CENTRO DE CIENCIAS BÁSICAS

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

OPTIMIZACIÓN INTELIGENTE

5° "A"

PRÁCTICA 3: COLORACIÓN DE UN GRAFO

Profesor: Aurora Torres Soto

Alumno: Joel Alejandro Espinoza Sánchez

Fecha de Entrega: Aguascalientes, Ags.**, 5** de octubre de 2020

Práctica 3: Coloración de un grafo

Objetivo:

Mediante el desarrollo de esta práctica, implementar el algoritmo de la sección.

Introducción:

La coloración de los vértices de un grafo es un caso especial de etiquetado de grafos que consiste en la asignación de etiquetas llamadas colores a los vértices, de manera tal que ningún vértice adyacente comparta el mismo. Similarmente, una coloración de aristas asigna colores a cada arista de forma tal que aristas adyacentes no compartan el mismo color.

Pregunta de Investigación:

¿De qué manera pueden adaptarse las ideas de las matemáticas discretas para la coloración de grafos en un algoritmo de programación para que una computadora desarrolle el método de forma más eficiente?

Predicción:

Creo que la complejidad de este problema requerirá herramientas de modelado y formas de representación de la solución que nos pueda servir para poder llevar a cabo la coloración de un grafo determinado.

Materiales:

Una computadora con compilador de C.	Dos hojas de papel
Una calculadora.	Lápiz o plumas

Método (Variables):

Dependiente: El resultado que arroje el programa según la coloración del grafo encontrada.
Independiente: El algoritmo que uno como alumno se focalizará en elaborar.
Controlada: El grafo que se usará para evaluar.

Seguridad:

Realmente no se trabajó en campo, por lo que no se corren riesgos al elaborar el experimento.

Procedimiento:

1.- Haciendo uso del lenguaje de programación C, se realizó un programa que implemente la siguiente secuencia de eventos.

- Se solicita la información del grafo y se construye su matriz de adyacencias. El programa supone un máximo de 15 vértices y que los grafos son no dirigidos.

- Se ordenan los vértices por grado de mayor a menor.
- Se colorea el grafo en el orden dispuesto en el paso anterior.
- Se muestra el coloreo en la forma de una tabla con: Vértice y Color.

2. Posteriormente, se implementó el algoritmo de Welsh y Powell. Donde el algoritmo es el siguiente:

a) Se ordenan los vértices de mayor a menor grado
b) Se selecciona el primero en la lista y se colorea o se etiqueta con el primer color disponible.
c) Se toma el siguiente vértice y se colorea o etiqueta con el primer color admisible; es decir, se verifican adyacencias. Suponga que las etiquetas de los colores corresponden con enteros de 0 en adelante.
d) Se repite el paso (c) hasta que todos los vértices hayan sido coloreados.
e) Se imprime la solución.

3.- El programa se probó con algunos grafos realizados previamente bajo este algoritmo. Los resultados se procesaron en el presente documento a continuación.

Obtención y Procesamiento de Datos:

La elaboración del código en C (véase anexo 1) fue más compleja que en otras prácticas. Tomando las partes más importantes del código (véase anexo 2 para las capturas a mayor detalle del código), el primer paso que realiza el programa es pedir los datos del grafo para llevarlos a una matriz de adyacencia como puede observarse en la figura 1.

```
//Procedemos a llenar la matriz de adyacencia
for(i = 0; i < orden; i++)
{
    j = i;
    do
    {
        printf("--------------------------------\n");
        printf("¿El vértice %d está conectado con el vértice %d?\n",i+1,j+1);
        printf("1. Sí\n");
        printf("2. No\n");
        scanf("%d",&f);
        if(f == 1)
        {
            matady[i][j] = 1;
            matady[j][i] = 1;
        }
        if(f == 2)
        {
            matady[i][j] = 0;
            matady[j][i] = 0;
        }
        j++;
    }
    while(j < orden);
    matady[i][orden] = 0;
    vertices[i] = i + 1;
}
```

Figura 1: Sección de código correspondiente a la calibración del grafo en la que se determinan las uniones de los vértices.

De modo que, acabado este punto, el programa tiene registrado un grafo con un número n de vértices guardados en una matriz de $n \times n + 1$, esto porque existe un renglón extra para guardar el total de conexiones que cada vértice tiene. El llenado de este renglón extra se realiza en un ciclo aparte.

```
//La matriz tiene un renglón de más hecho para guardar el total de conexiones. Esto se calculará a continuación
for(i = 0; i < orden; i++)
{
    for(j = 0; j < orden; j++)
    {
        if(matady[i][j] == 1)
        {
            matady[i][orden] = matady[i][orden] + 1;
        }
    }
}
```

Figura 2: Sección de código correspondiente al llenado del renglón extra de la matriz, el cual indica el total de conexiones de cada vértice.

Una vez que se llega aquí, la matriz es dispuesta en pantalla puesto que el renglón extra se verá modificado a partir de este punto.

En los pasos anteriores se han tratado dos estructuras de datos llamados `matady` (que es la estructura en el código que guarda la matriz de $n \times n + 1$) y `vertices`, que entre ellos existe una biyección de existencia, pues mientras se guardaban los datos de la matriz, también se fue llenando el arreglo llamado `vertices` ordenadamente, pues como la columna $n + 1$ de la matriz y el arreglo tienen la misma longitud y en este punto se encuentran ordenados, diremos que la biyección presentada aquí es que el orden del arreglo `vertices` representará el vértice correspondiente a la cantidad de conexiones en la columna $n + 1$ de la matriz.

Por lo que en el siguiente paso que será ordenar de mayor número de conexiones a menor, se modificarán ambas estructuras simultáneamente, utilizando el algoritmo del método "*Bubble Sort*" para llevar a las primeras posiciones los números mayores. Así como las comparaciones son entre el elemento i y el $i + 1$, cuando se haga un cambio entre los números de la columna $n + 1$ de la matriz entre el elemento i y el $i + 1$, también se hará el mismo cambio en el arreglo `vertices` con los mismos elementos. Este proceso se observa en la figura 3.

```
//Ordenaremos los vértices de mayor número de conexiones a menor
for(i = 0; i + 1 < orden; i++)
{
    if(matady[i][orden] < matady[i + 1][orden])
    {
        u = matady[i + 1][orden];
        matady[i + 1][orden] = matady[i][orden];
        matady[i][orden] = u;

        u = vertices[i + 1];
        vertices[i + 1] = vertices[i];
        vertices[i] = u;

        i = 0;
    }
}
```

Figura 3: Sección de código correspondiente a ordenar la estructura `vertices` y `colores`.

Para comenzar el coloreo de grafos, se usarán dos nuevas estructuras. La primera llamada `colores` que se trata de un arreglo igualmente biyectivo entre los elementos de éste y los elementos del arreglo `vértices`; pues la intención es que cada elemento del arreglo `colores` indique el color correspondiente de su respectivo elemento en el arreglo `vertices`.

También se añade la estructura `coloresaux` que va a guardar de cada iteración todos los colores registrados anteriormente como aquellos que tocan al vértice que se está analizando actualmente, para no elegir los colores dentro de este arreglo.

Conociendo estas dos estructuras, lo primero que se realiza es asignarle el primer color al vértice al principio del arreglo `vertices`, pues éste es el de mayor número de conexiones. Posteriormente se revisará cada uno de los vértices. En cada vértice se revisarán todos los anteriores hasta el anterior al que se está revisando actualmente. Si los dos vértices que se comparan, dentro de la matriz de adyacencia muestran que sí se encuentran conectados, el color se guardará en lo más próximo del arreglo `coloresaux`. Este proceso se hace en cada comparación, en todos los vértices.

Para cada vértice, también se revisará el arreglo `coloresaux` y buscará el número mayor que existe en este arreglo, se le sumará uno y este número será el que se le asignará a su respectivo espacio dentro del arreglo `colores` para obtener finalmente el arreglo que indica de manera biyectiva la correspondencia entre los elementos del arreglo `vertices` y los elementos del arreglo `colores`.

```
//Comienza el coloreo de grafos
colores[0] = 1; //El primer vértice es fácil pues no se ha revisado ninguno anteriormente

//Se revisará de uno en uno los próximos vértices en orden como se guardaron por el arreglo vertices
for(i = 1; i < orden; i++)
{
    u = 0;

    //En cada revisión tenemos que checar todos los vértices previos a éste
    for(j = 0; j < i; j++)
    {

        //Si el vértice j tiene conexión con el vértice i, entonces guardaremos en coloresaux ese color como color imposible
        if(matady[vertices[i] - 1][vertices[j] - 1] == 1)
        {
            coloresaux[u] = colores[j];
            u++;
        }
    }

    colores[i] = 1;
    for(j = 0; j < i; j++)
    {
        if(coloresaux[j] == colores[i])
        {
            colores[i] = colores[i] + 1;
            j = 0;
        }
    }
}
```

Figura 4: Sección de código correspondiente a al coloreo de los vértices del grafo.

El último paso es mostrar estos resultados en pantalla.

Para entender en primera instancia previa a la implementación en el lenguaje C, se había realizado un pequeño bosquejo a mano (véase anexo 4) con un grafo pequeño, el cual se observa en la figura 5.

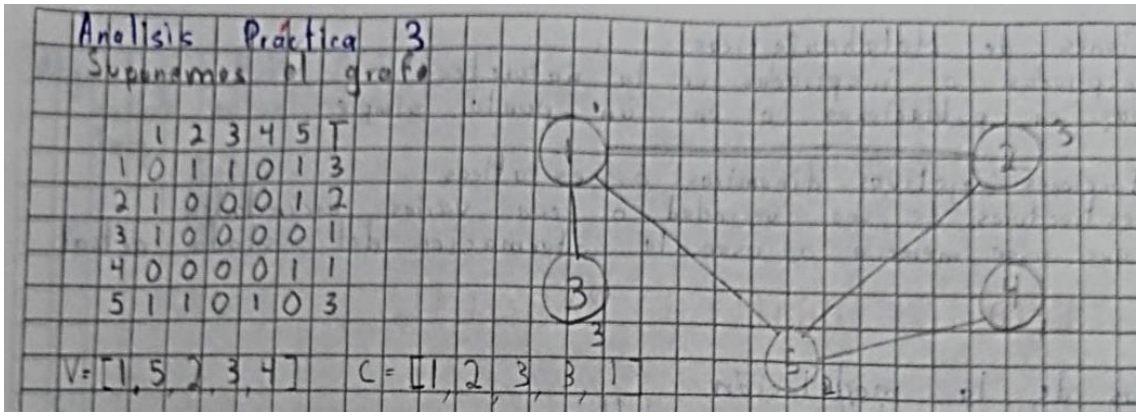


Figura 5: Grafo de prueba para entender el algoritmo.

En la figura 5 se observa el grafo. Dentro de cada vértice se señala el número del vértice; afuera de cada vértice está el color final asignado, que aunque no se alcanza a ver, existe anotado y se explicará más adelante.

Al lado del grafo se tiene su matriz de adyacencia y debajo se tiene el equivalente al arreglo `vertices` ordenado y el arreglo `colores` con los colores asignados. Como observamos y se ha recalcado a lo largo del documento, la biyección en este caso representa que al vértice 1 se le asignará el color 1, al vértice 5 se le asignará

el color 2, al vértice 2 se le asignará el color 3, al vértice 3 también se le asignará el color 3 y finalmente al vértice 4 se le asignará el color 1.

De forma más gráfica, podemos decir que el color 1 representa el rojo, el color 2 simboliza al verde y el azul está representado bajo el nombre del color 3. Observemos entonces que el grafo obtenido es el que se muestra en la figura 6.

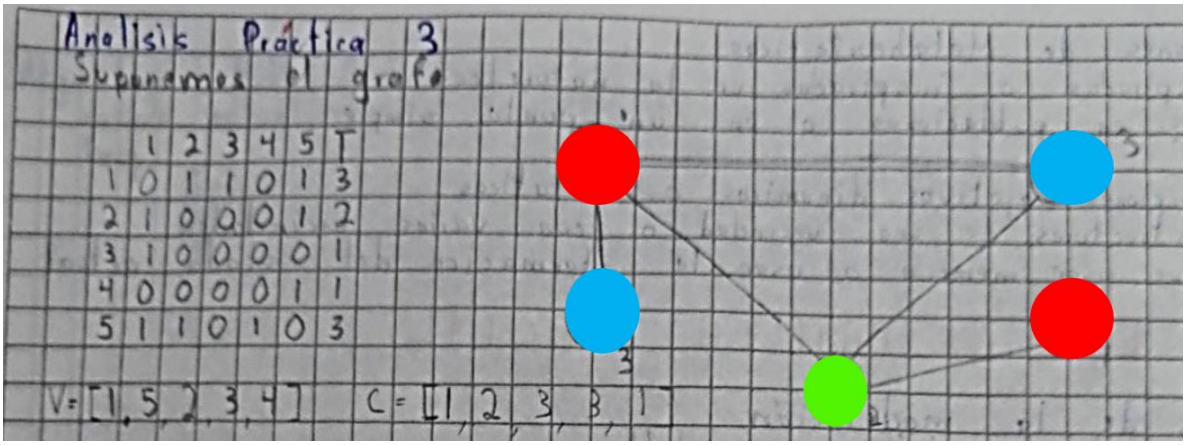


Figura 6: Simulación de coloreo de grafos del mostrado en la figura 5.

El mismo resultado de los arreglos `vertices` y `colores` deberían desplegarse en el programa. Agregando este grafo al programa observamos los resultados que se encuentran en la figura 7 (si se quiere probar el código en el anexo, para evitar aclarar cada vértice de este grafo en específico, la calibración de este mismo grafo puede hacerse insertando la línea: `5 2 1 1 2 1 2 2 2 1 2 2 2 2 1 2`).

```
Matriz de adyacencia:
[0] [1] [1] [0] [1] [3]
[1] [0] [0] [0] [1] [2]
[1] [0] [0] [0] [0] [1]
[0] [0] [0] [0] [1] [1]
[1] [1] [0] [1] [0] [3]
--------------------------------
El orden de los vértices según el algoritmo de Welsh Powell es el siguiente:
V = [1, 5, 2, 3, 4]
--------------------------------
El color de los vértices según el algoritmo de Welsh Powell es el siguiente:
C = [1, 2, 3, 3, 1]

--------------------------------
```

Figura 7: Simulación de coloreo de grafos del mostrado en la figura 5 dentro del programa implementado.

Observamos que la misma biyección existe y se realiza la misma interpretación para ambos arreglos.

Asimismo, se trató de probar con otro grafo. El grafo se muestra en la figura 8. Este grafo se había analizado ya previamente y observamos ya la conclusión a la que se llegó coloreando el grafo con los colores rojo, verde y azul. Observamos los

números de cada vértice en negro y en rojo la cantidad de conexiones que cada vértice tiene. No hay que confundir que en la imagen se observa el conjunto C que es el conjunto de colores que se planteaba usar. Éste no es igual al arreglo `colores` en el programa el cual sí presenta una relación de correspondencia con el arreglo `vertices`.

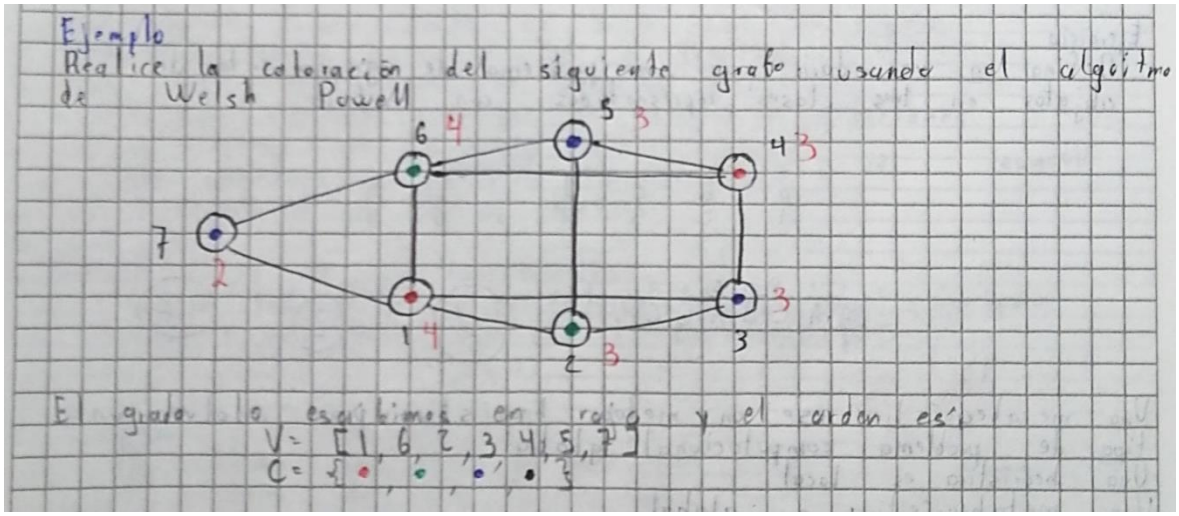


Figura 8: Segundo grafo de prueba.

Podemos asignarles números tales como 1 al rojo, 2 al verde y 3 al azul como ya se ha hecho en el mismo análisis e insertarlo dentro del programa realizado y se obtendrán los resultados mostrados en la figura 9 (si se quiere probar el código en el anexo, para evitar aclarar cada vértice de este grafo en específico, la calibración de este mismo grafo puede hacerse insertando la línea: `7 2 1 1 2 2 1 1 2 1 2 1 2 2 2 1 2 2 2 2 1 1 2 2 1 2 2 1 2`).

```
Matriz de adyacencia:
[0] [1] [1] [0] [0] [1] [1] [4]
[1] [0] [1] [0] [1] [0] [0] [3]
[1] [1] [0] [1] [0] [0] [0] [3]
[0] [0] [1] [0] [1] [1] [0] [3]
[0] [1] [0] [1] [0] [1] [0] [3]
[1] [0] [0] [1] [1] [0] [1] [4]
[1] [0] [0] [0] [0] [1] [0] [2]
--------------------------------
El orden de los vértices según el algoritmo de Welsh Powell es el siguiente:
V = [1, 6, 2, 3, 4, 5, 7]
--------------------------------
El color de los vértices según el algoritmo de Welsh Powell es el siguiente:
C = [1, 2, 2, 3, 1, 3, 3]

--------------------------------
```

Figura 9: Simulación de coloreo de grafos del mostrado en la figura 8 dentro del programa implementado.

Interpretando la información observamos que el vértice 1 tiene asignado el color 1, el vértice 6 tiene asignado el color 2, el vértice 2 tiene asignado el color 2 también,

el vértice 3 tiene asignado el color 3, el vértice 4 tiene asignado el color 1, el vértice 5 tiene asignado el color 3 y finalmente, el vértice 7 tiene asignado el color 3, que es el mismo resultado obtenido a mano.

Conclusiones:

Creo que es importante observar que el algoritmo también está realizado de manera que desempata los vértices de igual número de conexiones manteniendo el menor al principio y estos resultados pueden causar variación entre otros criterios de desempate, sin embargo, la importancia de esta práctica me permitió desarrollar criterios de modelado para usar el trabajo de los datos y las relaciones de correspondencia entre las estructuras de datos planteadas y usadas.

Concluyo que es importante el uso de los ciclos y el planteamiento de cómo se pueden resolver los problemas, entendiendo el problema que se quiere resolver de modo que podamos modelarlo a mano, usando algún caso pequeño pero probarlo para casos grandes y permitir al algoritmo darle la condición de generalidad y resolver grandes problemas y la mayor cantidad de ellos.

Referencias:

Purcell, E. (2007). *Cálculo.* Londres: Pearson Education.

Talbi, E. (2009). *Metaheuristics from design to implementation.* New Jersey: John Wiley & Sons Publication.

Torres, A. (2020). *Apuntes: Optimización Inteligente.* 5° ICI. México: Universidad Autónoma de Aguascalientes.

Anexos:

Anexo 1: Código del programa en lenguaje C:

```
/*
    Universidad Autónoma de Aguascalientes

             Centro de Ciencias Básicas
  Departamento de Ciencias de la Computación
             Optimización Inteligente

                          5° "A"

   Práctica 3: Coloración de un Grafo

             Doctora Aurora Torres Soto

   Alumno: Joel Alejandro Espinoza Sánchez

  Fecha de Entrega: 5 de octubre del 2020

Descripción:
*/
//Cargamos las librerías
#include <stdio.h>
#include <locale.h>
#include <math.h>

main()
{
      setlocale(LC_ALL,"");

      //Declaramos las variables que usaremos
      /*
            f: Auxiliador detector de opciones cuando se pregunta al
usuario
            i: Iterador principal
            j: Iterador secundario
            orden: Es la cantidad de vértices que tendrá el grafo
            u: Múltiples usos (Detecta la condición para repetir el
proceso, auxiliar para ordenar los vértices)
      */
      int f, i, j, orden, u = 0;
```

```
      printf("========================== COLORACIÓN DE UN GRAFO
==========================\n");
      printf("\n");

      do
      {
            //Pedimos que inserten el grafo
            printf("-------------------------------\n");
            printf("Otorgue la cantidad de vértices del grafo: ");
            scanf("%d",&orden);

            //Sabiendo la cantidad de vértices del grafo, hacemos la
matriz de adyacencia vacía y un arreglo de vértices
            int matady[orden][orden + 1], vertices[orden],
colores[orden], coloresaux[orden];

            //Procedemos a llenar la matriz de adyacencia
            for(i = 0; i < orden; i++)
            {
                  j = i;
                  do
                  {
                        printf("--------------------------------\n");
                        printf("¿El vértice %d está conectado con el
vértice %d?\n",i+1,j+1);
                        printf("1. Sí\n");
                        printf("2. No\n");
                        scanf("%d",&f);
                        if(f == 1)
                        {
                              matady[i][j] = 1;
                              matady[j][i] = 1;
                        }
                        if(f == 2)
                        {
                              matady[i][j] = 0;
                              matady[j][i] = 0;
                        }
                        j++;
                  }
                  while(j < orden);
```

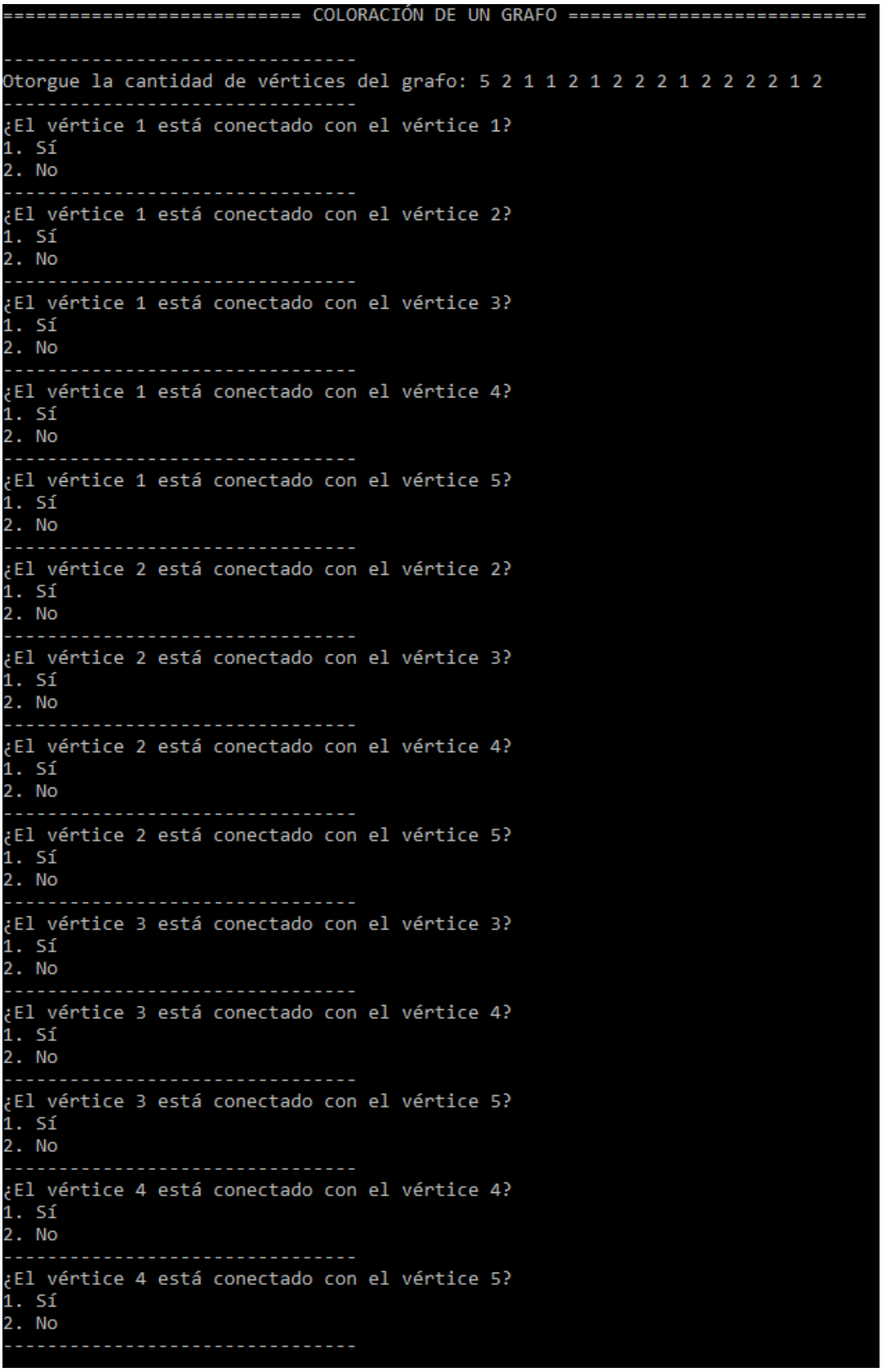
```
                matady[i][orden] = 0;
                vertices[i] = i + 1;
          }

          //La matriz tiene un renglón de más hecho para guardar
el total de conexiones. Esto se calculará a continuación
          for(i = 0; i < orden; i++)
          {
                for(j = 0; j < orden; j++)
                {
                      if(matady[i][j] == 1)
                      {
                            matady[i][orden] = matady[i][orden] + 1;
                      }
                }
          }

          //Mostramos la matriz de adyacencia
          printf("--------------------------------\n\n");
          printf("Matriz de adyacencia:\n");
          for(i = 0; i < orden; i++)
          {
                for(j = 0; j <= orden; j++)
                {
                      printf("[%d] ",matady[i][j]);
                }
                printf("\n");
          }

          //Ordenaremos los vértices de mayor número de conexiones
a menor
          for(i = 0; i + 1 < orden; i++)
          {
                if(matady[i][orden] < matady[i + 1][orden])
                {
                      u = matady[i + 1][orden];
                      matady[i + 1][orden] = matady[i][orden];
                      matady[i][orden] = u;

                      u = vertices[i + 1];
                      vertices[i + 1] = vertices[i];
                      vertices[i] = u;
```

```
                    i = 0;
               }
          }

          printf("--------------------------------\n");
          printf("El orden de los vértices según el algoritmo de
Welsh Powell es el siguiente:\n");
          printf("V = [%d", vertices[0]);
          for(i = 1; i < orden; i++)
          {
               printf(", %d", vertices[i]);
          }
          printf("]\n");

          //Comienza el coloreo de grafos
          colores[0] = 1; //El primer vértice es fácil pues no se
ha revisado ninguno anteriormente

          //Se revisará de uno en uno los próximos vértices en
orden como se guardaron por el arreglo vertices
          for(i = 1; i < orden; i++)
          {
//             printf("DEBUGGING: Estoy revisando al vértice %d
(%d°)\n", vertices[i], i + 1);
               u = 0;

               //En cada revisión tenemos que checar todos los
vértices previos a éste
               for(j = 0; j < i; j++)
               {
//                  printf("          DEBUGGING: con el vértice
%d (%d°)\n", vertices[j], j + 1);

                    //Si el vértice j tiene conexión con el
vértice i, entonces guardaremos en coloresaux ese color como color
imposible
//                  printf("          if: vertices[i] - 1 = %d
vertices[j] - 1 = %d      por tanto matady[vertices[i] -
1][vertices[j] - 1] = %d\n", vertices[i] - 1, vertices[j] - 1,
matady[vertices[i] - 1][vertices[j] - 1]);
```

```
                    if(matady[vertices[i] - 1][vertices[j] - 1]
== 1)
                    {
//                        printf("                    DEBUGGING:
y sí tienen conexión, así que no podemos asignar el color %d\n",
colores[j]);
                        coloresaux[u] = colores[j];
                        u++;
                    }
               }

               colores[i] = 1;
               for(j = 0; j < i; j++)
               {
                    if(coloresaux[j] == colores[i])
                    {
                        colores[i] = colores[i] + 1;
                        j = 0;
                    }
               }

//             DEBUGGING: Vamos a ver cómo se guardó coloresaux
/*             printf("                    DEBUGGING: coloresaux
= ");
               for(j = 0; j < i; j++)
               {
                    printf("%d ",coloresaux[j]);
               }
               printf("\n");
*/        }

          printf("--------------------------------\n");
          printf("El color de los vértices según el algoritmo de
Welsh Powell es el siguiente:\n");
          printf("C = [%d", colores[0]);
          for(i = 1; i < orden; i++)
          {
               printf(", %d", colores[i]);
          }
          printf("]\n\n");
          printf("--------------------------------\n\n");
```

```
            //Opción para repetir el procedimiento
            printf("¿Desea repetir el procedimiento?\n");
            printf("1. Sí\n");
            printf("2. No\n");
            scanf("%d",&u);
      }
      while(u == 1);

      getchar();
}
```

Anexo 2: Impresión de pantalla completa al ejecutar el programa



```
¿El vértice 5 está conectado con el vértice 5?
1. Sí
2. No
--------------------------------

Matriz de adyacencia:
[0] [1] [1] [0] [1] [3]
[1] [0] [0] [0] [1] [2]
[1] [0] [0] [0] [0] [1]
[0] [0] [0] [0] [1] [1]
[1] [1] [0] [1] [0] [3]
--------------------------------
El orden de los vértices según el algoritmo de Welsh Powell es el siguiente:
V = [1, 5, 2, 3, 4]
--------------------------------
El color de los vértices según el algoritmo de Welsh Powell es el siguiente:
C = [1, 2, 3, 3, 1]

--------------------------------

¿Desea repetir el procedimiento?
1. Sí
2. No
```

Anexo 3: Algunas capturas del código en el IDE

```c
/*
    Universidad Autónoma de Aguascalientes

        Centro de Ciencias Básicas
  Departamento de Ciencias de la Computación
          Optimización Inteligente

                5º "A"

   Práctica 3: Coloración de un Grafo

        Doctora Aurora Torres Soto

   Alumno: Joel Alejandro Espinoza Sánchez

  Fecha de Entrega: 5 de octubre del 2020

Descripción:
*/
//Cargamos las librerías
#include <stdio.h>
#include <locale.h>
#include <math.h>

main()
{
    setlocale(LC_ALL,"");

    //Declaramos las variables que usaremos
    /*
        f: Auxiliador detector de opciones cuando se pregunta al usuario
        i: Iterador principal
        j: Iterador secundario
        orden: Es la cantidad de vértices que tendrá el grafo
        u: Múltiples usos (Detecta la condición para repetir el proceso, auxiliar para ordenar los vértices)
    */
```

```c
   int f, i, j, orden, u = 0;

   printf("========================== COLORACIÓN DE UN GRAFO ==========================\n");
   printf("\n");

   do
   {
       //Pedimos que inserten el grafo
       printf("--------------------------------\n");
       printf("Otorgue la cantidad de vértices del grafo: ");
       scanf("%d",&orden);

       //Sabiendo la cantidad de vértices del grafo, hacemos la matriz de adyacencia vacía y un arreglo de vértices
       int matady[orden][orden + 1], vertices[orden], colores[orden], coloresaux[orden];

       //Procedemos a llenar la matriz de adyacencia
       for(i = 0; i < orden; i++)
       {
           j = i;
           do
           {
               printf("--------------------------------\n");
               printf("¿El vértice %d está conectado con el vértice %d?\n",i+1,j+1);
               printf("1. Si\n");
               printf("2. No\n");
               scanf("%d",&f);
               if(f == 1)
               {
                   matady[i][j] = 1;
                   matady[j][i] = 1;
               }
               if(f == 2)
               {
                   matady[i][j] = 0;
                   matady[j][i] = 0;
               }
```

```c
//La matriz tiene un renglón de más hecho para guardar el total de conexiones. Esto se calculará a continuación
for(i = 0; i < orden; i++)
{
    for(j = 0; j < orden; j++)
    {
        if(matady[i][j] == 1)
        {
            matady[i][orden] = matady[i][orden] + 1;
        }
    }
}

//Mostramos la matriz de adyacencia
printf("-------------------------------\n\n");
printf("Matriz de adyacencia:\n");
for(i = 0; i < orden; i++)
{
    for(j = 0; j <= orden; j++)
    {
        printf("[%d] ",matady[i][j]);
    }
    printf("\n");
}
```

```c
//Ordenaremos los vértices de mayor número de conexiones a menor
for(i = 0; i + 1 < orden; i++)
{
    if(matady[i][orden] < matady[i + 1][orden])
    {
        u = matady[i + 1][orden];
        matady[i + 1][orden] = matady[i][orden];
        matady[i][orden] = u;

        u = vertices[i + 1];
        vertices[i + 1] = vertices[i];
        vertices[i] = u;

        i = 0;
    }
}

printf("--------------------------------\n");
printf("El orden de los vértices según el algoritmo de Welsh Powell es el siguiente:\n");
printf("V = [%d", vertices[0]);
for(i = 1; i < orden; i++)
{
    printf(", %d", vertices[i]);
}
printf("]\n");
```

```
        //Comienza el coloreo de grafos
        colores[0] = 1; //El primer vértice es fácil pues no se ha revisado ninguno anteriormente

        //Se revisará de uno en uno los próximos vértices en orden como se guardaron por el arreglo vertices
        for(i = 1; i < orden; i++)
        {
//          printf("DEBUGGING: Estoy revisando al vértice %d (%dº)\n", vertices[i], i + 1);
            u = 0;

            //En cada revisión tenemos que checar todos los vértices previos a éste
            for(j = 0; j < i; j++)
            {
//              printf("          DEBUGGING: con el vértice %d (%dº)\n", vertices[j], j + 1);

                //Si el vértice j tiene conexión con el vértice i, entonces guardaremos en coloresaux ese color como color imposible
//              printf("          if: vertices[i] - 1 = %d      vertices[j] - 1 = %d      por tanto matady[vertices[i] - 1][vertices[j] - 1] =
                if(matady[vertices[i] - 1][vertices[j] - 1] == 1)
                {
//                  printf("                    DEBUGGING: y sí tienen conexión, así que no podemos asignar el color %d\n", colores[j]);
                    coloresaux[u] = colores[j];
                    u++;
                }
            }

            colores[i] = 1;
            for(j = 0; j < i; j++)
            {
                if(coloresaux[j] == colores[i])
                {
                    colores[i] = colores[i] + 1;
                    j = 0;
                }
            }
```

```
        printf("---------------------------------\n");
        printf("El color de los vértices según el algoritmo de Welsh Powell es el siguiente:\n");
        printf("C = [%d", colores[0]);
        for(i = 1; i < orden; i++)
        {
            printf(", %d", colores[i]);
        }
        printf("]\n\n");
        printf("---------------------------------\n\n");

        //Opción para repetir el procedimiento
        printf("¿Desea repetir el procedimiento?\n");
        printf("1. Sí\n");
        printf("2. No\n");
        scanf("%d",&u);
    }
    while(u == 1);

    getchar();
}
```

Anexo 4: Bosquejo completo sobre el entendimiento del algoritmo