Informe del Trabajo Práctico N°2

Aplicaciones de estructuras jerárquicas y grafos

Materia: Algoritmos y Estructuras de Datos - FIUNER

Integrantes:

Flores, Valentina

Aguilar Gonzales, Andrea Fernanda

Lell, Camila Luciana

Fecha de entrega: 6 de junio del 2025

ACTIVIDAD 3

Palomas Mensajeras

En esta tercera actividad se nos planteó un problema inspirado en un sistema de comunicación antigua: el envío de mensajes entre aldeas utilizando palomas mensajeras. Cada aldea solo puede enviar mensajes a aquellas con las que tiene un camino directo, y el objetivo era lograr que la noticia enviada desde la aldea *Peligros* llegara a las demás de forma eficiente y sin repeticiones, minimizando el total de distancias recorridas por las palomas.

Para resolver este problema, modelamos la situación como un grafo no dirigido y ponderado, en el que cada vértice representa una aldea y cada arista un camino entre dos aldeas, con una distancia (en leguas) como peso. Aplicamos el algoritmo de Prim, que permite construir un Árbol de Expansión Mínima (MST). Este árbol asegura que todas las aldeas estén conectadas con la menor cantidad de distancia posible, cumpliendo además con la condición de que cada aldea reciba el mensaje una única vez y pueda replicarlo a otras según su posición en el árbol.

Implementación

La solución se implementó con los siguientes módulos:

- **grafo.py**: Define Vertice (aldea con conexiones y pesos) y Grafo (grafo no dirigido). cargar_grafo lee aldeas.txt, validando 22 aldeas, la presencia de "Peligros", y conexiones válidas.
- **monticulo.py**: Implementa MonticuloBinario para optimizar el algoritmo de Prim, soportando inserción, eliminación del mínimo, y actualización en O(log n).
- **algoritmo_prim.py**: Contiene prim (construye el MST desde "Peligros") y calcular_sumas_distancias (calcula distancias y suma total).
- **main.py**: Coordina la carga del grafo, muestra los entregables, y maneja errores..

Funcionalidad y métodos implementados

- **grafo.py**: Define Vertice (aldea con conexiones y pesos) y Grafo (grafo no dirigido). cargar_grafo lee aldeas.txt, validando 22 aldeas, la presencia de "Peligros", y conexiones válidas.
- **monticulo.py**: Implementa MonticuloBinario para optimizar el algoritmo de Prim, soportando inserción, eliminación del mínimo, y actualización en O(log n).
- **algoritmo_prim.py**: Contiene prim (construye el MST desde "Peligros") y calcular_sumas_distancias (calcula distancias y suma total).

- **main.py**: Coordina la carga del grafo, muestra los entregables, y maneja errores.

Resultados y comportamiento observado

Una vez cargado el grafo desde el archivo y ejecutado el algoritmo, se generó correctamente el Árbol de Expansión Mínima, donde cada aldea quedó conectada sin formar ciclos y con la mínima distancia total.

Se mostró primero el listado de las 22 aldeas ordenadas alfabéticamente, seguido de un resumen para cada una indicando:

- De qué aldea recibe el mensaje (predecesor en el árbol).
- A qué aldeas debe reenviar la noticia (hijos en el árbol).

Validaciones: Se manejaron errores como:

- Archivo no encontrado.
- Grafo con menos/más de 22 aldeas.
- "Peligros" sin conexiones o no presente.
- Distancias no enteras o negativas.

Finalmente, se calculó y mostró la distancia total recorrida por las palomas mensajeras, que corresponde a la suma de todas las distancias entre cada aldea y su predecesora. Esto representa el costo mínimo de propagación del mensaje, cumpliendo con el objetivo planteado.

Observaciones

- La implementación del algoritmo de Prim garantizó una solución óptima en cuanto al uso de recursos, evitando duplicaciones y reduciendo al mínimo la distancia total.
- El uso del montículo binario redujo la complejidad de prim a O(E log V), optimizando la selección de aristas, clave para mantener la eficiencia del algoritmo, especialmente al aumentar la cantidad de conexiones.
- El diseño del MST aseguró que no hubiera ciclos en la propagación, cumpliendo con la consigna de que cada aldea reciba la noticia una sola vez.
- Se validaron correctamente los datos del archivo de entrada, lo que permitió evitar errores comunes como aldeas sin conexiones o datos mal formateados.

Conclusiones

La solución desarrollada para este problema fue efectiva y adecuada, combinando estructuras de grafos y algoritmos eficientes para lograr una propagación óptima de mensajes entre las aldeas. La actividad permitió aplicar de forma concreta los

conocimientos adquiridos sobre árboles de expansión mínima, estructuras jerárquicas y análisis algorítmico, reforzando la comprensión de cómo estas herramientas se utilizan en contextos reales.

Además, el enfoque modular y validado del desarrollo permitió asegurar que los resultados fueran correctos, claros y fácilmente interpretables, respetando los principios de diseño de estructuras eficientes estudiados a lo largo de la materia.