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

Tema: **“Grafos en C#”**

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 10.
- Computadora con programa Microsoft Visual C#.

Introducción Teórica

Un grafo G = (V, A) está formado por un conjunto de elementos llamados vértices “V” y un conjunto de aristas “A” que conectan a los distintos vértices. En ocasiones los vértices son llamados nodos y las aristas arcos.

Las aristas se definen como el par de vértices que conectan y pueden tener asociado un valor el cual representa el peso o dificultad para desplazarse de un vértice a otro.

Ejemplo gráfico de un grafo:

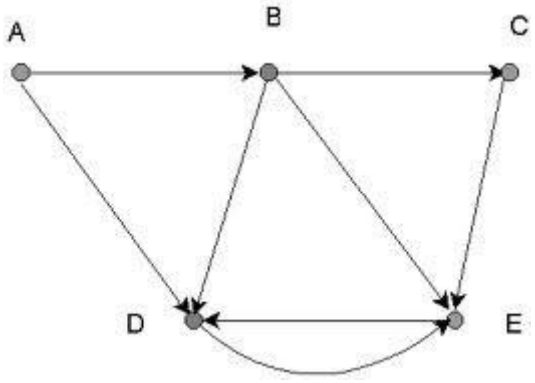


Figura 1.

Donde:

Vértices = {A, B, C, D, E}

Aristas = {(A, B), (A, D), (B, C), (B, D), (B, E), (C, E), (D, E), (E, D)}

Tipos de grafos.

Existen dos tipos de grafos: ***los no dirigidos*** y ***los dirigidos***.

Grafos No Dirigidos.

Son aquellos en los cuales las aristas no están orientadas (no se representan con flechas). Cada arista se representa entre paréntesis, separando sus vértices por comas, y teniendo en cuenta que ambos vértices son origen y destino a la vez: (Vi, Vj) = (Vj, Vi).

Grafos Dirigidos (Dígrafos).

Son aquellos en los cuales las aristas están orientadas (se representan con flechas). Cada arista se representa entre paréntesis, separando sus vértices por comas y teniendo en cuenta que (Vi, Vj) ≠ (Vj, Vi).

Los grafos pueden representarse de varias formas en una computadora:

- **Listas adyacentes.**

  Cada vértice tiene una lista de vértices los cuales son adyacentes a él.

  Representación del ejemplo (grafo de Figura 1):

  (A) → B → D

  (B) → C → D → E

  (C) → E

  (D) → E

  (E) → D

- **Listas de pares ordenados (incidentes).**

  Las aristas se representan como un arreglo de pares ordenados.

  Representación del ejemplo (grafo de Figura 1):

  Aristas = {(A, B), (A, D), (B, C), (B, D), (B, E), (C, E), (D, E), (E, D)}

- **Matriz de adyacencia.**

  El grafo se representa por una matriz de tamaño VxV, donde V son los vértices que unen a los nodos, la conjunción de los dos vértices representa la arista.

  Representación del ejemplo (grafo de Figura 1):

	A	B	C	D	E
A	0	1	0	1	0
B	0	0	1	1	1
C	0	0	0	0	1
D	0	0	0	0	1
E	0	0	0	1	0

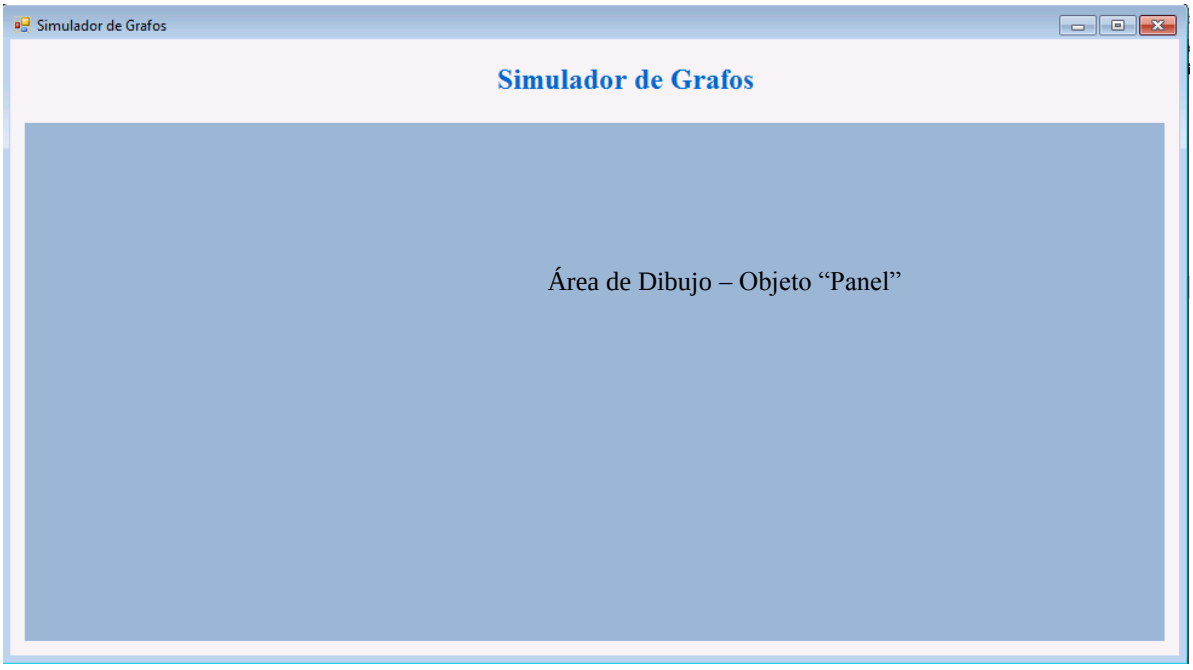
Figura 2.

Procedimiento

Ejemplo 1.

En los códigos siguientes, se muestra la implementación de un grafo a partir de listas adyacentes. Tomar como base el grafo del ejemplo de la sección teórica (ver Figura 1) y crearlo usando la implementación.

Crear un proyecto llamado Grafos y luego renombrar el formulario con el nombre: **Simulador**. Luego insertamos los objetos necesarios, de tal manera que el formulario luzca similar al mostrado en la figura siguiente (imagen sugerida, el estudiante decide el diseño final):



Se han agregado al formulario un Label y un Panel.

Al objeto Panel le cambiamos la propiedad “Name” a “**Pizarra**”.

El siguiente paso es agregar una clase al proyecto, le colocaremos como nombre "**CVertice**", la cual nos servirá para definir los nodos (vértices) del grafo.

Los pasos son: clic derecho sobre el proyecto "Grafos" en el árbol Explorador de Soluciones, clic en agregar y clic en clase. Dar el nombre especificado en el recuadro y clic en Agregar.

En esta clase colocamos el siguiente código:

CVertice.cs

```
using System;
using System.Collections.Generic;
using System.Drawing;                // Librería agregada, para poder dibujar
using System.Drawing.Drawing2D;  // Librería agregada, para poder dibujar
using System.Linq;
using System.Text;
using System.Threading.Tasks;      // Librería agregada, para el manejo de hilos de ejecución

namespace Grafos
{
   class Cvertice
   {                        //  Atributos
      public string Valor;                          // Valor que almacena (representa) el nodo
      public List <CArco> ListaAdyacencia;    // Lista de adyacencia del nodo
      /* Los diccionarios representan una colección de claves y valores. El primer parámetro representa
        el tipo de las claves del diccionario, el segundo, el tipo de los valores del diccionario. */
      Dictionary <string, short> _banderas;
      Dictionary <string, short> _banderas_predeterminado;

                        // Propiedades
      public Color Color
      {
         get { return color_nodo; }
         set { color_nodo = value; }
      }

      public Color FontColor
      {
         get { return color_fuente; }
         set { color_fuente = value; }
      }

      public Point Posicion
      {
         get { return _posicion; }
         set { _posicion = value; }
      }

      public Size Dimensiones
      {
         get { return dimensiones; }
         set
         {
```

```
            radio = value.Width / 2;
            dimensiones = value;
        }
    }

    static int size = 35;          // Tamaño del nodo
    Size dimensiones;
    Color color_nodo;              // Color definido para el nodo
    Color color_fuente;            // Color definido para la fuente del nombre del nodo
    Point _posicion;               // Dónde se dibujará el nodo
    int radio;                     //  Radio del objeto que representa el nodo

            //  Métodos

        // Constructor de la clase, recibe como parámetro el nombre del nodo (el valor que tendrá)
    public CVertice (string Valor)
    {
        this.Valor = Valor;
        this.ListaAdyacencia = new List <CArco> ( );
        this._banderas = new Dictionary <string, short> ( );
        this._banderas_predeterminado = new Dictionary <string, short> ( );
        this.Color = Color.Green;                          // Definimos el color del nodo
        this.Dimensiones = new Size (size, size);          // Definimos las dimensiones del circulo
        this.FontColor = Color.White;                      // Color de la fuente
    }

    public CVertice ( ) : this (" ") { }        // Constructor por defecto

                // Método para dibujar el nodo
    public void DibujarVertice (Graphics g)
    {
        SolidBrush b = new SolidBrush (this.color_nodo);

                    // 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 ( );     // Para liberar los recursos utilizados por el objeto
    }

                // Método para dibujar los arcos
    public void DibujarArco (Graphics g)
    {
        float distancia;
        int difY, difX;
```

```
            foreach (CArco arco in ListaAdyacencia)
            {
               difX = this.Posicion.X - arco.nDestino.Posicion.X;
               difY = this.Posicion.Y - arco.nDestino.Posicion.Y;

               distancia = (float) Math.Sqrt ((difX * difX + difY * difY));

               AdjustableArrowCap bigArrow = new AdjustableArrowCap (4, 4, true);
               bigArrow.BaseCap = System.Drawing.Drawing2D.LineCap.Triangle;

               g.DrawLine (new Pen (new SolidBrush (arco.color), arco.grosor_flecha)
               {    CustomEndCap = bigArrow, Alignment = PenAlignment.Center },
                    _posicion,
                    new Point (arco.nDestino.Posicion.X + (int) (radio * difX / distancia),
                               arco.nDestino.Posicion.Y + (int) (radio * difY / distancia)
                    )
               );

               g.DrawString (
                  arco.peso.ToString ( ),
                  new Font ("Times New Roman ", 12),
                  new SolidBrush (Color.White),
                  this._posicion.X - (int) ((difX / 3)),
                  this._posicion.Y - (int) ((difY / 3)),
                  new StringFormat ()
                  {
                     Alignment = StringAlignment.Center,
                     LineAlignment = StringAlignment.Far
                  }
               );
            }
         }

                  // Método para detectar posición en el panel donde se dibujará el nodo
         public bool DetectarPunto (Point p)
         {
            GraphicsPath posicion = new GraphicsPath ( );

            posicion.AddEllipse (new Rectangle (this._posicion.X - this.dimensiones.Width / 2,
                                                this._posicion.Y - this.dimensiones.Height / 2,
                                                this.dimensiones.Width, this.dimensiones.Height));

            bool retval = posicion.IsVisible (p);
            posicion.Dispose ( );
            return retval;
         }

         public string ToString ( )
         {
            return this.Valor;
         }
      }
   }
```

A continuación agregamos otra clase al proyecto, le colocaremos como nombre "**CArco**", la cual nos servirá para definir los arcos (aristas) del grafo.

En esta clase colocamos el siguiente código:

CArco.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;

namespace Grafos
{
   class CArco
   {                    // Atributos
          public CVertice nDestino;
          public int peso;                    // Peso (valor) de cada arco (Arista)

          public float grosor_flecha;
          public Color color;

                    // Métodos
          public CArco (CVertice destino) : this (destino, 1)
          {
                    this.nDestino = destino;
          }

          public CArco (CVertice destino, int peso)
          {
              this.nDestino = destino;
              this.peso = peso;
              this.grosor_flecha = 2;
              this.color = Color.Red;
          }
   }
}
```

Ahora agregamos la clase que nos servirá para implementar la estructura de datos grafo con listas de adyacencia; le llamaremos a esta clase "**CLista**"

En esta clase colocamos el siguiente código:

CLista.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Grafos
{
```

```
class CLista
{           //  Atributos
    private CVertice aElemento;
    private CLista aSubLista;
    private int aPeso;

            // Constructores
    public CLista ( )
    {       aElemento = null;
            aSubLista = null;
            aPeso = 0;
    }

    public CLista (CLista pLista)
    {       if (pLista != null)
            {       aElemento = pLista.aElemento;
                    aSubLista = pLista.aSubLista;
                    aPeso = pLista.aPeso;
            }
    }

    public CLista (CVertice pElemento, CLista pSubLista, int pPeso)
    {       aElemento = pElemento;
            aSubLista = pSubLista;
            aPeso = pPeso;
    }

            // Propiedades
    public CVertice Elemento
    {       get
            {       return aElemento; }

            set
            {       aElemento = value; }
    }

    public CLista SubLista
    {       get
            {       return aSubLista; }
            set
            {        aSubLista = value; }
    }

    public int Peso
    {       get
            {        return aPeso; }

            set
            {       aPeso = value; }
    }

            // Métodos
            public bool EsVacia ( )             // Verificar si la lista está vacía
    {       return aElemento == null;
    }
```

```
public void Agregar (CVertice pElemento, int pPeso)
{       if (pElemento != null)
        {       if (aElemento == null)
                {       aElemento = new CVertice (pElemento.nombre);
                        aPeso = pPeso;
                        aSubLista = new CLista ( );
                }
                else
                {       if (!ExisteElemento (pElemento))
                            aSubLista.Agregar (pElemento, pPeso);
                }
        }
}

public void Eliminar (CVertice pElemento)
{       if (aElemento != null)
        {       if (aElemento.Equals (pElemento))
                {       aElemento = aSubLista.aElemento;
                        aSubLista = aSubLista.SubLista;
                }
                else
                        aSubLista.Eliminar (pElemento);
        }
}

public int NroElementos ( )
{       if (aElemento != null)
                return 1 + aSubLista.NroElementos ( );
        else
                return 0;
}

public object IesimoElemento (int posicion)
{       if ((posicion > 0) && (posicion <= NroElementos ( )))
                if (posicion == 1)
                        return aElemento;
                else
                        return aSubLista.IesimoElemento (posicion - 1);
        else
                return null;
}

public object IesimoElementoPeso (int posicion)
{       if ((posicion > 0) && (posicion <= NroElementos ( )))
                if (posicion == 1)
                        return aPeso;
                else
                        return aSubLista.IesimoElementoPeso (posicion - 1);
        else
                return 0;
}

public bool ExisteElemento (CVertice pElemento)
{       if ((aElemento != null) && (pElemento != null))
```

```
                {       return (aElemento.Equals (pElemento) ||
                                (aSubLista.ExisteElemento (pElemento)));
                }
                else
                        return false;
        }

        public int PosicionElemento (CVertice pElemento)
        {       if ((aElemento != null) || (ExisteElemento (pElemento)))
                        if (aElemento.Equals (pElemento))
                                return 1;
                        else
                                return 1 + aSubLista.PosicionElemento (pElemento);
                else
                        return 0;
        }
    }
}
```

Finalmente agregamos la clase donde definiremos la estructura de datos grafo, la llamaremos "CGrafo"**.**

En esta clase colocamos el siguiente código:

CGrafo.cs

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace Grafos
{
    class CGrafo
    {
        public List <CVertice> nodos;        // Lista de nodos del grafo

        public CGrafo ( )                    // Constructor
        {
            nodos = new List <CVertice> ( );
        }

//====================Operaciones Básicas================================

            //Construye un nodo a partir de su valor y lo agrega a la lista de nodos
        public CVertice AgregarVertice (string valor)
        {
            CVertice nodo = new CVertice (valor);
            nodos.Add (nodo);
            return nodo;
        }
```

```
        //Agrega un nodo a la lista de nodos del grafo
public void AgregarVertice (CVertice nuevonodo)
{
   nodos.Add (nuevonodo);
}

       //Busca un nodo en la lista de nodos del grafo
public CVertice BuscarVertice (string valor)
{
   return nodos.Find (v => v.Valor == valor);
}

    //Crea una arista a partir de los valores de los nodos de origen y de destino
public bool AgregarArco (string origen, string nDestino, int peso = 1)
{
   CVertice vOrigen, vnDestino;

            //Si alguno de los nodos no existe, se activa una excepción
   if ((vOrigen = nodos.Find (v => v.Valor == origen)) == null)
      throw new Exception ("El nodo " + origen + " no existe dentro del grafo");

   if ((vnDestino = nodos.Find (v => v.Valor == nDestino)) == null)
      throw new Exception ("El nodo " + nDestino + " no existe dentro del grafo");

   return AgregarArco (vOrigen, vnDestino);
}

            // Crea la arista a partir de los nodos de origen y de destino
public bool AgregarArco (CVertice origen, CVertice nDestino, int peso = 1)
{
   if (origen.ListaAdyacencia.Find (v => v.nDestino == nDestino) == null)
   {
      origen.ListaAdyacencia.Add (new CArco (nDestino, peso));
      return true;
   }
   return false;
}

    // Método para dibujar el grafo
public void DibujarGrafo (Graphics g)
{
        // Dibujando los arcos
   foreach (CVertice nodo in nodos)
      nodo.DibujarArco (g);

       // Dibujando los vértices
   foreach (CVertice nodo in nodos)
      nodo.DibujarVertice (g);
}

    // Método para detectar si se ha posicionado sobre algún nodo y lo devuelve
public CVertice DetectarPunto (Point posicionMouse)
{
   foreach (CVertice nodoActual in nodos)
      if (nodoActual.DetectarPunto (posicionMouse)) return nodoActual;
```

```
            return null;
        }

            // Método para regresar al estado original
        public void ReestablecerGrafo (Graphics g)
        {
            foreach (CVertice nodo in nodos)
            {
                nodo.Color = Color.White;
                nodo.FontColor = Color.Black;
                foreach (CArco arco in nodo.ListaAdyacencia)
                {
                    arco.grosor_flecha = 1;
                    arco.color = Color.Black;
                }
            }

            DibujarGrafo (g);
        }
    }
}
```

Hasta el momento hemos definido todas las clases que necesitaremos en nuestra aplicación.

A continuación, programaremos la funcionalidad de la aplicación, es decir el código para poder dibujar los vértices y los arcos de nuestro grafo.

Vamos a trabajar con la codificación de nuestro formulario "**Simulador**".

Agregamos la librería "Drawing2D".

A continuación definimos todas las variables que utilizaremos, así:

```
private CGrafo grafo;                // instanciamos la clase CGrafo
private CVertice nuevoNodo;          // instanciamos la clase Cvertice para crear el nodo "nuevoNodo"
private CVertice NodoOrigen;         // instanciamos la clase Cvertice para crear el nodo "NodoOrigen"
private CVertice NodoDestino;        // instanciamos la clase Cvertice para crear el nodo "NodoDestino"
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 Vertice ventanaVertice;   // ventana para agregar los vértices
```

El código en el constructor del formulario es el siguiente:

```
public Simulador ( )
{
        InitializeComponent ( );
        grafo = new CGrafo ( );
        nuevoNodo = null;
        var_control = 0;
```

```
            ventanaVertice = new Vertice ( );

            this.SetStyle (ControlStyles.AllPaintingInWmPaint | ControlStyles.UserPaint |
                           ControlStyles.DoubleBuffer, true);
 }
```

El código en los eventos “Paint” y “MouseLeave” del objeto “Panel” es el siguiente:

```
private void Pizarra_Paint (object sender, PaintEventArgs e)
{
        try
        {
           e.Graphics.SmoothingMode = SmoothingMode.HighQuality;
           grafo.DibujarGrafo (e.Graphics);
        }
        catch (Exception ex)
        {
           MessageBox.Show (ex.Message);
        }
 }

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

El evento “MouseLeave” ocurre cuando el mouse ya no se encuentra en la parte visible del control.

Agregaremos un nuevo objeto al formulario “**Simulador**”.

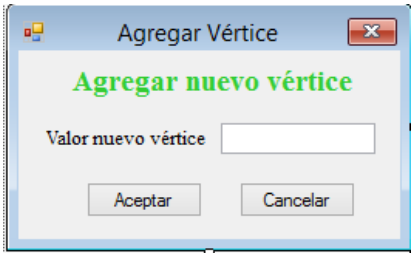
Agregamos un ContextMenuStrip, le agregamos el ítem “Nuevo Vértice” y le cambiamos la propiedad “Name” a “**CMSCrearVertice**”.

El código asociado a este objeto es el siguiente:

```
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 acción, 1: Dibujando arco, 2: Nuevo vértice
}
```

Vamos a agregar un nuevo formulario a la aplicación, el cual nos servirá para proporcionar la etiqueta de los vértices del grafo. Le daremos el nombre de “**Vértice**”.

Se sugiere que luzca parecido a este formulario:



Se han agregado al formulario dos Labels, dos Buttons y un TextBox.

Al objeto TextBox le cambiamos la propiedad “Name” a “**txtVertice**” y le cambiamos la propiedad “Modifiers” a “**Public**”**.** La propiedad “Modifiers” indica el nivel de visibilidad del objeto.

Al primer botón insertado lo rotulamos “Aceptar” y le cambiamos la propiedad “Name” a “**btnAceptar**”.

Al segundo botón insertado lo rotulamos “Cancelar” y le cambiamos la propiedad “Name” a “**btnCancelar**”.

El código que necesitamos en este formulario es el siguiente:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Grafos
{
   public partial class Vertice : Form
   {
      public bool control;       // variable de control
      public string dato;        // el dato que almacenará el vértice

   public Vertice ( )
   {
         InitializeComponent ( );
         control = false;
         dato = " ";
   }

private void btnAceptar_Click (object sender, EventArgs e)
      {
         string valor = txtVertice.Text.Trim ( );

          if ((valor == "") || (valor == " "))
         {
```

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

    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);
        }
    }
  }
}
```

Para dibujar tanto los vértices (nodos) como los arcos (aristas) de nuestro grafo utilizaremos el mouse y algunos de sus eventos dentro del objeto “Panel”.

Haremos uso de los eventos del mouse sobre el objeto “Panel”.

El código a agregar en el formulario “**Simulador**” es el siguiente:

```
private void Pizarra_MouseUp (object sender, MouseEventArgs e)
{
    switch (var_control)
    {
        case 1:          // Dibujando arco
            if ((NodoDestino = grafo.DetectarPunto (e.Location)) != null && NodoOrigen != NodoDestino)
            {
                if (grafo.AgregarArco (NodoOrigen, NodoDestino))            //Se procede a crear la arista
                {
                    int distancia = 0;
                    NodoOrigen.ListaAdyacencia.Find(v => v.nDestino == NodoDestino).peso = distancia;
                }
            }
            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 acción, 1: Dibujando arco, 2: Nuevo vértice
            }

            if (nuevoNodo != null && NodoOrigen == null)
            {
                ventanaVertice.Visible = false;
                ventanaVertice.control = false;
                grafo.AgregarVertice (nuevoNodo);
                ventanaVertice.ShowDialog ( );

                if (ventanaVertice.control)
                {
                    if (grafo.BuscarVertice (ventanaVertice.txtVertice.Text) == null)
                    {
                        nuevoNodo.Valor = ventanaVertice.txtVertice.Text;
                    }
                    else
                    {
                        MessageBox.Show ("El Nodo " + ventanaVertice.txtVertice.Text + " ya existe en el
                                          grafo", "Error nuevo Nodo", MessageBoxButtons.OK,
                                          MessageBoxIcon.Exclamation);
                    }
                }
                nuevoNodo = null;
                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)
                {
                    CMSCrearVertice.Text = "Nodo " + NodoOrigen.Valor;
                }
                else
                    Pizarra.ContextMenuStrip = this.CMSCrearVertice;
```

```
        }
      }
    }
```

El evento "**MouseUp**" ocurre cuando el puntero del mouse se encuentra sobre el componente y se suelta un botón del mouse que se había presionado previamente.
El evento "**MouseDown**" ocurre cuando el puntero del mouse se encuentra sobre el componente y se presiona un botón del mouse.
El evento "**MouseMove**" ocurre cuando el puntero del mouse se mueve sobre el componente.

Ahora si podemos probar la aplicación, sabiendo que para dibujar un vértice lo haremos dando click derecho con el mouse sobre el objeto "Panel", y para dibujar un arco lo haremos dando click izquierdo sobre un vértice (origen) y sin soltar lo unimos a otro vértice (destino).

Análisis de resultados

Ejercicio 1.

Analizar el código proporcionado y realizar la documentación del mismo (agregar comentarios al código), considerando lo siguiente:

- Identificar el tipo de estructura utilizada para el manejo del grafo.
- Identificar el objetivo de cada atributo de las clases utilizadas y colocarlo como comentario en el código del programa.
- Determinar la utilidad de cada función proporcionada en las clases y colocarlo como comentario en el código del programa.
- Identificar para cada opción del menú, el método invocado y los subsecuentes métodos para llevar a cabo la tarea.

Ejercicio 2.

Para el ejemplo desarrollado, agregar al menú, las siguientes opciones:

- ✓ Eliminar Vértice.
- ✓ Eliminar Arco.

Investigación Complementaria

Para la siguiente semana:

Basándose en el código de ejemplo proporcionado, implementar la simulación del Grafo en una interfaz gráfica de formulario (Windows Forms).

Deben considerarse las siguientes operaciones:

A. Crear grafo
B. Agregar vértice
C. Agregar arco, solicitando el valor del peso al usuario
D. Eliminar vértice
E. Eliminar arco