# REPORTE

## TÉCNICO

*Milton Cruz Pánuco Castillo*

# 1. DISEÑO DEL MODELO

## 1.1 Descripción del Mapa

El proyecto modela una red urbana simplificada compuesta por 5 ciudades que representan diferentes zonas conectadas. Cada vértice del grafo corresponde a una ciudad estratégica:

- Ciudad A: Núcleo central de la red, punto de mayor conectividad
- Ciudad B: Zona norte conectada al centro
- Ciudad C: Zona oeste con conexiones múltiples
- Ciudad D: Nodo central que conecta múltiples ciudades
- Ciudad E: Ciudad periférica al este

## 1.2 Justificación de Aristas

El diseño incluye 7 conexiones bidireccionales que representan carreteras de doble sentido entre las ciudades:

Aristas Bidireccionales (7 conexiones):

- Ciudad A ↔ Ciudad B (50.5 km): Carretera principal centro-norte
- Ciudad A ↔ Ciudad C (80.0 km): Ruta más larga hacia el oeste
- Ciudad A ↔ Ciudad D (95.0 km): Conexión extensa centro-este
- Ciudad B ↔ Ciudad D (30.0 km): Ruta corta norte-este
- Ciudad C ↔ Ciudad D (45.5 km): Conexión oeste-este
- Ciudad C ↔ Ciudad E (70.0 km): Acceso a zona periférica
- Ciudad D ↔ Ciudad E (25.0 km): Ruta más corta de la red

Esta configuración 100% bidireccional refleja una red de carreteras interurbanas donde todas las rutas permiten tráfico en ambas direcciones, simulando condiciones de autopistas y carreteras estatales.

## 1.3 Criterio de Pesos

Los pesos representan distancias en kilómetros basadas en:

1. Distancias cortas (25-30 km): Conexiones entre ciudades cercanas (B-D: 30 km, D-E: 25 km)
2. Distancias medias (45-50 km): Rutas regionales (C-D: 45.5 km, A-B: 50.5 km)

3. Distancias largas (70-95 km): Conexiones extensas (C-E: 70 km, A-C: 80 km, A-D: 95 km)

Esta distribución simula una red regional donde la Ciudad D actúa como hub central con conexiones más cortas, mientras que las rutas desde la Ciudad A (centro principal) tienden a ser más extensas.

### 1.4 Diagrama del Grafo

Conexiones:

- Ciudad A: conecta con B, C, D (grado 3)
- Ciudad B: conecta con A, D (grado 2)
- Ciudad C: conecta con A, D, E (grado 3)
- Ciudad D: conecta con A, B, C, E (grado 4) - HUB CENTRAL
- Ciudad E: conecta con C, D (grado 2)

## 2. DECISIONES DE IMPLEMENTACIÓN

### 2.1 Lista de Adyacencia vs Matriz de Adyacencia

Elección: Lista de adyacencia implementada con Dictionary<T, List<Arista<T>>> en C# y diccionarios en Python.

Justificación cuantitativa:

Para nuestro grafo específico (n=5 vértices, m=7 aristas no dirigidas = 14 aristas dirigidas):

- Lista de adyacencia: Memoria ≈ O(V + E) = 5 + 14 = 19 elementos almacenados
- Matriz de adyacencia: Memoria = $O(V^2)$ = 5×5 = 25 elementos (matriz completa)
- Ahorro de memoria: (25-19)/25 = 24%

Densidad del grafo:

Densidad = E / (V × (V-1)) = 7 / (5 × 4) = 0.35 (35%)

Con una densidad menor al 50%, el grafo se clasifica como disperso, lo que favorece el uso de listas de adyacencia.

Análisis de operaciones:

| Operación | Lista de Adyacencia | Matriz de Adyacencia |
|---|---|---|
| **Verificar arista u→v** | O(grado(u)) ≈ O(2-3) | O(1) |

| | | |
|---|---|---|
| **Obtener vecinos de u** | O(grado(u)) | O(V) = O(5) |
| **Agregar arista** | O(1) | O(1) |
| **Espacio** | O(V+E) = O(19) | $O(V^2) = O(25)$ |

### 2.2 Manejo de Direccionalidad

Estrategia implementada:

En la clase Grafo<T>, el constructor acepta un parámetro booleano dirigido y el método AgregarArista implementa la lógica bidireccional:

Ventajas de este enfoque:

- Única estructura de datos para ambos tipos de grafos
- Flexibilidad para cambiar entre dirigido/no dirigido
- Código mantenible y reutilizable

```
public void AgregarArista(T origen, T destino, double peso = 1.0)
{
    AgregarVertice(origen);
    AgregarVertice(destino);

    listaAdyacencia[origen].Add(new Arista<T>(destino, peso));

    if (!esDirigido)
    {
        listaAdyacencia[destino].Add(new Arista<T>(origen, peso));
    }

    string tipo = esDirigido ? "->" : "<->";
    Console.WriteLine("Arista " + origen + " " + tipo + " " + destino + " (peso: " + peso + ") agregada");
}
```

Manejo en exportación: Para evitar duplicación de aristas bidireccionales en archivos, se implementa comparación de strings y uso de HashSet para registrar pares procesados:

### 2.3 Trade-offs Identificados

**Perdidas:**

1. Consulta de existencia de arista más lenta: O(grado) vs O(1) en matriz

2. Mayor complejidad de código (manejo de listas dinámicas)

3. Sin acceso directo bidimensional como matriz[i][j]

**Ganancias:**

1. Ahorro de memoria (24% en nuestro caso)

2. Iteración eficiente de vecinos (solo conexiones reales)

3. Escalabilidad superior para grafos dispersos

4. Menor overhead con grados variables por vértice

**2.4 Escalabilidad**

Análisis prospectivo:

Si la red crece a 50 ciudades manteniendo la misma densidad (~35%):

**- Lista:** O(50 + 350) = 400 elementos ≈ 3.2 KB

**- Matriz:** O($50^2$) = 2,500 elementos ≈ 20 KB

**- Factor de escalamiento:** 6.25x diferencia

Con 100 ciudades:

**- Lista:** O(100 + 1,400) = 1,500 elementos ≈ 12 KB

**- Matriz:** O($100^2$) = 10,000 elementos ≈ 80 KB

**- Factor de escalamiento:** 6.67x diferencia

La ventaja de listas de adyacencia aumenta considerablemente con el tamaño del grafo porque la densidad se mantiene constante mientras el espacio de la matriz crece cuadráticamente.

## 4. RESULTADOS

```
source,target,weight
Ciudad A,Ciudad B,50.5
Ciudad A,Ciudad C,80.0
Ciudad A,Ciudad D,95.0
Ciudad B,Ciudad D,30.0
Ciudad C,Ciudad D,45.5
Ciudad C,Ciudad E,70.0
Ciudad D,Ciudad E,25.0
Ciudad B,Ciudad C,60.0
Ciudad A,Ciudad E,120.0
```

```
Ana Carlos
Ana Diana
Ana Elena
Bruno Ana
Bruno Carlos
Bruno Diana
Bruno Elena
Carlos Ana
Carlos Bruno
Carlos Diana
Carlos Elena
Diana Ana
Diana Bruno
Diana Elena
Elena Bruno
Elena Carlos
Elena Diana
```

## 5. CONCLUSIONES

El proyecto logró exitosamente modelar una red interurbana de 5 ciudades usando grafos no dirigidos, implementando una solución eficiente en C# y Python. La elección de listas de adyacencia demostró ser apropiada para este grafo disperso (densidad 35%), ofreciendo un ahorro de memoria del 24% comparado con matrices.

**Resultados clave:**

- Grafo con 5 vértices y 7 aristas correctamente implementado
- Ciudad D identificada como hub central con grado 4
- Suma de grados = 14, verificando consistencia (2m = 14)
- Grafo totalmente conexo validado mediante BFS

**La experiencia reforzó la importancia de:**

1. Analizar trade-offs cuantitativamente antes de elegir estructuras de datos
2. Validar resultados manualmente antes de confiar en el código
3. Documentar decisiones técnicas con justificaciones medibles
4. Diseñar sistemas escalables que funcionen con 5 o 500 vértices

Este proyecto sienta las bases sólidas para algoritmos más avanzados de optimización que se desarrollarán en las siguientes semanas del curso, como árboles generadores mínimos y detección de ciclos.