PED941
Ciclo 01-2024
G01T

UDB VIRTUAL
Universidad Don Bosco, El Salvador

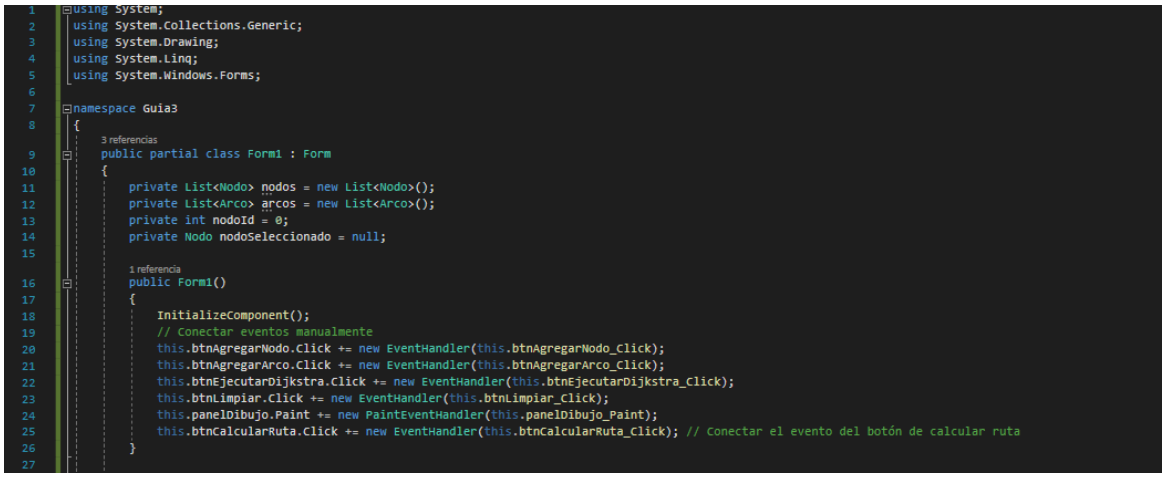
Guia 3

Docente: Carmen Celia Morales Samayoa

Apellidos	Nombres	Carné
González Romero	Irvin Eduardo	GR202825

REPOSITORIO: https://github.com/Irvin-g/Guia3-PED

El código presentado es parte de una aplicación de Windows Forms en C#. Define una clase Form1 que hereda de Form. Dentro de Form1, se inicializan listas para almacenar nodos y arcos de un grafo, un identificador para los nodos, y una referencia al nodo seleccionado. El constructor de Form1 conecta manualmente los eventos de varios botones (btnAgregarNodo, btnAgregarArco, btnEjecutarDijkstra, btnLimpiar, btnCalcularRuta) y del panel de dibujo (panelDibujo) a sus respectivos manejadores de eventos. Esto permite que, al hacer clic en los botones, se ejecuten funciones específicas para agregar nodos, agregar arcos, ejecutar el algoritmo de Dijkstra, limpiar el grafo y calcular rutas, mientras que el evento de pintura del panel se utiliza para redibujar el grafo en la interfaz gráfica.



```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Windows.Forms;

namespace Guia3
{
    3 referencias
    public partial class Form1 : Form
    {
        private List<Nodo> nodos = new List<Nodo>();
        private List<Arco> arcos = new List<Arco>();
        private int nodoId = 0;
        private Nodo nodoSeleccionado = null;

        1 referencia
        public Form1()
        {
            InitializeComponent();
            // Conectar eventos manualmente
            this.btnAgregarNodo.Click += new EventHandler(this.btnAgregarNodo_Click);
            this.btnAgregarArco.Click += new EventHandler(this.btnAgregarArco_Click);
            this.btnEjecutarDijkstra.Click += new EventHandler(this.btnEjecutarDijkstra_Click);
            this.btnLimpiar.Click += new EventHandler(this.btnLimpiar_Click);
            this.panelDibujo.Paint += new PaintEventHandler(this.panelDibujo_Paint);
            this.btnCalcularRuta.Click += new EventHandler(this.btnCalcularRuta_Click); // Conectar el evento del botón de calcular ruta
        }
```

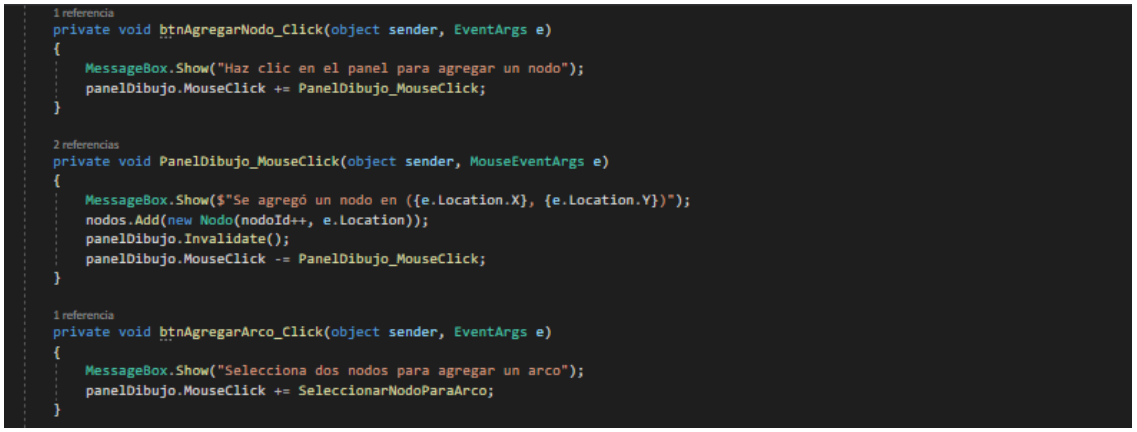
El código presentado define manejadores de eventos para agregar nodos y arcos en un grafo interactivo.

btnAgregarNodo_Click: Al hacer clic en el botón para agregar un nodo, muestra un mensaje y habilita la detección de clics en el panel de dibujo.

PanelDibujo_MouseClick: Al hacer clic en el panel, agrega un nodo en la ubicación del clic, muestra un mensaje con las coordenadas, redibuja el panel, y desactiva la detección de clics.

btnAgregarArco_Click: Al hacer clic en el botón para agregar un arco, muestra un mensaje y habilita la selección de nodos mediante clics en el panel.

Estos manejadores permiten al usuario agregar nodos y definir arcos interactivamente en el panel de dibujo.



```
1 referencia
private void btnAgregarNodo_Click(object sender, EventArgs e)
{
    MessageBox.Show("Haz clic en el panel para agregar un nodo");
    panelDibujo.MouseClick += PanelDibujo_MouseClick;
}

2 referencias
private void PanelDibujo_MouseClick(object sender, MouseEventArgs e)
{
    MessageBox.Show($"Se agregó un nodo en ({e.Location.X}, {e.Location.Y})");
    nodos.Add(new Nodo(nodoId++, e.Location));
    panelDibujo.Invalidate();
    panelDibujo.MouseClick -= PanelDibujo_MouseClick;
}

1 referencia
private void btnAgregarArco_Click(object sender, EventArgs e)
{
    MessageBox.Show("Selecciona dos nodos para agregar un arco");
    panelDibujo.MouseClick += SeleccionarNodoParaArco;
}
```

El código permite seleccionar nodos para crear un arco en el grafo:

SeleccionarNodoParaArco: Al hacer clic en el panel, busca un nodo en la ubicación del clic.

Si no hay un nodo seleccionado, selecciona el nodo y pide seleccionar el nodo de destino.

Si ya hay un nodo seleccionado, pide el peso del arco y, si es válido, crea el arco, muestra un mensaje de confirmación, restablece la selección y actualiza el panel. Luego desactiva la detección de clics para arcos.

Esto permite al usuario interactuar con el grafo para definir arcos entre nodos.

```
2 referencias
private void SeleccionarNodoParaArco(object sender, MouseEventArgs e)
{
    var nodo = nodos.FirstOrDefault(n => EsNodoSeleccionado(n, e.Location));
    if (nodo != null)
    {
        if (nodoSeleccionado == null)
        {
            nodoSeleccionado = nodo;
            MessageBox.Show($"Nodo {nodo.Id} seleccionado. Ahora selecciona el nodo de destino.");
        }
        else
        {
            int peso = PromptForPeso();
            if (peso > 0)
            {
                arcos.Add(new Arco(nodoSeleccionado, nodo, peso));
                MessageBox.Show($"Arco agregado desde {nodoSeleccionado.Id} hasta {nodo.Id} con peso {peso}");
                nodoSeleccionado = null;
                panelDibujo.Invalidate();
                panelDibujo.MouseClick -= SeleccionarNodoParaArco;
            }
        }
    }
}
```

El código proporciona dos funciones:

EsNodoSeleccionado: Esta función verifica si un punto dado está dentro del radio de un nodo en el panel. Utiliza la fórmula de distancia entre dos puntos en un plano cartesiano para calcular la distancia entre el punto y el centro del nodo. Si esta distancia es menor o igual al radio del nodo, se considera que el punto está dentro del nodo.

btnEjecutarDijkstra_Click: Este manejador de eventos se activa cuando se hace clic en el botón para ejecutar el algoritmo de Dijkstra. Verifica si se ingresó un número de nodo inicial válido en un campo de texto (txtNodoInicial). Si el número es válido, muestra un mensaje indicando que se está ejecutando Dijkstra desde el nodo especificado. Luego, crea una instancia del algoritmo de Dijkstra y lo ejecuta con

los nodos y arcos actuales. Finalmente, muestra las distancias calculadas por el algoritmo en la interfaz gráfica. Si el número de nodo inicial no es válido, muestra un mensaje de error.

```
1 referencia
private bool EsNodoSeleccionado(Nodo nodo, Point punto)
{
    int radio = 10;
    return Math.Sqrt(Math.Pow(nodo.Posicion.X - punto.X, 2) + Math.Pow(nodo.Posicion.Y - punto.Y, 2)) <= radio;
}

1 referencia
private void btnEjecutarDijkstra_Click(object sender, EventArgs e)
{
    if (int.TryParse(txtNodoInicial.Text, out int nodoInicial) && nodoInicial >= 0 && nodoInicial < nodos.Count)
    {
        MessageBox.Show($"Ejecutando Dijkstra desde el nodo {nodoInicial}");
        var algoritmo = new AlgoritmoDijkstra(nodos, arcos);
        algoritmo.Ejecutar(nodoInicial);

        var distancias = algoritmo.ObtenerDistancias();
        MostrarDistancias(distancias);
    }
    else
    {
        MessageBox.Show("Por favor, ingresa un nodo inicial válido.");
    }
}
```

Estas funciones tienen como objetivo visualizar los resultados del algoritmo de Dijkstra y dibujar el grafo en la interfaz gráfica:

MostrarDistancias: Borra los resultados anteriores y muestra las distancias calculadas para cada nodo en un control de lista.

panelDibujo_Paint: Dibuja los nodos como círculos azules con su identificador y los arcos como líneas negras entre los nodos, mostrando también el peso de los arcos en el punto medio de cada arco.

```
1 referencia
96  private void MostrarDistancias(int[] distancias)
97  {
98      lstResultados.Items.Clear();
99      for (int i = 0; i < distancias.Length; i++)
100     {
101         lstResultados.Items.Add($"Distancia al nodo {i}: {distancias[i]}");
102     }
103     MessageBox.Show("Distancias calculadas y mostradas.");
104 }
105
    1 referencia
106 private void panelDibujo_Paint(object sender, PaintEventArgs e)
107 {
108     Graphics g = e.Graphics;
109
110     foreach (var nodo in nodos)
111     {
112         g.FillEllipse(Brushes.Blue, nodo.Posicion.X - 5, nodo.Posicion.Y - 5, 10, 10);
113         g.DrawString(nodo.Id.ToString(), DefaultFont, Brushes.White, nodo.Posicion);
114     }
115
116     foreach (var arco in arcos)
117     {
118         g.DrawLine(Pens.Black, arco.Origen.Posicion, arco.Destino.Posicion);
119         var puntoMedio = new Point((arco.Origen.Posicion.X + arco.Destino.Posicion.X) / 2, (arco.Origen.Posicion.Y + arco.Destino.Posicion.Y) / 2);
120         g.DrawString(arco.Peso.ToString(), DefaultFont, Brushes.Red, puntoMedio);
121     }
122 }
123
```

Esta función muestra un cuadro de diálogo para que el usuario ingrese el peso del arco que desea agregar. El cuadro de diálogo contiene un campo de texto para ingresar el peso y un botón "OK" para confirmar la entrada.

```
1 referencia
private int PromptForPeso()
{
    using (var form = new Form())
    {
        var lbl = new Label() { Left = 50, Top = 20, Text = "Peso del arco:" };
        var txt = new TextBox() { Left = 50, Top = 50, Width = 200 };
        var btnOk = new Button() { Text = "OK", Left = 150, Width = 100, Top = 80, DialogResult = DialogResult.OK };

        form.Controls.Add(lbl);
        form.Controls.Add(txt);
        form.Controls.Add(btnOk);
        form.AcceptButton = btnOk;

        if (form.ShowDialog() == DialogResult.OK && int.TryParse(txt.Text, out int peso))
        {
            return peso;
        }
        else
        {
            MessageBox.Show("Por favor, ingresa un peso válido.");
            return 0;
        }
    }
}
```

Se limpian los componentes

```
2 referencias
private void btnLimpiar_Click(object sender, EventArgs e)
{
    // Limpiar nodos y arcos
    nodos.Clear();
    arcos.Clear();
    nodoId = 0;
    nodoSeleccionado = null;

    // Limpiar cuadro de texto
    txtNodoInicial.Text = string.Empty;

    // Limpiar ListBox
    lstResultados.Items.Clear();

    // Limpiar el panel de dibujo
    panelDibujo.Invalidate();
}
```

Esta función se activa cuando se hace clic en el botón para calcular la ruta más corta. Verifica si hay distancias calculadas en el control de lista lstResultados. Si no hay ninguna distancia calculada, muestra un mensaje de advertencia y sale de la función. Luego, busca la menor distancia calculada entre los nodos y muestra esta distancia como la ruta más corta en un mensaje emergente.

```
2 referencias
private void btnCalcularRuta_Click(object sender, EventArgs e)
{
    if (lstResultados.Items.Count == 0)
    {
        MessageBox.Show("No hay distancias calculadas.");
        return;
    }

    int menorDistancia = int.MaxValue;

    foreach (var item in lstResultados.Items)
    {
        string[] partes = item.ToString().Split(':');
        if (partes.Length == 2 && int.TryParse(partes[1].Trim(), out int distancia) && distancia > 0)
        {
            if (distancia < menorDistancia)
            {
                menorDistancia = distancia;
            }
        }
    }

    if (menorDistancia == int.MaxValue)
    {
        MessageBox.Show("No se encontraron distancias");
    }
    else
    {
        MessageBox.Show($"La menor distancia mas corta es: {menorDistancia}");
    }
}
}
```

Estas clases definen la estructura de un grafo y el algoritmo de Dijkstra para encontrar las distancias más cortas en un grafo ponderado.

Nodo: Representa un nodo en el grafo con un identificador (Id) y una posición (Posicion).

Arco: Representa un arco dirigido entre dos nodos, con un nodo de origen (Origen), un nodo de destino (Destino) y un peso (Peso).

AlgoritmoDijkstra: Implementa el algoritmo de Dijkstra para encontrar las distancias más cortas entre un nodo inicial y todos los demás nodos en el grafo. Usa listas de nodos y arcos, y calcula las distancias y predecesores para cada nodo.

```
12 referencias
public class Nodo
{
    12 referencias
    public int Id { get; set; }
    12 referencias
    public Point Posicion { get; set; }

    1 referencia
    public Nodo(int id, Point posicion)
    {
        Id = id;
        Posicion = posicion;
    }
}

6 referencias
public class Arco
{
    5 referencias
    public Nodo Origen { get; set; }
    10 referencias
    public Nodo Destino { get; set; }
    public int Peso { get; set; }


    public Arco(Nodo origen, Nodo destino, int peso)
    {
        Origen = origen;
        Destino = destino;
        Peso = peso;
    }
}
```

```
2 referencias
public class AlgoritmoDijkstra
{
    private List<Nodo> nodos;
    private List<Arco> arcos;
    private int[] distancias;
    private int[] predecesores;

    1 referencia
    public AlgoritmoDijkstra(List<Nodo> nodos, List<Arco> arcos)
    {
        this.nodos = nodos;
        this.arcos = arcos;
        distancias = new int[nodos.Count];
        predecesores = new int[nodos.Count];
    }

    1 referencia
    public void Ejecutar(int nodoInicial)
    {
        for (int i = 0; i < nodos.Count; i++)
        {
            distancias[i] = int.MaxValue;
            predecesores[i] = -1;
        }
        distancias[nodoInicial] = 0;

        var colaPrioridad = new SortedSet<Tuple<int, int>>(Comparer<Tuple<int, int>>.Create((x, y) => x.Item2 == y.Item2 ? x.Item1.CompareTo(y.Item1) : x.Item2.CompareTo(y.Item2)));
        colaPrioridad.Add(Tuple.Create(nodoInicial, 0));

        while (colaPrioridad.Any())
        {
            var (nodoActual, distanciaActual) = colaPrioridad.Min;
            colaPrioridad.Remove(colaPrioridad.Min);
```

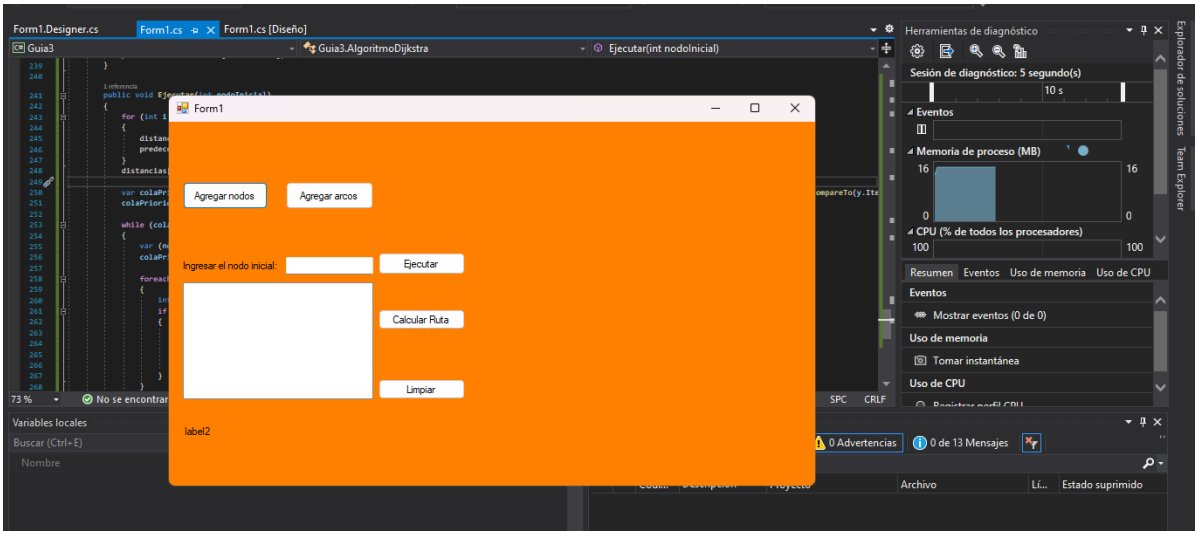
Estas son las funciones principales del algoritmo de Dijkstra:

Ejecutar: Inicializa las distancias y predecesores para todos los nodos, estableciendo la distancia del nodo inicial como 0. Luego, utiliza una cola de prioridad para explorar los nodos de acuerdo a sus distancias actuales. Para cada nodo explorado, actualiza las distancias de sus nodos vecinos si encuentra una ruta más corta. Este proceso continúa hasta que todos los nodos hayan sido explorados.

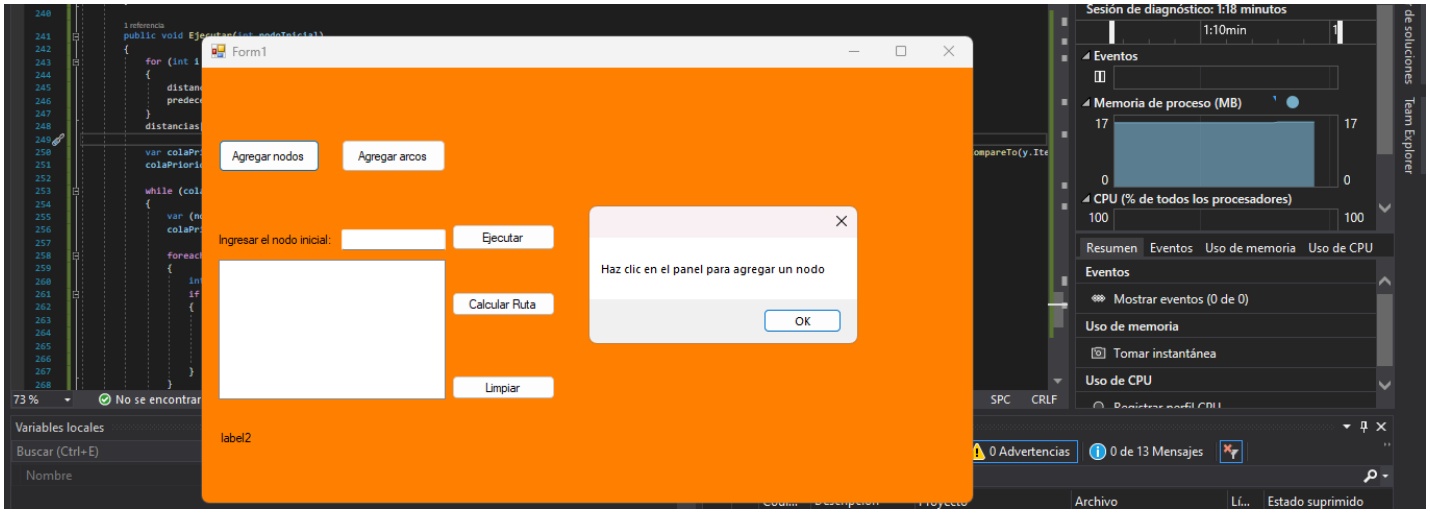
ObtenerDistancias: Devuelve un array que contiene las distancias más cortas desde el nodo inicial a todos los demás nodos.

```
        {
            distancias[i] = int.MaxValue;
            predecesores[i] = -1;
        }
        distancias[nodoInicial] = 0;
namespace Guia3
    public class AlgoritmoDijkstr
        var colaPrioridad = new SortedSet<Tuple<int, int>>(Comparer<Tuple<int, int>>.Create((x, y) => x.Item2 == y.Item2 ? x.Item1.CompareTo(y.Item1) : x.Item2.CompareTo(y.Item2)));
        colaPrioridad.Add(Tuple.Create(nodoInicial, 0));

        while (colaPrioridad.Any())
        {
            var (nodoActual, distanciaActual) = colaPrioridad.Min;
            colaPrioridad.Remove(colaPrioridad.Min);

            foreach (var arco in arcos.Where(a => a.Origen.Id == nodoActual))
            {
                int nuevoCosto = distanciaActual + arco.Peso;
                if (nuevoCosto < distancias[arco.Destino.Id])
                {
                    colaPrioridad.Remove(Tuple.Create(arco.Destino.Id, distancias[arco.Destino.Id]));
                    distancias[arco.Destino.Id] = nuevoCosto;
                    predecesores[arco.Destino.Id] = nodoActual;
                    colaPrioridad.Add(Tuple.Create(arco.Destino.Id, nuevoCosto));
                }
            }
        }
    }

    1 referencia
    public int[] ObtenerDistancias() => distancias;
}
```

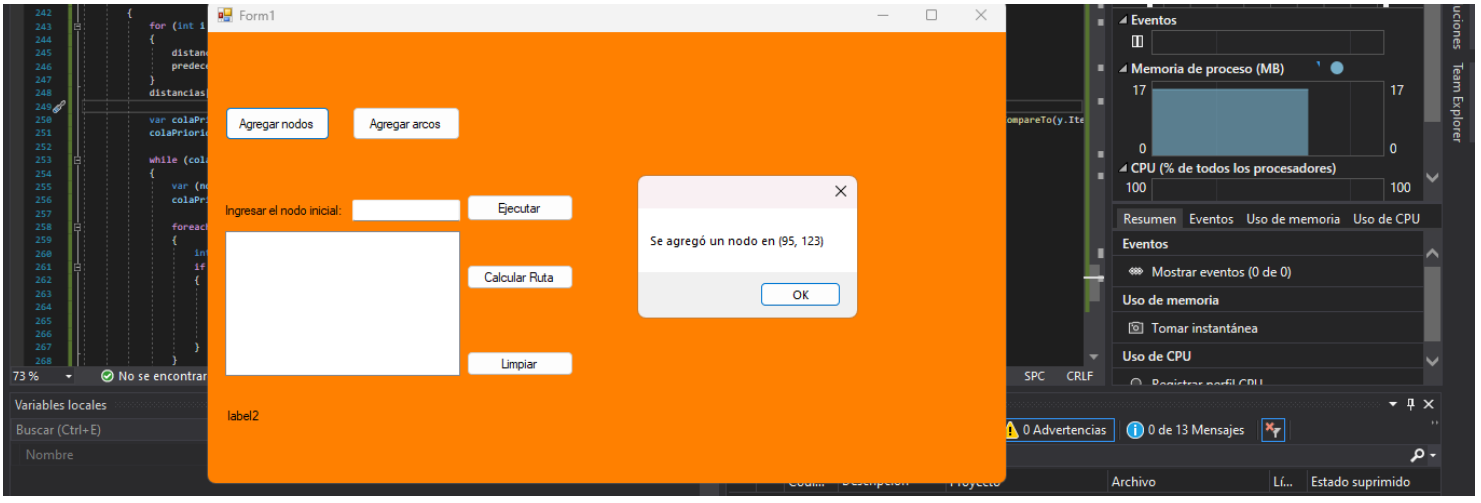
Resultados:, acá observamos el diseño, para empezar daremos click en agregar nodo y después damos click en el panel



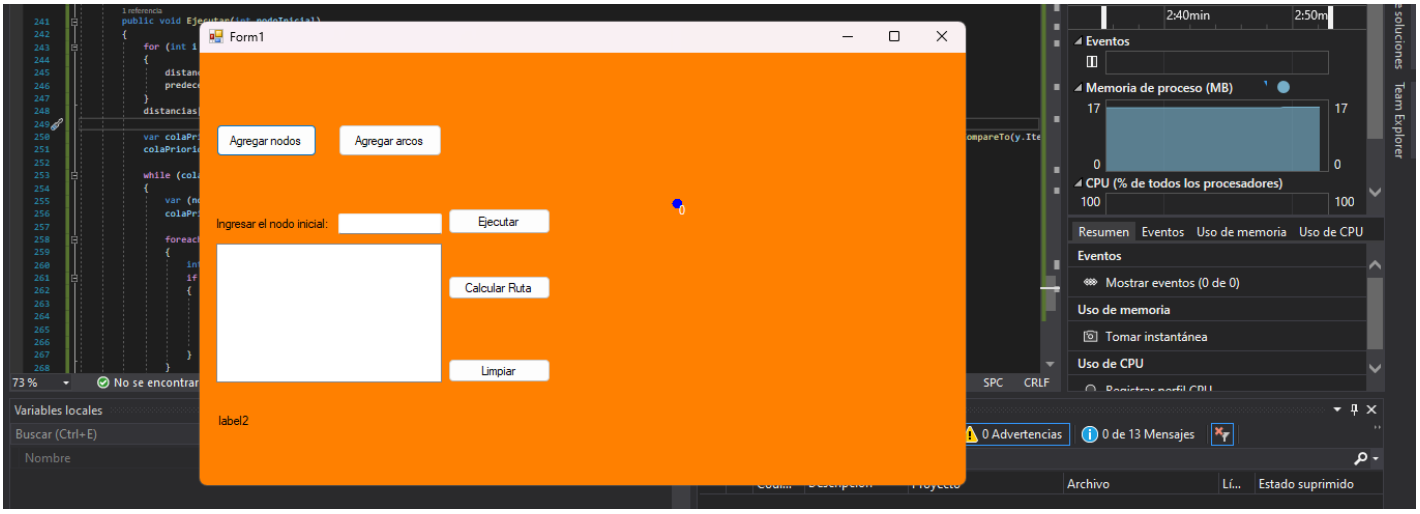
Al dar click sobre el botón se nos muestra esto, que es para presionar en el panel ahora



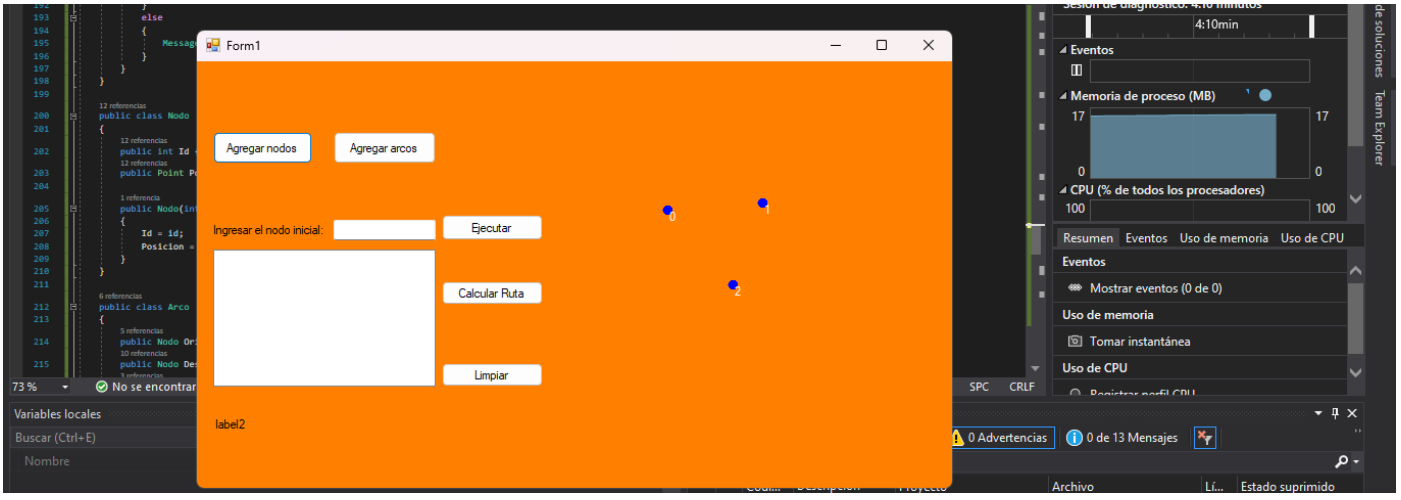
Luego de apretar en el panel nos sale el mensaje con coordenadas



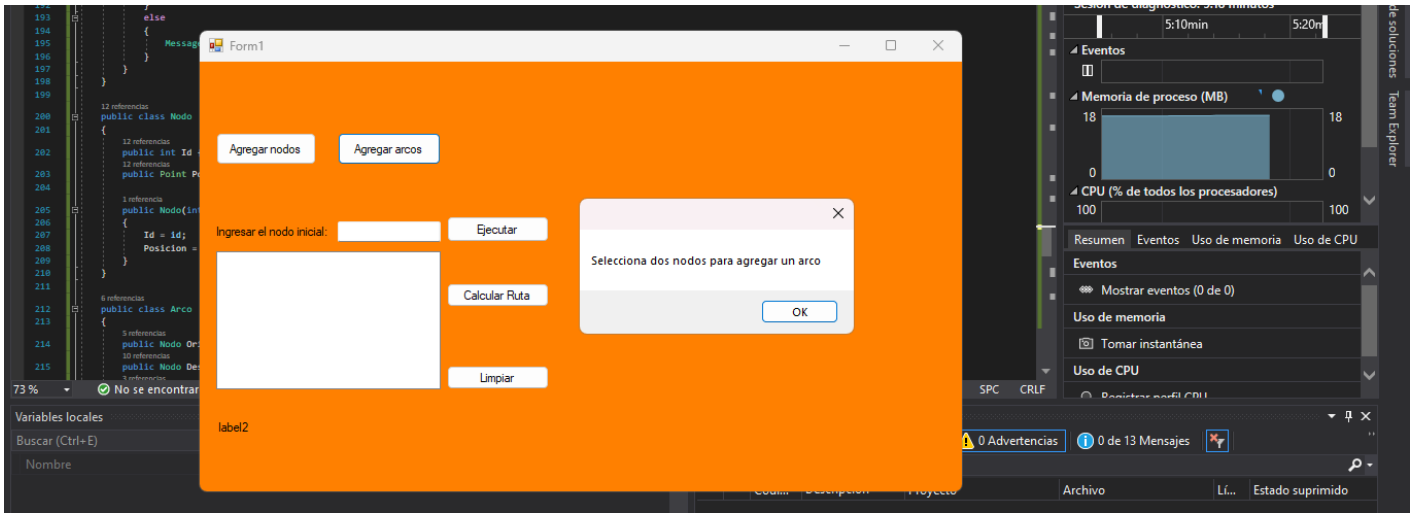
Observamos que si se agrego correctamente el nodo, ahora agregamos los demas



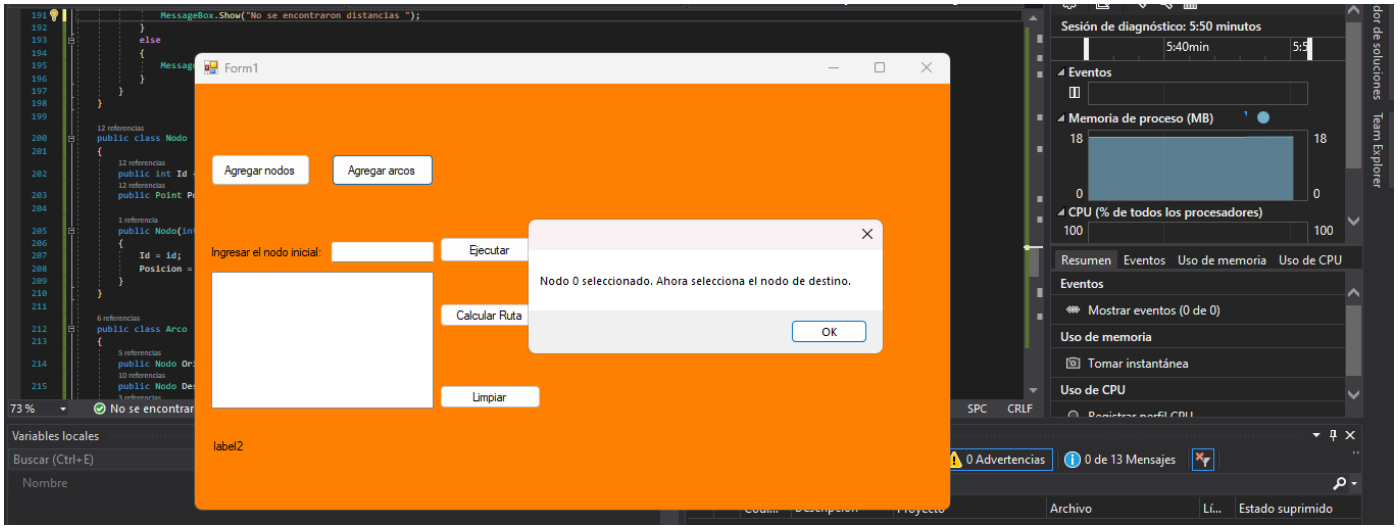
Agregamos 2 nodos mas



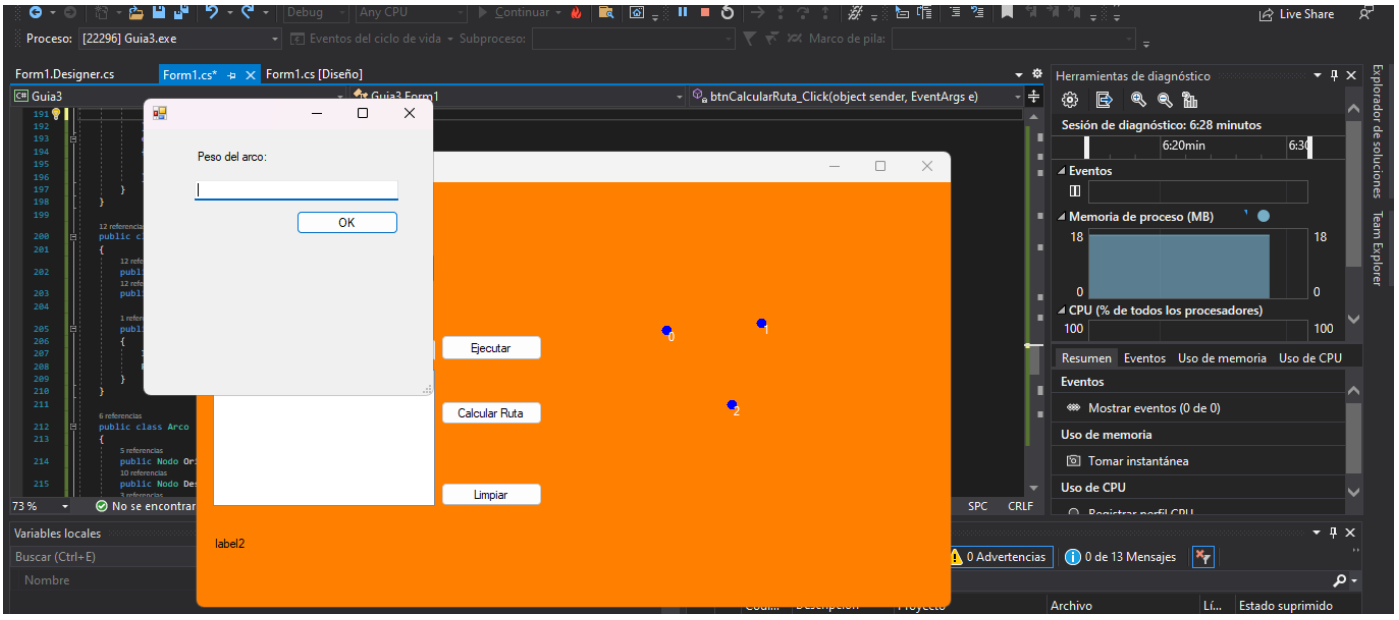
Para agregar los arcos, apretamos el botón y luego el nodo inicial y después el nodo final, al apretar el botón nos muestra



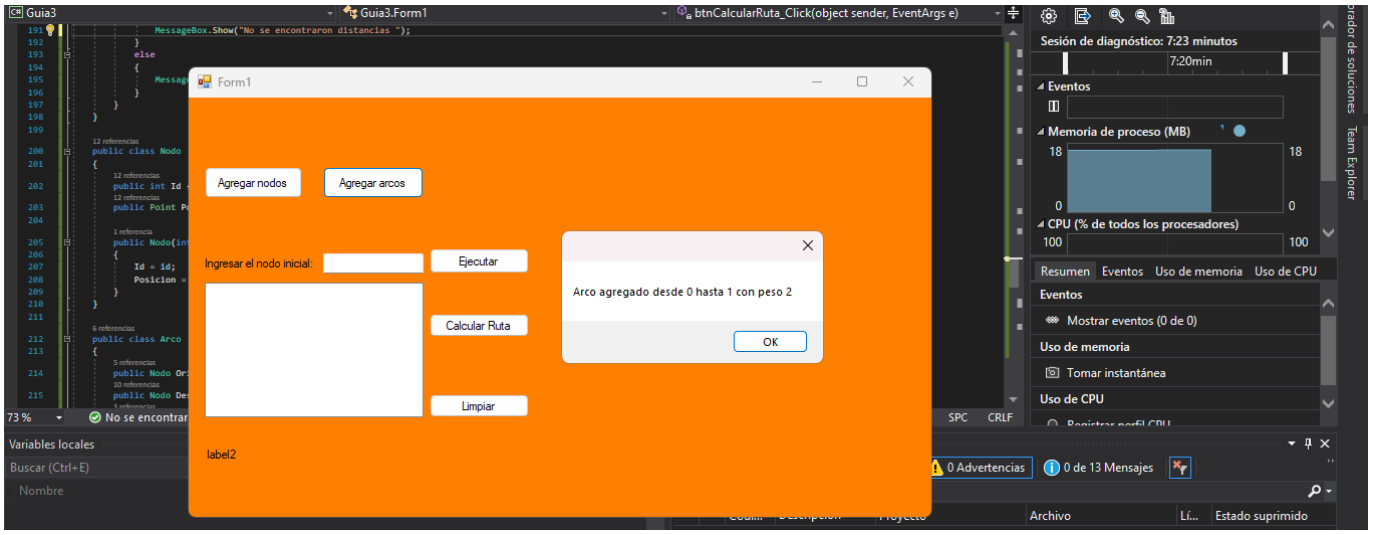
Después de seleccionar el primer nodo, sale



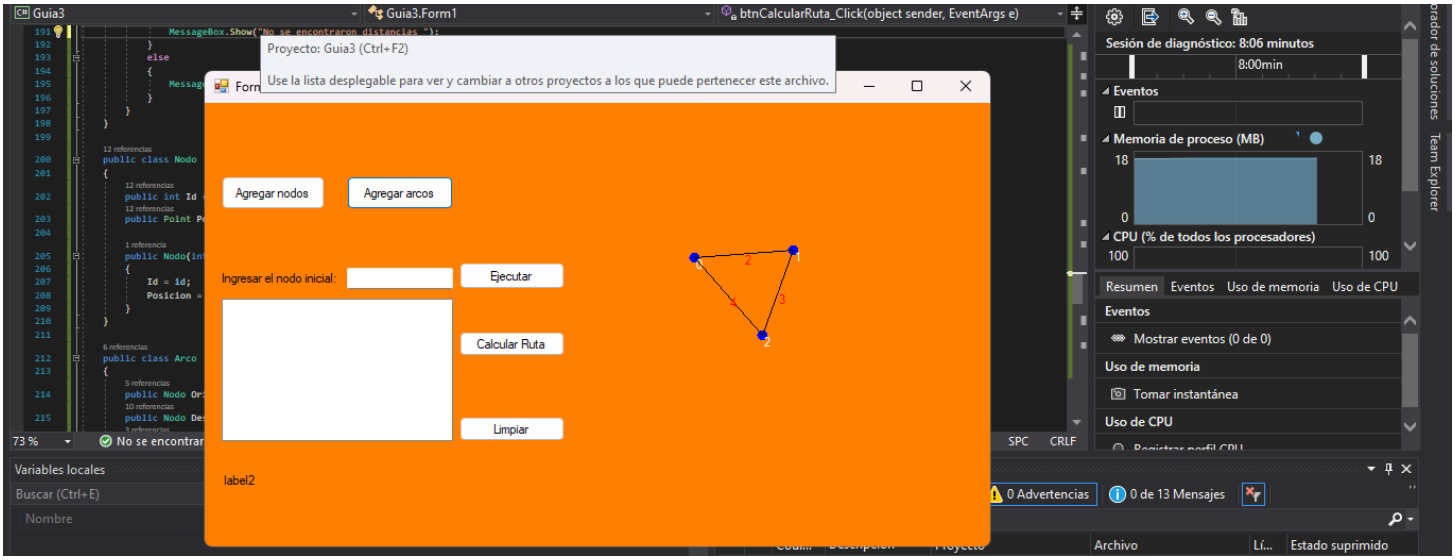
Al presionar nodo destino nos sale



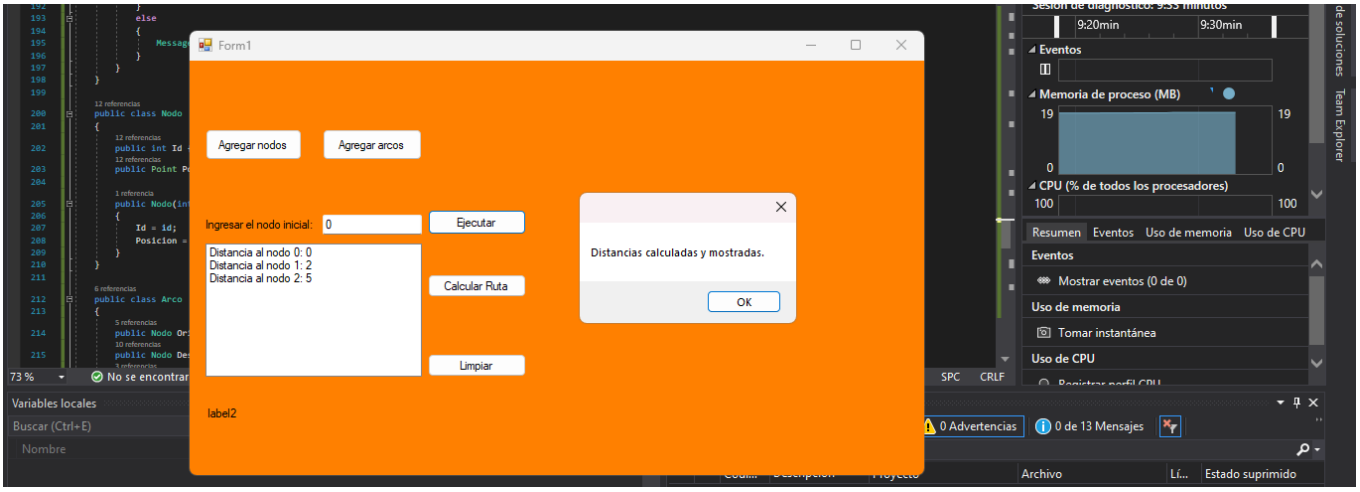
Agregar peso, así haremos en los demás arcos con los otros dos nodos



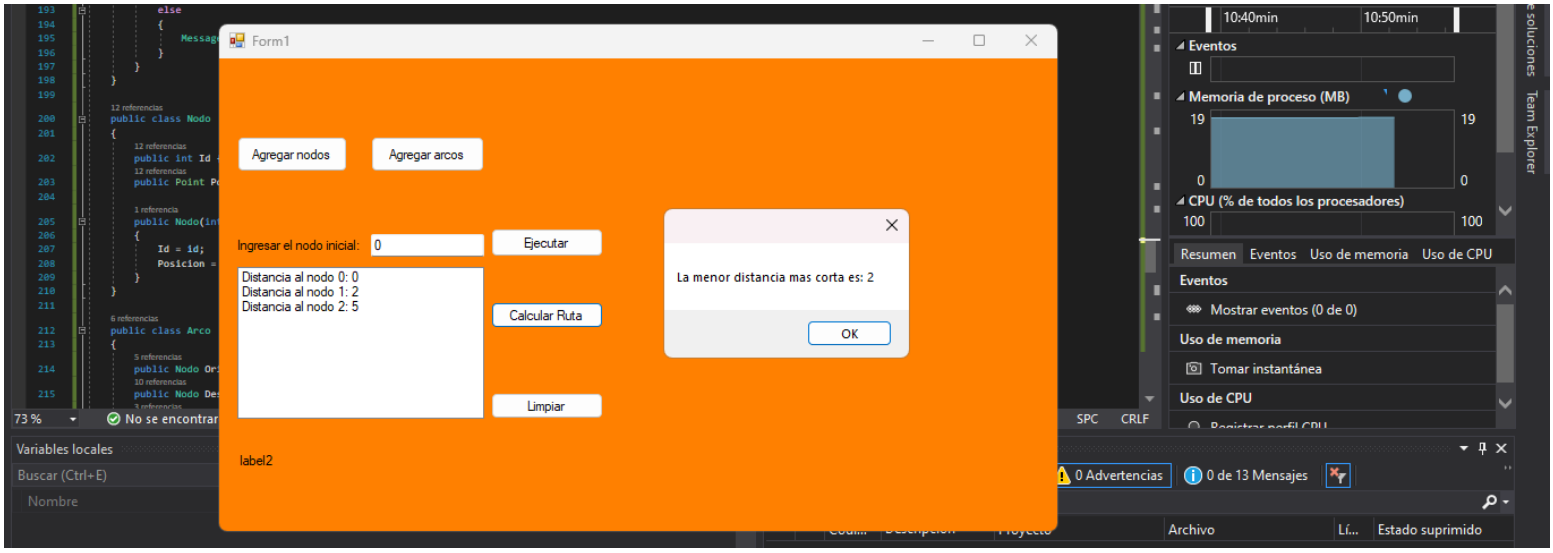
Acá se nos muestran todos los arcos realizados



Ahora ingresamos en el textbox el nodo inicial para poder ver sus resultados, en este caso nodo 0 y observamos los resultados



Luego damos en calcular ruta, y nos saldrá la ruta mas corta de las que se presentaron en el listbox de dicho nodo, en este caso es 2, recordemos que no es 0, porque el nodo que estamos calculando es el mismo nodo 0, entonces por eso sale 0, del nodo 0 al nodo 0, hay 0 de distancia, por eso se toma el valor menor y diferente a 0 para ser ruta mas corta



Por último limpiamos