******REPÚBLICA BOLIVARIANA DE VENEZUELA**

**MINISTERIO DEL PODER PUPULAR PARA LA DEFENSA**

**UNIVERSIDAD NACIONAL EXPERIMENTAL POLITÉCNICA**

**DE LA FUERZA ARMADA NACIONAL BOLIVARIANA**

**NÚCLEO FALCON - EXTENSIÓN PUNTO FIJO**

Enunciado 3

Grafos Bipartitos

**Integrantes**

* Chirinos, Enmanuel. 26.058.259
* Petit, Anyelo. 26.057.580
* Vargas, Paola. 25.723.540

Ingeniería de Sistemas,

5to Semestre, Sección “A”.

Punto Fijo, Mayo del 2017

**TEORÍA APLICADA PARA LA RESOLUCIÓN DE LOS PROBLEMAS**

Dos términos resaltantes, en lo que grafos se refiere, fueron estudiados con detalle para la resolución de la problemática planteada. La primera determinante que nos presenta el enunciado es que toda la resolución será girará en torno a lo que se denomina como un grafo simple. La teoría referente a este término nos indica que un grafo simple no es más que un grafo que no posee ni lazos (bucles) ni aristas múltiples, esta última característica nos dice que no debe existir más de una arista que conecte dos vértices cualquieras, por ende, desde vértice A hacia el vértice B solo debe existir una arista que los conecte.

Siendo este el primer condicional, del cual derivan otros, que el ejercicio nos plantea proseguimos a lo que en realidad éste nos pide, siendo el objetivo verificar si es o no bipartito un grafo dado. Otra teoría en la cual fijarnos es que un **grafo bipartito** se le denomina a un grafo cuyos vértices pueden ser separados en dossubconjuntos disjuntos, V1 (G) y V2 (G), como se muestra en la Fig.1, tomando en cuenta que las aristas siempre unen vértices de un subconjunto con vértices de otro subconjunto, es decir, que la unión de estos subconjuntos es la cantidad de vértices o nodos totales que el grafo posee y que su intersección es vacía.

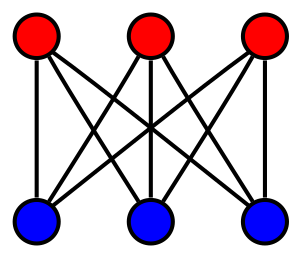


Fig. 1

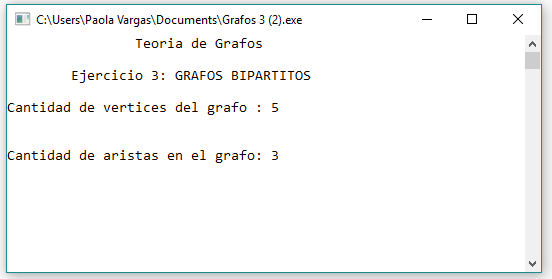
El problema de verificar si es un grafobipartito o no, es equivalente al problema de colorabilidad con dos colores, pero con diferente nomenclatura. Cualquier grafo con un ciclo de longitud impar es claramente no colorable para dos colores, por ende, podemos decir que un grafo que no es colorable con dos colores posee ciclos impares en él y de esta manera, no puede ser un grafo bipartito.

Para colorear un grafo con dos colores, dado un color asignado a un vértice v, debe colorearse el resto del grafo asignando el otro color a cada vértice adyacente a v. Este proceso es semejante a alternar los dos colores para los nodos en cada nivel de un recorrido DFS (**D**epth **F**irst **S**earch), y durante el retorno de las llamadas recursivas, verificar si no hay arista que conecte dos nodos del mismo color, ya que tal arista es evidencia de un ciclo de longitud impar.

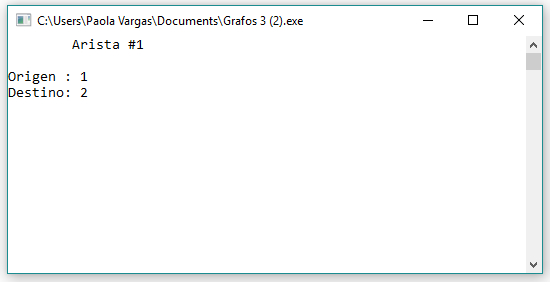
Recorrer un grafo consiste en “visitar” cada uno de los nodos a través de las aristas del mismo. Se trata de realizar recorridos de grafos de manera eficiente. Para ello, se pondrá una marca en un nodo en el momento en que éste es visitado, de tal manera que, inicialmente, no está marcado ningún nodo del grafo.

**ESPECIFICACIONES DE ENTRADA**

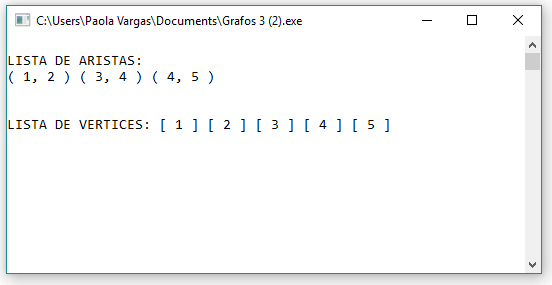
El tipo de entrada establecido en la solución del problema es dictada por el enunciado, diciendo éste que el grafo deber ser introducido por lista de aristas, entonces se puede asumir que la entrada presente en el algoritmo serán de pares ordenados, recibiendo como primer término del par ordenado (x), el vértice inicial de la arista, y como segundo término (y), el vértice final de ésta.

La primera entrada que pide el archivo de extensión .exe en ejecución con la solución del ejercicio será la cantidad de vértices que tendrá el grafo, almacenando dicho dato en una variable. El siguiente dato solicitado será la cantidad de aristas que el grafo poseerá.

Por consiguiente, comenzará a solicitar los términos de cada arista por lista de pares ordenados, enumerados los vértices automáticamente del número 1 hasta *n* cantidad establecida en la primera variable, la cual lo especifica la cantidad de vértices que insertó el usuario.

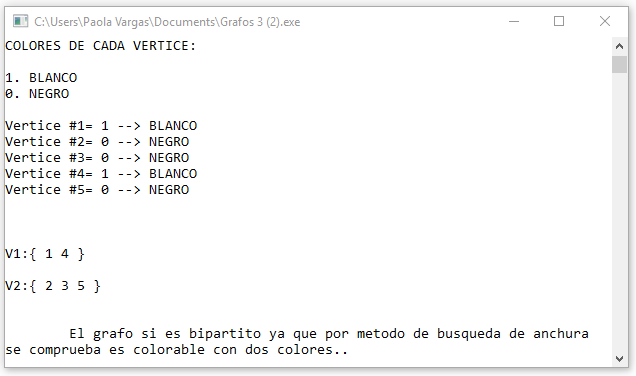


**ESPECIFICACIONES DE SALIDA**

La salida que es generada por el algoritmo de acuerdo al problema en general muestra la secuencia de pasos que sigue para la resolución de éste, donde primeramente al terminar la introducción de todos los datos, se genera una lista de aristas por pares ordenados: “(1,2) (3,4) (4,5)”. Siguiente a esa impresión de datos se muestra la lista de todos los vértices presentes dentro del grafo cada uno dentro de sus respectivos corchetes “[1] [2] [3] [4] [5]”.

Comienza con el proceso de recorrido DFS, el cual colorea de manera lógica cada vértice en base a dos colores, y dependiendo del color con el que se coloreó el nodo anterior. Imprimiendo el valor de cada variable mostrando el proceso del algoritmo paso por paso.

El valor AUX impreso es el nodo en el que está situado actualmente el algoritmo, debajo muestra el tamaño del vector antes de agregar los vértices incidentes con el nodo instanciado en AUX y se muestra los nodos que se encuentran dentro del vector mismo. Luego de haber agregado los vértices incidentes al vector, se imprime nuevamente el tamaño del mismo y los nodos que ahora residen. Para finalizar el bloque de código, se imprime por última vez los vértices dentro del vector, para verificar que el primer nodo del vector que, en este caso, y en los siguientes será AUX, sea eliminado y la variable AUX tome como valor el nodo siguiente.

Finalizado el proceso de coloreo lógico de vértices y de evaluar cada condición, dependiendo de cuál sea el resultado del recorrido DFS, se da a conocer un resultado final, en caso de que el grafo dado resulte no ser bipartito, se imprime por pantalla una justificación pertinente de acuerdo al caso que se presente, y en caso contrario, se procede a mostrar los vértices con sus respectivos colores y a su vez, muestra los subconjuntos V1 y V2, que son formados a partir de la relación entre ellos.

**ESTRUCTURAS DE DATOS UTILIZADAS**

**Variables de tipo ENTERO**

**aristas=0:** variable que almacena la cantidad de aristas del grafo

**vertices=0:** variable que almacena cantidad de vértices en el grafo

**fir:** Variable que almacena temporalmente el primer vértice que pertenece a la arista a insertar

**sec:** Variable que almacena temporalmente el segundo vértice que pertenece a la arista a insertar

**limite:** variable que posee la cantidad de aristas máxima que puede poseer un grafo con respecto a los vértices dados

**cont=0:** Variable bandera que señala el hecho de que posea vértices aislados

**flag1=0:** Variable bandera que se incrementa en caso de que el primer vértice se encuentre ya en el arreglo de vértices.

**flag2=0:** Variable bandera que se incrementa en caso de que el segundo vértice se encuentre ya en el arreglo de vértices.

**indice=0:** Variable utilizada como índice en los diferentes ciclos presentes en el algoritmo.

**index=0:** Variable utilizada como índice en los diferentes ciclos presentes en el algoritmo.

**i=0:** Variable utilizada como índice en los diferentes ciclos presentes en el algoritmo.

**j=0**: Variable utilizada como índice en los diferentes ciclos presentes en el algoritmo.

**primero:** Variable que instancia posición inicial en la que comenzará el recorrido.

**aux:** Variable que cumple el papel de tomar temporalmente el primer valor del vector **“asd”:** para poder manejar las adyacencias a él.

**Variables de tipo ARREGLO**

**arrayV[vertices]:** Arreglo que contiene los vértices presentes en el grafo.

**arrayV2[vertices]**: Arreglo que contiene los vértices que se encuentran relacionados entre ellos presentes en el grafo.

**visited[vertices]:** Un arreglo el cual es inicializado en 0 cada espacio en el para llevar un control de los vértices que ya han sido visitados y cuáles no.

**colors[vertices]:** Un arreglo el cual es inicializado en 0 cada espacio en el para llevar un control de los colores lógicos de cada vértice del grafo. Siendo el número 1 blanco y el número 2 negro.

**Variables de tipo VECTOR**

**vert:** Vector que almacena por pares ordenados las aristas ingresadas por el usuario.

**asd:** Vector por el cual se maneja los vértices en los que se encuentra el recorrido y los vértices adyacentes a él que son agregados.

**Variables de tipo PAR (PAIR)**

Librería estática la cual permite almacenar los números por pares ordenados dentro de una misma variable

**Variables de tipo BOOLEANO**

**validation=false:** Variable que se utiliza para las diferentes validaciones al momento de ingresar aristas. La cual es cambiada a verdadero (true) en caso de que apruebe las validaciones.

**tree=false:** Variable bandera que indica cuando un grafo al saber sus vértices y cantidad de aristas se deduce que es un árbol.

**bipartito=true:** Variable booleana que permite indicar si el grafo cumple con las condiciones a la que es sometido para verificar si es bipartito o no.

**ALGORITMO DISEÑADO PARA RESOLVER EL PROBLEMA**

Método **DFS** (origen):

creamos una pila S

agregamos origen a la pila S

marcamos origen como visitado

**mientras** S no esté vacío:

sacamos un elemento de la pila S llamado v

**Para** cada vértice w adyacente a v en el Grafo:

**Si** w no ha sido visitado:

marcamos como visitado w

insertamos w dentro de la pila S