Universidad de los Andes > FACULTAD DE INGENIERÍA Y CIENCIAS APLICADAS

# Estructuras de Datos y Algoritmos (EDA)
# Tarea 2: Procesador de Imágenes (ImagePRO)

Prof: José M. Saavedra Rondo
Ayudantes: Lukas Pavez & Vicente Romero

**Fecha de entrega:** 13 de octubre de 2024

## 1. Objetivo

Comprender y aplicar estructuras de datos dinámicas en el contexto de procesamiento de imágenes.

## 2. Descripción

En esta tarea deberás implementar un programa para realizar operaciones básicas sobre imágenes como binarizar, detectar y contar objetos presentes en una imagen. Por ejemplo, tu programa debería permitir contar e identificar cada objeto de la Figura 1. Para este fin, deberás diseñar eficientemente estructuras dinámicas que te permitan realizar las tareas definidas más adelante.

Figura 1: Imagen con 8 monedas. El programa ImagePRO permitirá contar e identificar cada unos de los objetos presentes en la imagen.

Para facilitar la implementación, tendrás disponible un código base que puedes descargar de `https://github.com/jmsaavedrar/eda_cpp/tree/main/imagepro`.

## 3. Diseño General de la Solución

Para resolver el problema planteado, necesitarás resolver los siguientes subproblemas:

1. Crear una función para leer una imagen. Aquí trabajaremos con imágenes con un formato simple como BMP, que pueden ser fácilmente leídas con un pequeño código C++. De hecho, el código base trae la función **readImage** que te permitirá leer una imagen BMP. La función readImage devuelve un puntero a un objeto de la clase **Image**, la cual describiremos más adelante. Recuerda que una imagen no es más que una matriz de enteros de 8-bits (tipo char). Cada valor de la matriz representa una intensidad de luminosidad (color), donde 0 indica negro y 255 blanco. Los valores de la matriz se guardan en un *buffer* de memoria (array) de tipo **char**.

2. Para identificar los diferentes objetos en una imagen, será necesario aplicar un proceso de binarización de modo que los valores de los pixeles tengas solamente dos posibles valores 0 y 1. Para ello definimos un umbral, de modo que los valores menores a ese umbral tomen el valor 0 y los mayores o iguales tomen el valor 1. Suponiendo que los objetos aparecen más claro que el fondo, entonces los pixeles de los objetos tendrán el valor 1 y los del fondo quedarán en 0. El código base también incluye un proceso para binarizar (threshold).

3. **Ahora es tú turno!**, deberás extender el código base para permitir identificar cada objeto de la imagen binaria correspondiente a una imagen de entrada. Para ello debes aplicar las estrategias vistas en clase que te permiten recorrer componentes conectadas a partir de una posición. Aquí deberás crear y hace uso de estructuras dinámicas como Pilas, Colas o Listas Enlazadas.

# 4. Clases Involucradas

Para extender el código base deberás extender las siguientes clases:

## 4.1. Clase Imagen

La Figura 2 define la estructura y algunas operaciones de la clase Imagen.

- **Estructura**: esta estructura permite guardar el alto (height) y ancho (width) de la imagen, así como los valores de intensidad. En el último caso, los valores se guardan en el arreglo llamado *data*. Siéntete libre de extender la clase Imagen de la mejor manera.

- **Operaciones**

  1. getRegions: este método debe descomponer la representación binaria de la imagen en una lista de regiones (u objetos presentes). Cada región es una componente conexa, es decir desde cualquier punto al interior de la región existe un camino para llegar a cualquier otro dentro de la misma región. El resultado de este método debe ser una lista enlazada de regiones, donde cada región debe ser representada como un objeto de la clase Region, descrita en la Sección 4.2. Es importante notar que el tamaño de la ListOfRegion, es decir el número de objetos de la lista, indica la cantidad de objetos en la imagen.
  2. show: muestra la imagen binaria. El código base trae un ejemplo del uso de **show** de Image.

## 4.2. Clase Region

- **Estructura**: la clase **Region** permite representar a cada una de las regiones conexas encontradas en una imagen. La Figura 3 presenta el diseño básico de esta clase. Su estructura de datos debe permitir guardar un identificador, el tamaño de la región (esto es la cantidad de puntos que la componen) y una lista enlazada de coordenadas (Point2D) que definen a la región.

- **Operaciones**

  1. showRegion: debe imprimir la región a manera de imagen.


Figura 2: Esquema base de la clase Image. Siéntete libre de mejorarla.

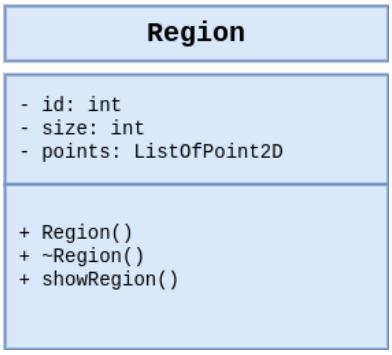


Figura 3: Esquema base de la clase Region. Siéntete libre de mejorarla.

## 5. Funcionalidad de tu Programa ImagePro

Tu procesador de imágenes, al que llamamos ImagePRO, debe ser implementado de modo que permita interactuar con el usuario de la siguiente manera:

```
$ ./imagepro    # ejecutamos  el programa imagepro
>> Bievenido a ImagePro
>> im1 = read <path>
>> im2 = read <path>
>> show im1
11100001110000
11100001110000
00000000000000
00000001110000
00000001110000
>> getRegions im1
La imagen de im1 tienes 3 regiones
 Region 1 -> size 6
 Region 2 -> size 6
 Region 3 -> size 6
>> showRegion im1 1
11100000000000
11100000000000
00000000000000
00000000000000
00000000000000
>> getRegions im2
La imagen de im2 tienes 2 regiones
```

```
 Region 1 -> size 6
 Region 2 -> size 10
 ...
>> exit
```

Los comandos permitidos durante la interacción con el usuario son:

- **read** : lee una image desde dado la ruta completa del archivo. Esta operación devuelve un identificador de la imagen leída.
- **show** : muestra imagen binaria indicando el id de la imagen leída.
- **showRegion**: muestra una región de una imagen indicando el id de la región.
- **exit**: cierra el programa.

## 6. Informe

1. **Abstract o Resumen**: es el resumen del trabajo.
2. **Introducción**: aquí se describe cada uno de los algoritmos implementados y evaluados (10 %)
3. **Desarrollo**: aquí se describe el diseño e implementación de las estructuras y algoritmos que has desarrollado para resolver la tarea. Además, presenta una descripción y ejemplos de cómo usar tus programas. En esta sección es importante justificar la selección del algoritmo de ordenación. (40 %)
4. **Resultados Experimentales y Discusión**: aquí se presentan los resultados, pero lo más importante es analizarlos. Se debe observar y describir el comportamiento de los métodos. Es recomendable mostrar el desempeño de tu solución con ejemplos prácticos. Puedes ocupar las imágenes que vienen en el repositorio del código base como casos de estudio. (40 %).
5. **Conclusiones**: ideas o hallazgos principales sobre el trabajo. (10 %)

## 7. Restricciones

1. Pueden trabajar en grupos de 2 estudiantes.
2. Todos los programas deben ser propios, permitiendo solamente utilizar el código disponible en el repositorio del curso `https://github.com/jmsaavedrar/eda_cpp/`.
3. El hallazgo de plagio será penalizado con nota 1.0, para todos los grupos involucrados.
4. Todos las implementaciones deben ser realizadas en C++.
5. **La entrega del informe es obligatorio**. Un trabajo sin informe no será calificado, asignando la nota mínima igual a 1.0.

## 8. Entrega

La entrega se debe realizar por canvas hasta el domingo 13 de octubre, 2024, 23:50 hrs. La entrega debe incluir:

1. Código fuente (en C++), junto a un README con los pasos de compilación.
2. Informe