Proyecto de Matemática Discreta 2

Integrantes:

* Alan Bracco.
* Mario Ferreyra.

Tema: Coloreo de grafos

1 de Junio de 2015, Córdoba Argentina

Especificación

Aspectos generales

La estructura con la cual trabajamos para poder representar el grafo fue una llamada “GrafoPlus” en la que incluimos los siguientes campos:
* Uno en el que almacenamos la cantidad de vértices,
* Uno en el que almacenamos la cantidad de lados,
* Uno en el que almacenamos la cantidad de colores,
* Un arreglo en el que se guarda el orden de los vértices,
* Un arreglo guarda la estructura de cada vértice.

A su vez, esta estructura de cada vértice incluye los siguientes campos:
* Su grado,
* Su color,
* Su Identificador o nombre original del vértice, ya que en el programa se utilizan posiciones donde se almacenan los vértices y no los nombres originales de cada uno de ellos.
* Un arreglo de vecinos el cual contiene los vecinos de cada vértice (usando la metodología anterior, es decir sin usar su nombre original). Este es el arreglo que representa a Γ(x).

El orden en el que se agregan los vértices a la estructura es el orden que hay en el grafo ingresado, a medida que se los ve, es decir, no existe ningún caso en el que se vea un vértice y no se analice. Si ya fue leído anteriormente, simplemente no se agrega.

Con la posibilidad de que los vértices pueden ser u32, nos restringimos a trabajar solamente con valores van a poder estar entre UINT32_MIN (exactamente 0) y UINT32_MAX (exactamente 2^32 – 1).

MAIN

El numero de iteraciones que va a realizar Greedy se estableció en 10 para que devuelva un resultado lo mas rápido posible.

Los porcentajes con los que se eligen los tipos de ordenamiento también están fijados.
La probabilidad de hacer GrandeChico (%p) es 25%, al igual que la probabilidad de hacer ChicoGrande.
La probabilidad de hacer Revierte (%r) es 44% y 6% (%a) hacer un OrdenAleatorio.

El resto del main, realiza lo pedido en el enunciado.

API

La estructura utilizada fue la explicada anteriormente en “Aspectos generales”.
La función NuevoGraf() y DestruirGraf() simplemente inicializa, aloca y desaloca memoria respectivamente.

Lectura del grafo:

Al momento de leer grafo, se recorren
los datos una primera vez para inicializar los valores del grafo, tal como cantidad de vértices, cantidad de aristas y cantidad de colores (inicialmente la cantidad de vértices), y la información de los vértices tales como su identificador, orden, grado y el nombre (o posición) con el que se manejará o referenciará durante la ejecución del programa.
Para esto se utiliza una función auxiliar (add_vertex_id_color_grado) que realiza ese procedimiento.
A su vez se utilizan arreglos auxiliares para guardar la información de las aristas, para luego completar la información faltarte de los vértices (los vecinos).
Con esto se utiliza mas memoria, pero solo momentánea ya que esos arreglos auxiliares se liberan al momento de finalizar la lectura del grafo.
Además se evita la realocación de memoria, y consumo de tiempo realizando este proceso.
Al conocer los grados de cada vértice se puede alocar la memoria justa para cada uno de ellos.

Para completar esa información de vecinos de cada vértice se utiliza una función (add_vecino) que si la información no estaba previamente la agrega.
Una vez realizado esto, el grafo contiene todos los datos necesarios, así como cada vértice tiene su información necesaria.

Impresión del grafo:

Para imprimir el grafo, como la estructura del grafo contiene los datos de cantidad de vértices y aristas, es simple obtenerlos e imprimirlos.
Para imprimir los lados, se observa una de las ventajas de guardas las posiciones en las cuales están los vértices en lugar de su nombre original, ya que por cada vértice hay que imprimir el lado con todos sus vecinos, siempre y cuando su vecino no haya sido analizado anteriormente y haya lados impresos repetidos.
Para saber si ya fue impreso o no, simplemente debe fijarse que el vecino a imprimir este en una posición mas alta del arreglo, es decir que no fue analizado.
Y así se imprimen los lados en donde los vértices sean el vértice actual y su vecino no analizado previamente.

Número de vértices de color x:

Ya que la estructura de cada vértice consta de un campo “color”, se busca la cantidad de veces que ese mismo campo coincide a lo largo del arreglo de vértices con el color x ingresado como argumento.

Impresión de vértices de color x:

Recorre el arreglo de vértices hasta encontrar un vértice con el color x ingresado.
Si no lo encuentra, lo avisa mediante un mensaje.
En caso contrario imprime el identificador del vértice, mostrandosé así el nombre con el que se ingresó y no la posición utilizada internamente.
Así sucesivamente va imprimiendo los vértices de ese color hasta llegar al final del arreglo de vértices.
No es necesario un arreglo temporal para guardar los vértices ya que se van imprimiendo los valores a medida que se encuentra la igualdad entre el color del vértice y el color x ingresado.

Cantidad de Colores:

Como ventaja de la estructura del grafo, esta función solo retorna el valor de un campo (color_count), evitando buscar cuantos colores distintos existen entre los vértices.

Greedy:

Como no es necesario tener en cuenta ningún parámetro para la elección de los vértices, se colorea el primer vértice, y el resto de los vértices es coloreado con un color no válido (color 0) para indicar que no está coloreado con el color final.
Luego de esto se recorre el arreglo de los vértices sin necesidad de fijarse si está o no coloreado ya que colorea cada vértice que aparece en el arreglo según el orden en el que están.
Por cada vértice a colorear, para la correcta elección del color, lo que se hace es ir probando que el posible color con el que se coloreará el vértice no sea un color de alguno de sus vecinos. Si esto ocurre, el posible color es el siguiente, así hasta encontrar un color que cumpla la condición de que es distinto a el color de todos los vecinos.
En ese momento se colorea el vértice actual, y se compara el color reciente con el máximo número de color hasta el momento, para que al finalizar el algoritmo pueda retornar la cantidad de colores utilizados de una manera rápida y simple, sin necesidad de volver a fijarse en los colores de los vértices.

OrdenWheshPowell:

Se usa el algoritmo de ordenación “Quick Sort”, pero adaptado a lo que necesitamos lo cual es que nos ordene el arreglo de orden según el grado de los vértices del grafo (de mayor a menor), para esto se comparan los grados de los vértices y cuando uno tiene mayor grado que otro se intercambia los valores del arreglo de orden, así sucesivamente hasta que estén todos los vértices ordenados.
La ventaja que nos proporciona este algoritmo es que no complejidad es bastante buena, la cual seria:
En el mejor caso y caso promedio: O(cantidad_vertices * log (cantidad_vertices))
En el peor caso: O(cantidad_vertices^2)

DSATUR:

Inicialmente todos los colores de los vértices se inicializan para indicar que no están coloreados, o indicado con un color no válido (color 0) .
Como en un principio todos los vértices tienen el mismo grado de saturación, se busca entre los vértices aquel que tenga mayor grado, en caso de empates, se queda con el primero que encontró (un caso análogo a elegir por orden alfabético).
Una vez seleccionado el vértice, se colorea, y es sabido que todos sus vecinos tendrán grado de saturación 1, por lo que ese es el valor que se pone en el arreglo de grados de saturación en las posiciones correspondientes a los vértices vecinos del vértice coloreado.
Luego de esto, comienza el bucle que tendrá fin una vez que todos los vértices estén coloreados, chequeando esto viendo que ningún campo color de los vértices tenga un color no válido.
Para seleccionar el vértice a colorear, se agregó un arreglo auxiliar que contiene los grados de saturaciones de todos los vértices, siendo necesaria simplemente la búsqueda del mayor elemento del arreglo.

Para el caso de los empates, se busca entre los vértices que tienen ese grado de saturación, aquel que mayor grado (cantidad de vecinos) tenga.
Al igual que Greedy, en caso de empates, se queda con el primero encontrado.
Una vez seleccionado el vértice se realiza el mismo procedimiento de Greedy para la elección del color.
Una vez conocido el color, se debe actualizar el grado de saturación, cambiando su valor si es necesario.
Para esto, se recorren los arreglos de los vecinos de los vecinos del vértice coloreado, fijándose que el nuevo color no aparezca entre los colores de ellos.
De ser así el grado de saturación se mantiene. Caso contrario, se actualiza.
Al igual que en Greedy, para una fácil obtención del valor de retorno de la función indicando la cantidad de colores usados, se va guardando el máximo número de color utilizado hasta el momento.

GrandeChico:

Aquí nos creamos un arreglo auxiliar del tamaño de la cantidad de colores que tiene el grafo mas 1, esto es para poder utilizar los índices del arreglo como los colores que tiene el grafo y guardamos en cada celda la cantidad de vértices de color como lo indica el índice de esa celda.
De ahí recorremos el dicho arreglo, buscando el máximo elemento, y seteamos esa celda en 0 para que no la analice de nuevo en las próximas corridas, nos quedamos con la posición de esa celda (que nos indica el color) y buscamos en el arreglo de vértices, aquellos que tengan dicho color e intercambiamos en el arreglo de orden, seguimos de esa manera hasta que todas las posiciones del arreglo que mencionamos al principio estén en 0, con lo cual tendríamos ordenados los vértices según lo indica el enunciado.

ChicoGrande:

Es exactamente la misma idea que GrandeChico, excepto que en vez de buscar el máximo elemento, buscamos el mínimo elemento, y en vez setear la celda en 0, la seteamos en UINT32_MAX para que no la tenga en cuenta en las siguientes recorridas.

Revierte:

Se empieza buscando los vértices con el color K (el color mas grande), y una vez encontrados, se intercambia en el arreglo de orden, poniendo los vértices con dicho color al principio.
Luego se buscan los de color K – 1, y así sucesivamente hasta que ya no tenga mas colores.

OrdenAleatorio

Aquí creamos un arreglo auxiliar del tamaño de la cantidad de colores que tiene el grafo mas 1, esto es para poder utilizar los índices del arreglo como los colores que tiene el grafo, pero las celdas del arreglo se setean en 0 (ese color no fue visto, osea el índice) o en 1 (ese color ya fue visto).
Se tienen dos flags, uno que corresponde si se debe hacer intercambio en el arreglo de orden o no, en el primer caso es que el numero aleatorio obtenido no fue nunca visto (en esta situación el flag se setea en True para que pueda realizar el intercambio) y en el segundo caso, el numero aleatorio es ya fue obtenido anteriormente (aquí el flag se seta en False para que no realice el intercambio, si se da este caso hay que obtener otro numero aleatorio). El otro flag es para que la primera vez no se busque en el arreglo auxiliar ya que es la primera corrida, de ahí es mas se busca en el arreglo para ver si fue visto o no. El tema del intercambio es el mismo que siempre, se buscan los vértices con el color igual al numero aleatorio y se intercambia en el arreglo de orden, así sucesivamente hasta que no queden mas colores por ver.

Tarea de los integrantes

Alan Bracco:

* ImprimeGrafo,
* CantidadDeColores,
* Greedy,
* DSATUR,
* NumeroVerticesDeColor,
* ImprimirColor,

Mario Ferreyra

* LeerGrafo,
* OrdenWhelshPowell,
* GrandeChico,
* ChicoGrande,
* Revierte,
* OrdenAleatorio

Tanto estructura del grafo, como la de los vértices, las funciones no listadas anteriormente y el resto del proyecto, fueron pensadas y echas por ambos integrantes.