Ramo: Estructura de datos, Ingeniería en Computación, Universidad de la Serena
Prof: Mauro San Martín.
Por: Sebastian Rojas Cortez
Fecha: Lunes 14 de Septiembre, 2015.

Teoría de grafos
Métodos de recorrido BFS y DFS aplicados a la interacción entre alumnos de un curso

Comenzaremos anotando los datos entregados en la situación que se nos plantea:

> *En un curso G existen nueve alumnos (alumno A, alumno B, alumno C, alumno D, alumno E, alumno F, alumno G, alumno H, alumno I), los cuales no todos interactuad directamente entre sí. En este curso encontraremos tres grupos, de tres alumnos cada uno, en donde dentro de cada grupo todos los alumnos interactúan directamente entre los del mismo grupo, a excepción de ellos mismos. Los grupos son los siguientes:*
>
> *Grupo 1:* *alumno A, alumno B y alumno C*
>
> *Grupo 2:* *alumno D, alumno E y alumno F*
>
> *Grupo 3:* *alumno G, alumno H y alumno I*
>
> *Dentro de cada grupo puede existir uno o más alumnos que interactúen con otros alumnos relacionados a los demás grupos. En este caso los alumnos que se relacionarán a alumnos de otros grupos serán los siguientes:*
>
> *Alumno A: Interactuará con todo su grupo (alumno B y alumno C), sumando al alumno D y al alumno E*
>
> *Alumno D: Interactuará con todo su grupo (alumno E y alumno F), sumado el alumno A*
>
> *Alumno E: Interactuará con todo su grupo (alumno D y alumno F), sumado el alumno A*
>
> *Alumno F: interactuará con todo su grupo (alumno D y alumno E), sumado el alumno I*
>
> *Alumno I: Interactuará con todo su grupo (alumno G y alumno H), sumado al alumno F*

Por consiguiente , como podemos ver, tendremos las siguientes interacciones entre los alumnos del curso:

Alumno A interactúa con:	*alumno B, alumno C, alumno D y alumno E*
Alumno B interactúa con:	*alumno A y alumno C*
Alumno C interactúa con:	*alumno B y alumno C*
Alumno D interactúa con:	*alumno A, alumno E y alumno F*
Alumno E interactúa con:	*alumno A, alumno D y alumno F*
Alumno F interactúa con:	*alumno D, alumno E y alumno I*
Alumno G interactúa con:	*alumno H y alumno I*
Alumno H interactúa con:	*alumno G y alumno I*
Alumno I interactúa con:	*alumno F, alumno G y alumno H*

Según la situación planteada anteriormente podemos sacar varios datos y relacionarlos a la teoría de grafos. Por ejemplo:

Podemos representar el curso G como un grafo del mismo nombre (G), siendo así cada alumno uno de sus nodos, mientras que las interacciones podrías ir representadas en grafo como los arcos.
Siendo de la siguiente manera:

Curso G = G (V , A)

Donde V es el total de nodos y A el total de aristas. V contaría con un total de 9 nodos, los que se relacionan al total de los alumnos en el curso G de la siguiente manera:

Alumno A = Nodo A

Alumno B = Nodo B

Alumno C = Nodo C

Alumno D = Nodo D

Alumno E = Nodo E

Alumno F = Nodo F

Alumno G = Nodo G

Alumno H = Nodo H

Alumno I = Nodo I

Siguiendo con lo mencionado, y teniendo ya relacionado cada nodo con un alumno, procedemos a relacionar las aristas del grafo con las interacciones entre los alumnos, y ocupamos estas aristas para hacer las uniones entre los nodos del grafo acorde al alumno que representan. Esto sería de la siguiente manera:

Nodo A comparte arista con: nodo B, nodo C, nodo D y nodo E

Nodo B comparte arista con: nodo A y nodo C

Nodo C comparte arista con: nodo B y nodo C

Nodo D comparte arista con: nodo A, nodo E y nodo F

Nodo E comparte arista con: nodo A, nodo D y nodo F

Nodo F comparte arista con: nodo D, nodo E y nodo I

Nodo G comparte arista con: nodo H y nodo I

Nodo H comparte arista con: nodo G y nodo I

Nodo I comparte arista con: nodo F, nodo G y nodo H

Puesto ya los datos en nuestro grafo procedemos a definir las estructuras.

1.- Estructura tipo Nodo (V):

```
typedef struct tipo_nodo{
        char letra;
        int prof;
        int pos;
        int color;
        }       Alumno;
```

En la estructura tipo nodo, como podrán ver en la imagen, cuenta con cuatro datos primitivos, los cuales son Letra (tipo carácter), y prof, pos y color. Estos últimos tres de tipo entero. Inicializaremos color 0, prof en -1 y pos será acorde a la posición que ocupará el nodo en nuestra matriz de adyacencia y arreglo de alumnos que vendrá contenida en el grafo, y así en cada nodo.

```
(alumnoS[0].letra)='a'; (alumnoS[0].prof)=-1;(alumnoS[0].pos)=0;(alumnoS[0].color)=0;
(alumnoS[1].letra)='b'; (alumnoS[1].prof)=-1;(alumnoS[1].pos)=1;(alumnoS[1].color)=0;
(alumnoS[2].letra)='c'; (alumnoS[2].prof)=-1;(alumnoS[2].pos)=2;(alumnoS[2].color)=0;
(alumnoS[3].letra)='d'; (alumnoS[3].prof)=-1;(alumnoS[3].pos)=3;(alumnoS[3].color)=0;
(alumnoS[4].letra)='e'; (alumnoS[4].prof)=-1;(alumnoS[4].pos)=4;(alumnoS[4].color)=0;
(alumnoS[5].letra)='f'; (alumnoS[5].prof)=-1;(alumnoS[5].pos)=5;(alumnoS[5].color)=0;
(alumnoS[6].letra)='g'; (alumnoS[6].prof)=-1;(alumnoS[6].pos)=6;(alumnoS[6].color)=0;
(alumnoS[7].letra)='h'; (alumnoS[7].prof)=-1;(alumnoS[7].pos)=7;(alumnoS[7].color)=0;
(alumnoS[8].letra)='i'; (alumnoS[8].prof)=-1;(alumnoS[8].pos)=8;(alumnoS[8].color)=0;
```

[Imagen de como se inicializaron los nodos alumnos en el programa]

2.- Estructura tipo Grafo (G)

```
typedef struct tipo_grafo{
        int interaccion [tam][tam];
        Alumno alumnos[tam];
        }       Curso;
```

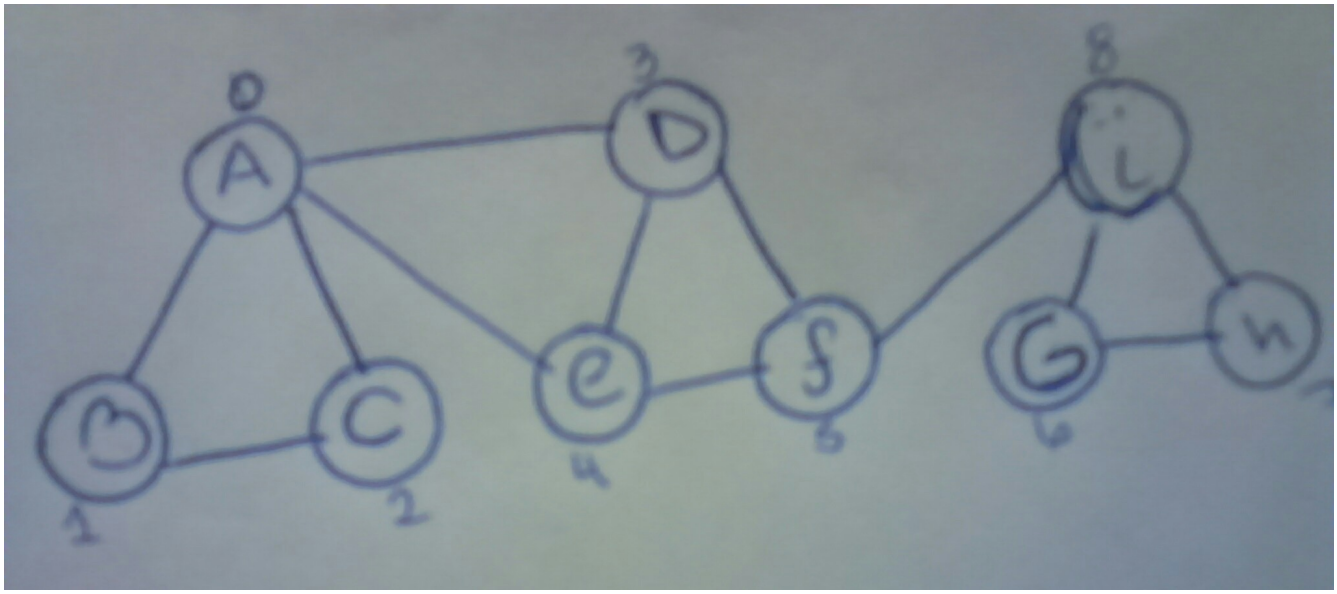
La Estructura de tipo grafo, será un poco más compleja y como ven cuenta con dos tipos de variable en forma de arreglos. Un arreglo tipo entero de dos dimensiones llamado interacción, donde asignaremos una cola y una fila a cada nodo, y así podremos tener guardada de forma binaria si estos comparten arista con otros marcando la posición de la matriz con un 1 y un 0 si no. El otro arreglo será de tipo Alumno (nodo), en este tendremos guardado cada nodo del programa. Estos arreglos tendrán una dimensión 'tam', siendo tam igual a la cantidad de nodos (alumnos) que tenga el grafo (curso).

```
 a.b.c.d.e.f.g.h.i
 0.1.2.3.4.5.6.7.8
{0,1,1,1,1,0,0,0,0},//0a
{1,0,1,0,0,0,0,0,0},//1b
{1,1,0,0,0,0,0,0,0},//2c
{0,0,0,0,1,1,0,0,0},//3d
{0,0,1,1,0,1,0,0,0},//4e
{0,0,0,1,1,0,0,0,1},//5f
{0,0,0,0,0,0,0,1,1},//6g
{0,0,0,0,0,0,1,0,1},//7h
{0,0,0,0,0,5,1,1,0} //8i
};
```

[Imagen de la matriz de adyacencia en el programa]

En cuanto a como llenamos el arreglo alumnos de tipo Alumno, se puede ver en la imagen anterior a la matriz, donde se mostraba como inicializábamos cada nodo (alumno).

Ahora que tenemos todos nuestros datos relacionados, proseguiré a mostrar la imagen del grafo que represento en mi programa.



Una vez definidas e inicializadas las estructuras, procedemos a trabajar en los algoritmos de recorrido de grafo asignados. BFS y DFS.

BFS

Comenzaremos con la implementación del Algoritmo BFS, el cual en mi caso me tomó más tiempo realizar (al rededor de 5 días).

Comencé el mismo día definiendo las estructuras y durante la semana intenté implementar mi propio algoritmo que llevara a cabo un recorrido de tipo BFS, pero al darme cuenta que el tiempo no estaba de mi lado y que el algoritmo que estaba desarrollando no lograba satisfacer las necesidades del BFS, decidí programar un algoritmo que ya estuviese hecho en internet. En mi caso tomé el siguiente algoritmo como referencia.

```
1  método BFS(Grafo,origen):
2     creamos una cola Q
3     agregamos origen a la cola Q
4     marcamos origen como visitado
5     mientras Q no este vacío:
6        sacamos un elemento de la cola Q llamado v
7        para cada vertice w adyacente a v en el Grafo:
8           si w no ah sido visitado:
9              marcamos como visitado w
10             insertamos w dentro de la cola Q
```

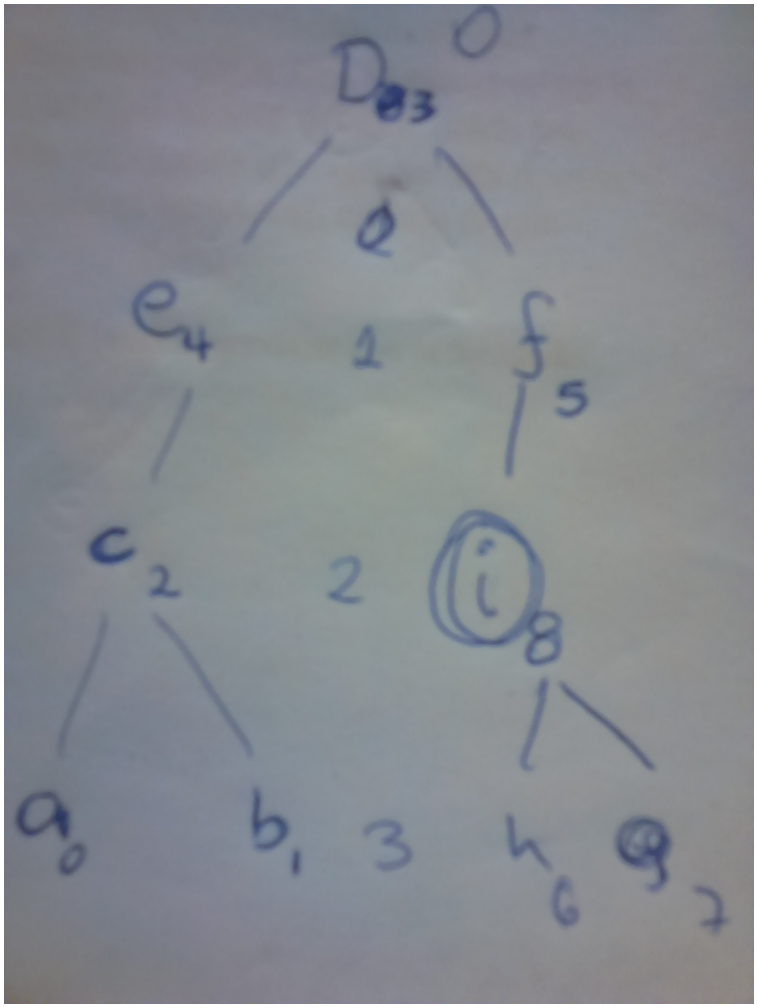
No encontré que el algoritmo fuera difícil de comprender, al contrario, se me hizo muy simple, pero al momento de programarlo, algunos problemas con el lenguaje que intenté resolver agregando variables me hicieron tardar más de la cuenta, ya que por problemas tontos de sintaxis no lograba compilar el programa y el haber agregado demasiadas variables me produjo un enredo en la programación. Pero llegado el momento de resolver mis dudas sobre la sintaxis y luego de un buen debugeo de variables logré hacer que corriera mi programa. El resultado de mi código fue el siguiente:

```
115 void metodo_BFS (Curso curso){
116         int lugar, profundidad=lugar=0;
117         Alumno Q[tam];          inicializar_Q(Q);       Alumno alumno;
118         curso.alumnos[3].prof=profundidad; curso.alumnos[3].color=1;
119         Q[0]=curso.alumnos[3];  lugar++;
120         curso.alumnos[(Q[0].pos)].prof=profundidad;
121         while (Q_vacio(Q)==1){
122                 curso.alumnos[(Q[0].pos)].color++;
123                 Q[0].color++;   alumno=nuevo_v(Q,&lugar);
124                 if (  alumno.prof==Q[0].prof || profundidad==0){
125                         profundidad++;
126                 }
127                 if (existen_adyacentes(alumno, curso)==1){
128                         marco_adyacentes(&curso,Q,alumno,profundidad,&lugar);
129                 }
130         }
131         imprimir_curso(curso);getchar();
132 }
```

Como se puede apreciar en el código, comencé a recorrer el grafo curso a partir del alumno[3], qué vendría siendo el alumno D. Definí una cola auxiliar llamada Q de tamaño tam (9), en cual iba metiendo todos los nodos marcados como grises (color=1), y al momento de sacar el primero de la cola, ese quedaba automáticamente marcado de color negro (color=2). Todo estro ocurría dentro de un ciclo, que tenía como condición repetirse mientras la cola Q no estuviese vacía.

Entonces, primero (Linea 122 y 123) marcaba de negro el alumno en la estructura curso, luego hacía lo mismo con el alumno de la cola Q (Esto dos veces debido a que copiaba el alumno y al hacerle la modificación a uno, no se le hacía al otro, por lo que debía hacerlo en ambas estructuras. Esto pudo deberse a una poco eficiente utilización de las variables). Segundo, sacaba el primer valor de la cola (Linea 123), y seguían dos sentencias de evaluación. La primera me sirvió para aumentar la profundidad de los nodos que iba a marcar solo en caso de que la profundidad del elemento fuera 0 (osea la primera pasada) o que el nodo que seguía para evaluar no era de la misma profundidad del que se estaba evaluando, lo que significaba un salto de nivel en el recorrido. La segunda sentencia de evaluación me sirvió para evaluar si habían y/o quedaban nodos que no estuvieran marcados y compartieran arista con el nodo que estaba siendo evaluado, en caso de que quedaran, estos eran automáticamente marcados y agregados a la cola Q. Y así se repetía hasta que la cola Q estuviese vacía. Esto código puede dar problemas en caso de que el grafo que se entregue no esté del todo unido por aristas, ya que no lograría llegar a todos los nodos, para solucionar esto habría que agregar una función que revisara si todos los nodos del grafo han sido agregados a la cola (o marcados), en caso de que alguno no, se debería de ingresar automáticamente a ella.

En conclusión, el recorrido de mi grafo, partiendo por el alumno[3] (nodo), quedaría con la forma del siguiente árbol.



De esta manera podríamos ver y saber la distancia que existe entre cada alumno de un curso, en caso de que uno necesitada algún dato del otro, ya sea en favores, actitudes, conocimientos, etc. De esto se puede además inferir información, como forma de relacionarse y mucho más.

DFS

Este problema me tomó un poco menos de tiempo resolverlo y programardo, algo así como 50 minutos, esto pudo ser debido al fácil entendimiento de su funcionalidad una vez que ya tenía definida las estructuras y la manera de recorrer un grafo se me hizo mucho más fácil.

Me basé en vídeos vistos en youtube de como se recorría el grafo, por lo que pensé en usar recursividad y mi algoritmo resultante fue el siguiente.

```
134 void metodo_DFS(Curso *curso, int profundidad, int pos){
135         (*curso).alumnos[pos].prof=profundidad;
136         (*curso).alumnos[pos].color=2;
137         imprimir_alumno((*curso).alumnos[pos]);
138         //MARCO
139         for (int i=0; i<tam; i++){
140                 if ((*curso).alumnos[i].color!=2 && (*curso).interaccion[pos][i]==1){
141                         pos=i; profundidad++;
142                         metodo_DFS(&(*curso), profundidad, pos);
143                 }
144         //BUSCO POR VERTICE. con un ciclo.
145         }
146         //RETORNO
147 }
```

Como se puede ver, paso el grafo por referencia para asegurar de que al retornar los nodos ya estén marcados. Al entrar a la función primero marco el nodo de inicio, luego entro a dos for anidados para poder recorrer la matriz, y cuando encuentro un nodo adyacente al actual vuelvo a repetir la función en base al nodo encontrado. Así hasta ya no encontrar más nodos adyacentes en el nodo actual y comenzar a retornar y buscar en los anteriores (algo así como 1.marcar, 2.izquierda, 3.derecha, 4.retornar de forma recursiva). El contador en el que iba encontrando los nodos los guardé en la variable prof, la cual quizás no fue la más apta (por el nombre), pero pensé que sería más practico que declarar una nueva variable.

El paso por valor de variables como profundidad me fue necesario para ir marcando los nodos y pos, para saber qué nodo dentro del arreglo alumnos era el actual a marcar y revisar sus adyacentes dentro de la recursividad. En este método utilicé el mismo grafo curso y comencé por el mismo nodo alumno[3] (D), por lo que las imágenes referentes al grafo son las mismas.