Semana 3: Proyecto de Grafos - Mapa de Tráfico Urbano

Descripción

Este proyecto implementa un **sistema de modelado de tráfico urbano** utilizando **teoría de grafos**. Incluye implementaciones en C# y Python para crear, analizar y comparar grafos dirigidos y no dirigidos.

Contenido del Proyecto

Archivos Principales

1. **Program.cs** - Implementación completa de la clase Graph en C#
2. **analyze_graph.py** - Script de análisis y estadísticas en Python
3. **edges_undirected.txt** - Datos del grafo no dirigido (generado por C#)
4. **edges_directed.txt** - Datos del grafo dirigido (generado por C#)
5. **Reporte_Semana3.md** - Documentación completa del proyecto
6. **README.md** - Este archivo

Características del Proyecto

Implementación en C#

- ☑ Clase genérica `Graph<T>` con soporte para cualquier tipo de dato
- ☑ Soporte para grafos dirigidos y no dirigidos
- ☑ Listas de adyacencia para eficiencia de memoria
- ☑ Exportación a archivos de texto
- ☑ Métodos de consulta (grado, vecinos, existencia de aristas)
- ☑ Manejo robusto de errores

Análisis en Python

- ☑ Carga de grafos desde archivos
- ☑ Cálculo de estadísticas (densidad, grados, conectividad)
- ☑ Identificación del vértice más conectado
- ☑ Comparación de representaciones (matriz vs lista)
- ☑ Visualización de resultados

Estructura del Mapa

Vértices (8 intersecciones)

- **A**: Centro Comercial
- **B**: Zona Norte
- **C**: Zona Sur
- **D**: Este Industrial
- **E**: Oeste Residencial

- **F**: Zona Industrial
- **G**: Hospital
- **H**: Estadio

Aristas (12 calles)

- **7 bidireccionales**: A↔B, A↔C, B↔D, C↔E, D↔F, E↔F, G↔H
- **5 unidireccionales**: A→G, B→H, C→D, F→E, H→A

Cómo Ejecutar

Requisitos Previos

- **.NET 8.0 SDK** o superior
- **Python 3.8** o superior

Paso 1: Generar Archivos de Datos

```
cd Semana3_Grafos
dotnet run
```

Salida esperada:

```
🌍 === Generando Mapa de Tráfico === 🌍
🛣️ Agregando calles bidireccionales...
☑️ Archivo 'edges_undirected.txt' exportado exitosamente.
🚦 Creando mapa completo con calles direccionales...
☑️ Archivo 'edges_directed.txt' exportado exitosamente.
```

Paso 2: Analizar con Python

```
python analyze_graph.py
```

Salida esperada:

```
🕸️ === Análisis de Grafos con Python === 🕸️
📊 Estadísticas generales:
   • Vértices: 8
   • Aristas: 19
   • Densidad: 0.339
```

Resultados Destacados

Eficiencia de Memoria

Representación	Espacios Usados	Porcentaje
Matriz de Adyacencia	64 (8×8)	100%
Lista de Adyacencia	19	29.7%
Ahorro	**45**	**70.3%**

Vértice Más Conectado

- **Vértice A (Centro Comercial)**
- Grado de salida: 3
- Grado de entrada: 2
- **Grado total: 5**

Densidad del Grafo

- **Densidad**: 0.339 (33.9%)
- **Clasificación**: Grafo disperso
- **Implicación**: Listas de adyacencia son óptimas

Conceptos Clave Implementados

1. Grafos Dirigidos vs No Dirigidos

No Dirigido (↔):

```
graph.AddEdge("A", "B", 2.0, isDirected: false);
// Crea: A → B y B → A automáticamente
```

Dirigido (→):

```
graph.AddEdge("A", "G", 1.0, isDirected: true);
// Crea solo: A → G
```

2. Listas de Adyacencia

```
Dictionary<T, List<(T to, double weight)>> adjacencyList
```

Ventajas:

- Memoria: O(n + m) vs O(n^2) de matrices
- Recorrido de vecinos: O(grado(v)) vs O(n)
- Ideal para grafos dispersos

3. Operaciones Principales

Operación	Complejidad	Descripción
AddVertex	O(1)	Agregar vértice
AddEdge	O(1)	Agregar arista
HasEdge	O(grado(u))	Verificar existencia
GetNeighbors	O(1)	Obtener vecinos
GetOutDegree	O(1)	Grado de salida
GetInDegree	O(n+m)	Grado de entrada

Análisis de Trade-offs

¿Cuándo usar Matriz de Adyacencia?

☑ Usar si:

- Grafo denso (>50% de aristas)
- Consultas frecuentes de "¿existe arista (u,v)?"
- Número pequeño de vértices

❌ No usar si:

- Grafo disperso (<50% de aristas)
- Recorridos frecuentes de vecinos
- Memoria es limitada

¿Cuándo usar Lista de Adyacencia?

☑ Usar si:

- Grafo disperso (nuestro caso: 33.9%)
- Algoritmos de búsqueda (BFS, DFS)
- Escalabilidad es importante

❌ No usar si:

- Necesitas consultas O(1) de existencia de aristas
- El grafo es muy denso

Extensiones Futuras

Algoritmos a Implementar

- ☐ **BFS** (Breadth-First Search) - Camino más corto
- ☐ **DFS** (Depth-First Search) - Detección de ciclos
- ☐ **Dijkstra** - Camino más corto ponderado
- ☐ **A*** - Búsqueda heurística

Mejoras Propuestas

- ☐ Visualización gráfica con NetworkX
- ☐ Interfaz web interactiva
- ☐ Simulación de tráfico
- ☐ Integración con APIs de mapas

Estructura de Archivos Generados

edges_undirected.txt

```
A B 2.0
A C 3.0
B D 1.0
...
```

edges_directed.txt

```
A B 2.0
A C 3.0
A G 1.0
B A 2.0
B D 1.0
B H 3.0
...
```

Pruebas de Funcionalidad

El programa incluye pruebas automáticas:

```
☑ Grado de A (no dirigido): 2 (esperado: 2)
☑ ¿Existe A↔B no dirigido? True
☑ ¿Existe A→G dirigido? True
☑ ¿Existe G→A dirigido? False
```

Aplicaciones Reales

Este proyecto es la base para:

1. **Sistemas de Navegación**
    - Google Maps, Waze
    - Cálculo de rutas óptimas
2. **Redes Sociales**

    - Conexiones entre usuarios
    - Recomendaciones de amigos

3. **Redes de Computadoras**
    - Routing de paquetes
    - Optimización de tráfico

4. **Logística**
    - Rutas de entrega
    - Optimización de costos

Recursos Adicionales

- **Reporte Completo**: Ver `Reporte_Semana3.md`
- **Código Fuente**: `Program.cs` y `analyze_graph.py`
- **Teoría**: Consultar material de la Semana 3

Curso: Estructuras de Datos Avanzadas
Semana: 3
Tema: Teoría de Grafos - Representación y Modelado
Fecha: Diciembre 2025