Proyecto de Matemática Discreta II-2015

1 Introducción

Deberán implementar una API en C (de hecho, C99, i.e., pueden usar comentarios // u otras cosas de C99). La API servirá para desarrollar un programa que cargará un grafo y usará diversos algoritmos para colorear sus vertices. El proyecto llevará una nota entre 0 y 10, la cual se promediara con un peso del 20% a las notas del práctico y del teórico del final para obtener la nota final de la materia. No pueden rendir el final mientras el proyecto no este aprobado por la cátedra, o si la nota del proyecto es menor a 4. El proyecto sera revisado por la catedra, y si no se lo considera aprobado será devuelto para ser corregido. Cada devolución implica un descuento de puntos de la nota final máxima que puede tener el proyecto. Cuando el total de puntos descontado sume mas de 6, ya no pueden volver a entregar el proyecto y no pueden rendir, debiendo recursar la materia. En la mayoria de los casos, cada devolución implica una perdida de 2 puntos, pero en algunos casos de errores menores puede ser menos, y en casos de errores graves, la perdida puede ser de hasta 6 puntos.

Deben hacer la API según las especificaciones indicadas abajo.

La API sera usada por uno o varios programas de la cátedra al cual uds. NO TENDRAN ACCESO, asi que deben asegurarse que su API cumpla las especificaciones.

Ademas de la API deben entregar un main.c que use la API en la forma que detallamos mas adelante. Se evaluará por separado la API y el main.c.

Deben entregar un archivo API.h en el cual se hagan las llamadas que uds. consideren convenientes a librerias generales, ademas de las especificaciones de los tipos de datos y funciones descriptas abajo. El programa que la cátedra usará para testear su programa solo tendrá includes de stdio, la stdlib, stdint.h y API.h por lo cual, por ejemplo, si van a usar string.h el include lo deben escribir uds. en la API. No es necesario (ni conveniente) que todo este definido en API.h, pero si definen algo en otros archivos auxiliares, ademas de (obviamente) entregarlos, la llamada a esos archivos auxiliares debe estar dentro de API.h. En API.h deben figurar (comentados, obvio) los nombres de todos los integrantes del grupo, junto con sus mails.

A continuación se detalla la especificación de la API, y la documentación que se debe entregar junto al código. Su programa DEBE RESPONDER a estas especificaciones.

2 Especificación de la API

2.1 Tipos de datos

La API deberá proveer los siguientes tipos de datos.

2.1.1 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. Uds deben definir u32 en API.h.

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

2.1.2 GrafSt:

Es una estructura, la cual debe contener toda la información necesaria para correr su implementación. La definición interna de la 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 (como grado de los vertices, numero de vecinos sin colorear, etc). Ademas, como Greedy usa un orden de los vertices, en esta estructura tiene que estar guardado algún orden de los vertices.

Basicamente, debe tener lo que uds. consideren necesario para poder implementar las funciones descriptas abajo, de la forma que a uds. les parezca mas conveniente.

Una forma posible de definir la estructura es por ejemplo:

```
typedef struct Grafoplus{
    //aquí van los parámetros específicos de cada equipo
} GrafSt;
```

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

2.1.3 GrafP:

es un puntero a una estructura de datos *GrafSt*. Esto se define de la siguiente forma:

```
typedef GrafSt *GrafP;
```

Las funciones que deberan implementar son en general de la forma Funcion(GrafP G), para poder cambiar la estructura interna del grafo.

2.2 Funciones

Se deben implementar las siguientes funciones.

2.2.1 NuevoGraf()

Prototipo de función:

```
GrafP NuevoGraf();
```

La función aloca memoria e inicializa una estructura GrafSt y devuelve un puntero a ésta. En caso de error, la función devolverá un puntero a `null`.

2.2.2 DestruirGraf()

Prototipo de función:

```
int DestruirGraf(GrafP G);
```

Destruye G y libera la memoria alocada. Retorna 1 si todo anduvo bien y 0 si no.

2.2.3 LeerGrafo()

Prototipo de función:

```
int LeerGrafo(GrafP G);
```

Lee un grafo desde standard input en el formato especificado a continuación.

Devuelve la cantidad de vertices del grafo si la lectura se completa sin problema y -1 si no.

Ademas de cargar el grafo, colorea cada vertice con un color distinto. (coloreo inicial)

El formato de entrada sera DIMACS, que es un formato estandard para representar grafos. El formato es asi:

1) Ninguna linea tiene mas de 80 caracteres.

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

3)Luego hay una linea de la forma:

p edge n m

donde n y m son dos enteros.

4) Luego siguen m lineas todas comenzando con e y dos enteros, representando un lado. Luego de esas m lineas deben detener la carga.

5) Cada lado vw aparecerá exactamente una vez en alguna linea de la forma e v w, o bien en una linea e w v. Si aparece una no aparecerá la otra. En general usaremos e v w con v menor que w, pero eso no esta especificado y no pueden asumir que la entrada será asi.

6) En el formato DIMACS no parece estar especificado si hay algun limite para los enteros, pero en nuestro caso los limitaremos a u32s.

Ejemplo:

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

Que un grafo tenga el vértice v (un u32) no implica 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.

Como dijimos, la función debe leer desde standard input. En general le daremos de comer una serie de lineas en ese formato, a mano, o bien usaremos en standard input el redireccionador "<" para redireccionar a un archivo que sea de esa forma.

2.2.4 ImprimeGrafo()

Prototipo de función:

```
int ImprimeGrafo(GrafPG);
```

Imprime en standard output una serie de lineas de acuerdo al formato DIMACS especificado arriba que describan el grafo. Devuelve 1 si todo anduvo bien, 0 si hubo algun problema.

2.2.5 GenerarGrafoAleatorio()

Prototipo de función:

```
GrafP GenerarGrafoAleatorio(int n,int m);
```

Crea un grafo con n vertices y m lados. Los vertices serán 1,2,...,n. Los lados deben ser creados aleatoriamente a partir de los vertices de la siguiente forma: calculan $q = \frac{2m}{n(n-1)}$. Luego iterativamente hasta obtener m lados eligen v y w al azar, con $v < w$, y tal que vw no haya sido todavia agregado al grafo y deciden con probabilidad q si agregan el lado vw al grafo o no. Devuelve un puntero a la estructura GrafSt creada.

2.2.6 NumeroVerticesDeColor()

Prototipo de función:

```
u32 NumeroVerticesDeColor(GrafP G,u32 i);
```

Retorna el número de vértices de color i.

2.2.7 ImprimirColor()

Prototipo de función:

```
u32 ImprimirColor(GrafP G,u32 i);
```

Imprime una linea que dice "Vertices de Color i:" y a continuacion una lista de los vertices que tienen el color i, separados por comas y con un punto luego del último color. Por ejemplo:

Vertices de Color 3: 15,17,1.

Los vertices no tienen porque estar ordenados.

Si no hay vertices de color i debe imprimir "No hay vertices de color i"

Retorna el número de vertices de color i.

2.2.8 CantidadDeColores()

Prototipo de función:

```
u32 CantidadDeColores(GrafP G);
```

Devuelve la cantidad de colores usados en el coloreo de G.

2.2.9 Greedy()

Prototipo de función:

```
u32 Greedy(GrafP G);
```

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

2.2.10 OrdenWelshPowell()

Prototipo de función:

```
void OrdenWelshPowell(GrafP G);
```

Ordena los vertices de G de acuerdo con el orden Welsh-Powell.

2.2.11 OrdenWelshPowellMod()

Prototipo de función:

```
void OrdenWelshPowellMod(GrafP G);
```

Ordena los vertices de G de acuerdo con el orden Welsh-Powell Modificado.

2.2.12 RLF()

Prototipo de función:

```
u32 RLF(GrafP G);
```

Corre RLF en G. Devuelve el numero de colores que se obtiene.

2.2.13 DSATUR()

Prototipo de función:

```
u32 DSATUR(GrafP G);
```

Corre DSATUR en G. Devuelve el numero de colores que se obtiene.

2.2.14 GrandeChico()

Prototipo de función:

```
void GrandeChico(GrafP G);
```

Si G esta coloreado con r colores y W_1 son los vertices coloreados con 1, W_2 los coloreados con 2, etc, entonces esta función ordena los vertices poniendo primero los vertices de W_{j_1} (en algún orden) luego los de W_{j_2} (en algún orden), etc, donde $j_1, j_2, ...$ son tales que $|W_{j_1}| \geq |W_{j_2}| \geq ... \geq |W_{j_r}|$.

2.2.15 ChicoGrande()

Prototipo de función:

```
void ChicoGrande(GrafP G);
```

Idem que el anterior excepto que ahora el orden es tal que $|W_{j_1}| \leq |W_{j_2}| \leq ... \leq |W_{j_r}|$.

2.2.16 Revierte()

Prototipo de función:

```
void Revierte(GrafP G);
```

Si G esta coloreado con r colores y W_1 son los vertices coloreados con 1, W_2 los coloreados con 2, etc, entonces esta función ordena los vertices poniendo primero los vertices de W_r (en algún orden) luego los de W_{r-1} (en algún orden), etc.

2.2.17 OrdenAleatorio()

Prototipo de función:

```
void OrdenAleatorio(GrafP G);
```

Si G esta coloreado con r colores y W_1 son los vertices coloreados con 1, W_2 los coloreados con 2, etc, entonces esta función ordena los vertices poniendo primero los vertices de W_{j_1} (en algún orden) luego los de W_{j_2} (en algún orden), etc, donde $j_1, j_2, ...$ son aleatorios (pero distintos).

3 Main(s)

Ademas de la API, deben entregar dos archivos en donde figure un main (obviamente en la API no debe haber ningún main) en donde se use la API para hacer lo que se detalla a continuación:

3.1 main.c

Este debe usarse para leer un grafo desde standard input, donde la entrada será como es descripta arriba y que haga lo siguiente:

1. Carga el grafo
2. Si $m = 0$ imprimen X(G)=1 y paran. Los items siguientes se hacen solo si $m > 0$.
3. Si el grafo es bipartito, debe imprimir en standard output una linea que diga “Grafo Bipartito” (sin las comillas).
4. Si el grafo no es bipartito debe Correr Welsh-Powell, Welsh-Powell modificado, RLF y DSATUR y decir cuantos colores obtiene en cada caso. Debe imprimir en standard output cuatro lineas por separado, en cada linea debe figurar el nombre del algoritmo, luego dos puntos y luego “Colores usados: k” (sin las comillas) donde k será el número de colores que obtuvieron.
5. Si estan en este paso es porque el grafo no es bipartito, por lo tanto si alguno de esos algoritmos obtiene un coloreo con tres colores, ya saben que $\chi(G) = 3$ asi que deben imprimir una linea que diga “X(G)=3”.
6. Si no obtuvieron 3 colores con ninguno de los algoritmos, deben tomar el mejor coloreo de esos cuatro y correr Greedy Nv veces cambiando el orden de los colores, siguiendo el siguiente patrón:

   - p% GrandeChico
   - q% ChicoGrande
   - r% Revierte
   - a% OrdenAleatorio

   luego de lo cual deben imprimir “Mejor coloreo con Greedy iterado Nv veces: k colores”, donde k es el menor número de colores que hayan obtenido en las Nv iteraciones.

   Nv,p,q,r y a seran parametros fijos de su algoritmo. Pueden usar lo que quieran con las siguientes restricciones:

   (a) Nv debe ser al menos 10.

   (b) La suma de p,q,r y a debe ser 100.

   (c) a no puede ser superior a 10 ni inferior a 4, y debe ser el mas chico de todos.

Los valores pueden ser fijos o, si lo desean, pueden hacerlos variables y dependientes del usuario. En ese caso deben explicar en el informe que entregarán con el proyecto como debe correrse el programa una vez compilado. Si lo hacen de esta forma tengan en cuenta que queremos correr el programa de forma tal de poder redireccionar a un archivo, asi que deben ser argumentos y no ser pedidos al usuario mediante preguntas. Obviamente las técnicas de buena programación requieren que debe haber un mensaje de help que indique como se deberia usar el programa si no se usa el numero o tipo de argumentos requeridos

Si son fijos, en el informe deben aclarar cuales valores usaron. Una posibilidad interesante es hacerlos fijos pero Nv dependiente del número de lados.

Si quieren pueden entregar mas de un main.c en donde implementen distintas elecciones de estos parametros, o algunos con parámetros fijos y otros variables. El que ustedes consideren el "principal" se debe llamar main.c, los otros main<algo>.c donde <algo> puede ser un número o algo mas descriptivo como por ejemplo mainvariable.c Nosotros solo evaluaremos para descontar puntos el main.c (i.e., los otros no producirán descuentos de puntos) y si leyendo el informe nos parece que alguno de los otros es interesante, o el main.c tiene errores, tambien lo evaluaremos para posibles puntos extras. (pero la nota final nunca superará el 10, esto sólo compensaría algún error, por ejemplo del main.c). Por razones de sanidad mental, sólo pueden entregar a lo sumo 4 versiones (incluyendo la principal).

Las condiciones p% GrandeChico, q% ChicoGrande, r% Revierte y a% Aleatorio no significan que exactamente un p% van a usar GrandeChico, exactamente un q% van a usar ChicoGrande, etc, sino que en cada iteración, cuando el algoritmo va a decidir cual orden usará, elegirá GrandeChico con una probabilidad del p%, ChicoGrande con una probabilidad del q%, etc.

3.2 test.c

Este otro hace lo mismo que el anterior, excepto que en vez de leer desde standard input, genera grafos aleatorios con un número prescripto de vertices y lados y muestra para cada algoritmo cual es el promedio de los colores obtenidos. Debe o bien usar parámetros de entrada, o bien preguntarle al usuario por tres números: n, m y k. (debe estar bien explicado en el informe como funciona y debe haber una help programada por si los parametros no son correctos). El algoritmo entonces generará k grafos aleatorios con n vertices y m lados, correrá Welsh-Powell,Welsh-Powel modificado, RLF, DSATUR y GreedyIterado en cada uno de ellos e imprimirá para cada uno de ellos el promedio de los colores que obtuvo. Solo hay que imprimir el promedio obtenido, no los resultados individuales de las k corridas. Este lo necesitarán para el Informe (ver mas abajo). Al igual que con main.c, pueden hacer hasta 4 versiones distintas dependiendo de la elección de los parametros Nv, p,q,r,a.

4 Informe

Deberán entregar un informe sobre el proyecto. El informe puede ser un .pdf o un README que sea leible con cualquier lector de textos ASCII (asi que NO usen Word o FreeOffice o cualquier editor no ASCII). Pueden hacer tanto un .pdf como un README, por ejemplo, en el .pdf poner un informe detallado, y en el README algo mas corto.

El informe debe ser claro, y debe mostrar que ustedes entienden su propio proyecto y no lo copiaron de algun lado. (esta permitido tomar ideas de otros lados, e incluso fragmentos de código si lo desean, pero debe quedar claro que las/los entienden y no que solo hicieron cut and paste). Algunos detalles del código nos pueden hacer sospechar que lo copiaron, en ese caso podemos requerir una exposición oral intensiva y si no nos convencen el proyecto se devuelve con 5 puntos de descuento.

En el informe deben figurar el nombre del o los integrantes. Ademas, el informe debe contener al menos las siguientes secciones:

4.1 Especificación

El informe debe proveer una especificación del proyecto, tal que alguien, leyendo el informe, pudiera hacer una implementación "black room" de su proyecto sin tener que leer el código. Es decir,

deben explicar que hace su algoritmo en forma resumida. Copiar en bloque el código no es una especificación. No hace falta que expliquen que es Greedy o RLF o DSATUR, etc, pero si hace falta que expliquen que tipo de estructura/s usaron para representar el grafo, como implementaron $\Gamma(x)$ (si es que lo hicieron, y si no, como hicieron para buscar vecinos), como implementaron el orden de los vértices, como hicieron para resolver el problema de que los vertices pueden ser cualquier u32, como implementaron la aleatoriedad que se necesita en diversas partes del algoritmo, etc. Para repetir una vez mas, simplemente copiar fragmentos de código no es lo que buscamos, para eso leemos la API. Queremos que muestren que entienden su código lo suficientemente bien como para explicarlo en sus propias palabras.

Si los parametros para hacer Greedy iterado son fijos, deben explicitarlos acá, si son variables deben decirlo también acá.

4.2 Tareas de los Integrantes

Si hay un solo integrante, esta sección no es necesaria. Si hay dos integrantes se debe describir aproximadamente que hizo cada integrante del grupo. Ambos integrantes deben haber trabajado en la API. Esta bien que uno haga por ejemplo DSATUR completamente y otro RLF completamente, pero no esta bien que uno haga toda la API y el otro solo el main.

4.3 Tests

Debe haber una sección con resultados de tests hechos a su algoritmo. No es necesario que esten todos los tests que les hayan hecho. Lo que se requiere es lo siguiente: deben correr su algoritmo en una serie de categorias de grafos. Deben tener al menos 5 categorias de grafos. Una categoria de grafos quedará determinada por dos números: el número de vertices y el número de lados. Las cinco categorias mínimas que deben incluir son: (10,20) (con al menos 40 grafos) (20,100) (con al menos 20 grafos), (100,2500), (con al menos 15 grafos), (1000,5000) (con al menos 10 grafos), (1000,200000) (con al menos 5 grafos). Para cada categoria de grafo deben detallar cuantos grafos dentro de esa categoria testearon y cuantos colores obtuvieron en promedio con Welsh-Powell,Welsh-Powell modificado, RLF, DSATUR y con Greedy iterado y cuanto demoraron en promedio en correr el programa completo. (no estoy buscando suma precisión en la medición del tiempo, aca basta con que lo corran con la instrucción time).

Los grafos en cada categoria deben haber sido creados con su función GenerarGrafoAleatorio, y deben haber usado test.c para hacer el informe. (si usaron mas de un test.c, incluyan el informe de cada uno en secciones separadas)

5 Cosas para tener en cuenta

- El uso de macros esta permitido pero como siempre, sean cuidadosos si los usan.
- Debe haber MUCHO COMENTARIO, las cosas deben entenderse. En particular, en cada funcion o macro que usen debe haber una buena explicación, entendible, de lo que hace. Ademas cada variable que se use debe tener un comentario al lado indicando su significado y para que se usará. Tambien seria bueno que los nombres sean mas o menos significativos. (por ejemplo:

  ```
  int i,j; //indices

  int ncolores; //numero de colores

  bool Stop; //una booleana que sera 0 si...  y 1 si...
  ```

  etc;
- El código debe estar bien modularizado.
- Presten atencion al formato especificado, no solo al de entrada sino al de salida: uds. deben asumir que otro grupo de personas usará la salida de su programa como input de algun programa propio, de ahi las especificaciones de salida.
- Como se dijo varias veces, la entrada y la salida son estandards. Imprimir a un archivo implica devolución automática del proyecto.

- No se pueden usar librerias externas que no sean las básicas de C. (eg, stdlib.h, stdio.h, strings.h, stdbool.h, assert.h etc, si, pero otras no). Pueden tomar ideas de ellas, pero su código debe ser lo mas portable posible, algunas librerias externas no siempre lo son, o son demasiado grandes. La experiencia de años anteriores indica que nos crea problemas.
- Su programa no debe usar nunca mas de 256 MB de memoria RAM.
- El grafo de entrada puede tener miles o incluso cientos de miles de vertices y lados. Testeen para casos grandes. (pero nunca va a haber millones de vertices)
- Su programa debe ser razonablemente rápido. Algunos lo harán mas rápido que otros, pero no deberia demorar horas (y menos aun dias) en terminar. Sin embargo, como es el primer año que hacemos este proyecto, seremos un poco mas lenientes en este punto que años anteriores.

6 Entrega

6.1 Protocolo

Deben entregar vía e-mail los archivos que implementan la API dada arriba y al menos un main.c y un test.c No debe haber ningun ejecutable. Entrega de un ejecutable implica desaprobación instantánea.

Deben empaquetar todo de la siguiente forma: Debe haber un directorio de primer nivel cuyo nombre consiste en el apellido de los miembros del equipo.

En ese directorio deben poner el informe, ya sea en README o en .pdf (o ambos, si hacen los dos). Ademas, en ese directorio habrá dos directorios de segundo nivel, uno llamado apifiles y otro llamado dirmain. En apifiles deben estar todos los archivos de la API, incluidos archivos auxiliares, pero no debe haber ningun main. **Todo .c que este en apifiles se considera que debe formar parte de la compilación.** (usaremos *.c en apifiles para compilar). En dirmain deben estar main.c, test.c y si hay otras versiones, de main o test, tambien deben estar aqui. Empaqueten todo en formato .tgz.
No hace falta un make. Compilaremos con gcc, -Wall, -Wextra, -O3, -std=c99. Sin embargo, nuestro compilador de gcc puede no ser el mismo que el suyo, traten de hacer las cosas lo mas portable posible.

Luego de enviado, se les responderá con un “recibido”. Si no reciben confirmación dentro de las 24hs pregunten si lo recibí.Excepción: para los fines de semana pueden tener que esperar 48hs.

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.

Una vez entregado el proyecto, lo podemos aprobar, ya sea con nota 10 o con alguna nota menor con descuento de puntos por errores diversos. Dependiendo de los errores podemos considerar que el proyecto no esta para ser aprobado y en ese caso se devuelve para que lo corrijan. Esto implica un descuento de al menos 2 puntos.

Pueden entregarlo tantas veces como quieran, pero cada vez suma puntos de descuento.

Una vez que se hayan descontado 6 puntos, no pueden volver a entregar, el proyecto queda rechazado definitivamente y deben recursar la materia

6.2 Fecha de Entrega

El 1 de Junio a las 7:30AM o antes deben entregar el proyecto.

6.2.1 Entrega Alternativa

Si no lo entregan en esa fecha o lo entregan y se los devolvemos, quedan libres y no pueden rendir en la primera fecha.

Para rendir en la segunda fecha de examen deben entregar para el 22 de Junio a las 730AM. Para rendir en la tercera fecha de examen deben entregar para el primer lunes despues de las vacaciones de julio.

Para rendir en Noviembre-Diciembre-Febrero-Marzo tienen que entregarlo como máximo el 30 de Septiembre. Luego de esa fecha no se aceptan más proyectos.

7 Devolución del proyecto y Descuento de Puntos

En general la experiencia demuestra que algunos alumnos son muy creativos a la hora de cometer errores, asi que es imposible para nosotros listar todos los errores posibles que puedan cometer.

La nota por default del proyecto es 10, es decir, no juzgaremos calidades distintas de proyectos. Sin embargo, a partir del 10, descontaremos puntos por fallas en el proyecto, en algunos casos devolviendo el proyecto.

Si juzgamos que los comentarios no son los suficientes o no lo suficientemente claros, les descontaremos entre 0,5 puntos y 1 punto, y podemos devolver el proyecto requiriendo que agreguen mas comentarios.

Idem si consideramos que el informe no es claro.

SI MANDAN EL INFORME EN UN FORMATO QUE NO ES EL QUE ESPECIFICAMOS LES DESCONTAREMOS 4 PUNTOS.

La siguiente lista de errores de programación implica una devolución automatica del proyecto, con descuento de puntos. Obviamente, seguramente habra errores que no hemos pensado.

1. No compila. Aca hay dos casos. Si no compila con su main y su API, les descontaremos entre 2 y 4 puntos. Si no compila cuando lo intentamos con nuestro main y su API (o al reves) entonces es un error de compilación por incompatibilidad entre su API y nuestro main. Esto ocurre si no respetan las especificaciones. Cosas que pueden ocurrir (y ocurrieron en otros años) es que no tipeen bien las funciones de la API (descontaremos 0,5 puntos), que sus funciones usen mas argumentos o argumentos de distinto tipo que los especificados o retornen valores que no son los especificados (descontaremos entre 2 a 3 puntos) o bien que hagan cosas completamente ridiculas, que es dificil listar aqui pero han ocurrido. (descontaremos entre 3 y 4 puntos).

2. El algoritmo dice que un grafo es bipartito cuando no lo es o viceversa: -4 puntos.

3. Hay memory leaks. -1 punto.

4. Runtime error. (por ejemplo, segmentation fault, o peor aún, clava la maquina). Dependiendo del error, entre -1 y -4 puntos.

5. Error de impresión de la salida. entre -0,5 puntos si es un error menor, a -2 puntos.

6. No acepta inputs válidos. (por ejemplo, es incapaz de leer desde standard input, o bien espera un formato que no es el que especificamos) -2 puntos.

7. Errores de carga en los datos: continua cargando mas alla de lo que deberia o carga mal algunos lados, o directamente no los carga, o inicializa mal algo. Entre -1 y -2 puntos.

8. Alguno de los algoritmos que les pedimos no es correcto. Si es Greedy, -6 puntos. Si es alguno de los otros, -2 puntos.