Facultad:	Ingeniería
Escuela:	Computación
Asignatura:	Programación con Estructuras de Datos

Tema: **“Recorrido de Grafos”**

Competencia

Desarrolla sistemas de información informáticos mediante la integración de principios matemáticos, ciencia computacional y prácticas de ingeniería, considerando estándares de calidad y mejores prácticas validadas por la industria del software.

Materiales y Equipo

- Guía Número 11.
- Computadora con programa Microsoft Visual C#.

Introducción Teórica

Recorrido de Grafos.

La operación de recorrer una estructura de datos consiste en visitar (procesar) cada uno de los nodos a partir de uno dado. Así, para recorrer un árbol se parte del nodo raíz y según el orden se visitan todos los nodos. De igual forma, recorrer un grafo consiste en visitar todos los vértices alcanzables a partir de uno dado.

Hay dos formas de recorrer un grafo: recorrido en profundidad y recorrido en anchura.

Si el conjunto de nodos marcados se trata como una **cola**, entonces el recorrido es en anchura; si se trata como una **pila,** el recorrido es en profundidad.

Recorrido en Anchura.

El recorrido de búsqueda en anchura, en amplitud o expansión, es una estrategia aplicable indistintamente al caso de grafos dirigidos y no dirigidos.
El recorrido en anchura es una generalización del recorrido por niveles de un árbol.

Se trata de visitar un nodo inicial y luego a todos los nodos que están a un arco de distancia de éste, luego a todos los nodos que están a dos arcos de distancia de éste y así sucesivamente, hasta alcanzar a todos los nodos a los que se pueda llegar desde el nodo inicial.

Una aplicación típica de este recorrido es la ***resolución de problemas de planificación***.
La búsqueda en amplitud se puede utilizar para hallar la distancia más corta entre algún nodo inicial y los nodos restantes del grafo. Esta distancia más corta es el mínimo número de aristas que hay que recorrer para pasar desde el nodo inicial hasta el nodo concreto que se esté examinando.

Comenzando en el nodo “s”, esta distancia se calcula examinando todas las aristas incidentes en el nodo “s”, y pasando después a un nodo adyacente “w”, repitiéndose entonces todo el proceso. El recorrido continúa hasta que se hayan examinado todos los nodos del grafo.
En una búsqueda en amplitud, cada nodo se visita o es procesado en algún sentido, dependiendo de la aplicación concreta. La búsqueda comienza en un nodo concreto del grafo. A continuación la búsqueda se extiende a los nodos del grafo que estén más próximos al nodo inicial antes de visitar ningún otro. Estos nodos están relacionados de alguna forma con el nodo especificado, y forman parte de un grupo que depende de la aplicación concreta.
Inicialmente, se visitan todos aquellos nodos que sean adyacentes al nodo inicial. A continuación se visitan todos los nodos que estén a una distancia “2” del nodo inicial. Este proceso se repite hasta que se haya visitado todos los nodos posibles.
Sin embargo, este enfoque de búsqueda puede dar lugar a problemas. Los nodos ya visitados no deben visitarse de nuevo. Se puede evitar esta situación marcando aquellos nodos que se hayan visitado. Si un nodo ya ha sido marcado con anterioridad (esto es, si ya ha sido alcanzado o visitado), nunca vuelve a ser visitado.
Este método utiliza una cola como estructura auxiliar en la que se mantienen los vértices que se vayan a procesar posteriormente.

El recorrido en anchura trabaja de la siguiente manera:

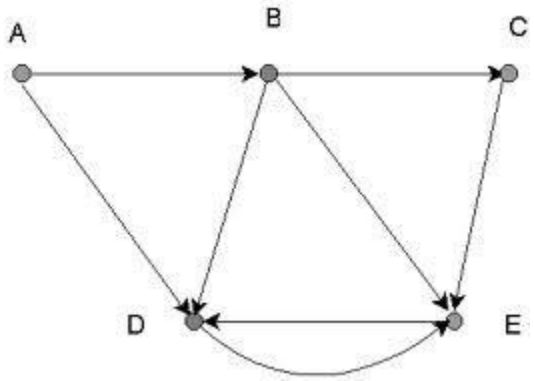
1. Se visita el nodo inicial.
2. Después de visitar el nodo inicial se visitan todos los sucesores.
3. Después los sucesores de los sucesores.
4. Se repiten los pasos 1 y 2 hasta encontrar un nodo sin arcos salientes o ya visitados.
5. Si después de visitar todos los descendientes del primer nodo, todavía quedan más nodos por visitar, se repite el proceso reiniciando.

Este tipo de recorrido es muy usado para problemas de planificación (problema de laberinto).
Al igual que el recorrido en profundidad, en el recorrido en anchura no existe un único recorrido de grafo, sino un conjunto de ellos.

Para el siguiente grafo, habría varios recorridos en anchura posibles, entre los cuales podríamos mencionar:

Visitando primero el nodo izquierdo seria:
A → B → D → C → E

Visitando primero el derecho seria:
A → D → B → E → C



Recorrido en Profundidad.
Este método es una forma básica de recorrer un grafo implementando recursividad.
En este recorrido se basan los recorridos preorden y postorden para árboles binarios.

Supóngase que una persona se encuentra en algún sistema de cuevas interconectadas, o en un laberinto, en una cierta intersección (o nodo), y que se le pide a esta persona que busque la salida, que se encuentra en un determinado nodo. En esta búsqueda se podrían emplear varias opciones.
Una posibilidad que probablemente no se utilizase sería la búsqueda en amplitud. Intuitivamente, una estrategia que comienza en algún nodo de la cueva y que después visita todos y cada uno de los nodos adyacentes siguientes de la cueva, no parece demasiado prometedora.

Una estrategia de aspecto mucho mejor es la que consiste en comenzar en un cierto punto y seguir, digamos, la cueva situada más a la derecha desde esa cueva hasta la próxima intersección. Al llegar a esa intersección, se sigue de nuevo la salida situada más a la derecha, y así sucesivamente hasta llegar al nodo deseado, si este proceso de seguimiento del camino situado más a la derecha no tiene éxito, porque no se hallan nuevas intersecciones, entonces el individuo deshace el camino andado hasta la última intersección, y sigue por la salida contigua a la situada más a la derecha, y así sucesivamente.

El retroceso podría muy bien devolver a la persona al nodo de partida, y entonces se sigue el proceso con la próxima salida de las situadas a la derecha. Los movimientos hacia delante, que van seguidos en ocasiones por retrocesos, son en esencia un método de búsqueda en profundidad que se estudiará a continuación con más detalle.

Así se puede utilizar una búsqueda en profundidad en un grafo arbitrario para realizar el recorrido de un grafo general. A medida que se encuentra cada nuevo nodo, se marca el mismo para evitar volver a visitarlo de nuevo.

La estrategia de recorrido en profundidad es la siguiente:

1. Se toma un nodo “s” como comienzo, y se marca.
2. A continuación se toma y se marca un nodo no marcado adyacente a “s”, y ese nodo pasa a ser el nuevo nodo de partida, dejando posiblemente por el momento al nodo inicial original con nodos no explorados.
3. La búsqueda continúa por el grafo hasta que el camino en curso finalice con un grafo de salida igual a cero, o bien en un nodo en que todos los nodos adyacentes estén marcados.
4. A continuación la búsqueda vuelve al último nodo que todavía tenga nodos adyacentes sin marcar, y continúa marcando todos los nodos de forma recursiva hasta que ya no queden nodos sin marcar.

Es preciso marcar los nodos en la búsqueda en profundidad. Si no se hiciera esto, los grafos que contuvieran ciclos darían lugar a bucles infinitos.

Un algoritmo de búsqueda o recorrido en profundidad consta de una única rutina principal que invoca a un procedimiento recursivo de la manera siguiente:

Principal

1. Se marcan todos los nodos del grafo como no visitados.

2. Se invoca a recorrido_en_profundidad (s) para algún nodo inicial “s”.

Procedimiento recorrido_en_profundidad

1. Se marca y visita “s”.
2. Para cada vecino “w” de “s”

    Si el vecino “w” no está marcado

        Se invoca a recorrido_en_profundidad (w).

El recorrido en profundidad persigue el mismo objetivo que el recorrido en anchura: visitar todos los vértices del grafo alcanzables desde un vértice dado.
La búsqueda en profundidad empieza por un vértice “V” del grafo “G”; “V” se marca como visitado. Después se recorre en profundidad cada vértice adyacente a “V” no visitado; así hasta que no haya más vértices adyacentes no visitados.
Esta técnica se denomina en profundidad porque la dirección de visitar es hacia adelante mientras que sea posible; al contrario que la búsqueda en anchura, que primero visita todos los vértices posibles en amplitud.
La definición recursiva del recorrido en profundidad nos indica que tenemos que utilizar una “pila” como estructura para guardar los vértices adyacentes no visitados a uno dado. De igual forma que en el recorrido en anchura, hacemos uso de una lista de vértices para controlar los ya visitados.

Los usos que pueden darse a este tipo de recorrido son:

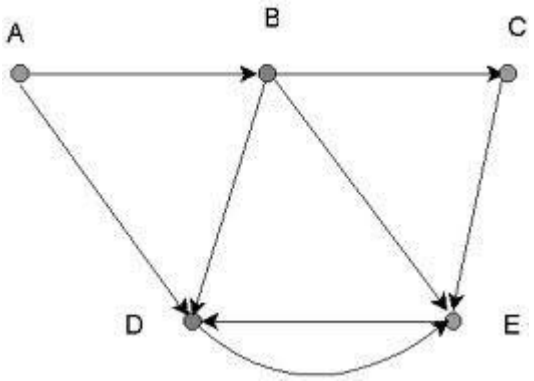
- Simular una red, a partir de un grafo y analizar su robustez para transferencia de información.
- Identificar bucles, los cuales son potenciales causantes de elevación de tráfico en recorridos.

Hay que tomar en cuenta que, para recorridos primero en profundidad, no existe un único recorrido de grafo, sino un conjunto de ellos.

Para el siguiente grafo, habría varios recorridos en profundidad posibles, entre los cuales podríamos mencionar:

Visitando primero el nodo izquierdo seria:
A → D → E → B → C

Visitando primero el nodo derecho seria:
A → B → C → E → D



Aplicaciones de Grafos.

Prácticamente cualquier red puede ser modelada con un grafo: una red de carreteras que conecta ciudades, una red eléctrica o un sistema de alcantarillados, una red de computadoras. Al visitar una página web y hacer clic a un enlace, visto como un grafo los vértices son los sitios, y cuyas aristas son lógicamente los enlaces.

Gracias a la teoría de Grafos se pueden resolver diversos problemas como por ejemplo la síntesis de circuitos secuenciales.
Los grafos se utilizan también para modelar trayectos como el de una línea de autobús a través de las calles de una ciudad, en el que podemos obtener caminos óptimos para el trayecto aplicando diversos algoritmos como puede ser el algoritmo de Floyd.

Para la administración de proyectos, utilizamos técnicas como PERT en las que se modelan los proyectos utilizando grafos y optimizando los tiempos para concretar las distintas tareas asociadas al desarrollo de un proyecto en particular.

Desarrollo de Habilidades

Ejercicio 1.

Basados en el código de la guía anterior (guía 10. Grafos en C#), se le agregará más funcionalidades.

NOTA IMPORTANTE: en esta guía se incluye nuevamente TODO el código incluyendo la guía anterior, sin embargo usted SOLAMENTE deberá agregar aquellos elementos que no poseía anteriormente.

Agregar todo el código de nuevo es solamente por cuestiones didácticas y para explicar mejor qué cosa va en cada sección.

1. En la clase **CVertice.sv** se modificará o agregará, únicamente este código:

1.1 En las sección de las variables, al inicio de la clase

```
public int distancianodo;//guarda la distancia que hay entre el nodo inicio en el algoritmo de Dijkstra
public Boolean Visitado;//variable que sirve para marcar como visto el nodo en un recorrido
public CVertice Padre;//nodo que sirve en los recorridos como el antecesor
public Boolean pesoasignado;//variable que sirve se usa en el algoritmo de Dijkstra
```

1.2 En el constructor que recibe el parámetro string valor modificarán y agregaran

```
this.Color = Color.FromArgb(51, 204, 255); // Definimos el color del nodo
this.FontColor = Color.Black; // Color de la fuente
this.Visitado = false;
```

1.3 Finalmente agregarán el siguiente código al final de la clase

```
public void colorear(Graphics g)
{
    SolidBrush b = new SolidBrush(Color.GreenYellow);
    // Definimos donde dibujaremos el nodo
    Rectangle areaNodo = new Rectangle(this._posicion.X - radio, this._posicion.Y - radio,
    this.dimensiones.Width, this.dimensiones.Height);
    g.FillEllipse(b, areaNodo);
    g.DrawString(this.Valor, new Font("Times New Roman", 14), new SolidBrush(color_fuente),
    this._posicion.X, this._posicion.Y,
    new StringFormat()
    {
        Alignment = StringAlignment.Center,
        LineAlignment = StringAlignment.Center
    });
    g.DrawEllipse(new Pen(Brushes.Black, (float)1.0), areaNodo);
    b.Dispose();
}
```

2. En la clase **CArco.cs** no haremos ninguna modificación

3. En la clase **CLista.cs** tampoco haremos ninguna modificación

4. En la clase **CGrafo.cs** añadiremos los siguientes métodos:

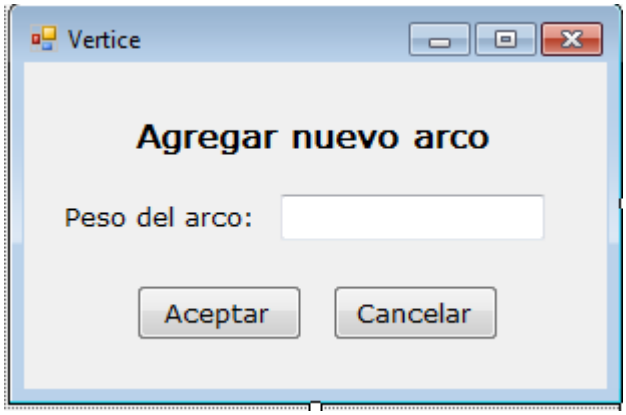
```
public void ColoArista(string o, string d)
{
    foreach (CVertice nodo in nodos)
    {
        foreach (CArco a in nodo.ListaAdyacencia)
        {
            if (nodo.ListaAdyacencia != null && nodo.Valor == o && a.nDestino.Valor == d)
            {
                a.color = Color.Red;
                a.grosor_flecha = 4;
            }
        }
    }
}
```

```
    public void Colorear(CVertice nodo)
    {
        nodo.Color = Color.AliceBlue;
        nodo.FontColor = Color.Black;
    }

     public CVertice nododistanciaminima()
     {
         int min = int.MaxValue;
         CVertice temp = null;
         foreach (CVertice origen in nodos)
         {
             if (origen.Visitado)
             {
                 foreach (CVertice destino in nodos)
                 {
                     if (!destino.Visitado)
                     {
                         foreach (CArco a in origen.ListaAdyacencia)
                         {
                             if (a.nDestino == destino && min > a.peso)
                             {
                                 min = a.peso;
                                 temp = destino;
                             }  }  } } } }
         return temp;
     }

     public int posicionNodo(string Nodo)
     {
         for (int i = 0; i < nodos.Count; i++)
         {
             if (String.Compare(nodos[i].Valor, Nodo) == 0)
                 return i;
         }
         return -1;
     }
//Funcion para re-dibujar los arcos que llegan a un nodo
public void DibujarEntrantes(CVertice nDestino)
{
    foreach (CVertice nodo in nodos)
    {
        foreach (CArco a in nodo.ListaAdyacencia)
        {
            if (nodo.ListaAdyacencia != null && nodo != nDestino)
            {
                if (a.nDestino == nDestino)
                {
                    a.color = Color.Black;
                    a.grosor_flecha = 2;
                    break;
                }  } } }  }
```

```
//funcion que desmarca como visitados todos los nodos del grafo
public void Desmarcar()
{
    foreach (CVertice n in nodos)
    {
        n.Visitado = false;
        n.Padre = null;
        n.distancianodo = int.MaxValue;
        n.pesoasignado = false;
    }
}
```

5. El formulario **Vértice** no sufrirá ningún cambio

6. Agregaremos otro formulario y lo nombraremos Arco.



El código de ese formulario es el siguiente:

```
public partial class Arco : Form
{
    public bool control; // variable de control
    public int dato; // el dato que almacenará el vértice
    public Arco()
    {
        InitializeComponent();
        control = false;
        dato = 0;
    }

    private void btnAceptar_Click(object sender, EventArgs e)
    {
        try
        {
            dato = Convert.ToInt16(txtVertice.Text.Trim());

            if (dato < 0)
            {
                MessageBox.Show("Debes ingresar un valor positivo", "Error", MessageBoxButtons.OK,
            MessageBoxIcon.Exclamation);
            }
            else
            {
                control = true;
                Hide();
            }
        }
```

```
        catch (Exception ex) {
            MessageBox.Show("Debes ingresar un valor numérico");
        }

    }

    private void btnCancelar_Click(object sender, EventArgs e)
    {
        control = false;
        Hide();
    }

    private void Vertice_Load(object sender, EventArgs e)
    {
        txtVertice.Focus();
    }

    private void Vertice_FormClosing(object sender, FormClosingEventArgs e)
    {
        this.Hide();
        e.Cancel = true;
    }

    private void Vertice_Shown(object sender, EventArgs e)
    {
        txtVertice.Clear();
        txtVertice.Focus();
    }


     private void txtVertice_KeyDown(object sender, KeyEventArgs e)
     {
         if (e.KeyCode == Keys.Enter)
         {
             btnAceptar_Click(null, null);
         }
     }
```

7. Haremos modificaciones a la interfaz (formulario **Simulador**), de forma que luzca como se muestra en la imagen. ***El panel siempre se mantiene y se sigue llamando Pizarra***

En este caso se incluye el código completo para este formulario:


```
public partial class Simulador : Form
{
    private CGrafo grafo; // instanciamos la clase CGrafo
    private CVertice nuevoNodo; // instanciamos la clase CVertice
    private CVertice NodoOrigen; // instanciamos la clase CVertice
    private CVertice NodoDestino; // instanciamos la clase CVertice
    private int var_control = 0; // la utilizaremos para determinar el estado en la pizarra:
    // 0 -> sin acción, 1 -> Dibujando arco, 2 -> Nuevo vértice
    // variables para el control de ventanas modales
    //private Recorrido ventanaRecorrido; // ventana para seleccionar el nodo inicial del recorrido
    private Vertice ventanaVertice; // ventana para agregar los vértices
    private Arco ventanaArco; // ventana para agregar los arcos
    List<CVertice> nodosRuta; // Lista de nodos utilizada para almacenar la ruta
    List<CVertice> nodosOrdenados; // Lista de nodos ordenadas a partir del nodo origen
    bool buscarRuta = false, nuevoVertice = false, nuevoArco = false;
    private int numeronodos = 0; //Enteros para definir las diferentes opciones y el numero de nodos
    bool profundidad = false, anchura = false, nodoEncontrado = false;
    Queue cola = new Queue(); //para el recorrido de anchura
    private string destino = "", origen = "";
    private int distancia = 0;
```

```
public Simulador()
{
    InitializeComponent();
    grafo = new CGrafo();
    nuevoNodo = null;
    var_control = 0;
    ventanaVertice = new Vertice();
    ventanaArco = new Arco();
    nodosRuta = new List<CVertice>();
    nodosOrdenados = new List<CVertice>();
    this.SetStyle(ControlStyles.AllPaintingInWmPaint | ControlStyles.UserPaint |
    ControlStyles.DoubleBuffer, true);
}

 private void Pizarra_Paint(object sender, PaintEventArgs e)
 {
     try
     {
         e.Graphics.SmoothingMode = SmoothingMode.HighQuality;
         grafo.DibujarGrafo(e.Graphics);
         if (nuevoVertice)
         {
             CBVertice.Items.Clear();
             CBVertice.SelectedIndex = -1;
             CBNodoPartida.Items.Clear();
             CBNodoPartida.SelectedIndex = -1;
             foreach (CVertice nodo in grafo.nodos)
             {
                 CBVertice.Items.Add(nodo.Valor);
                 CBNodoPartida.Items.Add(nodo.Valor);
             }
             nuevoVertice = false;
         }
         if (nuevoArco)
         {
             CBArco.Items.Clear();
```

```
    CBArco.SelectedIndex = -1;
    foreach (CVertice nodo in grafo.nodos)
    {
        foreach (CArco arco in nodo.ListaAdyacencia)
            CBArco.Items.Add("(" + nodo.Valor + "," + arco.nDestino.Valor + ") peso: " + arco.peso);
    }
    nuevoArco = false;
}
if (buscarRuta)
{
    foreach (CVertice nodo in nodosRuta)
    {
        nodo.colorear(e.Graphics);
        Thread.Sleep(1000);
        nodo.DibujarVertice(e.Graphics);
    }
    buscarRuta = false;
}
```

```
    if (profundidad)
    {
        //ordenando los nodos desde el que indica el usuario
        ordenarNodos();
        foreach (CVertice nodo in nodosOrdenados)
        {
            if (!nodo.Visitado)
                recorridoProfundidad(nodo, e.Graphics);
        }
        profundidad = false;
        //reestablecer los valroes
        foreach (CVertice nodo in grafo.nodos)
            nodo.Visitado = false;

    }

     if (anchura)
     {
         distancia = 0;
         //ordenando los nodos desde el que indica el usuario
         cola = new Queue();
         ordenarNodos();
         foreach (CVertice nodo in nodosOrdenados)
         {
             if (!nodo.Visitado && !nodoEncontrado)
                 recorridoAnchura(nodo, e.Graphics, destino);
         }
         anchura = false;
         nodoEncontrado = false;
         //reestablecer los valroes
         foreach (CVertice nodo in grafo.nodos)
             nodo.Visitado = false;
     }
}

      catch (Exception ex)
      {
          MessageBox.Show(ex.Message);
      }
 }
```

```
public void ordenarNodos()
{
    nodosOrdenados = new List<CVertice>();
    bool est = false;
    foreach (CVertice nodo in grafo.nodos)
    {
        if (nodo.Valor == origen)
        {
            nodosOrdenados.Add(nodo);
            est = true;
        }
        else if (est)
            nodosOrdenados.Add(nodo);
    }
    foreach (CVertice nodo in grafo.nodos)
    {
        if (nodo.Valor == origen)
        {
            est = false;
            break;
        }
        else if (est)
            nodosOrdenados.Add(nodo);
    }
}

private void Pizarra_MouseLeave(object sender, EventArgs e)
{
    Pizarra.Refresh();
}

private void nuevoVerticeToolStripMenuItem_Click(object sender, EventArgs e)
{
    nuevoNodo = new CVertice();
    var_control = 2; // recordemos que es usado para indicar el estado en la pizarra: 0 ->
    // sin accion, 1 -> Dibujando arco, 2 -> Nuevo vértice
}

private void Pizarra_MouseUp(object sender, MouseEventArgs e)
{
    switch (var_control)
    {
        case 1: // Dibujando arco
            if ((NodoDestino = grafo.DetectarPunto(e.Location)) != null && NodoOrigen !=
            NodoDestino)
            {
                ventanaArco.Visible = false;
                ventanaArco.control = false;
                ventanaArco.ShowDialog();
                if (ventanaArco.control)
                {
                    if (grafo.AgregarArco(NodoOrigen, NodoDestino, ventanaArco.dato)) //Se procede a crear la arista
                    {
                        int distancia = ventanaArco.dato;
                        NodoOrigen.ListaAdyacencia.Find(v => v.nDestino == NodoDestino).peso =
                       distancia;
                    }
                    nuevoArco = true;
                }
            }
```

```
            var_control = 0;
            NodoOrigen = null;
            NodoDestino = null;
            Pizarra.Refresh();
            break;
    }
}

private void Pizarra_MouseMove(object sender, MouseEventArgs e)
{
    switch (var_control)
    {
        case 2: //Creando nuevo nodo
            if (nuevoNodo != null)
            {
                int posX = e.Location.X;
                int posY = e.Location.Y;
                if (posX < nuevoNodo.Dimensiones.Width / 2)
                    posX = nuevoNodo.Dimensiones.Width / 2;
                else if (posX > Pizarra.Size.Width - nuevoNodo.Dimensiones.Width / 2)
                    posX = Pizarra.Size.Width - nuevoNodo.Dimensiones.Width / 2;
                if (posY < nuevoNodo.Dimensiones.Height / 2)
                    posY = nuevoNodo.Dimensiones.Height / 2;
                else if (posY > Pizarra.Size.Height - nuevoNodo.Dimensiones.Width / 2)
                    posY = Pizarra.Size.Height - nuevoNodo.Dimensiones.Width / 2;
                nuevoNodo.Posicion = new Point(posX, posY);
                Pizarra.Refresh();
                nuevoNodo.DibujarVertice(Pizarra.CreateGraphics());
            }
            break;
```

```
        case 1: // Dibujar arco
            AdjustableArrowCap bigArrow = new AdjustableArrowCap(4, 4, true);
            bigArrow.BaseCap = System.Drawing.Drawing2D.LineCap.Triangle;
            Pizarra.Refresh();
            Pizarra.CreateGraphics().DrawLine(new Pen(Brushes.Black, 2) { CustomEndCap = bigArrow },
                NodoOrigen.Posicion, e.Location);
            break;
    }
}
```

```
private void Pizarra_MouseDown(object sender, MouseEventArgs e)
{
    if (e.Button == System.Windows.Forms.MouseButtons.Left) // Si se ha presionado el botón
    // izquierdo del mouse
    {
        if ((NodoOrigen = grafo.DetectarPunto(e.Location)) != null)
        {
            var_control = 1; // recordemos que es usado para indicar el estado en la pizarra:
            // 0 -> sin accion, 1 -> Dibujando arco, 2 -> Nuevo vértice
        }
        if (nuevoNodo != null && NodoOrigen == null)
        {
            ventanaVertice.Visible = false;
            ventanaVertice.control = false;
            ventanaVertice.ShowDialog();
            numeronodos = grafo.nodos.Count;//cuenta cuantos nodos hay en el grafo
            if (ventanaVertice.control)
            {
                if (grafo.BuscarVertice(ventanaVertice.dato) == null)
                {
                    grafo.AgregarVertice(nuevoNodo);
                    nuevoNodo.Valor = ventanaVertice.dato;
                }
```

```
            else
            {
                lblRespuesta.Text = "El Nodo " + ventanaVertice.dato + " ya existe en el grafo";
                lblRespuesta.ForeColor = Color.Red;
            }
        }
        nuevoNodo = null;
        nuevoVertice = true;
        var_control = 0;
        Pizarra.Refresh();
    }
    if (e.Button == System.Windows.Forms.MouseButtons.Right) // Si se ha presionado el botón
    // derecho del mouse
    {
        if (var_control == 0)
        {
            if ((NodoOrigen = grafo.DetectarPunto(e.Location)) != null)
            {
                nuevoVerticeToolStripMenuItem.Text = "Nodo " + NodoOrigen.Valor;
            }
            else
                Pizarra.ContextMenuStrip = this.contextMenuStrip1;
        }
```

```
        }
    }
    if (e.Button == System.Windows.Forms.MouseButtons.Right) // Si se ha presionado el botón
    // derecho del mouse
    {
        if (var_control == 0)
        {
            if ((NodoOrigen = grafo.DetectarPunto(e.Location)) != null)
            {
                nuevoVerticeToolStripMenuItem.Text = "Nodo " + NodoOrigen.Valor;
            }
            else
                Pizarra.ContextMenuStrip = this.contextMenuStrip1;
        }
    }
}

private void BtnEliminarVer_Click(object sender, EventArgs e)
{
    if (CBVertice.SelectedIndex > -1)
    {
        foreach (CVertice nodo in grafo.nodos)
        {
```

```
    if (nodo.Valor == CBVertice.SelectedItem.ToString())
    {
        grafo.nodos.Remove(nodo);
        //Borrando arcos que posea el nodo eliminado
        nodo.ListaAdyacencia = new List<CArco>();
        break;
    }
}
foreach (CVertice nodo in grafo.nodos)
{
    foreach (CArco arco in nodo.ListaAdyacencia)
    {
        if (arco.nDestino.Valor == CBVertice.SelectedItem.ToString())
        {
            nodo.ListaAdyacencia.Remove(arco);
            break;
        }
    }
}
nuevoArco = true;
nuevoVertice = true;
CBVertice.SelectedIndex = -1;
Pizarra.Refresh();
```

```
        }
        else
        {
            lblRespuesta.Text = "Seleccione un nodo";
            lblRespuesta.ForeColor = Color.Red;
        }
    }

    private void BtnElArc_Click(object sender, EventArgs e)
    {
        if (CBArco.SelectedIndex > -1)
        {
            foreach (CVertice nodo in grafo.nodos)
            {
                foreach (CArco arco in nodo.ListaAdyacencia)
                {
                    if ("(" + nodo.Valor + "," + arco.nDestino.Valor + ") peso: " + arco.peso ==
                   CBArco.SelectedItem.ToString())
                    {
                        nodo.ListaAdyacencia.Remove(arco);
                        break;
                    }
                }

            }
            nuevoVertice = true;
            nuevoArco = true;
            CBArco.SelectedIndex = -1;
            Pizarra.Refresh();
        }
        else
        {
            lblRespuesta.Text = "Seleccione un arco";
            lblRespuesta.ForeColor = Color.Red;
        }
    }

    private void recorridoProfundidad(CVertice vertice, Graphics g)
    {
        vertice.Visitado = true;
        vertice.colorear(g);
        Thread.Sleep(1000);
        vertice.DibujarVertice(g);
        foreach (CArco adya in vertice.ListaAdyacencia)
        {
            if (!adya.nDestino.Visitado) recorridoProfundidad(adya.nDestino, g);
        }
```

```
}
private void recorridoAnchura(CVertice vertice, Graphics g, string destino)
{
    vertice.Visitado = true;
    cola.Enqueue(vertice);
    vertice.colorear(g);
    Thread.Sleep(1000);
    vertice.DibujarVertice(g);
    if (vertice.Valor == destino)
    {
        nodoEncontrado = true;
        return;
    }
    while (cola.Count > 0)
    {
        CVertice aux = (CVertice)cola.Dequeue();
        foreach (CArco adya in aux.ListaAdyacencia)
        {
            if (!adya.nDestino.Visitado)
            {
                if (!nodoEncontrado)
                {
                    adya.nDestino.Visitado = true;

                        adya.nDestino.colorear(g);
                        Thread.Sleep(1000);
                        adya.nDestino.DibujarVertice(g);
                        if (destino != "")
                            distancia += adya.peso;
                        cola.Enqueue(adya.nDestino);
                        if (adya.nDestino.Valor == destino)
                        {
                            nodoEncontrado = true;
                            return;
                        }
                    }
                }
            }
        }
    }

    private void BtnProf_Click(object sender, EventArgs e)
    {
        if (CBNodoPartida.SelectedIndex > -1)
        {
            profundidad = true;
            origen = CBNodoPartida.SelectedItem.ToString();
```

```
            Pizarra.Refresh();
            CBNodoPartida.SelectedIndex = -1;
        }
        else
        {
            lblRespuesta.Text = "Seleccione un nodo de partida";
            lblRespuesta.ForeColor = Color.Red;
        }
    }

    private void BtnAnch_Click(object sender, EventArgs e)
    {
        if (CBNodoPartida.SelectedIndex > -1)
        {
            origen = CBNodoPartida.SelectedItem.ToString();
            anchura = true;
            Pizarra.Refresh();
            CBNodoPartida.SelectedIndex = -1;
        }
        else
        {
            lblRespuesta.Text = "Seleccione un nodo de partida";
            lblRespuesta.ForeColor = Color.Red;

        }
    }


    private void BtnBuscar_Click(object sender, EventArgs e)
    {
        if (txtBuscar.Text.Trim() != "")
        {
            if (grafo.BuscarVertice(txtBuscar.Text) != null)
            {
                lblRespuesta.Text = "Si se encuentra el vértice " + txtBuscar.Text;
                lblRespuesta.ForeColor = Color.Blue;
            }
            else
            {
                lblRespuesta.Text = "No se encuentra el vértice " + txtBuscar.Text;
                lblRespuesta.ForeColor = Color.Red;
            }
        }
    }
```

```
private int totalNodos; //lista de nodos
int[] parent; // padre del nodo
bool[] visitados;// variable para comprobar los nodos ya visitados

private void calcularMatricesIniciales() // se calculan las matrices iniciales de distancia y de nodos
{

    nodosRuta = new List<CVertice>(); //lista de nodos
    totalNodos = grafo.nodos.Count; //cuenta el numero de nodos en la lista "nodos"
    parent = new int[totalNodos];
    visitados = new bool[totalNodos];
    //calculamos la matriz inicial de distancias
    for (int i = 0; i < totalNodos; i++)
    {
        List<int> filaDistancia = new List<int>();
        for (int j = 0; j < totalNodos; j++)
        {
            //si el origen = al destino
            if (i == j)
            {
                filaDistancia.Add(0);
            }
            else
            {
                //buscamos si existe la relacion i,j; de existir obtenemos la distancia
                int distancia = -1;
                for (int k = 0; k < grafo.nodos[i].ListaAdyacencia.Count; k++)
                {
                    if (grafo.nodos[i].ListaAdyacencia[k].nDestino == grafo.nodos[j])
                        distancia = grafo.nodos[i].ListaAdyacencia[k].peso;
                }
                filaDistancia.Add(distancia);
            }
        }

    }
}
```

Análisis de Resultados

Ejercicio 1.

a. Al desarrollo del ejercicio anterior añada la funcionalidad para el botón distancia, de forma que sepa cuánta distancia hay de un nodo a otro

b. Modifique el ejercicio de forma que envíe en un label o en un texbox (u otra herramienta) cuál es la ruta del recorrido en anchura y en profundidad.