Práctica N° 17 - FdLP

Christian Omar Rodriguez Huamanñahui
crodriguezh@ulasalle.edu.pe

NOV 2022

Índice

1. Ejercicio 1

1. Implemente la adición de imágenes con los elementos de la Figura 1. Considere el problema de *overflow* en los pixeles, esto sucede cuando un pixel supera el valor de 255 o es inferior a 0; por ejemplo si un pixel resulta con un valor mayor a 255, solo le asigna el valor de 255.

Figura 1: Imágenes de muestra.

- CÓDIGO EN GOLANG (CONCURRENCIA Y PARALELISMO):

```
// --------EJERCICIO1: CONCURRENCIA Y PARALELISMO--------
// IMPORTANDO LIBRERIAS NECESARIAS
package main

import (
        "fmt"
        "image"
        "image/color"
        "image/jpeg"
        "log"
        "os"
        "path/filepath"
        "strings"
        "sync"
        "time"
)

func main() {
        //VARIABLE DEL FADE (BRILLO) ENTRE IMÁGENES
        FADE := 0.75
        //SE REGISTRA LA RUTA DE LA PRIMERA IMÁGEN
        IMPORTAR_IMAGEN1 := "src/OG/AFTER_HOURS.jpg"
        f, err := os.Open(IMPORTAR_IMAGEN1)
        check(err)

        //SE CAPTURAN LOS VALORES DE LA 1RA IMAGEN
        img, _, err := image.Decode(f)

        //SE REGISTRA LA RUTA DE LA SEGUNDA IMÁGEN
        IMPORTAR_IMAGEN2 := "src/OG/DAWN_FM.jpg"
        f2, err2 := os.Open(IMPORTAR_IMAGEN2)
        check(err2)

        //SE CAPTURAN LOS VALORES DE LA 2DA IMAGEN
        img2, _, err := image.Decode(f2)

        //SE RECONOCE EL TAMAÑO DE CADA IMAGEN
        TAMAÑO := img.Bounds().Size()
        rect := image.Rect(0, 0, TAMAÑO.X, TAMAÑO.Y)
```

```
wImg := image.NewRGBA(rect)

//GOROUTINE
wg := new(sync.WaitGroup)

//TEMPORIZADOR: INICIAR
start := time.Now()

//CICLO QUE RECORRE TODO Y (ALTO)
for ALTO := 0; ALTO < TAMAÑO.Y; ALTO++ {

        //GOROUTINE
        wg.Add(1)
        ALTO := ALTO
        go func() {

                // CICLO QUE RECORRE TODO X (ANCHO)
                for ANCHO := 0; ANCHO < TAMAÑO.X; ANCHO++ {
                        pixel := img.At(ANCHO, ALTO)
                        pixel2 := img2.At(ANCHO, ALTO)

                        OG_COLOR1 := color.RGBAModel.Convert(pixel).(color.RGBA)
                        OG_COLOR2 := color.RGBAModel.Convert(pixel2).(color.RGBA)

                        // CONVIRTIENDO ANÁLISIS RGB A VALORES FLOAT (IMÁGEN 1 Y 2)
                        RED_CHANNEL := float64(OG_COLOR1.R)
                        GREEN_CHANNEL := float64(OG_COLOR1.G)
                        BLUE_CHANNEL := float64(OG_COLOR1.B)

                        RED_CHANNEL2 := float64(OG_COLOR2.R)
                        GREEN_CHANNEL2 := float64(OG_COLOR2.G)
                        BLUE_CHANNEL2 := float64(OG_COLOR2.B)

                        // COMBINANDO VALORES RGB DE AMBAS IMÁGENES
                        RED_CHANNEL3 := uint8((RED_CHANNEL + RED_CHANNEL2) * FADE)
                        GREEN_CHANNEL3 := uint8((GREEN_CHANNEL + GREEN_CHANNEL2) * FADE)
                        BLUE_CHANNEL3 := uint8((BLUE_CHANNEL + BLUE_CHANNEL2) * FADE)

                        //ASIGNANDO VALOR MÁXIMO DE CANT. DE PIXELES POR CADA CANAL DE LA IMÁGEN 3
                        if RED_CHANNEL3 > 255 || GREEN_CHANNEL3 > 255 || BLUE_CHANNEL3 > 255 {
                                RED_CHANNEL3 = 255
                                GREEN_CHANNEL3 = 255
                                BLUE_CHANNEL3 = 255
                        }
                        if RED_CHANNEL3 < 0 || GREEN_CHANNEL3 < 0 || BLUE_CHANNEL3 < 0 {
                                RED_CHANNEL3 = 0
                                GREEN_CHANNEL3 = 0
                                BLUE_CHANNEL3 = 0
                        }

                        //SETTEANDO LOS VALORES RGBA DE CADA CANAL
                        COLOR := color.RGBA{
                                R: RED_CHANNEL3, G: GREEN_CHANNEL3, B: BLUE_CHANNEL3, A: OG_COLOR1.A,
                        }
                        wImg.Set(ANCHO, ALTO, COLOR)
                }
                defer wg.Done()
        }()
}
wg.Wait()
```

```
        //FINALIZAR CRONÓMETRO E IMPRIMIR EL TIEMPO
        elapsed := time.Since(start)
        print("Se guardó correctamente la nueva imágen en la carpeta: 'OUTPUT'")
        log.Printf("\nTIEMPO | EJERCICIO1 | GOROUTINE: %s", elapsed)

        //DETERMINANDO CARPETA DONDE SE GUARDARÁ EL RESULTADO
        imgOut := "src/OUT/"

        //CREAR EL RESULTADO
        ext := filepath.Ext(IMPORTAR_IMAGEN1)
        name := strings.TrimSuffix(filepath.Base("OUTPUT"), ext)
        newImagePath := fmt.Sprintf("%s/%s_CONCURRENCIA-EJ1%s", filepath.Dir(imgOut), name, ext)
        fg, err := os.Create(newImagePath)
        defer fg.Close()
        check(err)
        err = jpeg.Encode(fg, wImg, nil)
        check(err)
}

// POR SI HAY ERRORES
func check(err error) {
        if err != nil {
                panic(err)
        }
}
```

- CÓDIGO EN GOLANG (SECUENCIAL):

```
// --------EJERCICIO1: SECUENCIAL--------
// IMPORTANDO LIBRERIAS NECESARIAS
package main

import (
        "fmt"
        "image"
        "image/color"
        "image/jpeg"
        "log"
        "os"
        "path/filepath"
        "strings"
        "time"
)

func main() {
        //VARIABLE DEL FADE (BRILLO) ENTRE IMÁGENES
        FADE := 0.75
        //SE REGISTRA LA RUTA DE LA PRIMERA IMÁGEN
        IMPORTAR_IMAGEN1 := "src/OG/AFTER_HOURS.jpg"
        f, err := os.Open(IMPORTAR_IMAGEN1)
        check(err)

        //SE CAPTURAN LOS VALORES DE LA 1RA IMAGEN
        img, _, err := image.Decode(f)

        //SE REGISTRA LA RUTA DE LA SEGUNDA IMÁGEN
        IMPORTAR_IMAGEN2 := "src/OG/DAWN_FM.jpg"
        f2, err2 := os.Open(IMPORTAR_IMAGEN2)
        check(err2)
```

```
//SE CAPTURAN LOS VALORES DE LA 2DA IMAGEN
img2, _, err := image.Decode(f2)

//SE RECONOCE EL TAMAÑO DE CADA IMAGEN
TAMAÑO := img.Bounds().Size()
rect := image.Rect(0, 0, TAMAÑO.X, TAMAÑO.Y)
wImg := image.NewRGBA(rect)

//TEMPORIZADOR: INICIAR
start := time.Now()
//CICLO QUE RECORRE TODO Y (ALTO)
for ALTO := 0; ALTO < TAMAÑO.Y; ALTO++ {
        // CICLO QUE RECORRE TODO X (ANCHO)
        for ANCHO := 0; ANCHO < TAMAÑO.X; ANCHO++ {
                pixel := img.At(ANCHO, ALTO)
                pixel2 := img2.At(ANCHO, ALTO)

                OG_COLOR1 := color.RGBAModel.Convert(pixel).(color.RGBA)
                OG_COLOR2 := color.RGBAModel.Convert(pixel2).(color.RGBA)

                // CONVIRTIENDO ANÁLISIS RGB A VALORES FLOAT (IMÁGEN 1 Y 2)
                RED_CHANNEL := float64(OG_COLOR1.R)
                GREEN_CHANNEL := float64(OG_COLOR1.G)
                BLUE_CHANNEL := float64(OG_COLOR1.B)

                RED_CHANNEL2 := float64(OG_COLOR2.R)
                GREEN_CHANNEL2 := float64(OG_COLOR2.G)
                BLUE_CHANNEL2 := float64(OG_COLOR2.B)

                // COMBINANDO VALORES RGB DE AMBAS IMÁGENES
                RED_CHANNEL3 := uint8((RED_CHANNEL + RED_CHANNEL2) * FADE)
                GREEN_CHANNEL3 := uint8((GREEN_CHANNEL + GREEN_CHANNEL2) * FADE)
                BLUE_CHANNEL3 := uint8((BLUE_CHANNEL + BLUE_CHANNEL2) * FADE)

                //ASIGNANDO VALOR MÁXIMO DE CANT. DE PIXELES POR CADA CANAL DE LA IMÁGEN 3
                if RED_CHANNEL3 > 255 || GREEN_CHANNEL3 > 255 || BLUE_CHANNEL3 > 255 {
                        RED_CHANNEL3 = 255
                        GREEN_CHANNEL3 = 255
                        BLUE_CHANNEL3 = 255
                }
                if RED_CHANNEL3 < 0 || GREEN_CHANNEL3 < 0 || BLUE_CHANNEL3 < 0 {
                        RED_CHANNEL3 = 0
                        GREEN_CHANNEL3 = 0
                        BLUE_CHANNEL3 = 0
                }

                //SETTEANDO LOS VALORES RGBA DE CADA CANAL
                COLOR := color.RGBA{
                        R: RED_CHANNEL3, G: GREEN_CHANNEL3, B: BLUE_CHANNEL3, A: OG_COLOR1.A,
                }
                wImg.Set(ANCHO, ALTO, COLOR)
        }
}

//FINALIZAR CRONÓMETRO E IMPRIMIR EL TIEMPO
elapsed := time.Since(start)
print("Se guardó correctamente la nueva imágen en la carpeta: 'OUTPUT'")
log.Printf("\nTIEMPO | EJERCICIO1 | SECUENCIAL: %s", elapsed)

//DETERMINANDO CARPETA DONDE SE GUARDARÁ EL RESULTADO
```

```
        imgOut := "src/OUT/"

        //CREAR EL RESULTADO
        ext := filepath.Ext(IMPORTAR_IMAGEN1)
        name := strings.TrimSuffix(filepath.Base("OUTPUT"), ext)
        newImagePath := fmt.Sprintf("%s/%s_SECUENCIAL-EJ1%s", filepath.Dir(imgOut), name, ext)
        fg, err := os.Create(newImagePath)
        defer fg.Close()
        check(err)
        err = jpeg.Encode(fg, wImg, nil)
        check(err)
}

// POR SI HAY ERRORES
func check(err error) {
        if err != nil {
                panic(err)
        }
}
```

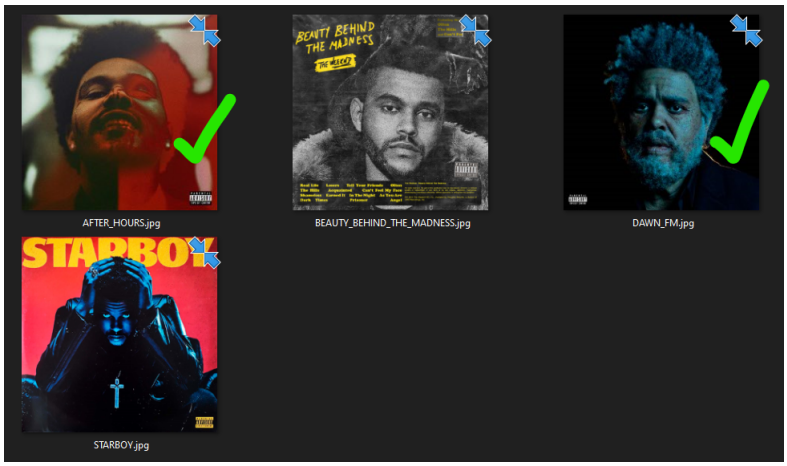
- CONCURRENCIA Y PARALELISTMO: EJECUCIÓN Y TIEMPO EN CONSOLA (ADMIN/POWERSHELL):

```
PS C:\Users\nuez_\OneDrive\Escritorio\img_process_golang> go run 1SEC.go
Se guardó correctamente la nueva imágen en la carpeta: 'OUTPUT'2022/11/25 03:47:22
TIEMPO | EJERCICIO1 | GOROUTINE: 3.693ms
```

- SECUENCIAL: EJECUCIÓN Y TIEMPO EN CONSOLA (ADMIN/POWERSHELL):

```
PS C:\Users\nuez_\OneDrive\Escritorio\img_process_golang> go run 1S.go
Se guardó correctamente la nueva imágen en la carpeta: 'OUTPUT'2022/11/25 04:37:30
TIEMPO | EJERCICIO1 | SECUENCIAL: 7.437ms
```

- INPUT DE IMÁGENES (ARCHIVOS: .ªFTER HOURS"Y "DAWN FM"):



- OUTPUT DE IMAGEN RESULTANTE:

2. Ejercicio 2

2. El operador *blending*, es un operador que suma dos imágenes, pero nosotros podemos decidir que imágen tendra mas presencia en el resultado. Implemente el operador *Blending* (ecuación 1) y evalue sus resultados con imágenes de su preferencia, tambien pruebe diferentes valores de x.

$$Q(i,j) = X * P_1(i,j) + (1 - X) * P_2(i,j) \tag{1}$$

Donde: $Q(i,j)$, es un pixel de la imagen resultado; $P_1(i,j)$, es un pixel de la imagen de entrada; $P_2(i,j)$, es un pixel de la otra imagen de entrada.

Por ejemplo, con las imágenes de entrada de la Figura 2, se puede obtener el resultado de la Figura 3, para un $x = 0{,}25$, $x = 0{,}5$ y $x = 0{,}75$.

Figura 3: Imagen resultado de *blending*, para $x = 0{,}25$, $x = 0{,}5$ y $x = 0{,}75$.

- CÓDIGO EN GOLANG (CONCURRENCIA Y PARALELISMO):

```
// --------EJERCICIO2: CONCURRENCIA Y PARALELISMO--------
// IMPORTANDO LIBRERIAS NECESARIAS
package main

import (
        "fmt"
        "image"
        "image/color"
        "image/jpeg"
        "log"
        "os"
        "path/filepath"
        "strings"
        "sync"
```

```
        "time"
)

func main() {
        //VARIABLE DEL FADE (BRILLO) ENTRE IMÁGENES / VALORES ENTRE 0 - 1 (PROBAR: 0.25 - 0.50 - 0.75)
        FADE := 0.75
        //SE REGISTRA LA RUTA DE LA PRIMERA IMÁGEN
        IMPORTAR_IMAGEN1 := "src/OG/BEAUTY_BEHIND_THE_MADNESS.jpg"
        f, err := os.Open(IMPORTAR_IMAGEN1)
        check(err)

        //SE CAPTURAN LOS VALORES DE LA 1RA IMAGEN
        img, _, err := image.Decode(f)

        //SE REGISTRA LA RUTA DE LA SEGUNDA IMÁGEN
        IMPORTAR_IMAGEN2 := "src/OG/STARBOY.jpg"
        f2, err2 := os.Open(IMPORTAR_IMAGEN2)
        check(err2)

        //SE CAPTURAN LOS VALORES DE LA 2DA IMAGEN
        img2, _, err := image.Decode(f2)

        //SE RECONOCE EL TAMAÑO DE CADA IMAGEN
        TAMAÑO := img.Bounds().Size()
        rect := image.Rect(0, 0, TAMAÑO.X, TAMAÑO.Y)
        wImg := image.NewRGBA(rect)

        //GOROUTINE
        wg := new(sync.WaitGroup)

        //TEMPORIZADOR: INICIAR
        start := time.Now()

        //CICLO QUE RECORRE TODO Y (ALTO)
        for ALTO := 0; ALTO < TAMAÑO.Y; ALTO++ {

                //GOROUTINE
                wg.Add(1)
                ALTO := ALTO
                go func() {

                        // CICLO QUE RECORRE TODO X (ANCHO)
                        for ANCHO := 0; ANCHO < TAMAÑO.X; ANCHO++ {
                                pixel := img.At(ANCHO, ALTO)
                                pixel2 := img2.At(ANCHO, ALTO)

                                OG_COLOR1 := color.RGBAModel.Convert(pixel).(color.RGBA)
                                OG_COLOR2 := color.RGBAModel.Convert(pixel2).(color.RGBA)

                                // CONVIRTIENDO ANÁLISIS RGB A VALORES FLOAT (IMÁGEN 1 Y 2)
                                RED_CHANNEL := float64(OG_COLOR1.R)
                                GREEN_CHANNEL := float64(OG_COLOR1.G)
                                BLUE_CHANNEL := float64(OG_COLOR1.B)

                                RED_CHANNEL2 := float64(OG_COLOR2.R)
                                GREEN_CHANNEL2 := float64(OG_COLOR2.G)
                                BLUE_CHANNEL2 := float64(OG_COLOR2.B)

                                // REALIZANDO MEZCLA FADE CON LA FÓRMULA ESTABLECIDA EN EL EJERCICIO2
                                RED_CHANNEL3 := uint8(FADE*RED_CHANNEL + (1-FADE)*RED_CHANNEL2)
```

```
                                GREEN_CHANNEL3 := uint8(FADE*GREEN_CHANNEL + (1-FADE)*GREEN_CHANNEL2)
                                BLUE_CHANNEL3 := uint8(FADE*BLUE_CHANNEL + (1-FADE)*BLUE_CHANNEL2)
                                COLOR := color.RGBA{
                                        R: RED_CHANNEL3, G: GREEN_CHANNEL3, B: BLUE_CHANNEL3, A: OG_COLOR1.A,
                                }
                                wImg.Set(ANCHO, ALTO, COLOR)
                        }
                        defer wg.Done()
                }()
        }
        wg.Wait()

        //FINALIZAR CRONÓMETRO E IMPRIMIR EL TIEMPO
        elapsed := time.Since(start)
        print("Se guardó correctamente la nueva imágen en la carpeta: 'OUTPUT'")
        log.Printf("\nTIEMPO | EJERCICIO2 | GOROUTINE: %s", elapsed)

        //DETERMINANDO CARPETA DONDE SE GUARDARÁ EL RESULTADO
        imgOut := "src/OUT/"

        //CREAR EL RESULTADO
        ext := filepath.Ext(IMPORTAR_IMAGEN1)
        name := strings.TrimSuffix(filepath.Base("OUTPUT"), ext)
        newImagePath := fmt.Sprintf("%s/%s_CONCURRENCIA-EJ2%s", filepath.Dir(imgOut), name, ext)
        fg, err := os.Create(newImagePath)
        defer fg.Close()
        check(err)
        err = jpeg.Encode(fg, wImg, nil)
        check(err)
}

// POR SI HAY ERRORES
func check(err error) {
        if err != nil {
                panic(err)
        }
}
```

- CÓDIGO EN GOLANG (SECUENCIAL):

```
// --------EJERCICIO2: SECUENCIAL--------
// IMPORTANDO LIBRERIAS NECESARIAS
package main

import (
        "fmt"
        "image"
        "image/color"
        "image/jpeg"
        "log"
        "os"
        "path/filepath"
        "strings"
        "time"
)

func main() {
        //VARIABLE DEL FADE (BRILLO) ENTRE IMÁGENES / VALORES ENTRE 0 - 1 (PROBAR: 0.25 - 0.50 - 0.75)
        FADE := 0.25
        //SE REGISTRA LA RUTA DE LA PRIMERA IMÁGEN
        IMPORTAR_IMAGEN1 := "src/OG/BEAUTY_BEHIND_THE_MADNESS.jpg"
```

```
f, err := os.Open(IMPORTAR_IMAGEN1)
check(err)

//SE CAPTURAN LOS VALORES DE LA 1RA IMAGEN
img, _, err := image.Decode(f)

//SE REGISTRA LA RUTA DE LA SEGUNDA IMÁGEN
IMPORTAR_IMAGEN2 := "src/OG/STARBOY.jpg"
f2, err2 := os.Open(IMPORTAR_IMAGEN2)
check(err2)

//SE CAPTURAN LOS VALORES DE LA 2DA IMAGEN
img2, _, err := image.Decode(f2)

//SE RECONOCE EL TAMAÑO DE CADA IMAGEN
TAMAÑO := img.Bounds().Size()
rect := image.Rect(0, 0, TAMAÑO.X, TAMAÑO.Y)
wImg := image.NewRGBA(rect)

start := time.Now()
// loop though all the x

for ALTO := 0; ALTO < TAMAÑO.Y; ALTO++ {
        for ANCHO := 0; ANCHO < TAMAÑO.X; ANCHO++ {
                pixel := img.At(ANCHO, ALTO)
                pixel2 := img2.At(ANCHO, ALTO)

                OG_COLOR1 := color.RGBAModel.Convert(pixel).(color.RGBA)
                OG_COLOR2 := color.RGBAModel.Convert(pixel2).(color.RGBA)

                // CONVIRTIENDO ANÁLISIS RGB A VALORES FLOAT (IMÁGEN 1 Y 2)
                RED_CHANNEL := float64(OG_COLOR1.R)
                GREEN_CHANNEL := float64(OG_COLOR1.G)
                BLUE_CHANNEL := float64(OG_COLOR1.B)

                RED_CHANNEL2 := float64(OG_COLOR2.R)
                GREEN_CHANNEL2 := float64(OG_COLOR2.G)
                BLUE_CHANNEL2 := float64(OG_COLOR2.B)

                // REALIZANDO MEZCLA FADE CON LA FÓRMULA ESTABLECIDA EN EL EJERCICIO2
                RED_CHANNEL3 := uint8(FADE*RED_CHANNEL + (1-FADE)*RED_CHANNEL2)
                GREEN_CHANNEL3 := uint8(FADE*GREEN_CHANNEL + (1-FADE)*GREEN_CHANNEL2)
                BLUE_CHANNEL3 := uint8(FADE*BLUE_CHANNEL + (1-FADE)*BLUE_CHANNEL2)
                COLOR := color.RGBA{
                        R: RED_CHANNEL3, G: GREEN_CHANNEL3, B: BLUE_CHANNEL3, A: OG_COLOR1.A,
                }
                wImg.Set(ANCHO, ALTO, COLOR)
        }
}

//FINALIZAR CRONÓMETRO E IMPRIMIR EL TIEMPO
elapsed := time.Since(start)
print("Se guardó correctamente la nueva imágen en la carpeta: 'OUTPUT'")
log.Printf("\nTIEMPO | EJERCICIO2 | SECUENCIAL: %s", elapsed)

//DETERMINANDO CARPETA DONDE SE GUARDARÁ EL RESULTADO
imgOut := "src/OUT/"

//CREAR EL RESULTADO
ext := filepath.Ext(IMPORTAR_IMAGEN1)
```

```
        name := strings.TrimSuffix(filepath.Base("OUTPUT"), ext)
        newImagePath := fmt.Sprintf("%s/%s_SECUENCIAL-EJ2%s", filepath.Dir(imgOut), name, ext)
        fg, err := os.Create(newImagePath)
        defer fg.Close()
        check(err)
        err = jpeg.Encode(fg, wImg, nil)
        check(err)
}

// POR SI HAY ERRORES
func check(err error) {
        if err != nil {
                panic(err)
        }
}
```

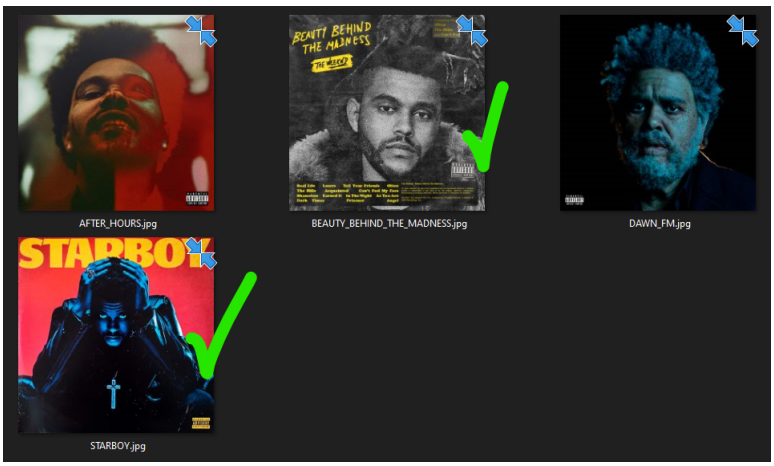
- CONCURRENCIA Y PARALELISTMO: EJECUCIÓN Y TIEMPO EN CONSOLA (ADMIN/POWERSHELL):

```
PS C:\Users\nuez_\OneDrive\Escritorio\img_process_golang> go run 2CP.go
Se guardó correctamente la nueva imágen en la carpeta: 'OUTPUT'2022/11/25 05:41:39
TIEMPO | EJERCICIO2 | GOROUTINE: 3.8578ms
PS C:\Users\nuez_\OneDrive\Escritorio\img_process_golang> go run 2CP.go
Se guardó correctamente la nueva imágen en la carpeta: 'OUTPUT'2022/11/25 05:41:57
TIEMPO | EJERCICIO2 | GOROUTINE: 4.0909ms
PS C:\Users\nuez_\OneDrive\Escritorio\img_process_golang> go run 2CP.go
Se guardó correctamente la nueva imágen en la carpeta: 'OUTPUT'2022/11/25 05:42:08
TIEMPO | EJERCICIO2 | GOROUTINE: 2.8881ms
```

- SECUENCIAL: EJECUCIÓN Y TIEMPO EN CONSOLA (ADMIN/POWERSHELL):

```
PS C:\Users\nuez_\OneDrive\Escritorio\img_process_golang> go run 2S.go
Se guardó correctamente la nueva imágen en la carpeta: 'OUTPUT'2022/11/25 05:38:28
TIEMPO | EJERCICIO2 | SECUENCIAL: 7.284ms
PS C:\Users\nuez_\OneDrive\Escritorio\img_process_golang> go run 2S.go
Se guardó correctamente la nueva imágen en la carpeta: 'OUTPUT'2022/11/25 05:39:16
TIEMPO | EJERCICIO2 | SECUENCIAL: 8.2403ms
PS C:\Users\nuez_\OneDrive\Escritorio\img_process_golang> go run 2S.go
Se guardó correctamente la nueva imágen en la carpeta: 'OUTPUT'2022/11/25 05:39:45
TIEMPO | EJERCICIO2 | SECUENCIAL: 7.1838ms
```

- INPUT DE IMÁGENES (ARCHIVOS: "BEAUTY BEHIND THE MADNESS"Y "STARBOY"):



- OUTPUT DE IMAGEN RESULTANTE:

3. Ejercicio 3

3. Obtenga el histograma de una imágen, este representa la frecuencia de intensidad de un colores de los pixeles.Como un pixel tiene tres canales de colores, se puede obtener tres histogramas como se ve en la Figura 4.

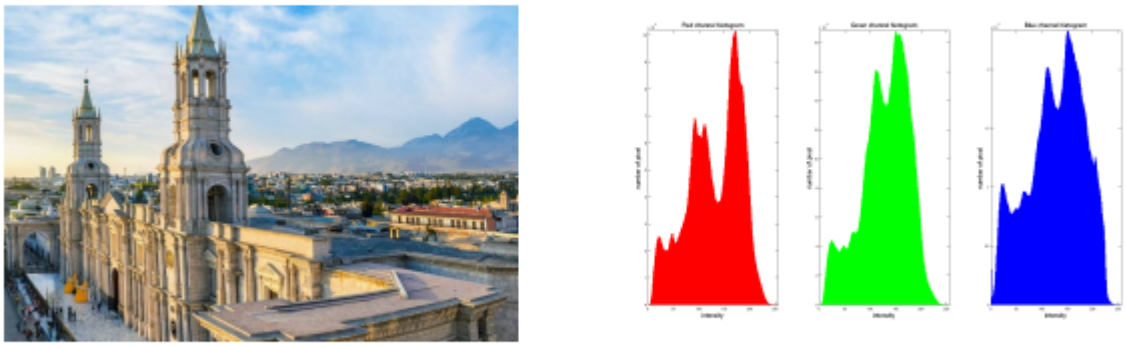


Figura 4: Ejemplo de histograma de una imagen.

- CÓDIGO EN GOLANG (CONCURRENCIA Y PARALELISMO):

```
// --------EJERCICIO3: CONCURRENCIA Y PARALELISMO--------
// IMPORTANDO LIBRERIAS NECESARIAS
package main

import (
        "fmt"
        "image"
        "image/color"
        "image/jpeg"
        "log"
        "os"
        "path/filepath"
        "strings"
        "sync"
        "time"
)

func main() {

        txt, err := os.Create("CANAL_ROJO.txt")
        if err != nil {
                fmt.Printf("error", err)
                return
        }
        txt1, err := os.Create("CANAL_VERDE.txt")
        if err != nil {
                fmt.Printf("error", err)
                return
        }
        txt2, err := os.Create("CANAL_AZUL.txt")
        if err != nil {
                fmt.Printf("error", err)
                return
        }
        //VARIABLE DEL FADE (BRILLO) ENTRE IMÁGENES / VALORES ENTRE 0 - 1 (PROBAR: 0.25 - 0.50 - 0.75)
        //SE REGISTRA LA RUTA DE LA PRIMERA IMÁGEN
        IMPORTAR_IMAGEN1 := "src/OG/ASTROWORLD.jpg"
```

```
f, err := os.Open(IMPORTAR_IMAGEN1)
check(err)

//SE CAPTURAN LOS VALORES DE LA IMAGEN
img, _, err := image.Decode(f)

TAMAÑO := img.Bounds().Size()
rect := image.Rect(0, 0, TAMAÑO.X, TAMAÑO.Y)
wImg := image.NewRGBA(rect)

//GOROUTINE
wg := new(sync.WaitGroup)

//TEMPORIZADOR: INICIAR
start := time.Now()

//CICLO QUE RECORRE TODO Y (ALTO)
for ALTO := 0; ALTO < TAMAÑO.Y; ALTO++ {

        //GOROUTINE
        wg.Add(1)
        ALTO := ALTO
        go func() {

                // CICLO QUE RECORRE TODO X (ANCHO)
                for ANCHO := 0; ANCHO < TAMAÑO.X; ANCHO++ {
                        pixel := img.At(ANCHO, ALTO)
                        OG_COLOR := color.RGBAModel.Convert(pixel).(color.RGBA)

                        // CONVIRTIENDO ANÁLISIS RGB A VALORES FLOAT (IMÁGEN 1 Y 2)
                        RED_CHANNEL := float64(OG_COLOR.R)
                        GREEN_CHANNEL := float64(OG_COLOR.G)
                        BLUE_CHANNEL := float64(OG_COLOR.B)

                        _, err = txt.WriteString(fmt.Sprintf("%d\n", uint8(RED_CHANNEL)))
                        if err != nil {
                                fmt.Printf("error", err)
                        }
                        _, err = txt1.WriteString(fmt.Sprintf("%d\n", uint8(GREEN_CHANNEL)))
                        if err != nil {
                                fmt.Printf("error", err)
                        }
                        _, err = txt2.WriteString(fmt.Sprintf("%d\n", uint8(BLUE_CHANNEL)))
                        if err != nil {
                                fmt.Printf("error", err)
                        }
                }
                defer wg.Done()
        }()
}
wg.Wait()

//FINALIZAR CRONÓMETRO E IMPRIMIR EL TIEMPO
elapsed := time.Since(start)
print("Se guardaron correctamente los txt con información de cada canal RGB en la carpeta: 'OUTPUT'")
log.Printf("\nTIEMPO | EJERCICIO3 | GOROUTINE: %s", elapsed)

//DETERMINANDO CARPETA DONDE SE GUARDARÁ EL RESULTADO
imgOut := "src/OUT/"
```

```
        //CREAR EL RESULTADO
        ext := filepath.Ext(IMPORTAR_IMAGEN1)
        name := strings.TrimSuffix(filepath.Base("OUTPUT"), ext)
        newImagePath := fmt.Sprintf("%s/%s_CONCURRENCIA-EJ3%s", filepath.Dir(imgOut), name, ext)
        fg, err := os.Create(newImagePath)
        defer fg.Close()
        check(err)
        err = jpeg.Encode(fg, wImg, nil)
        check(err)
}

// POR SI HAY ERRORES
func check(err error) {
        if err != nil {
                panic(err)
        }
}
```

- CÓDIGO EN GOLANG (SECUENCIAL):

```
// --------EJERCICIO3: SECUENCIAL--------
// IMPORTANDO LIBRERIAS NECESARIAS
package main

import (
        "fmt"
        "image"
        "image/color"
        "image/jpeg"
        "log"
        "os"
        "path/filepath"
        "strings"
        "time"
)

func main() {
        ROJO, err := os.Create("CANAL_ROJO.txt")
        if err != nil {
                fmt.Printf("error", err)
                return
        }
        VERDE, err := os.Create("CANAL_VERDE.txt")
        if err != nil {
                fmt.Printf("error", err)
                return
        }
        AZUL, err := os.Create("CANAL_AZUL.txt")
        if err != nil {
                fmt.Printf("error", err)
                return
        }
        //VARIABLE DEL FADE (BRILLO) ENTRE IMÁGENES / VALORES ENTRE 0 - 1 (PROBAR: 0.25 - 0.50 - 0.75)
        //SE REGISTRA LA RUTA DE LA PRIMERA IMÁGEN
        IMPORTAR_IMAGEN1 := "src/OG/ASTROWORLD.jpg"
        f, err := os.Open(IMPORTAR_IMAGEN1)
        check(err)

        //SE CAPTURAN LOS VALORES DE LA IMAGEN
        img, _, err := image.Decode(f)
```

```
        TAMAÑO := img.Bounds().Size()
        rect := image.Rect(0, 0, TAMAÑO.X, TAMAÑO.Y)
        wImg := image.NewRGBA(rect)

        //TEMPORIZADOR: INICIAR
        start := time.Now()

        //CICLO QUE RECORRE TODO Y (ALTO)
        for ALTO := 0; ALTO < TAMAÑO.Y; ALTO++ {
                // CICLO QUE RECORRE TODO X (ANCHO)
                for ANCHO := 0; ANCHO < TAMAÑO.X; ANCHO++ {
                        pixel := img.At(ANCHO, ALTO)
                        OG_COLOR := color.RGBAModel.Convert(pixel).(color.RGBA)

                        // CONVIRTIENDO ANÁLISIS RGB A VALORES FLOAT (IMÁGEN 1 Y 2)
                        RED_CHANNEL := float64(OG_COLOR.R)
                        GREEN_CHANNEL := float64(OG_COLOR.G)
                        BLUE_CHANNEL := float64(OG_COLOR.B)

                        _, err = ROJO.WriteString(fmt.Sprintf("%d\n", uint8(RED_CHANNEL)))
                        if err != nil {
                                fmt.Printf("error", err)
                        }
                        _, err = VERDE.WriteString(fmt.Sprintf("%d\n", uint8(GREEN_CHANNEL)))
                        if err != nil {
                                fmt.Printf("error", err)
                        }
                        _, err = AZUL.WriteString(fmt.Sprintf("%d\n", uint8(BLUE_CHANNEL)))
                        if err != nil {
                                fmt.Printf("error", err)
                        }
                }
        }

        //FINALIZAR CRONÓMETRO E IMPRIMIR EL TIEMPO
        elapsed := time.Since(start)
        print("Se guardó correctamente la nueva imágen en la carpeta: 'OUTPUT'")
        log.Printf("\nTIEMPO | EJERCICIO2 | SECUENCIAL: %s", elapsed)

        //DETERMINANDO CARPETA DONDE SE GUARDARÁ EL RESULTADO
        imgOut := "src/OUT/"

        //CREAR EL RESULTADO
        ext := filepath.Ext(IMPORTAR_IMAGEN1)
        name := strings.TrimSuffix(filepath.Base("OUTPUT"), ext)
        newImagePath := fmt.Sprintf("%s/%s_SECUENCIAL-EJ3%s", filepath.Dir(imgOut), name, ext)
        fg, err := os.Create(newImagePath)
        defer fg.Close()
        check(err)
        err = jpeg.Encode(fg, wImg, nil)
        check(err)
}

// POR SI HAY ERRORES
func check(err error) {
        if err != nil {
                panic(err)
        }
}
```

- CONCURRENCIA Y PARALELISTMO: EJECUCIÓN Y TIEMPO EN CONSOLA (ADMIN/POWERSHELL):

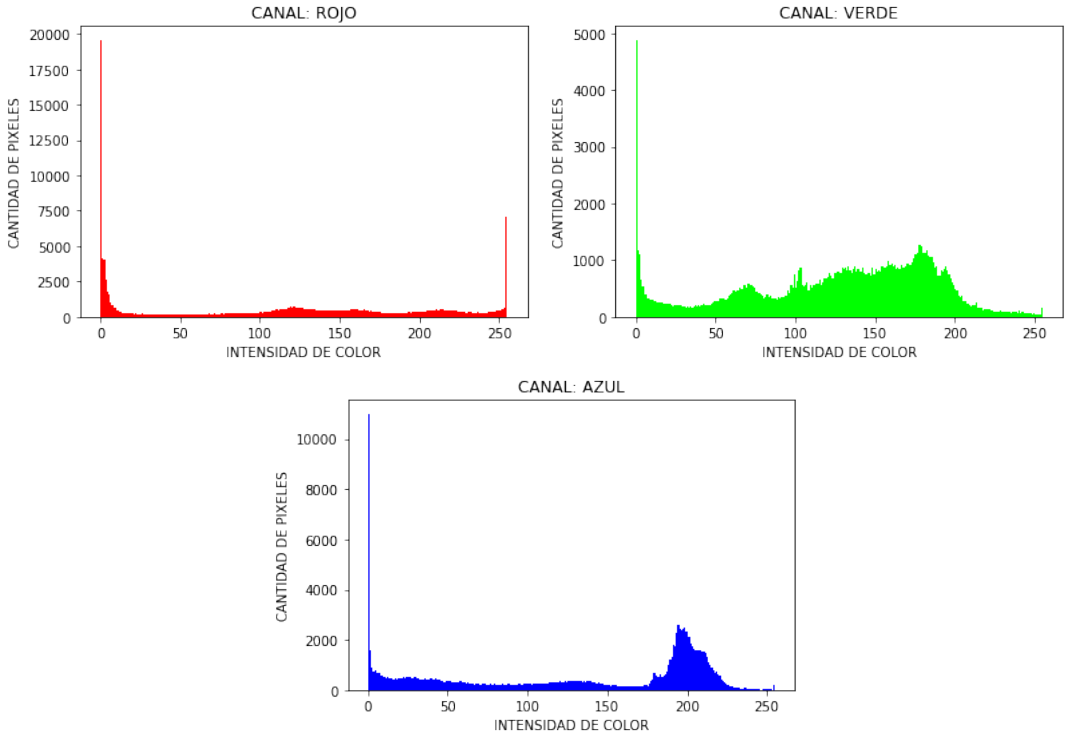
```
PS C:\Users\nuez_\OneDrive\Escritorio\img_process_golang> go run 3CP.go
Se guardaron correctamente los txt con información de cada canal RGB en la carpeta: 'OUTPUT'2022/11/25 06:39:13
TIEMPO | EJERCICIO3 | GOROUTINE: 2.4137804s
```

- SECUENCIAL: EJECUCIÓN Y TIEMPO EN CONSOLA (ADMIN/POWERSHELL):

```
PS C:\Users\nuez_\OneDrive\Escritorio\img_process_golang> go run 3CP.go
Se guardaron correctamente los txt con información de cada canal RGB en la carpeta: 'OUTPUT'2022/11/25 06:39:13
TIEMPO | EJERCICIO3 | GOROUTINE: 2.4137804s
PS C:\Users\nuez_\OneDrive\Escritorio\img_process_golang> go run 3S.go
Se guardó correctamente la nueva imágen en la carpeta: 'OUTPUT'2022/11/25 06:53:14
TIEMPO | EJERCICIO2 | SECUENCIAL: 4.1545348s
PS C:\Users\nuez_\OneDrive\Escritorio\img_process_golang>
```

- INPUT DE IMÁGEN (ARCHIVO: .ªSTROWORLD"): - OUTPUT DE IMAGEN RESULTANTE (se utilizó "Pandas",

librería de python para graficar los histogramas):



4. CONCLUSIONES

- GOLANG ES UN LENGUAJE CON EL QUE SE PUEDEN REALIZAR MÚLTIPLES FUNCIONES DE MANERA SENCILLA, ENTRE ELLAS ESTÁ PODER EDITAR IMÁGENES.
- "GOROUTINES"VUELVE MÁS EFICIENTE LA EJECUCIÓN DE FUNCIONES;PUES HACE USO DEL PARALELISMO, PERMITIENDO QUE NUESTRO CÓDIGO SEA MÁS VELOZ EN CUANTO A TIEMPO DE EJECUCIÓN.
- TENER EN CUENTA LA POTENCIA DEL PROCESADOR DEL EQUIPO CON EL QUE SE ESTÉ TRABAJANDO; NO EXCEDER LA CANTIDAD DE THREADS QUE SE ESTIMA; Y TENER EN CUENTA LOS POSIBLES FALLOS POR DEADLOCK.

GITHUB .