# Proyecto de Matemática Discreta II-2018

## Contents

# 1 Introducción

El proyecto puede ser hecho en forma individual o en grupos de hasta 3 personas. Deberán implementar lo que se detalla a continuación en C (C99, i.e., pueden usar comentarios // u otras cosas de C99).

## 1.1 Restricciones generales

El código debe ser razonablemente portable. Ademas:

1) No pueden usar getline
2) No pueden usar archivos llamados aux.c o aux.h
3) No pueden tener archivos tales que la unica diferencia en su nombre sea diferencia en la capitalización.
4) No pueden llamar a ninguna libreria que no sean las del estandard de C99.
5) No pueden tener archivos ni directorios que tengan un espacio en el nombre.

Pueden consultar con otros grupos, pero no pueden copiar grandes fragmentos de código y mucho menos hacer eso e intentar engañarnos haciendo cambios cosméticos en el código.

# 2 Tipos de datos

## 2.1 u32:

Se utilizará el tipo de dato `u32` para especificar un entero de 32 bits sin signo.

Todos los enteros sin signo de 32 bits que aparezcan en la implementación deberán usar este tipo de dato.

Los grafos a colorear tendran una lista de lados cuyos vertices seran todos u32.

## 2.2 GrafoSt

Es una estructura, la cual debe contener toda la información sobre el grafo necesaria para correr su implementación. La definición interna de esta estructura es a elección de ustedes y deberá soportar los métodos que se describirán más adelante, más los métodos que ustedes consideren necesarios para implementar los algoritmos que esten implementando.

Entre los parametros debe haber como mínimo los necesarios para guardar los datos de un grafo (vertices y lados) pero ademas los necesarios para guardar el coloreo que se tiene hasta ese momento en el grafo y cualquier información requerida en los algoritmos a implementar.

**IMPORTANTE: Cómo usaremos un algoritmo similar a Greedy, el cual usa un orden de los vertices, en esta estructura tiene que estar guardado algún orden de los vertices, y cómo vamos a cambiar ese orden repetidamente, debe ser algo que pueda ser cambiado.**

El coloreo siempre debe cumplir que si es un coloreo con $j$ colores **entonces los colores son** $1, 2, ..., j$**.**

## 2.3 Grafo

es un puntero a una estructura de datos *GrafoSt*.

# 3 Funciones De Construcción/Destrucción del grafo

## 3.1 ConstruccionDelGrafo()

Prototipo de función:

```
Grafo ConstruccionDelGrafo();
```

La función aloca memoria, inicializa lo que haya que inicializar de una estructura GrafoSt ,lee un grafo **desde standard input** en el formato indicado abajo, lo carga en la estructura y devuelve un puntero a ésta.

**WARNING: El año pasado hubo estudiantes que parecían no saber qué es stdin. Mas allá que esto es una falla en lo que deberían haber aprendido o les deberían haber enseñado, si no saben, existe algo muy útil llamado Google**

Como se explicó en 2.2, G debe tener un orden interno de los vértices, asi que en particular esta función debe dejar a G con algún tipo de orden interno de los vértices.

Ademas de cargar el grafo, **debe colorear todos los vertices con algún coloreo propio**, de alguna forma que prefieran.

Es decir, se asume que en todo momento luego de la carga del grafo, el grafo tiene algún coloreo propio (salvo durante las corridas de los dos algoritmos de coloreo que se detallan en la sección 6)

En caso de error, la función devolverá un puntero a `NULL`. (errores posibles pueden ser falla en alocar memoria, pero también que el formato de entrada no sea válido)

### 3.1.1 Formato de Entrada

El formato de entrada será una variación de DIMACS, que es un formato estandard para representar grafos, con algunos cambios.

La descripción **oficial** de DIMACS es asi:

- Ninguna linea tiene mas de 80 caracteres. PERO hemos visto archivos DIMACS en la web que NO cumplen esta especificación, asi que NO la pediremos.

  Su código debe poder procesar lineas con una cantidad arbitraria de caracteres.

- Al principio habrá cero o mas lineas que empiezan con c las cuales son lineas de comentario y deben ignorarse.

- Luego hay una linea de la forma:

  p edge n m

  donde n y m son dos enteros. Entre n y m puede haber una cantidad arbitraria de espacios en blancos, e igualmente luego de m.

  El primero número (n) en teoría representa el número de vértices y el segundo (m) el número de lados, pero hay ejemplos en la web en donde n es en realidad solo una COTA SUPERIOR del número de vertices y m una cota superior del número de lados. Sin embargo, **todos los grafos que nosotros usaremos para testear cumplirán que n será el número de vertices exacto y m el número de lados exacto**, asi que ustedes pueden asumir eso en su programa, pero si lo testean con algún grafo de la web, pueden tener problemas. De todos modos, en mi página hay muchos ejemplos bien formateados.

- Luego siguen m lineas todas comenzando con e y dos enteros, representando un lado. Es decir, lineas de la forma:

  e v w

  (Entre "v" y "w", asi como luego de "w", puede haber una cantidad arbitraria de espacios en blanco)

  Luego de esas m lineas deben **detener la carga sin leer ninguna otra linea**, aún si hay mas lineas. Estas lineas extras pueden tener una forma arbitraria, pueden o no ser comentarios, o extra lados, etc. y deben ser ignoradas. Pueden, por ejemplo, tener un SEGUNDO grafo, para que si esta función se llama dos veces por algún programa, el programa cargue dos grafos.

  Por otro lado, el archivo puede efectivamente terminar en la última de esas lineas, y su código debe poder procesar estos archivos también. (en particular en otros años hubo grupos que hicieron código que no funcionaba si la ultima linea no era una linea en blanco. Esto es un error.)

  **Nota:** Para testear usaremos tanto entrada manual desde stdin (con grafos chicos) como redireccionamiento desde stdin hacia un archivo con un grafo grande, usando el operador "<". En el primer caso puede suceder que deban usar Ctrl-D para que se termine la carga. Esto es aceptable.

  Ejemplo tomado de la web:

  c FILE: myciel3.col

  c SOURCE: Michael Trick (trick@cmu.edu)

  c DESCRIPTION: Graph based on Mycielski transformation.

  c Triangle free (clique number 2) but increasing

  c coloring number

  p edge 11 20

  e 1 2

  e 1 4

  e 1 7

  e 1 9

  e 2 3

  e 2 6

  e 2 8

  e 3 5

  e 3 7

  e 3 10

e 4 5

e 4 6

e 4 10

e 5 8

e 5 9

e 6 11

e 7 11

e 8 11

e 9 11

e 10 11

- En algunos archivos que figuran en la web, en la lista pueden aparecer tanto un lado de la forma

  e 7 9

  como el

  e 9 7

  Los grafos que usaremos nosotros **no son asi**.

  Es decir, ustedes pueden asumir en su programa que si aparece el lado e v w NO aparecerá el lado e w v.

- Nunca fijaremos $m = 0$, es decir, siempre habrá al menos un lado. (y por lo tanto, al menos dos vértices).

- En el formato DIMACS no parece estar especificado si hay algun limite para los enteros, pero en nuestro caso los limitaremos a enteros de 32 bits sin signo.

- Observar que en el ejemplo y en muchos otros casos en la web los vertices son 1,2,...,n, PERO ESO NO SIEMPRE SERÁ ASI.

  Que un grafo tenga el vértice $v$ no implicará que el grafo tenga el vértice $v'$ con $v' < v$.

  Por ejemplo, los vertices pueden ser solo cinco, y ser $0, 1, 10, 15768, 1000000$. Ustedes deben pensar bien como van a resolver este problema.

- El orden de los lados no tiene porqué ser en orden ascendente de los vertices.

  Ejemplo Válido:

  c vertices no consecutivos

  p edge 5 3

  e 1 10

  e 0 15768

  e 1000000 1

### 3.2 DestruccionDelGrafo()

Prototipo de función:

```
void DestruccionDelGrafo(Grafo G);
```

Destruye G y libera la memoria alocada.

## 4 Funciones para extraer información de datos del grafo

### 4.1 NumeroDeVertices()

Prototipo de función:

```
u32 NumeroDeVertices(Grafo G);
```

Devuelve el número de vértices de G.

### 4.2 NumeroDeLados()

Prototipo de función:

`u32 NumeroDeLados(Grafo G);`

Devuelve el número de lados de G.

### 4.3 NumeroDeColores()

Prototipo de función:

`u32 NumeroDeColores(Grafo G);`

Devuelve la cantidad de colores usados en el coloreo que tiene en ese momento G. (como dijimos en la sección 3.1, G siempre debe tener algún coloreo propio, salvo en el medio de la corrida de los dos algoritmos de la sección 6).

## 5 Funciones para extraer información de los vértices

### 5.1 NombreDelVertice()

Prototipo de Función:

`u32 NombreDelVertice(Grafo G, u32 i);`

Devuelve el nombre real del vértice número $i$ en el orden guardado en ese momento en G. (el índice 0 indica el primer vértice, el índice 1 el segundo, etc)

### 5.2 ColorDelVertice()

Prototipo de Función:

`u32 ColorDelVertice(Grafo G, u32 i);`

Devuelve el color con el que está coloreado el vértice número $i$ en el orden guardado en ese momento en G. (el índice 0 indica el primer vértice, el índice 1 el segundo, etc)

### 5.3 GradoDelVertice()

Prototipo de Función:

`u32 GradoDelVertice(Grafo G, u32 i);`

Devuelve el grado del vértice número $i$ en el orden guardado en ese momento en G. (el índice 0 indica el primer vértice, el índice 1 el segundo, etc)

### 5.4 ColorJotaesimoVecino()

Prototipo de función:

`u32 ColorJotaesimoVecino(Grafo G, u32 i,u32 j);`

Devuelve el color del vécino numero $j$ (en el orden en que lo tengan guardado uds. en G, con el índice 0 indicando el primer vécino, el índice 1 el segundo, etc)) del vértice número $i$ en el orden guardado en ese momento en G. (el índice 0 indica el primer vértice, el índice 1 el segundo, etc)

Como en las otras funciones, el orden de LOS VERTICES guardado en G es el orden en que supuestamente vamos a correr NotSoGreedy, pero el orden DE LOS VECINOS que usen ustedes es irrelevante, lo importante es que de esta forma podemos iterar sobre los vecinos para realizar tests. (por ejemplo, para saber si el coloreo es o no propio).

Para que quede claro: si el orden en el que vamos a correr NotSoGreedy es 4,10,7,15,100 y los vecinos de 7 son 10,100,4,15 (en ese orden) y los de 15 son 4,7,100 (en ese orden) y el color de 4 es 2, el de 7 es 5, el de 10 es 3, el de 15 es 4 y el de 100 es 1, entonces:

- ColorJotaesimoVecino(G,2,0)=3 (pues 10 es el primer vécino de 7)
- ColorJotaesimoVecino(G,2,2)=2 (pues 4 es el tercer vécino de 7)
- ColorJotaesimoVecino(G,3,0)=2 (pues 4 es el primer vécino de 15)
- ColorJotaesimoVecino(G,3,1)=5 (pues 7 es el segundo vécino de 15)

### 5.5 NombreJotaesimoVecino()

Prototipo de función:

`u32 NombreJotaesimoVecino(Grafo G, u32 i,u32 j);`

Devuelve el nombre del vécino numero $j$ (en el orden en que lo tengan guardado uds. en G, con el índice 0 indicando el primer vécino, el índice 1 el segundo, etc)) del vértice número $i$ en el orden guardado en ese momento en G. (el índice 0 indica el primer vértice, el índice 1 el segundo, etc)

### 5.6 GradoJotaesimoVecino()

Podriamos definir una función similar a las dos anteriores pero para grado. Sin embargo, no se preveen situaciónes de uso asi que no es requerida.

## 6 Funciones de coloreo

### 6.1 NotSoGreedy()

Prototipo de función:

`u32 NotSoGreedy(Grafo G,u32 semilla);`

Corre la variación de greedy que se indica mas abajo en G con el orden interno que debe estar guardado de alguna forma dentro de G. Devuelve el numero de colores que se obtiene.

La diferencia con Greedy normal es la siguiente: en Greedy se le da al vértice $v$ el MENOR color posible que no sea un color de alguno de los vecinos de $v$ ya coloreados. En NotSoGreedy en cambio, cuando se está por colorear al vértice $v$ se hace lo siguiente:

1. Si hasta ese momento el algoritmo está usando $b$ colores, y los colores $1, 2, ..., b$ son todos colores de algún vecino de $v$, entonces a $v$ se le da el color $b + 1$. (esto es igual que en Greedy)

2. Si hasta ese momento el algoritmo está usando $b$ colores y no todos los colores $1, 2, ..., b$ son colores de algún vecino, entonces se elige al azar ALGUNO de los colores $1, 2, ..., b$ que no sea color de algún vecino de $v$, y se colorea $v$ con ese color.

"Al azar" significa que van a tener que usar alguna función pseudoaleatoria para realizar la selección. La función que usen es a criterio suyo PERO NO PUEDEN USAR rand(). Esto pues he notado que varios grupos NO SABIAN USAR rand() correctamente, asi que es mejor prohibirles usarla, que de todos modos no es tan buena y ademas tengo curiosidad por ver que funciones de randomización inventan.

Esa función pseudoaleatoria debe depender determinísticamente del paramétro de entrada "semilla". (es decir, correr dos veces esta función (con el mismo orden de los vertices) usando semilla=0x4764 pej, debe dar el mismo resultado).

### 6.2 Bipartito()

Prototipo de función:

`int Bipartito(Grafo G);`

Si $k$ es el número de componentes conexas de G, devuelve $k$ si G es bipartito, y $-k$ si no.

Ademas, si es bipartito, colorea G con un coloreo propio de dos colores. Si no es bipartito, debe dejar a G coloreado con algún coloreo propio. (no necesariamente el mismo que tenía al principio).

## 7 Funciones de ordenación

Estas funciones cambian el orden interno guardado en G que se usa para correr NotSoGreedy.

### 7.1 OrdenNatural()

Prototipo de función:

`void OrdenNatural(Grafo G);`

Ordena los vertices en orden creciente de sus "nombres" reales. (recordar que el nombre real de un vértice es un u32)

### 7.2 OrdenWelshPowell()

Prototipo de función:

`void OrdenWelshPowell(Grafo G);`

Ordena los vertices de W de acuerdo con el orden Welsh-Powell, es decir, con los grados en orden **no creciente**.

Por ejemplo si hacemos un for en i de GradoDelVertice(G,i) luego de haber corrido OrdenWelshPowell(G), deberiamos obtener algo como 10,9,7,7,7,7,5,4,4,3,3,2,1,1,1,1

### 7.3 AleatorizarVertices()

Prototipo de función:

`void AleatorizarVertices(Grafo G,u32 semilla);`

Esta función ordena pseudoaleatoriamente los vertices de G, usando alguna función pseudoaleatoria que dependa determinísticamente de semilla. La función de pseudoaleatoriedad que usen es a criterio suyo, pero debe depender deterministicamente de la variable semilla. (es decir, correr dos veces esta función con semilla=4 pej, debe dar los mismos resultados, pero si semilla=12, debería dar otro resultado).

**IMPORTANTE: debe dar el mismo resultado independientemente del coloreo o del orden que tenga en ese momento G. Es decir, si esta función es llamada en dos lugares distintos de un programa con la misma semilla, el orden obtenido debe ser el mismo.**

Como dijimos en la función NotSoGreedy, pueden usar la función de pseudoaleatoriedad que deseen EXCEPTO rand().

### 7.4 ReordenManteniendoBloqueColores()

Prototipo de función:

`void ReordenManteniendoBloqueColores(Grafo G,u32 x);`

Si G esta coloreado con $r$ colores y $VC_1$ son los vertices coloreados con 1, $VC_2$ los coloreados con 2, etc, entonces esta función ordena los vertices poniendo primero los vertices de $VC_{j_1}$, luego los de $VC_{j_2}$ , etc, donde $j_1, j_2, ..., j_r$ se determinan a partir de x de la siguiente manera:

- Si x=0, entonces $j_1 = r$, $j_2 = r-1$,...,$j_{r-1} = 2$, $j_r = 1$.
- Si x=1, entonces $j_1, j_2, ...$ son tales que $|VC_{j_1}| \leq |VC_{j_2}| \leq ... \leq |VC_{j_r}|$
- Si x>1, entonces se usa alguna función de pseudoaleatoriedad que dependa determinísticamente de x para generar un orden aleatorio de los números $\{1, 2, ..., r\}$, obteniendo un orden $j_1, j_2, ..., j_r$.
  - Como antes, pueden usar la función de pseudoaleatoriedad que deseen EXCEPTO rand().

## 8 Velocidad

Su programa debe ser razonablemente rápido. En particular, en una máquina mas o menos similar a las máquinas del lab de Famaf, para grafos de menos de 100 vertices debe demorar pocos segundos en poder cargarlo y correr 1000 iteraciones de NotSoGreedy con los reordenes de la sección 7. Para grafos con algunos miles de vertices y menos de 100K lados no debería demorar mas de a lo sumo un par de minutos. Para grafos mas grandes con mas de 1 millón de lados no debería demorar mas de 15 minutos.

## 9 Entrega

Deben entregar vía e-mail los archivos que implementan el proyecto y un informe. Al final de este documento se detalla lo que debe ir en el informe. Los archivos del programa deben ser todos archivos .c o .h No debe haber ningun ejecutable. Entrega de un ejecutable implica devolución automática del proyecto, con descuento de 1 punto y pasaje al fondo de la pila de corrección para cuando lo devuelvan. **WARNING: los que lo hagan en una Mac, tengan cuidado, pues el SO a veces agrega por su cuenta ejecutables**

### 9.1 Fecha de Entrega

Deben tenerlo terminado aproximadamente para el Martes 3 de Abril de 2018, aprovechando el fin de semana extra largo. Luego tienen un par de semanas mas para testearlo exhaustivamente y hacer correcciones. El proyecto deben entregarlo antes del martes 17 de Abril de 2018 a las 13:00 hs.

*NO ESPEREN HASTA ÚLTIMO MOMENTO* pues tengo entendido que por ejemplo el 13 de Abril tienen que entregar un proyecto de otra de las materias, asi que si esperan a comenzar este entonces no van a llegar. Ademas el 19 de Abril tengo entendido que tienen un parcial, asi que...empiecen ahora el proyecto.

## 9.2 Archivos del programa

Deben entregar un archivo TheOutsider.h en el cual se hagan las llamadas que uds. consideren convenientes a librerias generales de C, ademas de las especificaciones de los tipos de datos y funciones descriptas abajo.

No es necesario que todo este definido en TheOutsider.h, pero si definen algo en otros archivos auxiliares la llamada a esos archivos auxiliares debe estar dentro de TheOutsider.h. En TheOutsider.h debe figurar (comentados, obvio) los nombres y los mails de contacto de todos los integrantes del grupo.

Junto con TheOutsider.h obviamente deben entregar uno o mas archivos .c que implementen las especificaciones indicadas abajo, pero pueden nombrarlos como quieran.

Deben empaquetar todo de la siguiente forma:

Debe haber un directorio de primer nivel cuyo nombre consiste en los apellidos de los integrantes.

En ese directorio deben poner cualquier información extra que quieran agregar. Pueden no poner nada, pero si lo hacen el formato debe ser .pdf o un archivo ASCII. Entrega de un .docx se castiga con limpieza de las letrinas de Sauron.

Ademas, en ese directorio habrá un directorio de segundo nivel, llamado Tiotimolina

En este directorio deben estar TheOutsider.h y todos los otros archivos que necesiten, incluidos archivos auxiliares, pero no debe haber ningun main. **Todo .c que este en este directorio se considera que debe formar parte de la compilación.** (usaremos *.c en Tiotimolina para compilar).

Empaqueten todo en formato .tgz y envienlo por mail a:

matematicadiscretaii.famaf arroba gmail.com

Compilaremos (con mains nuestros) desde el directorio de primer nivel con gcc, -ITiotimolina , -Wall, -Wextra, -std=c99 , -O3 (o -O2) y -DNDEBUG.

## 9.3 Protocolo

Luego de enviado, se les responderá con un "recibido". Si no reciben confirmación dentro de las 24hs pregunten si lo recibí. (ahora, en tiempos normales. Mas adelante (Septiembre, pej) puedo tomarme un par de dias).

Se les puede requerir que corran su programa en su máquina para algunos grafos que les mandaremos. Algunas razones por las cuales pediremos esto es que su algoritmo corra demasiado lento con grafos grandes, o que obtenga resultados distintos dependiendo del compilador que se use o en la maquina que se use. Traten de hacer el código lo mas portable posible.

## 9.4 Nota del Proyecto, Errores del programa y retorno

El Proyecto será evaluado con una nota entre 0 y 10, la cual sera promediada por medio de un promedio pesado con las notas del Teórico y del Práctico para obtener la nota final. Las tres partes deben aprobarse por separado, asi que si no aprueban el proyecto no aprueban el final.

Dependiendo de los errores, podemos darles la nota definitiva del proyecto luego de corregido, o bien podemos devolverles el proyecto para que corrijan algunos errores antes de fijar la nota, si es que consideramos estos errores como serios pero no lo suficientemente serios como para desaprobar el proyecto.

Si los errores son muy graves, se les pondrá una nota entre 0 y 3, y se les dará una oportunidad mas para enviar el proyecto corregido, el cual se volverá a evaluar, y la nota definitiva del proyecto será el promedio entre las dos notas. (excepto que si el promedio da menos de 4, se considera 4. Por ejemplo, si lo entregan, sacan un 1, luego lo entregan de vuelta, la nueva entrega tiene un 5, el promedio es 3, pero se considera un 4).

**Si aprueban el proyecto, la aprobación es válida para todas las fechas hasta Marzo 2020 incluido,***si yo sigo a cargo de la materia. Si no, sólo vale hasta Marzo 2019.*

## 9.5 Que pasa si el proyecto no es aprobado en esta primera entrega

Si el proyecto no es aprobado aún luego de la devolución y segunda entrega, (o bien no es presentado a tiempo) entonces tienen un descuento de 4 puntos (en el proyecto) y pierden el derecho a que se los corrija en forma no booleana en forma anticipada al final, pero tienen una oportunidad mas para entregar el proyecto hasta el mismo dia del examen que quieran rendir (pero deben entregarlo antes del examen). (ver WARNING abajo).

Pueden entregarlo mucho antes si quieren, pero no será corregido a menos que efectivamente se presenten. Y en ese caso, no se corregirá de forma exhaustiva, haciendo un balance de los errores, sino que el primer error serio que tengan estan desaprobados en esa fecha.

"Error serio" es a mi solo criterio, asi que mejor no cometan errores de ningún tipo, pero especialmente cualquier error que provoque que una función se comporte de una forma distinta a la especificada en este documento provocará la desaprobación.

Mas aún, dadas las limitaciones de tiempo en corregir un proyecto de esta forma, no realizaré un analísis exhaustivo del código para encontrar todos los errores sino que detendré mi analísis al primer error serio que encuentre, lo cual puede hacer que en una fecha desaprueben por un error A, lo corrijan, en la siguiente fecha desaprueben por un error B, etc.

(En caso de aprobar, debido al descuento, la nota máxima del proyecto en este caso es 6).

**WARNING** Si no aprueban o no entregan este proyecto, y rinden en las fechas de Junio-Julio-Agosto, es este mismo proyecto el que deben entregar hasta el dia del examen. **Pero si rinden en Diciembre-Febrero-Marzo el proyecto puede ser otro.** Revisar en la página de la materia en Septiembre, donde se anunciará si es este mismo proyecto u otro).

Obviamente, si rinden en o luego de Junio del 2019 y no aprobaron el proyecto de este año, deben hacer el proyecto del 2019, con las condiciones que se digan el año que viene.

# 10 Final Warnings

- El uso de macros esta permitido pero como siempre, sean cuidadosos si los usan.
- Debe haber MUCHO COMENTARIO, las cosas deben entenderse.

  En realidad no voy a bajar puntos por esto, pero si un proyecto se me hace dificil de entender, pasa al fondo de la pila de correcciones.

  Respecto de los nombres de las variables, sólo pido sentido común. Por ejemplo, para denotar el número de vértices pueden simplemente usar n, o bien una variable mas significativa como nvertices o si quieren usar un nombre como "Superman", no me opongo. Pero si al número de vértices le llaman nlados, nl o ncolores entonces quedan automáticamente desaprobados. Idem con los nombres de las funciones. (pej, una función que se llame OrdenarVertices pero en realidad ordene lados)
- No se puede incluir NINGUNA libreria externa que no sea una de las básicas de C. (eg, stdlib.h, stdio.h, strings.h, stdbool.h, assert.h etc, si, pero otras no. Especificamente, glibc NO).
- Respecto de la RAM, no voy a poner este año un limite definitivo, pero si lo que hacen es una desmesura carente de sentido común se les devolverá el proyecto para que lo corrijan.
- BE CAREFUL de no producir un stack overflow.
- El grafo de entrada puede tener miles, cientos de miles de vertices e incluso millones de vertices y lados. Testeen para casos grandes. En mi página hay algunos grafos de ejemplos y varios en otras paginas, y deberían hacer sus propios ejemplos.
- Testeen. Luego testeen un poco mas. Finalmente, testeen.
- Para el punto anterior deberán hacer por su cuenta uno o mas .c que incluyan un main que les ayude a testear sus funciones. (no deben entregar estos archivos)
- Deben testear la funcionalidad de cada una de las funciones que programan, con programas que testeen si las funciones efectivamente hacen lo que hacen o no.

  Programar sin errores es difícil, y algunos errores se les pueden pasar aún siendo cuidadosos y haciendo tests, porque somos humanos. Pero hay errores que no deberían quedar en el código que me entreguen, porque son errores que son fácilmente detectables con un mínimo de sentido común a la hora de testear. En particular, no sean idiotas: testeen que las funciones de reordenamiento realmente ordenan como se pide. (es increible la cantidad de estudiantes que se creen que son Mozart en vez de Salieri y asumen que sólo porque son ellos los que escriben el código, va a estar bien).
- Tengan en cuenta que uds. deben programar esto de acuerdo a las especificaciones porque no saben qué programa va a ser el que llame a sus funciones, ni cómo las va a usar. Asumir que serán usadas de una forma que no esté en las especificaciones es un error grave. En particular, ConstruccionDelGrafo carga UN grafo en una variable, pero esta función puede ser llamada múltiples veces por un programa, cargando mas de un grafo en distintas variables.
- If you use a global variable en este proyecto you are a fucking idiot.