FECHA: 18/10/2025

ESTRUCTURA DE DATOS AVANZADOS

UNIVERSIDAD AUTONÓMA DE NAYARIT

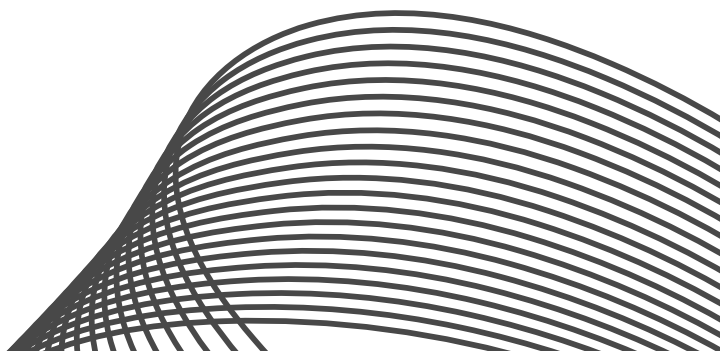
“Semana 3 — Grafos: Mapa de Ciudad (Red de Transporte)

DATOS

ALUMNO: CHRISTIAN GAEL ORTIZ RAMIREZ

DOCENTE:ELIGARDO CRUZ SANCHEZ

SISTEMAS COMPUTACIONALES



Introducción

En esta práctica se desarrolló un programa en C# para representar y analizar una red de transporte urbano mediante el uso de grafos dirigidos con pesos. Un grafo es una estructura de datos compuesta por vértices (nodos) y aristas (conexiones) que permite modelar relaciones o caminos entre diferentes elementos.

El objetivo principal fue simular un mapa de ciudad, donde cada zona o punto de interés se representa como un vértice y cada calle o ruta como una arista con un valor de peso correspondiente a la distancia o costo del trayecto. De esta manera, es posible identificar conexiones directas entre zonas, calcular grados de entrada y salida, y analizar la existencia de múltiples caminos entre dos puntos, característica propia de los multigrafos.

Este tipo de representación tiene aplicaciones reales en áreas como la planificación de rutas, logística, navegación GPSy optimización del tráfico, demostrando la importancia de los grafos como herramienta fundamental dentro de la programación y la ciencia de datos.

Desarrollo

1.- Diseñar Mapa

El Modelo esta basado en una ciudad con 8 vértices

Cada vértice funciona como un punto de referencia dentro de la ciudad, y se conecta con otros a través de calles (aristas). Las **calles** son las conexiones entre intersecciones. Se dividieron en **bidireccionales (↔)** y **unidireccionales (→)**:

No Dirigidas (Bidireccionales)

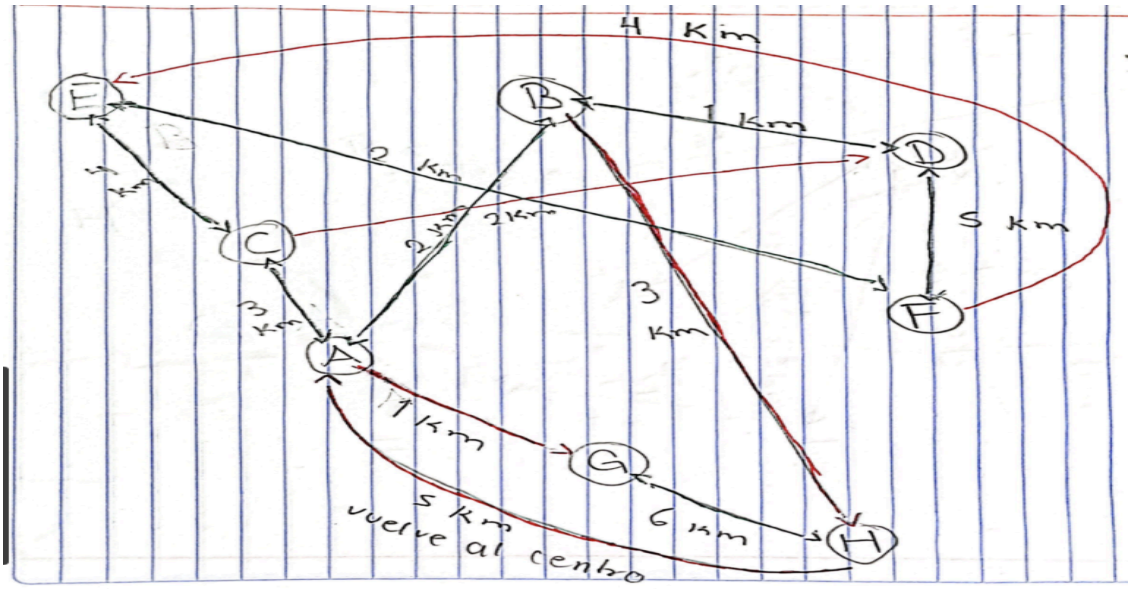
- **(↔):** Calles de doble sentido.
  - `A ↔ B: 2.0 km`
  - `A ↔ C: 3.0 km`
  - `B ↔ D: 1.0 km`
  - `C ↔ E: 4.0 km`
  - `D ↔ F: 5.0 km`
  - `E ↔ F: 2.0 km`
  - `G ↔ H: 6.0 km`

Dirigidas (Un solo sentido - Unidireccionales)

- **(→):** Calles de un solo sentido.
  - `A → G: 1.0 km (al hospital)`
  - `B → H: 3.0 km (al estadio)`

- C → D: 2.0 km
- F → E: 4.0 km
- H → A: 5.0 km (vuelta al centro)

En total, el grafo tiene **12 aristas**.



Cada vértice representa una intersección de calles o punto clave de la ciudad: A, B, C, D, E, F, G, H.

Las aristas representan calles con dirección y peso (distancia en km).

¿Cuántas conexiones totales tiene el vértice A?

Tiene 3 Conexiones
• No dirigidas: A↔B, A↔C (2 conexiones bidireccionales)
• Dirigidas: A→G (1 conexión saliente)

- **Implementación en Python**: Verán cómo los lenguajes se complementan

Aspecto	Lista de adyacencia	Matriz de adyacencia
Memoria	Baja	Alta
Recorrer	Rapido	Recorre por fila
Escalabilidad de grafos	Excelente	Ineficiente

Análisis de Trade-offs

¿Fue buena idea usar listas de adyacencia?

Eficiencia en memoria

- Solo almacenas los vértices y sus vecinos.
- Tu grafo tiene 8 vértices y 12–19 aristas → la lista ocupa mucho menos espacio que una matriz de adyacencia (8×8 = 64 posiciones).

Fácil de modificar

- Agregar o eliminar aristas es directo: solo añades o quitas de la lista del vértice.

¿Cuándo elegirías matriz de adyacencia?

Grafo muy denso y Operaciones matriciales.

Estructura del Código:

Graph

//Definicon del grafo

```
public class Graph<T> where T : notnull
{
   private readonly Dictionary<T, List<(T to, double weight)>> adj = new();
   private readonly bool isDirected;

   public Graph(bool directed = true)
   {
      isDirected = directed;
   }
}

//agregar aristas y vertices
public void AddVertex(T v)
{
   if (!adj.ContainsKey(v))
      adj[v] = new List<(T, double)>();
}

public void AddEdge(T u, T v, double weight = 1.0)
{
   AddVertex(u);
   AddVertex(v);
```

```
    adj[u].Add((v, weight));
    if (!isDirected) adj[v].Add((u, weight));
}

//Imprimir el Graph
public void Print()
{
        Console.WriteLine($"Grafo {(isDirected ? "dirigido" : "no dirigido")} con
{VertexCount()} vértices y {EdgeCount()} aristas:");
    foreach (var u in adj.Keys.OrderBy(k => k.ToString()))
    {
        Console.Write($"{u} → ");
        foreach (var (v, w) in adj[u])
        {
            Console.Write($"({v}, {w:F1}) ");
        }
        Console.WriteLine();
    }
}
```

Archivo Program

```
//Crear grafo Dirigido
var ciudad = new Graph<string>(directed: true);

//Lectura de aristas desde el archivo
string archivo = "edges_directed.txt";

if (File.Exists(archivo))
{
    string[] lineas = File.ReadAllLines(archivo);

    foreach (var linea in lineas)
    {
        if (string.IsNullOrWhiteSpace(linea)) continue;
        var partes = linea.Split(' ', StringSplitOptions.RemoveEmptyEntries);
        if (partes.Length < 3) continue;

        string origen = partes[0];
        string destino = partes[1];
        try
        {
            double peso = double.Parse(partes[2]);
            ciudad.AddEdge(origen, destino, peso);
        }
        catch (FormatException)
```

```
        {
            Console.WriteLine($"⚠️  Error de formato en la línea: '{linea}'. Se omitirá.");
        }
    }

    Console.WriteLine($"Archivo '{archivo}' cargado exitosamente ✅\n");
}
else
{
    Console.WriteLine($"⚠️ No se encontró el archivo '{archivo}', usando valores por defecto.\n");

    // Si no existe el archivo, usa datos por defecto
    ciudad.AddEdge("A", "G", 1.0);
    ciudad.AddEdge("B", "H", 3.0);
    ciudad.AddEdge("C", "D", 2.0);
    ciudad.AddEdge("A", "B", 2);
    ciudad.AddEdge("B", "A", 2);
    ciudad.AddEdge("A", "C", 3);
    ciudad.AddEdge("C", "A", 3);
    ciudad.AddEdge("A", "G", 1);
    ciudad.AddEdge("B", "D", 1);
    ciudad.AddEdge("C", "E", 4);
    ciudad.AddEdge("E", "C", 4);
    ciudad.AddEdge("D", "B", 1);
    ciudad.AddEdge("D", "F", 5);
    ciudad.AddEdge("F", "D", 5);
    ciudad.AddEdge("E", "F", 2);
    ciudad.AddEdge("F", "E", 2);
    ciudad.AddEdge("G", "H", 6);
    ciudad.AddEdge("H", "G", 6);
    ciudad.AddEdge("F", "E", 4);
    ciudad.AddEdge("H", "A", 5);
}
```

Resultados - EJECUCIÓN DEL PROGRAMA

```
christianortiz@192 Semana3 % dotnet run
=== MAPA DE CIUDAD - Red de Transporte ===

Archivo 'edges_directed.txt' cargado exitosamente ✅

Grafo dirigido con 8 vértices y 19 aristas:
A → (G, 1.0) (B, 2.0) (C, 3.0)
B → (H, 3.0) (A, 2.0) (D, 1.0)
C → (D, 2.0) (A, 3.0) (E, 4.0)
D → (B, 1.0) (F, 5.0)
E → (C, 4.0) (F, 2.0)
F → (E, 4.0) (D, 5.0) (E, 2.0)
G → (H, 6.0)
H → (A, 5.0) (G, 6.0)

=== ANÁLISIS: Conexión E–F (Multigrafo) ===

E→F existe? True
F→E existe? True

Rutas disponibles F→E: 2
  • Ruta con distancia: 4 km
  • Ruta con distancia: 2 km

📚 Concepto de MULTIGRAFO:
   - Permite múltiples aristas entre los mismos vértices.
   - Ejemplo práctico: rutas alternas con diferente distancia o costo.

=== CONSULTAS GENERALES ===

¿Puedo ir de A al Hospital (G)? True
¿Puedo volver de G a A? False

¿Existe F→E con 2.0 km? True
¿Existe F→E con 4.0 km? True

=== ANÁLISIS DE GRADOS ===

Zona A: salida=3, entrada=3
Zona B: salida=3, entrada=2
Zona C: salida=3, entrada=2
Zona D: salida=2, entrada=3
Zona E: salida=2, entrada=3
Zona F: salida=3, entrada=2
Zona G: salida=1, entrada=2
Zona H: salida=2, entrada=2

=== DESTINOS DESDE CENTRO COMERCIAL (A) ===

  → Zona G a 1 km
  → Zona B a 2 km
  → Zona C a 3 km
```

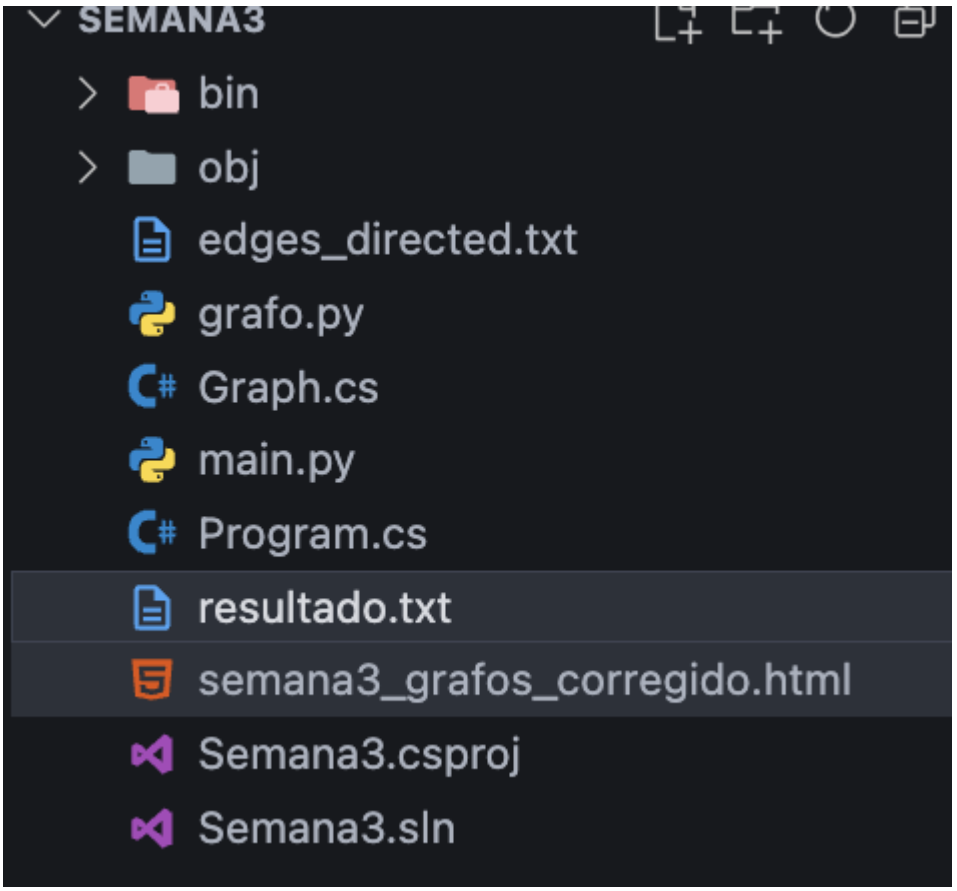
```
================================================================================
PRUEBA: Cierre temporal de calle por mantenimiento
================================================================================

Cerrando calle C→D...
Resultado: ✓ Eliminada

C→D: CERRADA
D→C: Cerrada

Archivo 'resultado.txt' generado exitosamente ✅

✓ Análisis completado.
```



Conclusión

En esta actividad se logró implementar y ejecutar correctamente un grafo dirigido con pesos utilizando listas de adyacencia. Se comprobó que el código permite manejar multi aristas, es decir, múltiples aristas entre los mismos vértices con diferentes pesos, lo cual refleja situaciones reales como rutas alternativas con distintas distancias.

A través de las consultas realizadas, se verificó la existencia de aristas, se calcularon grados de entrada y salida de cada vértice, y se demostraron las capacidades de análisis del grafo, como determinar rutas disponibles, verificar conexiones específicas y gestionar cierres de caminos.

El ejercicio permitió comprender cómo los grafos pueden representar de manera eficiente redes de transporte o cualquier sistema de conexiones, mostrando la importancia de estructuras de datos adecuadas para resolver problemas de análisis y optimización en escenarios reales.