Proyecto: Grafos Sistema de navegación urbana

Estudiante: María Chanto Hernández

Mapa de rutas

En Grafo.cs

Con la ayuda del diccionario asignaremos un número único a cada nodo del grafo

```
Dictionary<string, int> map = new Dictionary<string, int>(StringComparer.OrdinalIgnoreCase);
int[,] matriz;

1 referencia
public void InicializarMapa()
{
    map.Add("Policia", 0);
    map.Add("Restaurante", 1);
    map.Add("Parqueo Restaurante", 2);
    map.Add("Dentista", 3);
    map.Add("Familia Rojas", 4);
    map.Add("Heladeria", 5);
    map.Add("Universidad", 6);
    map.Add("Gasolinera", 7);
    map.Add("Central de Autobuses", 8);
    map.Add("Familia Ruiz", 9);
    map.Add("Parque Central", 10);
    map.Add("Iglesia", 11);
    map.Add("Dulceria", 12);
    map.Add("Panaderia", 13);
    map.Add("Supermercado", 14);
    map.Add("Familia Diaz", 15);
    map.Add("Hospital", 16);
    map.Add("Parque Sur", 17);
    map.Add("Zapateria", 18);
    map.Add("Escuela", 19);
```

Lista que contiene todas las conexiones con sus pesos, siendo asi fácil de manipular. Permite manejar un grafo ponderado y no dirigido.

```
List<(string, string, int)> list = new List<(string, string, int)>
    {
        ("Policia","Restaurante",3),
        ("Policia","Heladeria",5),
        ("Policia","Dentista",1),

        ("Parqueo Restaurante","Supermercado",3),

        ("Dentista","Familia Rojas",1),
        ("Heladeria","Universidad",1),

        ("Universidad","Supermercado",2),
        ("Familia Rojas","Escuela",2),

        ("Supermercado","Parque Sur",2),
        ("Supermercado","Familia Diaz",2),
        ("Supermercado","Zapateria",2),
        ("Supermercado","Gasolinera",4),

        ("Gasolinera","Familia Diaz",1),
        ("Gasolinera","Familia Ruiz",4),

        ("Parque Central","Familia Diaz",1),
        ("Parque Central","Central de Autobuses",5),
        ("Parque Central","Familia Ruiz",2),
        ("Iglesia","Parque Central",3),
        ("Iglesia","Dulceria",1),
```

```
        ("Zapateria","Hospital",2),
        ("Zapateria","Iglesia",3),
        ("Zapateria","Parque Central",4),

        ("Panaderia","Iglesia",4),
        ("Panaderia","Zapateria",3),

        ("Dulceria","Hospital",2),
        ("Dulceria","Escuela",1),
        ("Dulceria","Familia Diaz",3),

        ("Escuela","Central de Autobuses",1),
        ("Familia Ruiz","Iglesia",3),
        ("Familia Ruiz","Familia Rojas",4),
        ("Familia Ruiz","Dulceria",5)
};// final list
```

Matriz de Adyacencia

Se usa *n* para el total de nodos, luego se convierte los nombres de los indices usando map, esto coloca el peso en las posiciones correspondientes de la matriz.

```
    int n = map.Count;
    matriz = new int[n, n];

    foreach (var item in list)
    {
        string origen = item.Item1;
        string destino = item.Item2;
        int peso = item.Item3;

        if (map.ContainsKey(origen) && map.ContainsKey(destino))
        {
            int i = map[origen];
            int j = map[destino];

            matriz[j, i] = peso;
            matriz[i, j] = peso;
        }
    }//final foreach
}//final inicializar mapa
```

.

La idea de implementar un mapa y una lista es la facilidad que hay en agregar nodos o aristas nuevas. Solo habría que modificar *map* y *list*. Así se evita calcular manualmente que índice corresponde a cada nodo.

Algoritmo de Dijkstra

Este algoritmo nos muestra el camino mas corto en toda la ruta. Inicializamos 3 variables *int: origen, destino* y *n*. Estas variables nos indican el inicio y el fin de la ruta. Se convierten en índices numéricos usando el diccionario map.

Se utilizan 3 arreglos que guardan la mejor distancia hacia el lugar, otro para diferenciar cuales ya se visitaron y otro para recordar desde que lugar venimos cuando elegimos una ruta.

Al comenzar se asume que todos los lugares están lejos y que no se conoce como llegar todavía, por eso se utiliza infinito.

En el segundo ciclo for, en cada vuelta el programa elige el lugar que no ha sido revisado este parece más cercano según las distancias calculadas, si no queda alguna alcanzable termina.

El tercer ciclo for revisa todos los lugares conectados con él, los compara a ver cual es el mas corto. Si descubre que ir primero a ese punto y luego al vecino da un camino mas corto que el que teníamos guardado, actualiza la distancia del vecino y anota que para llegar a ese vecino pasamos por el lugar actual.

```
1 referencia
public void Dikjstra(string inicio, string fin, Control txtMensaje)
{
    int origen = map[inicio];
    int destino = map[fin];
    int n = map.Count;

    int[] dist = new int[n];
    bool[] visit = new bool[n];
    int[] prev = new int[n];

    for (int i = 0; i < n; i++)
    {
        dist[i] = int.MaxValue;
        visit[i] = false;
        prev[i] = -1;
    }
    dist[origen] = 0;

    for (int count = 0; count < n - 1; count++)
    {
        int u = MinDistancia(dist, visit);
        if (u == -1) break;
        visit[u] = true;

        for (int v = 0; v < n; v++)
        {
            int w = matriz[u, v];
            if (!visit[v] && w > 0 && dist[u] != int.MaxValue && dist[u] + w < dist[v])
            {
                dist[v] = dist[u] + w;
                prev[v] = u;
            }
        }
    }
```

En esta parte se crea una lista vacía para guardar el camino, pero todavía en forma de números, no con los nombres del lugar. El recorrido comienza en el destino y retrocede paso a paso siguiendo la información guardada en el arreglo *prev*. La variable *nodoActual* arranca con el índice destino y en cada repetición ese valor se añade a la lista del camino. Luego *nodoActual* se reemplaza por el índice del lugar desde el que se llegó, continua hasta que tome el valor de -1.

Toma el camino y lo acomoda en el orden correcto. Crea un nuevo diccionario para traducir números a nombres de lugares.

```
var caminoNumeros = new List<int>();
for (int nodoActual = destino; nodoActual != -1; nodoActual = prev[nodoActual])
{
    caminoNumeros.Add(nodoActual);
}
caminoNumeros.Reverse();

var indiceANombre = map.ToDictionary(kv => kv.Value, kv => kv.Key);
var rutaFinal = string.Join(" -> ", caminoNumeros.Select(i => indiceANombre[i]));

txtMensaje.Text = $"Ruta: {rutaFinal}\r\nPeso total: {dist[destino]}";
```

Este método nos ayuda a encontrar el siguiente lugar a procesar. Revisa todas las posiciones del arreglo *dist*, que guarda las distancias calculadas. Se considera solo los lugares que no han sido visitados y elige el que tenga la distancia mínima. Si encuentra uno más cercano actualiza el mínimo. Al terminar el recorrido devuelve el índice mas próximo.

```
1 referencia
private int MinDistancia(int[] dist, bool[] visit)
{
    int min = int.MaxValue, idx = -1;
    for (int i = 0; i < dist.Length; i++)
        if (!visit[i] && dist[i] <= min) { min = dist[i]; idx = i; }
    return idx;
}
```

Algoritmo de Floyd

Este algoritmo nos muestra las rutas mas cortas entre todos los pares. En la primera parte se cuenta cuantos lugares hay en el grafo y se crean dos matrices para guardar las distancias mínimas.

Luego inicializa ambas matrices. Recorre cada par de nodos y si existe arco directo guarda su peso y asigna el siguiente paso. Si no hay conexión la distancia es infinita.

Luego crea un diccionario para traducir los índices numéricos de los nodos a sus nombres reales. Se crea un *DataTable* que se utilizará para mostrar el resultado. Luego recorre cada par nodos evitando duplicados. Si la distancia es infinita agrega una fila indicando que no existe ruta, si existe el camino reconstruye la secuencia de nodos usando la matriz. El método termina devolviendo la tabla *DataTable*.

```
1 referencia
public DataTable FloydTodasTabla()
{
    int n = map.Count;
    int[,] dist = new int[n, n];
    int[,] next = new int[n, n];

    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
        {
            dist[i, j] = matriz[i, j] > 0 ? matriz[i, j] : (i == j ? 0 : int.MaxValue);
            next[i, j] = matriz[i, j] > 0 ? j : -1;
        }

    for (int k = 0; k < n; k++)
        for (int i = 0; i < n; i++)
        {
            if (dist[i, k] == int.MaxValue) continue;
            for (int j = 0; j < n; j++)
            {
                if (dist[k, j] == int.MaxValue) continue;
                int nueva = dist[i, k] + dist[k, j];
                if (nueva < dist[i, j])
                {
                    dist[i, j] = nueva;
                    next[i, j] = next[i, k];
                }
            }
        }

    var idxToName = map.ToDictionary(kv => kv.Value, kv => kv.Key);
```

```
    DataTable tabla = new DataTable();
    tabla.Columns.Add("Origen");
    tabla.Columns.Add("Destino");
    tabla.Columns.Add("Ruta");
    tabla.Columns.Add("Peso Total");

    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
        {
            if (dist[i, j] == int.MaxValue)
            {
                tabla.Rows.Add(idxToName[i], idxToName[j], "Sin ruta", "∞");
                continue;
            }

            var camino = new List<int> { i };
            int u = i;
            while (u != j && u != -1)
            {
                u = next[u, j];
                if (u != -1) camino.Add(u);
            }

            string ruta = string.Join(" -> ", camino.Select(x => idxToName[x]));
            tabla.Rows.Add(idxToName[i], idxToName[j], ruta, dist[i, j]);
        }

    return tabla;
}
```

Algoritmo de Warshall

Se toma cuantos nodos hay y se crea una matriz booleana A, se guardará en *A[i, j]* si existe algún cambio. Si hay arista directa se vuelve true, si no hay arista directa quedan en false.

Luego lo que hace es preguntar si puedo llegar de i a j pasando por k, si ya era alcanzable se mantiene, si no se vuelve true si existe un camino de i a k y de k a j. Al final convierte los nombres inicio y fin a índices y consulta la matriz booleana.

```
1 referencia
public void WarshallExiste(string inicio, string fin, System.Windows.Forms.Control txtMensaje)
{
    int n = map.Count;
    bool[,] A = new bool[n, n];

    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            A[i, j] = (i == j) || matriz[i, j] > 0;

    for (int k = 0; k < n; k++)
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                A[i, j] = A[i, j] || (A[i, k] && A[k, j]);

    int s = map[inicio], t = map[fin];
    txtMensaje.Text = A[s, t] ? "Sí existe camino." : "No existe camino.";
}
```

Form1.cs

El botón *btnCalcular* nos ayudara a mostrar los resultados, primero toma lo que el usuario escribió en las cajas de texto de *origen* y *destino*, les quita los espacios al principio y al final, esto como un método de validación. A su vez valida que sean letras y no números o símbolos, y muestra un mensaje si la validación encuentra un error.

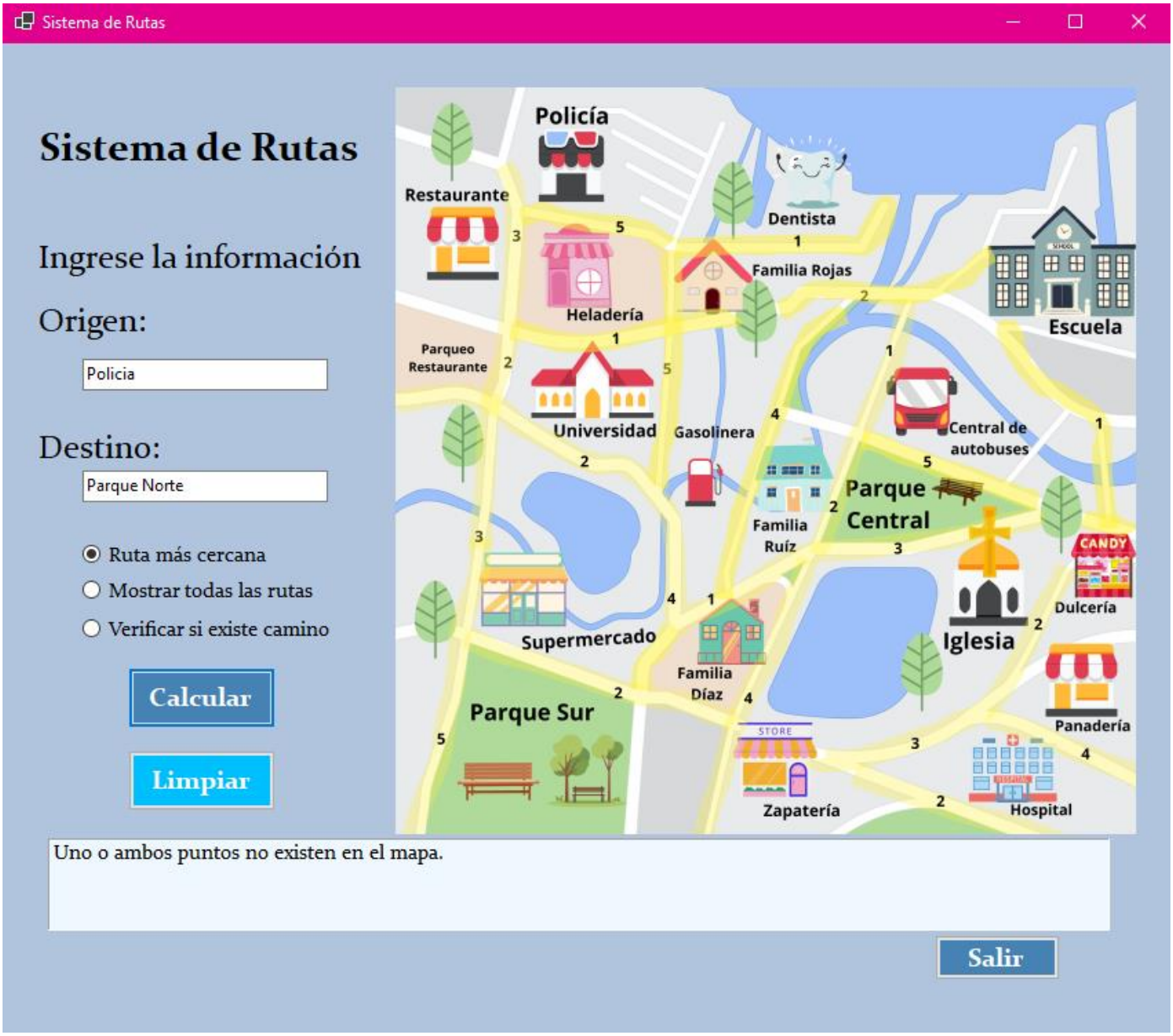
Luego confirma que los nombres escritos sí existen en el grafo, si alguno no aparece en el diccionario avisa y se detiene.

El *rdbtn* nos da la opción de elegir cual algoritmo queremos usar, si no lo marcamos nos muestra es un mensaje, diciendo que debemos seleccionar algún algoritmo para continuar.

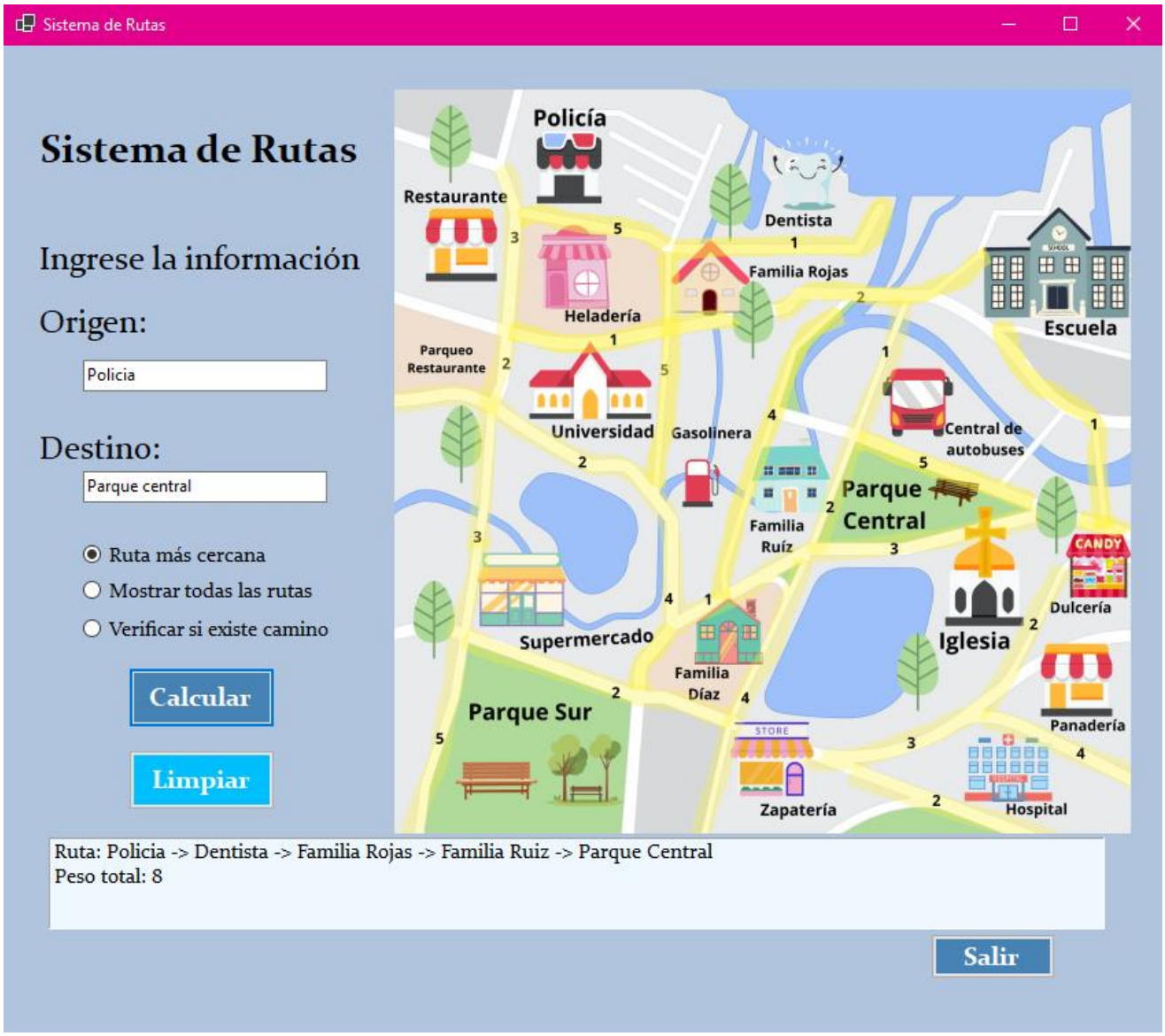
```
1 referencia
private void btnCalcular_Click(object sender, EventArgs e)
{
    if (rdbtnFlyod.Checked)
    {
        var datos = grafo.FloydTodasTabla();
        var ventanaFloyd = new FormFloyd();
        ventanaFloyd.CargarDatos(datos);
        ventanaFloyd.ShowDialog();
        return;
    }

    string inicio = txtOrigen.Text.Trim();
    string fin = txtDestino.Text.Trim();

    if (string.IsNullOrWhiteSpace(inicio) || string.IsNullOrWhiteSpace(fin) ||
        !inicio.All(c => char.IsLetter(c) || char.IsWhiteSpace(c)) ||
        !fin.All(c => char.IsLetter(c) || char.IsWhiteSpace(c)))
    {
        txtMensaje.Text = "Error: Origen y destino deben contener solo letras y no números ni símbolos";
        return;
    }

    if (!grafo.ExisteNodo(inicio) || !grafo.ExisteNodo(fin))
    {
        txtMensaje.Text = "Uno o ambos puntos no existen en el mapa.";
        return;
    }

    if (rdbtnDijkstra.Checked)
    {
        grafo.Dikjstra(inicio, fin, txtMensaje);
    }
    else if (rdbtnWarshall.Checked)
    {
        grafo.WarshallExiste(inicio, fin, txtMensaje);
    }
    else
    {
        txtMensaje.Text = "Seleccione un algoritmo antes de calcular.";
    }
}
```

Ejemplos de prueba.

1. Si queremos ir del punto “Policía” hacia “Parque Norte”, el programa debe validar si alguna de los puntos está disponible en la Lista y Diccionario. Por ejemplo “Parque Norte” no existe por lo que nos muestra un mensaje.



2. El siguiente ejemplo los dos puntos ingresados si existen en el mapa



3. El siguiente ejemplo nos validaría si la ruta existe o no.