Proyecto de Matemática Discreta II-2017 (primera parte)

1 Introducción

Deberán implementar lo que se detalla a continuación en C (C99, i.e., pueden usar comentarios // u otras cosas de C99).

Este año el proyecto se divide en dos partes: la primera parte se detalla en este documento y consiste en implementar las funciones detalladas aqui.

La segunda parte consistirá en hacer un archivo .c con un main que use estas funciones para hacer las cosas que se le pedirán cuando se detalle esa parte.

La primera parte puede ser hecha en forma individual o en grupos de hasta 3 personas. La segunda parte es individual.

El objetivo final es cargar un grafo y dar un coloreo propio de sus vertices, usando repetidamente Greedy de varias formas.

Contents

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 WinterSt

Es una estructura, la cual debe contener toda la información sobre el grafo 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.

IMPORTANTE: Cómo Greedy 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.

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.

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

2.3 WinterIsHere

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

3 Funciones De Construcción/Destrucción del grafo

3.1 WinterIsComing()

Prototipo de función:

```
WinterIsHere WinterIsComing();
```

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

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

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 A B

  donde A y B son dos enteros. Luego de B puede haber una cantidad arbitraria de espacios en blancos.

  El primero número (A) en teoría representa el número de vértices y el segundo (B) el número de lados, pero hay ejemplos en la web en donde A es en realidad solo una COTA SUPERIOR del número de vertices y B una cota superior del número de lados.

  Sin embargo, todos los grafos que nosotros usaremos para testear cumpliran que A será el número de vertices exacto y B 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 pagina hay muchos ejemplos bien formateados.

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

  e x y

  (luego de "y" puede haber una cantidad arbitraria de espacios en blanco)

  Luego de esas B lineas deben detener la carga.

  El archivo PUEDE tener lineas mas allá de esas B lineas. Estas lineas pueden tener una forma arbitraria, pueden o no ser comentarios, o extra lados, etc. Su programa debe simplemente ignorar esas lineas.

  Por otro lado, el archivo puede efectivamente terminar en la última de esas lineas, y su código debe poder procesar estos archivos también. (el año pasado, por alguna razón, varios grupos hicieron código que no funcionaba si la ultima linea no era una linea en blanco)

  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 dando vuelta 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 x y NO aparecerá el lado e y x.

- Nunca fijaremos $B = 0$, es decir, siempre habrá al menos un lado.

- 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. (reservar espacio para 2^{32} posibles entradas no es una buena idea)

- 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

Nota 3.1.1 *dado que los vertices pueden ser cualquier u32, una de las cosas que requeriremos es que a cada vertice se le asigne una "etiqueta" , que debe ser un número entre 0 y $n-1$, donde n es el número de vertices. Ese "etiqueteado" de los vertices debe proveer una biyección entre el conjunto de vertices y el conjunto $\{0, 1, ..., n-1\}$. Esta biyección, luego de terminado WinterIsComing, no debe ser cambiada por ninguna otra función.*

3.2 Primavera()

Prototipo de función:

```
int Primavera(WinterIsHere W);
```

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

4 Funciones de coloreo

4.1 Greedy()

Prototipo de función:

```
u32 Greedy(WinterIsHere W);
```

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

4.2 Bipartito()

Prototipo de función:

```
int Bipartito(WinterIsHere W);
```

Devuelve 1 si W es bipartito, 0 si no.

Ademas, si devuelve 1, colorea W con un coloreo propio de dos colores.

5 Funciones para extraer información de datos del grafo

5.1 NumeroDeVertices()

Prototipo de función:

```
u32 NumeroDeVertices(WinterIsHere W);
```

Devuelve el número de vértices de W.

5.2 NumeroDeLados()

Prototipo de función:

`u32 NumeroDeLados(WinterIsHere W);`

Devuelve el número de lados de W.

5.3 NumeroVerticesDeColor()

Prototipo de función:

`u32 NumeroVerticesDeColor(WinterIsHere W,u32 i);`

Retorna el número de vértices de color i. (debe retornar 0 si no hay vertices de color i).

5.4 NumeroDeColores()

Prototipo de función:

`u32 NumeroDeColores(WinterIsHere W);`

Devuelve la cantidad de colores usados en el coloreo que tiene en ese momento W.

5.5 IesimoVerticeEnElOrden()

Prototipo de función:

`u32 IesimoVerticeEnElOrden(WinterIsHere W,u32 i);`

Devuelve la etiqueta (ver nota 3.1.1) del vértice numero i en el orden guardado en ese momento en W. (el índice 0 indica el primer vértice, el índice 1 el segundo, etc)

6 Funciones de ordenación

Estas funciones cambian el orden interno guardado en W que se usa para correr Greedy.

6.1 OrdenNatural()

Prototipo de función:

`void OrdenNatural(WinterIsHere W);`

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

6.2 OrdenWelshPowell()

Prototipo de función:

`void OrdenWelshPowell(WinterIsHere W);`

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(W,IesimoVerticeEnElOrden(W,i)) luego de haber corrido OrdenWelshPowell(W), deberiamos obtener algo como 10,9,7,7,7,7,5,4,4

6.3 AleatorizarVertices()

Prototipo de función:

`void AleatorizarVertices(WinterIsHere W,u32 x);`

Esta función ordena pseudoaleatoriamente los vertices de W, usando alguna función pseudoaleatoria que dependa determinísticamente de x. La función de pseudoaleatoriedad que usen es a criterio suyo, pero debe depender deterministicamente de la variable x. (es decir, correr dos veces esta función con x=4 pej, debe dar los mismos resultados, pero si x=12, debería dar otro resultado). IMPORTANTE: debe dar el mismo resultado independientemente del coloreo o del orden que tenga en ese momento W. Es decir, si esta función es llamada en dos lugares distintos de un programa con el mismo x, el orden obtenido debe ser el mismo.

6.4 ReordenManteniendoBloqueColores()

Prototipo de función:

`void ReordenManteniendoBloqueColores(WinterIsHere W,u32 x);`

Si W 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 $j_1 = r$, $j_2 = 1$,$j_3 = 2$,...,$j_r = r - 1$.

- Si x=1 entonces $j_1 = r$, $j_2 = r-1$,...,$j_{r-1} = 2$, $j_r = 1$.
- Si x=2, entonces $j_1, j_2, ...$ son tales que $|VC_{j_1}| \leq |VC_{j_2}| \leq ... \leq |VC_{j_r}|$
- Si x=3 entonces $j_1, j_2, ...$ son tales que $|VC_{j_1}| \geq |VC_{j_2}| \geq ... \geq |VC_{j_r}|$.
- Si x>3, entonces se usa alguna función de pseudoaleatoriedad que dependa determinísticamente de x para generar un orden aleatorio de los números $\{1, 2, ..., j\}$, obteniendo un orden $j_1, j_2, ..., j_r$.

7 Funciones de los vertices

7.1 NombreDelVertice()

Prototipo de Función:

```
u32 NombreDelVertice(WinterIsHere W, u32 x);
```

Devuelve el nombre real del vértice cuya etiqueta es x. (ver nota 3.1.1)

7.2 ColorDelVertice()

Prototipo de Función:

```
u32 ColorDelVertice(WinterIsHere W, u32 x);
```

Devuelve el color con el que está coloreado el vértice cuya etiqueta es x. (ver nota 3.1.1)

7.3 GradoDelVertice()

Prototipo de Función:

```
u32 GradoDelVertice(WinterIsHere W, u32 x);
```

Devuelve el grado del vértice cuya etiqueta es x. (ver nota 3.1.1)

7.4 IesimoVecino()

Prototipo de función:

```
u32 IesimoVecino(WinterIsHere W, u32 x,u32 i);
```

Devuelve la etiqueta (ver nota 3.1.1) del vécino numero i (en el orden en que lo tengan guardado uds. en W) del vertice cuya etiqueta es x.

El orden que usen ustedes es irrelevante, lo importante es que de esta forma podemos pedir externamente la lista de vecinos para realizar tests. (el índice 0 indica el primer vécino, el índice 1 el segundo, etc)

8 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 ser exiliado en el universo de "The Walking Dead"

8.1 Archivos del programa

Deben entregar un archivo JonSnow.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 JonSnow.h, pero si definen algo en otros archivos auxiliares la llamada a esos archivos auxiliares debe estar dentro de JonSnow.h. En JonSnow.h debe figurar (comentados, obvio) los nombres y los mails de contacto de todos los integrantes del grupo.

Junto con JonSnow.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 primeraparte.

En este directorio deben estar JonSnow.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 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, -Iprimeraparte, -Wall, -Wextra, -O3, -std=c99 . Tambien podemos usar -DNDEBUG, o no, dependiendo de alguna reacción quimica en nuestro cerebro.

El código debe ser portable. En particular:

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 si vemos copiado de grandes fragmentos de código la van a pasar mal. Y la van a pasar peor si descubrimos intento de engañarnos haciendo cambios cosméticos en el código.

8.2 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.

8.3 Fecha de Entrega

La especificaremos en clase y en la pagina de la materia, pero será en algún dia en la segunda mitad de ABRIL, asi que ponganse a programar YA.

8.3.1 Entrega Alternativa

Si no lo entregan en esa fecha tienen un descuento de 5 puntos de la nota del final si rinden en Junio,Julio o Agosto. Es decir que para no ser bochados, deben obtener una nota en el final, antes del descuento, igual a 9 o 10. (y en esos casos, su nota final seria 4 o 5).

Para rendir en Noviembre-Diciembre-Febrero-Marzo sin descuento de puntos tienen que entregar (AMBAS partes, pues para esa epoca ya habremos dado la segunda parte) como máximo el 30 de Septiembre. De lo contrario, tienen un descuento de 5 puntos en la nota del final.

8.4 Que pasa si el proyecto no es aprobado

Hasta que no les digamos que el proyecto esta aprobado, si se presentan a rendir tienen un descuento de 5 puntos sobre la nota final que obtengan en el examen final, es decir, como explicamos arriba, deben sacar 9 o 10 para aprobar con 4 o 5.

En general tendremos paciencia y aunque haya errores, no los desaprobaremos sino que les retornaremos el proyecto para que corrijan los errores (ver punto siguiente). Sin embargo, en algunos casos, si hay muchos errores muy graves, o le hemos devuelto el proyecto muchas veces y siguen sin corregir los errores, podemos cansarnos y desaprobarlos.

8.5 Errores del programa y retorno

Dependiendo de los errores, podemos asignarles un descuento de puntos sobre la nota del final. Ese descuento sobre la primera entrega del proyecto nunca será mayor a dos puntos, y ademas, a diferencia del descuento de los que no aprueban el proyecto, no puede provocar que un alumno que apruebe el final sea bochado. (es decir, si tienen un descuento de 2 puntos y sacan 5 en el final, entonces su nota es 4, no 3).

Dependiendo de la severidad de los errores podemos devolverles el proyecto para que lo corrijan, y si cuando lo devuelven siguen teniendo errores podemos descontarles mas puntos. (los cuales sin embargo, tienen la "barrera" del 4, es decir, no pueden provocar que no aprueben). Por ejemplo, si lo envian numerosas veces, podriamos descontarles, por ejemplo, 6 puntos en total, pero serían mejores que los 5 de descuento del que no aprueba el proyecto porque no podrían hacerle bajar la nota a menos de 4 si tienen mas de 4.

9 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.
- El grafo de entrada puede tener miles o incluso cientos de miles de vertices y millones de lados. Testeen para casos grandes. En mi pagina 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. (obviamente, no deben entregar estos archivos)
- En la segunda parte, daremos las especificaciones de un archivo que usará estas funciones. En particular, Greedy se correrá miles de veces asi que debe estar eficientemente programado o no terminarán a tiempo. Lo mismo con las funciones de reordenación.