Trabajo Práctico 2

Organización del Computador II

Segundo Cuatrimestre de 2018

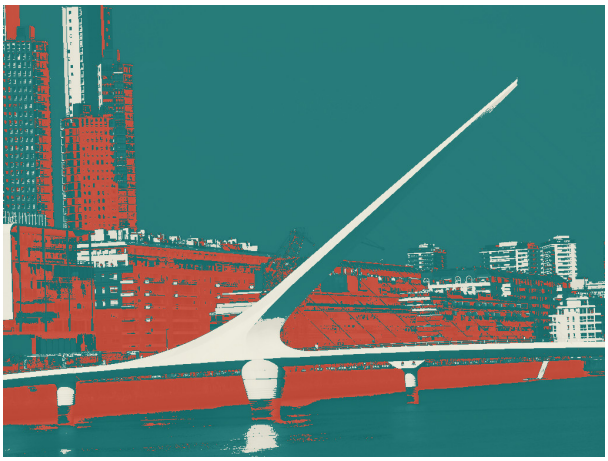
1. Introducción

En este trabajo práctico consiste en implementar filtros gráficos utilizando el modelo de procesamiento SIMD. El mismo consta en dos componentes igualmente importantes. En primer lugar aplicaremos lo estudiado en clase programando de manera vectorizada un conjunto de filtros. ~~Luego, considerando las características de microarquitectura del procesador se pedirá realizar un análisis experimental del funcionamiento de los filtros.~~ Estos serán codificados en lenguaje ensamblador ~~y lenguaje C~~.

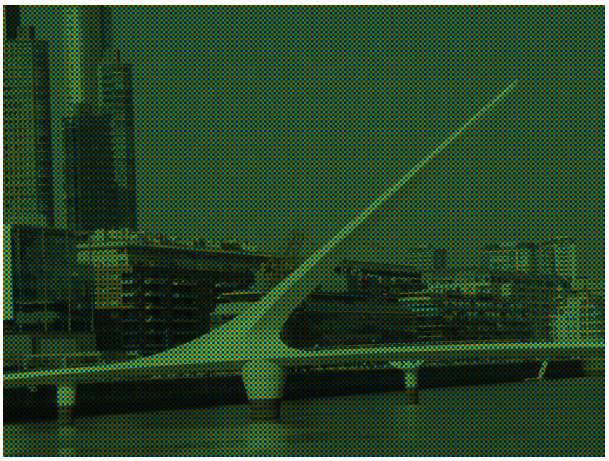
~~La etapa de experimentación se deberá realizar con un carácter científico y una metodología clara. Resulta fundamental entonces, la rigurosidad y exhaustividad del análisis que realicen.~~

2. Filtros

Imagen original

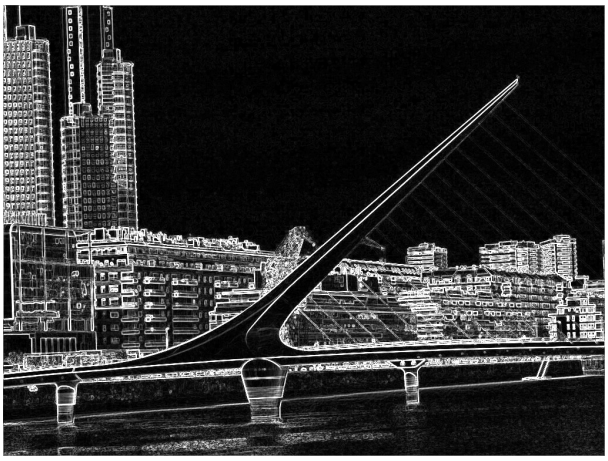


Tres Colores



Efecto Bayer

Cambia Color



Edge Sobel

2.1. Preliminares

Consideramos a una imagen como una matriz de píxeles. Cada píxel está determinado por cuatro componentes: los colores azul (b), verde (g) y rojo (r), y la transparencia (a). En nuestro caso particular cada una de estas componentes tendrá 8 bits (1 byte) de profundidad, es decir que estarán representadas por números enteros en el rango $[0, 255]$.

Dada una imagen src, notaremos $\mathsf{src}^k_{[i,j]}$ al valor de la componente $k \in \{\mathsf{r}, \mathsf{g}, \mathsf{b}, \mathsf{a}\}$ del píxel en la fila i y la columna j de la imagen. La fila 0 corresponde a la fila de más abajo de la imagen. La columna 0 a la de más a la izquierda.

Llamaremos dst a la imagen de salida generada por cada filtro. Por ejemplo, el filtro identidad estaría caracterizado por la fórmula

$$\forall k \in \{\mathsf{r}, \mathsf{g}, \mathsf{b}, \mathsf{a}\} \quad \mathsf{dst}^k_{[i,j]} = \mathsf{src}^k_{[i,j]}.$$

2.2. Tres Colores

Este filtro opera independientemente sobre cada píxel cambiando su color según el brillo del mismo. El píxel resultante será una combinación lineal entre el nuevo color y del brillo calculado. El brillo se calcula como el promedio de los componentes de cada píxel.

$$\mathsf{brillo}_{[i,j]} = \lfloor (\mathsf{src}^r_{[i,j]} + \mathsf{src}^g_{[i,j]} + \mathsf{src}^b_{[i,j]})/3 \rfloor$$

En función de este valor se debe determinar el color de la imagen destino entre tres colores posibles.

	R	G	B
CREMA	236	233	214
VERDE	0	112	110
ROJO	244	88	65

La combinación lineal para cada píxel estará dada por $\frac{3}{4}$ del nuevo color y $\frac{1}{4}$ de la componente de brillo.

$$\mathsf{dst}_{[i,j]\{r,g,b\}} = \begin{cases} \{\frac{3}{4} \cdot \texttt{0xec} + \frac{1}{4} \cdot W, \frac{3}{4} \cdot \texttt{0xe9} + \frac{1}{4} \cdot W, \frac{3}{4} \cdot \texttt{0xd6} + \frac{1}{4} \cdot W\} & \text{si } W \leq 32 \\ \{\frac{3}{4} \cdot \texttt{0x00} + \frac{1}{4} \cdot W, \frac{3}{4} \cdot \texttt{0x70} + \frac{1}{4} \cdot W, \frac{3}{4} \cdot \texttt{0x6e} + \frac{1}{4} \cdot W\} & \text{si } 32 < W \leq 170 \\ \{\frac{3}{4} \cdot \texttt{0xf4} + \frac{1}{4} \cdot W, \frac{3}{4} \cdot \texttt{0x58} + \frac{1}{4} \cdot W, \frac{3}{4} \cdot \texttt{0x41} + \frac{1}{4} \cdot W\} & \text{si } 170 < W \end{cases}$$

donde W corresponde al valor de brillo $\mathsf{brillo}_{[i,j]}$.

2.3. Efecto Bayer

Cada píxel en una imagen esta, formado por tres componentes de color, rojo, verde y azul. Este filtro construye un patrón de 4×4 píxeles sobre toda la imagen, donde cada conjunto de píxeles representa solamente uno de los colores. El Patrón respetará el siguiente orden, comenzando desde la esquina superior izquierda:

Rojo	Verde	
Verde	Azul	

Luego, formula a aplicar será:

$$\mathsf{dst}_{[i,j]\{r,g,b\}} = \begin{cases} \{\mathsf{src}^r_{[i,j]}, 0, 0\} & \text{si } i \equiv 0 \text{ o } 1 \text{ o } 2 \text{ o } 3 \mod 8 \text{ y } j \equiv 0 \text{ o } 1 \text{ o } 2 \text{ o } 3 \mod 8 \\ \{0, \mathsf{src}^g_{[i,j]}, 0\} & \text{si } i \equiv 0 \text{ o } 1 \text{ o } 2 \text{ o } 3 \mod 8 \text{ y } j \equiv 4 \text{ o } 5 \text{ o } 6 \text{ o } 7 \mod 8 \\ \{0, \mathsf{src}^g_{[i,j]}, 0\} & \text{si } i \equiv 4 \text{ o } 5 \text{ o } 6 \text{ o } 7 \mod 8 \text{ y } j \equiv 0 \text{ o } 1 \text{ o } 2 \text{ o } 3 \mod 8 \\ \{0, 0, \mathsf{src}^b_{[i,j]}\} & \text{si } i \equiv 4 \text{ o } 5 \text{ o } 6 \text{ o } 7 \mod 8 \text{ y } j \equiv 4 \text{ o } 5 \text{ o } 6 \text{ o } 7 \mod 8 \end{cases}$$

2.4. Cambia Color

Para cambiar un color por otro, se debe considerar la noción de distancia entre colores. Si el color objetivo se encuentra próximo al color por intercambiar, entonces estos son intercambiados.

Calcular la distancia entre colores es muy costo computacionalmente. En este caso consideraremos una aproximación:

$$d = \sqrt{2 \times \Delta R^2 + 4 \times \Delta G^2 + 3 \times \Delta B^2 + \frac{\bar{r} \times (\Delta R^2 - \Delta B^2)}{256}}$$

Donde:

$$\begin{aligned} \bar{r} &= \tfrac{C_{1,R}+C_{2,R}}{2} \\ \Delta R &= C_{1,R} - C_{2,R} \\ \Delta G &= C_{1,G} - C_{2,G} \\ \Delta B &= C_{1,B} - C_{2,B} \end{aligned}$$

Referencia: https://en.wikipedia.org/wiki/Color_difference

Este filtro calcula para cada píxel la distancia al color pasado por parámetro, si esta distancia se encuentra por debajo de un umbral dado, entonces se intercambian los colores, por medio de una combinación lineal.

$$\mathsf{dst}_{[i,j]\{r,g,b\}} = \begin{cases} \{\mathsf{src}^r_{[i,j]}, \mathsf{src}^g_{[i,j]}, \mathsf{src}^b_{[i,j]}\} & \text{si } d \geq lim \\ \{\mathsf{N}^r \cdot (1-c) + \mathsf{src}^r_{[i,j]} \cdot c, \mathsf{N}^g \cdot (1-c) + \mathsf{src}^g_{[i,j]} \cdot c, \mathsf{N}^b \cdot (1-c) + \mathsf{src}^b_{[i,j]}\} \cdot c & \text{si no} \end{cases}$$

Donde:

$$\begin{aligned} d &= \text{distancia entre color parámetro y } \mathsf{src} \\ \{\mathsf{N}^r, \mathsf{N}^g, \mathsf{N}^b\} &= \text{nuevo color} \\ c &= d^2/lim^2 \end{aligned}$$

2.5. Edge Sobel

Este es un filtro de detección de bordes, su funcionamiento esta basado en detectar cambios entre píxeles. Si consideramos una fila de píxeles en una imagen como muestras de una señal, un cambio abrupto en la pendiente determinaría un borde. De esta forma se utilizan dos operadores, uno que detecta los cambios en el sentido X y con el otro los cambios en Y.

Operador X

-1	0	+1
-2	0	+2
-1	0	+1

Operador Y

-1	-2	-1
0	0	0
+1	+2	+1

Para este filtro utilizaremos imágenes en escala de grises, obtenidas a partir del calculo del canal de brillo. Los operadores se utilizan como factores de multiplicación centrados sobre cada píxel en la imagen:

$$\mathsf{OP}_{\mathsf{z}[i,j]} = \left\{ \begin{array}{lcllcll} \mathrm{M}_{z[0,0]} \cdot \mathsf{src}_{[i-1,j-1]} & + & \mathrm{M}_{z[0,1]} \cdot \mathsf{src}_{[i-1,j]} & + & \mathrm{M}_{z[0,2]} \cdot \mathsf{src}_{[i-1,j+1]} & + \\ \mathrm{M}_{z[1,0]} \cdot \mathsf{src}_{[i,j-1]} & + & \mathrm{M}_{z[1,1]} \cdot \mathsf{src}_{[i,j]} & + & \mathrm{M}_{z[1,2]} \cdot \mathsf{src}_{[i,j+1]} & + \\ \mathrm{M}_{z[2,0]} \cdot \mathsf{src}_{[i+1,j-1]} & + & \mathrm{M}_{z[2,1]} \cdot \mathsf{src}_{[i+1,j]} & + & \mathrm{M}_{z[2,2]} \cdot \mathsf{src}_{[i+1,j+1]} & \end{array} \right\}$$

Donde:

$$\mathrm{M}_{z[i,j]} = \text{es el operador } z.$$
$$\mathsf{OP}_{\mathsf{z}[i,j]} = \text{el resultado de aplicar un operador } z.$$

La operación de este filtro consiste en aplicar para cada píxel el cálculo de ambos operadores:

$$\mathsf{dst}_{[i,j]} = \left\{ \begin{array}{ll} 255 & \text{si } \left|\mathsf{OP}_{\mathsf{x}[i,j]}\right| + \left|\mathsf{OP}_{\mathsf{y}[i,j]}\right| > 255 \\ \left|\mathsf{OP}_{\mathsf{x}[i,j]}\right| + \left|\mathsf{OP}_{\mathsf{y}[i,j]}\right| & \text{si no} \end{array} \right.$$

Los bordes de un píxel de la imagen deben estar completados con ceros.

3. Implementación

Para facilitar el desarrollo del trabajo práctico se cuenta con un *framework* que provee todo lo necesario para poder leer y escribir imágenes, así como también compilar y probar las funciones que vayan a implementar.

3.1. Archivos y uso

Dentro de los archivos presentados deben completar el código de las funciones pedidas. Puntualmente encontrarán el programa principal (de línea de comandos), denominado **tp2**, que se ocupa de parsear las opciones ingresadas por el usuario y ejecutar el filtro seleccionado sobre la imagen ingresada.
Los archivos entregados están organizados en las siguientes carpetas:

- documentos: Contiene este enunciado y un *template* de informe en LaTeX.
- codigo: Contiene el código fuente, junto con el framework de ejecución y testeo. Contiene los fuentes del programa principal, junto con el **Makefile** que permite compilarlo. Además contiene los siguientes subdirectorios:

- build: Contiene los archivos objeto y ejecutables del TP.
- filtros: Contiene las implementaciones de los filtros
- helper: Contiene los fuentes de la biblioteca BMP y de la herramienta de comparación de imágenes.
- img: Algunas imágenes de prueba.
- test: Contiene scripts para realizar tests sobre los filtros y uso de la memoria.

Compilación

Ejecutar `make` desde la carpeta `codigo`. Recordar que cada directorio tiene su propio Makefile, por lo que si se desea cambiar las opciones de compilación debe buscarse el Makefile correspondiente.

Uso

El uso del programa principal es el siguiente:

```
$ ./tp2 <nombre_filtro> <opciones> <nombre_archivo_entrada> [parámetros...]
```

Los filtros que se pueden aplicar y sus parámetros son los especificados en la sección 2.
Las opciones que acepta el programa son las siguientes:

- `-h, --help`
  Imprime la ayuda.
- `-i, --implementacion NOMBRE_MODO`
  Implementación sobre la que se ejecutará el proceso seleccionado. Los implementaciones disponibles son: `c`, `asm`.
- `-t, --tiempo CANT_ITERACIONES`
  Mide el tiempo que tarda en ejecutar el filtro sobre la imagen de entrada una cantidad de veces igual a `CANT_ITERACIONES`.
- `-o, --output DIRECTORIO`
  Genera el resultado en `DIRECTORIO`. De no incluirse, el resultado se guarda en el mismo directorio que el archivo fuente.
- `-v, --verbose`
  Imprime información adicional.

3.2. Código de los filtros

Para implementar los filtros descriptos anteriormente, tanto en C como en ASM se deberán implementar las funciones especificadas en la sección 2. Las imágenes se almacenan en memoria en color, en el orden B (blue), G (green), R (red), A (alpha).

Los parámetros genéricos de las funciones son:

- src : Es el puntero al inicio de la matriz de elementos de 32 bits sin signo (el primer byte corresponde al canal azul de la imagen (B), el segundo el verde (G), el tercero el rojo (R)), y el cuarto el alpha (A) que representa a la imagen de entrada. Es decir, como la imagen está en color, cada píxel está compuesto por 4 bytes.

- dst : Es el puntero al inicio de la matriz de elementos de 32 bits sin signo que representa a la imagen de salida.
- filas : Representa el alto en píxeles de la imagen, es decir, la cantidad de filas de las matrices de entrada y salida.
- cols : Representa el ancho en píxeles de la imagen, es decir, la cantidad de columnas de las matrices de entrada y salida.
- src_row_size : Representa el ancho en bytes de cada fila de la imagen incluyendo el padding en caso de que hubiere. Es decir, la cantidad de bytes que hay que avanzar para moverse a la misma columna de fila siguiente o anterior.

Consideraciones

Las funciones a implementar en lenguaje ensamblador deben utilizar el set de instrucciones **SSE**, a fin de optimizar la performance de las mismas.

Tener en cuenta lo siguiente:

- El ancho de las imágenes es siempre mayor a 16 píxeles y múltiplo de 8 píxeles.
- No se debe perder precisión en ninguno de los cálculos, a menos que se indique lo contrario.
- La implementación de cada filtro deberá estar optimizada para el filtro que se está implementando. No se puede hacer una función que aplique un filtro genérico y después usarla para implementar los que se piden.
- El procesamiento de los píxeles se deberá hacer **exclusivamente** con instrucciones **SSE**. No está permitido procesarlos con registros de propósito general, salvo para tratatimiento de casos borde. En tal caso se deberá justificarse debidamente y hacer un análisis del costo computacional.
- El TP se tiene que poder ejecutar en las máquinas del laboratorio.

3.3. Formato BMP

El formato BMP es uno de los formatos de imágenes más simples: tiene un encabezado y un mapa de bits que representa la información de los píxeles.

En este trabajo práctico se utilizará una biblioteca provista por la cátedra para operar con archivos en ese formato. Si bien esta biblioteca no permite operar con archivos con paleta, es posible leer tres tipos de formatos, tanto con o sin transparencia. Ambos formatos corresponden a los tipos de encabezado: BITMAPINFOHEADER (40 bytes), BITMAPV3INFOHEADER (56 bytes) y BITMAPV5HEADER (124 bytes).

El código fuente de la biblioteca está disponible como parte del material, deben seguirlo y entenderlo. Las funciones que deben implementar reciben como entrada un puntero a la imagen. Este puntero corresponde al mapa de bits almacenado en el archivo. El mismo está almacenado de forma particular: **las líneas de la imagen se encuentran almacenadas de forma invertida**. Es decir, en la primera fila de la matriz se encuentra la última línea de la imagen, en la segunda fila se encuentra la anteúltima y así sucesivamente. Dentro de cada línea los píxeles se almacenan de izquierda a derecha, y cada píxel **en memoria se guarda en el siguiente orden: B, G, R, A**.

3.4. Herramientas y tests

En el código provisto, podrán encontrar varias herramientas que permiten verificar si los filtros funcionan correctamente.

Diff

La herramienta diff permite comparar dos imágenes. El código de la misma se encuentra en `helper`, y se compila junto con el resto del trabajo práctico. El ejecutable, una vez compilado, se almacenará en `build/bmpdiff`.

La aplicación se utiliza desde linea de comandos de la forma:

```
$ ./build/bmpdiff [opciones] <archivo_1> <archivo_2> <epsilon>
```

Compara los dos archivos según las componentes de cada píxel, siendo `epsilon` la diferencia máxima permitida entre píxeles correspondientes de las dos imágenes. Tiene dos opciones: listar las diferencias o generar imágenes blanco y negro por cada componente, donde blanco es marca que hay diferencia y negro que no.

Las opciones soportadas por el programa son:

`-i`, `--image`	Genera imágenes de diferencias por cada componente.
`-v`, `--verbose`	Lista las diferencias de cada componente y su posición en la imagen.
`-a`, `--value`	Genera las imágenes mostrando el valor de la diferencia.
`-s`, `--summary`	Muestra un resumen de diferencias.

Tests

Para verificar el correcto funcionamiento de los filtros, además del comparador de imágenes, se provee un binario con la solución de la cátedra y varios scripts de test. El binario de la cátedra se encuentra en la carpeta test/.

El comparador de imágenes se ubica en la carpeta src/helper, y debe compilarse antes de correr los scripts.

Para ejecutar los script, en primer lugar, se deben generar las imagenes de prueba. Esto se realiza ejecutando el script `1_generar_imagenes.py`. Para que este script funcione correctamente se requiere la utilidad convert que se encuentra en la biblioteca imagemagick.[1].

Luego, el script `2_test_diff_cat_asm.py`, chequea diferencias entre las imagenes generadas por el binario de la catedria y su binario. Este script prueba por diferencias el resultado final solamente.

Por último, el script `3_correr_test_mem.sh`, chequea el correcto uso de la memoria. Al demorar mucho tiempo, este test se ejecuta solamente para los casos más chicos.

3.5. Mediciones de rendimiento

La forma de medir el rendimiento de nuestras implementaciones se realizará por medio de la toma de tiempos de ejecución. Como los tiempos de ejecución son muy pequeños, se utilizará uno de los contadores de performance que posee el procesador.

La instrucción de assembler `rdtsc` permite obtener el valor del Time Stamp Counter (TSC) del procesador. Este registro se incrementa en uno con cada ciclo del procesador. Obteniendo la

[1]Para instalar sudo apt-get install imagemagick

diferencia entre los contadores antes y después de la llamada a la función, podemos obtener la cantidad de ciclos de esa ejecución. Esta cantidad de ciclos no es siempre igual entre invocaciones de la función, ya que este registro es global del procesador y se ve afectado por una serie de factores.

Existen principalmente distintas problemáticas a solucionar:

a) La ejecución puede ser interrumpida por el *scheduler* para realizar un cambio de contexto, esto implicará contar muchos más ciclos (*outliers*) que si nuestra función se ejecutara sin interrupciones.

b) Los procesadores modernos varían su frecuencia de reloj, por lo que la forma de medir ciclos cambiará dependiendo del estado del procesador.

c) El comienzo y fin de la medición deben realizarse con la suficiente exactitud como para que se mida solamente la ejecución de los filtros, sin ser afectada por ruidos como la carga o el guardado de las imágenes.

Para medir tiempos deberán idear e implementar una metodología que les permita evitar estos tres problemas. En el archivo `tp2.c` se provee código para realizar una medición de tiempo básica. El mismo podrá ser modificado para mejorar y automatizar las mediciones.

4. Ejercicios

Se deberá implementar el código de los filtros, realizar un análisis de su performance y presentar un informe de los resultados.

4.1. Implementación

Deberán implementar todos los filtros mencionados en lenguaje ensamblador, utilizando instrucciones SSE y respetando los siguientes píxeles mínimos simultáneos:

- **Tres Colores**: 4 píxeles simultáneos.
- **Efecto Bayer**: 4 píxeles simultáneos.
- **Cambia Color**: 2 píxeles simultáneos.
- **Edge Sobel**: 4 píxeles simultáneos.

4.2. Informe

El informe debe incluir las siguientes secciones:

a) **Carátula** Contiene

   - número / nombre del grupo
   - nombre y apellido de cada integrante
   - número de libreta y mail de cada integrante

b) **Introducción** Describe lo realizado en el trabajo práctico.

c) **Desarrollo**
Describe cada una de las funciones que implementaron. Para la descripción de cada función deberán decir cómo opera una iteración del ciclo de la función. Es decir, cómo mueven los datos a los registros, cómo los reordenan para procesarlos, las operaciones que se aplican a los datos, etc. Además se agregará un detalle más profundo de las secciones de código que consideren más importantes. Para esto pueden utilizar pseudocódigo, diagramas (mostrando gráficamente el contenido de los registros `XMM`) o cualquier otro recurso que le sea útil para describir la adaptación del algoritmo al procesamiento simultáneo SIMD. No se deberá incluir el código assembler de las funciones (aunque se pueden incluir extractos en donde haga falta).

d) **Resultados**
~~Deberán **analizar** y **comparar** las implementaciones de las funciones en su versión **C** y **assembler** y mostrar los resultados obtenidos a través de tablas y gráficos. Para esto deberán plantear experimentos que les permitan comprobar las diferencias de performance e hipotetizar sobre sus causas.~~

~~Deberán además explicar detalladamente los resultados obtenidos y analizarlos. En el caso de encontrar anomalías o comportamientos no esperados deberán construir nuevos experimentos para entender qué es lo que sucede.~~

~~Utilizar como guía para la realización de experimentos las preguntas de la sección anterior. Al responder estas preguntas (y otras que vayan surgiendo), se deberán analizar y comparar las implementaciones de cada función en su versión `C` y `ASM`, mostrando los resultados obtenidos a través de tablas y gráficos. También se deberá *comentar* los resultados obtenidos.~~

Deberan entregar comparaciones de tiempo entre la versión C proporcionada por la cátedra, compilada con máximas optimizaciones (o3) y la versión ASM.

e) **Conclusión** Reflexión final sobre los alcances del trabajo práctico, la programación vectorial a bajo nivel, problemáticas encontradas, y todo lo que consideren pertinente.

~~**Importante**: El informe se evalúa de manera independiente del código. Puede reprobarse el informe, y en tal caso deberá ser reentregado para aprobar el trabajo práctico.~~

Se evaluará solamente el código entregado y su correcto funcionamiento.

5. Entrega y condiciones de aprobación

El presente trabajo es de carácter **grupal**, siendo los grupos de **3 personas**, pudiendo ser de 2 personas en casos excepcionales previa consulta y confirmación del cuerpo docente.

La fecha de entrega de este trabajo es **Jueves 4/10**. Deberá ser entregado a través de un repositorio GIT almacenado en `https://git.exactas.uba.ar` respetando el protocolo enviado por mail y publicado en la página web de la materia. La fecha límite para la **reentrega** es el día **Martes 06/11.**

Ante cualquier problema con la entrega, comunicarse por mail a la **lista de docentes**.