Reporte Semanal

Ramirez Moreno
Mauricio Damian

DISEÑO DEL MODELO

Descripción del Mapa

El proyecto modela una red urbana simplificada formada por 5 ciudades, representando distintas zonas conectadas. Cada vértice del grafo corresponde a una ciudad estratégica:

Ciudad A: Núcleo central de la red, con mayor conectividad.

Ciudad B: Zona norte conectada al centro.

Ciudad C: Zona oeste con múltiples conexiones.

Ciudad D: Nodo central que enlaza varias ciudades.

Ciudad E: Ciudad periférica al este.

Justificación de Aristas

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

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

Todas las conexiones son bidireccionales, simulando autopistas y carreteras estatales con tráfico en ambas direcciones.

1.3 Criterio de Pesos

Los pesos representan distancias en kilómetros:

Cortas (25-30 km): Ciudades cercanas (B-D: 30 km, D-E: 25 km).

Medias (45-50 km): Rutas regionales (C-D: 45.5 km, A-B: 50.5 km).

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

Esto refleja una red donde Ciudad D actúa como hub central con rutas cortas, mientras que desde Ciudad A (centro) las conexiones son más largas.

1.4 Diagrama del Grafo

Conexiones y grados de cada ciudad:

Ciudad A: B, C, D (grado 3)
Ciudad B: A, D (grado 2)
Ciudad C: A, D, E (grado 3)
Ciudad D: A, B, C, E (grado 4) – HUB CENTRAL
Ciudad E: 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:
Grafo: n=5 vértices, m=7 aristas no dirigidas → 14 aristas dirigidas.
Lista de adyacencia: O(V + E) = 5 + 14 = 19 elementos.
Matriz de adyacencia: O(V²) = 5 × 5 = 25 elementos.
Ahorro de memoria: (25-19)/25 = 24%.

Densidad del grafo

$$\text{Densidad} = \frac{E}{V(V-1)} = \frac{7}{5 \cdot 4} = 0.35(35\%)$$

- Grafos dispersos (<50%) favorecen listas de adyacencia.

Comparativa 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) = 19	O(V²) = 25

2.2 Manejo de Direccionalidad

La clase Grafo<T> recibe un parámetro booleano dirigido.
AgregarArista implementa lógica bidireccional si no es dirigido.

Ventajas:

Una sola estructura para grafos dirigidos/no dirigidos.
Flexibilidad de cambio sin modificar código.
Código mantenible y reutilizable.
Exportación: Se evita duplicación de aristas usando HashSet y comparación de strings.

2.3 Trade-offs Identificados

Desventajas:
Consulta de arista más lenta (O(grado) vs O(1)).
Código más complejo por manejo de listas dinámicas.
Sin acceso directo bidimensional como matriz[i][j].

Ventajas:
Ahorro de memoria (24%).
Iteración eficiente sobre vecinos reales.
Mejor escalabilidad en grafos dispersos.
Menor overhead con grados variables.

2.4 Escalabilidad

Proyección para redes mayores con misma densidad (~35%):

Ciudades	Lista de adyacencia	Matriz de adyacencia	Factor de escalado
50	400 elementos ≈ 3.2 KB	2,500 elementos ≈ 20 KB	6.25x
100	1,500 elementos ≈ 12 KB	10,000 elementos ≈ 80 KB	6.67x

4. RESULTADOS

Grafo de 5 vértices y 7 aristas correctamente implementado.
Ciudad D identificada como hub central (grado 4).
Suma de grados = 14 (verificación: 2m = 14).
Grafo totalmente conexo validado mediante BFS.

5. CONCLUSIONES

El proyecto modela exitosamente una red interurbana usando grafos no dirigidos.
La elección de listas de adyacencia fue adecuada para un grafo disperso (densidad 35%), con ahorro de memoria del 24%.
Se validó la estructura manualmente antes de confiar en el código.
La experiencia demuestra la importancia de:
Analizar trade-offs antes de elegir estructuras de datos.
Validar resultados manualmente.
Documentar decisiones técnicas.
Diseñar sistemas escalables (5 a 500 vértices).
Este proyecto prepara la base para algoritmos avanzados como árboles generadores mínimos y detección de ciclos en futuras semanas del curso.