Informe de Palomas mensajeras - Árbol de expansión mínima (MST)

UNIVERSIDAD NACIONAL DE ENTRE RÍOS

Facultad de Ingeniería

Carrera: T.U.P.E.D

Materia: Fundamentos de Algoritmos y Estructuras de Datos

Estudiante: Franco Tomiozzo

Profesor: Dr. Javier E. Diaz Zamboni

Bioing. Jordán F. Insfrán

Esp. Bioing. Francisco Rizzato

Comisión: 1

Fecha: Octubre 2025

Introducción

Partí del archivo de rutas aldeas.txt, con líneas como “Torralta, Villaviciosa, 8”, y ejecuté el programa usando “Peligros” como aldea de origen. El objetivo fue encontrar el plan más eficiente para que el mensaje salga desde Peligros y llegue a todas las demás aldeas, cumpliendo que cada una reciba la noticia una sola vez y minimizando la distancia total recorrida por las palomas. El programa logra esto construyendo un árbol de expansión mínima (**MST**) sobre el subgrafo de aldeas alcanzables desde Peligros, y luego “enraizando” ese árbol en Peligros para producir un plan de “recibe de / envía a”.

Desarrollo

El procesamiento se puede entender en tres capas: Limpieza/normalización de datos, Construcción y optimización del grafo, y Generación del plan operativo.

1. **Limpieza y normalización de dato:**

- Normalización de strings: antes de procesar, cada campo de texto se pasa por una función que colapsa espacios múltiples y recorta bordes. Esto evita que nombres con formatos irregulares (por ejemplo, “La Aparecida”) se traten como aldeas distintas.
- Lector robusto de aristas: el programa ignora líneas vacías, filas con menos de tres columnas, “nan”, pesos no enteros y pesos no positivos. También descarta cualquier ruido como la línea “Diosleguarde” sin columnas válidas. El resultado es una lista de aristas limpia, lista para armar el grafo.

2. **Construcción y optimización del grafo**

- El Grafo no dirigido mínimo por par: cuando hay varias rutas entre el mismo par de aldeas (en cualquier orden, “**A,B,w**” o “**B,A,w**”), el programa se queda con el menor peso observado. Esto se resuelve con una clave canónica del par (**min(A,B)**, **max(A,B)**). Esta consolidación es crítica: reduce la redundancia y garantiza que la optimización posterior (**Kruskal**) parta de los costos más bajos reales entre pares.
- Alcanzabilidad desde Peligros: antes de optimizar, se calcula el conjunto **R** de aldeas alcanzables desde Peligros mediante un **BFS (Breadth-First Search)**. Esta decisión fue

tomada por que si existiera otro componente desconectado, no tendría sentido incluirla en el costo; el **MST** sólo se calcula sobre **R**, que es lo que efectivamente puede recibir el mensaje desde Peligros.

- **Kruskal + Union-Find (DSU)**: para construir el **MST** sobre **R**, el programa:
    - Extrae todas las aristas entre nodos de **R**, las ordena por peso ascendente y las recorre de menor a mayor.
    - Usa **Union-Find** con compresión de caminos y unión por rango. La compresión hace que las búsquedas de representantes sean casi **O(1)** amortizado, y el rango evita árboles degenerados. Así se detectan ciclos de forma eficiente: una arista se agrega al **MST** sólo si conecta dos componentes distintos.
    - Valida que el **MST** tenga exactamente **|R|−1** aristas; si no, la estructura no sería un árbol (habría desconexión o ciclos).

3. Generación del plan operativo

- Enraizado del **MST** en Peligros: el **MST** es no dirigido, realice un **BFS** sobre el **MST** con raíz en Peligros y se determina, para cada aldea, su "padre" (quién le pasa la noticia) y sus "hijos" (a quién reenvía). El programa también ordena los "hijos" por nombre para que la salida sea estable y fácil de leer. El origen aparece sin "padre" y sólo como emisor hacia sus vecinos en el árbol.
- Formato de salida: el programa imprime:
    - La lista alfabética de aldeas alcanzables.
    - Para cada aldea, "recibe de **X (dist=d)** / envía a **Y**…".
    - La suma total de distancias del **MST**, que representa el costo total de todas las palomas en el plan óptimo.

Con los datos que usé, el sistema listó 22 aldeas (incluye Peligros) y el plan quedó enraizado en Peligros con envíos directos a "La Aparecida" (5) y "Lomaseca" (7). Desde ahí, se propaga sin repetir destinatarios ni formar ciclos: por ejemplo, "La Aparecida" reenvía a "Buenas Noches" (3) y "Silla" (5); "Silla" reenvía a "Torralta" (4); "Torralta" reenvía a "Humilladero" (9) y "Villaviciosa" (8); "Lomaseca" reenvía a "El Cerrillo" (5), "Los Infiernos" (2) y "Pepino" (3), y

así sucesivamente. La suma total de distancias del **MST** dio 94.

Detalles técnicos del código que marcan diferencia:

- Consolidación de aristas con “mejor peso” por par:
    - Clave canónica del par con **tuple(sorted((u, v)))** y diccionario **best** para guardar el mínimo. Esto evita que el MST se vea sesgado por duplicaciones o direcciones inversas y reduce el problema a un grafo simple con costos mínimos reales.
- DSU con dos optimizaciones:
    - La compresión de caminos en **find** acelera búsquedas sucesivas, esencial cuando hay muchas aristas.
    - Unión por rango mantiene árboles de conjuntos balanceados, lo que evita degradación de rendimiento.
    - Juntas, estas dos optimizaciones llevan las operaciones de Kruskal cerca de lineales en la práctica.
- Validación estructural del **MST**:
    - Más allá de “funcionar”, el programa chequea que el número de aristas del **MST** sea exactamente $|\mathbf{R}|-\mathbf{1}$. Esta validación simple pero contundente captura errores sutiles (datos desconectados, filtrados incorrectos, etc.).
- Enraizado estable del **MST**:
    - Convertir un MST no dirigido en un árbol dirigido desde Peligros no es trivial si querés que la salida sea legible y determinista. El **BFS** desde la raíz define padres/hijos y el ordenado por nombre fija la presentación para que no cambie entre corridas pese a empates de pesos.

Conclusión

El gráfico que se muestra representa el árbol (**MST**) en forma de “red”con sus respectivas aldeas y mensajes enviados: