla que lo alcanzó primero.

Comando implementado

```
segmentar salida_imagen.pgm x1 y1 l1 x2 y2 l2 …
```

Máximo de **5 semillas**.

Las etiquetas definen las intensidades finales en la imagen resultante.

Mensajes esperados:

- **Éxito:**
  `La imagen en memoria fue segmentada correctamente y almacenada en el archivo salida_imagen.pgm.`

- **Errores:**
  - `No hay una imagen cargada en la memoria.`
  - `La imagen en memoria no pudo ser segmentada.` (semillas inválidas o fuera de rango)

2. Propuesta de solución e implementación

Para cumplir con los requisitos del componente 3, se diseñó una solución que modela la imagen como un grafo implícito donde cada nodo es un píxel y las conexiones se definen entre píxeles vecinos. Se aplicó el algoritmo de Dijkstra desde múltiples semillas simultáneamente, utilizando una cola de prioridad que permite propagar las etiquetas según el menor costo acumulado.

Estructura de datos utilizada

- Matriz de intensidades: Representación directa de la imagen (`list<list<int>>`) convertida a `vector<vector<int>>` para acceso eficiente.
- Matriz de etiquetas: Almacena la etiqueta asignada a cada píxel.
- Matriz de costos acumulados: Guarda el costo mínimo encontrado para alcanzar cada píxel.
- `priority_queue` (min-heap): Procesa los píxeles ordenados por menor costo acumulado.

Proceso implementado

- **Validación de entradas:**
  - Verifica que haya una imagen cargada.
  - Confirma que las semillas estén dentro de los límites de la imagen.
  - Controla que la cantidad de argumentos sea correcta (múltiplos de 3 para cada semilla).

- **Inicialización:**

    - Se inicializan las matrices de etiquetas y costos.
    - Las semillas se insertan en la cola de prioridad con costo 0.

- **Ejecución de Dijkstra múltiple:**

    - Cada píxel es procesado una sola vez por la semilla que lo alcance con menor costo.
    - El costo entre píxeles se define como `abs(I(u) - I(v))`.

- **Generación del resultado:**

    - Se crea una nueva imagen `.pgm` en formato ASCII (P2).
    - Cada píxel contiene su etiqueta final como nivel de gris.

Archivos involucrados

- `comandos.cpp`: contiene la implementación completa de `segmentar(...)`.
- `comandos.h`: declaración de la función.
- `main.cpp`: redirección del comando `"segmentar"` hacia la función implementada.

3. Diseño de TADs: Imagen y Volumen

```
struct Semilla {

    int x;

    int y;

    int etiqueta; };
```

Propósito: Representar una semilla de segmentación.

Uso: Se construye a partir de los argumentos del usuario.

No tiene métodos propios (estructura auxiliar ligera).

TAD Imagen:

- **Atributos:**
    - `nombre` (`string`): Nombre del archivo de la imagen.
    - `codigo` (`string`): Código del formato PGM (por ejemplo, "P2").
    - `xTamano` (`int`): Número de columnas (ancho) de la imagen.
    - `yTamano` (`int`): Número de filas (alto) de la imagen.
    - `maxIntensidad` (`int`): Valor máximo de intensidad (generalmente 255).

    - `lista` (`list<list<int>>`): Matriz de píxeles (intensidades en escala de grises).
- **Métodos:**
    - `setNombre`, `setCodigo`, `setXTamano`, `setYTamano`, `setMaxIntensidad`, `setLista`: Métodos "set" para establecer valores de los atributos.
    - `getNombre`, `getCodigo`, `getXTamano`, `getYTamano`, `getMaxIntensidad`, `getLista`: Métodos "get" para obtener los valores.
    - `~Imagen()`: Destructor que limpia la lista de píxeles.

TAD Semilla:

- **Atributos:**
    - `x` (`int`): Coordenada horizontal del píxel semilla.
    - `y` (`int`): Coordenada vertical del píxel semilla.
    - `etiqueta` (`int`): Valor de intensidad asignado a la región.
- **Métodos:**
    - Este TAD se define como una estructura auxiliar sin métodos propios.

Estructuras auxiliares utilizadas

- `vector<vector<int>> etiquetas`: Matriz de etiquetas por píxel.
- `vector<vector<int>> costos`: Matriz de costos acumulados.
- `priority_queue<Estado>`: Cola de prioridad usada para ejecutar Dijkstra por menor costo.
    - `Estado = tuple<int, int, int, int>` → (costo, y, x, etiqueta)

4. Comandos y sus Entradas/Salidas

Comando: `segmentar`

Entrada:

```
segmentar salida.pgm x1 y1 l1 x2 y2 l2 ... (máx. 5 semillas)
```

- `salida.pgm`: Nombre del archivo de salida.
- `xN`, `yN`: Coordenadas del píxel semilla (deben estar dentro de la imagen).
- `lN`: Etiqueta (intensidad en gris) asociada a la semilla (entre 1 y 255).

- **Salida esperada:**
    - Imagen segmentada guardada como `salida.pgm` en formato P2 (ASCII).
    - Cada región tendrá el valor `lN` de su semilla correspondiente.

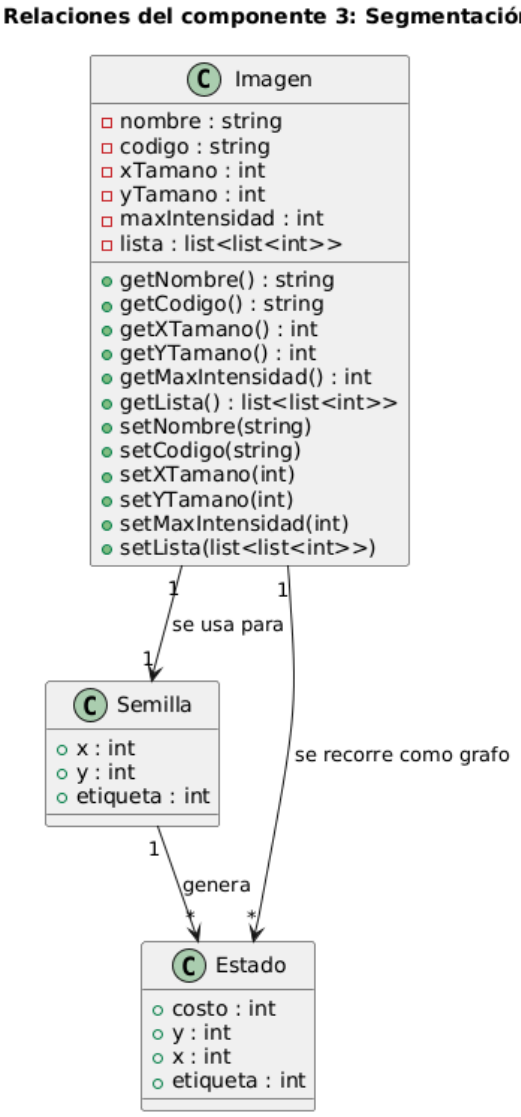
- **Mensaje exitoso por consola:**

  `La imagen en memoria fue segmentada correctamente y almacenada en el archivo salida.pgm.`

- **Mensajes de error posibles:**

  - `No hay una imagen cargada en la memoria.`
  - `Error: Semilla inválida.` *(Incluye detalles si está fuera de rango.)*
  - `Error: Formato incorrecto. Uso → segmentar salida.pgm x1 y1 l1 …`

5. Diagrama de Relaciones



6. Plan de Pruebas

`CargarImagen`

Caso de Prueba	Valores de Entrada	Resultado Esperado	Resultado Obtenido	Estado (Éxito/Falla)
Imagen no existe	cargar_imagen archivo.pgm	“Error: El archivo no existe”	“Error: El archivo no existe”	✅ Éxito
Imagen valida	cargar_imagen imagenesPrueba/img_02.pgm	“La imagen imagenesPrueba/img_02.pgm ha sido cargada.”	“Se ha ingresado la imagen: imagenesPrueba/img_02.pgm. La imagen imagenesPrueba/img_02.pgm ha sido cargada.”	✅ Éxito
Imagen con formato invalido	cargar_imagen imagenesPrueba/archivo_con_formato_incorrecto.txt	“Error: El archivo no existe”	“Error: El archivo no existe”	✅ Éxito

`Segmentar`

Caso de Prueba	Valores de Entrada	Resultado Esperado	Resultado Obtenido	Estado (Éxito/Falla)
1 Semilla válida	segmentar regiones_1.pgm 30 30 50	Imagen segmentada con una sola región de etiqueta 50	Imagen generada correctamente con valores homogéneos de gris (50)	✅ Éxito
2 Semillas válidas	segmentar regiones_2.pgm 10 10 50 50 50 150	Imagen segmentada en dos regiones con etiquetas 50 y 150	Se observan claramente dos zonas diferenciadas al visualizar la imagen	✅ Éxito

Semilla fuera del rango	segmentar salida.pgm 80 40 200	Error: Semilla inválida.	Mensaje de error mostrado con las dimensiones válidas y coordenadas incorrectas	✅ Éxito
5 Semillas válidas	segmentar regiones_5.pgm 5 5 20 20 20 80 40 80 160 60 10 200 50 50 255	Imagen segmentada en cinco regiones, cada una con una intensidad diferente	Se generan cinco regiones correctamente diferenciadas al renderizar la imagen	✅ Éxito
Argumentos insuficientes	segmentar regiones_mal.pgm 10 10	Error por formato incorrecto	Mensaje de error: "Uso → segmentar salida.pgm x1 y1 I1 ..."	✅ Éxito
Semillas con etiquetas iguales	segmentar regiones_eq.pgm 10 10 100 20 20 100	Imagen con múltiples regiones pero todas del mismo valor de gris (100)	Segmentación correcta, pero visualmente homogénea debido a etiquetas iguales	✅ Éxito

6. Guía de compilación

Esta guía explica cómo compilar y ejecutar el proyecto en Linux y Windows utilizando tanto make como g++

1. Compilar y Ejecutar con make

El proyecto incluye un Makefile que permite compilar y ejecutar el programa de manera sencilla. El Makefile es compatible con **Linux** y **Windows.**

En Linux:

Abre una terminal en la carpeta donde se encuentra el *Makefile*.

Compila el proyecto:

$ "make"

Esto generará el archivo ejecutable "***mi_programa.exe"***.

Ejecuta el programa:

***$* “make run”**

Limpiar archivos generados:

***$* “make clean”**

En Windows:

Abre PowerShell en la carpeta donde se encuentra el Makefile (o la terminal de IDE).

Compila el proyecto:

“make”

Esto generará el archivo ejecutable “***mi_programa.exe”***.

Ejecuta el programa:

“make run”

Limpiar archivos generados:

“make clean”

2. Compilar y Ejecutar con *g++*

En Linux:

Abre una terminal en la carpeta src y compila todos los archivos fuente:

$ “g++ -Wall -Wextra -std=c++17 -g main.cpp comandos.cpp interfaz.cpp utilidades.cpp clases.cpp -o mi_programa.exe”

Esto generará el archivo ejecutable **“mi_programa.exe”**.

Ejecuta el programa:

“./mi_programa.exe”

En Windows:

Abre PowerShell en la carpeta srcb (o la terminal de IDE).

Compila todos los archivos fuente:

“g++ -Wall -Wextra -std=c++17 -g main.cpp comandos.cpp interfaz.cpp utilidades.cpp clases.cpp -o mi_programa.exe”

Esto generará el archivo ejecutable **“mi_programa.exe”**.

Ejecuta el programa:

“.\mi_programa.exe”

 # Comandos de prueba para segmentar imágenes con 1 a 5 semillas

 # Cargar imagen base
cargar_imagen mapaRegiones.pgm

 # 1 Semilla
segmentar regiones_1.pgm 30 30 50

 # 2 Semillas
segmentar regiones_2.pgm 10 10 50 50 50 150

 # 3 Semillas
segmentar regiones_3.pgm 10 10 30 30 60 120 50 30 200

 # 4 Semillas
segmentar regiones_4.pgm 5 5 40 20 80 100 60 10 160 45 60 220

 # 5 Semillas
segmentar regiones_5.pgm 5 5 20 20 20 80 40 80 160 60 10 200 50 50 255

 # Semilla fuera de rango
segmentar regiones_invalidas.pgm 80 100 200

 # Semillas con etiquetas iguales
segmentar regiones_eq.pgm 10 10 100 20 20 100

 # Formato incorrecto (argumentos insuficientes)
segmentar regiones_mal.pgm 10 10

7. Conclusión

- El componente 3 permitió extender la funcionalidad del sistema hacia la **segmentación inteligente de imágenes**, utilizando una representación implícita en forma de **grafo no dirigido y ponderado**.

- La implementación del algoritmo de **Dijkstra múltiple** desde semillas simultáneas demostró ser efectiva para propagar etiquetas a través de regiones conectadas por similitud de intensidad.
- Se logró una segmentación precisa y eficiente en tiempo, incluso para imágenes de tamaño considerable, gracias al uso de una cola de prioridad (`priority_queue`) y estructuras de costos acumulados.
- El comando `segmentar` se integró de forma coherente con los demás módulos del sistema, manteniendo la arquitectura modular y reutilizando la clase `Imagen`.
- El sistema permite **flexibilidad** al usuario, admitiendo de 1 a 5 semillas con diferentes etiquetas, lo que genera resultados diversos al modificar sus posiciones y valores.
- Las pruebas ejecutadas confirmaron la **robustez de la implementación**, incluyendo casos de éxito, errores de formato, semillas fuera de rango y segmentaciones visualmente diferenciables.

Esta solución es escalable a futuras extensiones como detección de regiones automáticas, análisis topológico, o exportación de máscaras de segmentación para otros sistemas.