# Universidad Nacional de Entre Ríos

➢*Facultad:* Facultad de Ingeniería.

➢*Carrera:* Licenciatura en Bioinformática y Bioingeniería.

➢*Cátedra:* Algoritmos y estructuras de datos

➢*Docentes:* Javier E. Diaz Zamboni, Jordán F. Insfrán, Juan F. Rizzato

➢*Título del trabajo:* "Aplicaciones de estructuras jerárquicas y grafos -"*Problema 3 ""*.

➢*Alumnos:*


- Almirón Spahn, María Paz
- Leiva, Giuliana
- Saravia, Lucía Milagros


➢*Fecha de entrega:* 31 de Octubre del 2025

**Problema 3:**

En este proyecto se busca determinar la menor distancia total recorrida por las palomas mensajeras al entregar mensajes entre las distintas aldeas, partiendo desde la aldea inicial denominada "**Peligros**". Para resolver el problema se diseñaron e implementaron las estructuras "Grafo", "Vertice" y "ColaPrioridad", junto con la aplicación del algoritmo de Prim, que permite obtener el árbol de expansión mínima del grafo, representando el recorrido más eficiente para cubrir todas las aldeas.

El objetivo principal del problema consiste en conectar todas las aldeas con la *menor suma posible de distancias*, garantizando que cada una sea accesible desde la aldea inicial sin formar ciclos redundantes.

De esta forma, el árbol resultante representa las rutas que seguirán las palomas mensajeras para entregar los mensajes de la manera más corta y eficiente posible.

***Estructuras implementadas:***

- **Clase "Vertice":** Representa cada aldea del grafo. Es posible pensar que cada vértice es un nodo del grafo, y las conexiones entre las aldeas son las aristas.

Esta clase contiene como atributos:

- **"id"**: nombre de la aldea
- **"conectadoA"**: diccionario con los vecinos y sus distancias
- **"predecesor":** almacena la aldea anterior en el recorrido del árbol.
- **"distancia"**: costo temporal del vértice. Es el valor utilizado por el algoritmo para saber cual es el vértice más cercano al árbol parcial construido.

A su vez, incluye métodos importantes como:

- **"agregarVecino(vecino,distancia):** agrega una conexión entre aldeas.
- **"obtenerVecinos():** devuelve las aldeas conectadas con la actual.
- **"obtenerPonderacion(vecino):** devuelve el costo de viajar desde la aldea actual hasta este vecino.

- **Clase "Grafo":** reúne y organiza todos los vértices (aldeas) y sus conexiones (aristas). Se podría considerar al grafo como el mapa completo de las aldeas y distancias.

Algunos de sus atributos son:

- **"listaVertices":** es un diccionario donde cada clave es el nombre de la aldea y el valor es el objeto "Vertice" correspondiente.
- **"numVertices":** la cantidad total de aldeas en el grafo.

Además, tiene algunos métodos principales, como lo son:

- **"agregarVertice(nombre):** Esta función agrega un nuevo vértice de tipo Vertice (nodo) al grafo, que representa una aldea.

- **“agregarArista(origen, destino, costo)”:** conecta dos aldeas con la distancia indicada. Asimismo, agrega la conexión en ambos sentidos, ya que el grafo es no dirigido (es decir, que la distancia de $A \rightarrow B$ es la misma que de $B \rightarrow A$).
- **“obtenerVertice(nombre)”:** Devuelve un vértice específico (una aldea) a partir de su nombre.
- **“__iter__”:** Hace que el grafo se pueda recorrer con un for.

Dentro del algoritmo, el grafo es quien guarda toda la información del problema (aldeas, conexiones y distancias). El algoritmo de Prim recorre este grafo y elige las rutas más “baratas” para construir el árbol de expansión mínima.

- **Clase “ColaPrioridad”:** esta clase implementa un montículo mínimo, una estructura que permite obtener el elemento con menor valor (en este caso, la distancia más corta) de manera muy eficiente.

En este contexto, esta clase se utiliza debido a que el algoritmo de Prim necesita, en cada paso, elegir la siguiente aldea más cercana al árbol ya construido. En lugar de revisar todas las aldeas cada vez, se usa la cola de prioridad para obtener el vértice con la menor distancia en O (log n) tiempo.

Algunos de sus atributos son:

- **“listaMonticulo”:** es una lista interna que guarda tuplas “(distancia, vertice)”.
- **“tamanoActual”:** cantidad de elementos en la cola.

Unos de sus métodos más significativos son:

- **“construirMonticulo(distancias)”:** crea el montículo a partir de una lista inicial
- **“eliminarMin()”:** Quita y devuelve el valor mínimo (la raíz) del montículo, y luego lo reorganiza para mantener su estructura ordenada.
- **“decrementarClave(vertice, nueva_distancia)”:** Actualiza (disminuye) la distancia de un vértice dentro del montículo y luego lo reorganiza infiltrando hacia arriba para mantener su orden correcto.
- **“__contains__(vertice)”:** Comprueba si existe un vértice dentro del monticulo

***Algoritmo de Prim***

El algoritmo de Prim es una técnica que permite hallar el árbol de expansión mínima (MST) en un grafo ponderado y no dirigido. En el contexto del problema, Prim nos ayuda a determinar el recorrido más eficiente que deben seguir las palomas mensajeras para entregar los mensajes a todas las aldeas, partiendo desde “Peligros”.

***Funciones auxiliares***

- **“leer_aldeas”:** Lee un archivo de datos que contiene las aldeas y sus conexiones, y a partir de estos datos construye un grafo.
- **“recorridoMensaje”:** Devuelve una lista de lista donde se almacena el nombre de la aldea, el predecesor y los sucesores
- **“distanciaTotal”:** Suma las distancias del árbol de expansión mínima

***Análisis del orden de complejidad***

Para logra entender cuál es el **orden de complejidad** del algoritmo prim, hacemos el siguiente análisis:

- Primero se insertan todos los vértices del grafo en una cola de prioridad. Como hay $|V|$ vértices, esta operación tiene un costo de O(|V|)
- Luego, en el ciclo while, el algoritmo extrae el vértice de menor ponderación de la cola de prioridad en cada iteración, entonces el bucle se ejecuta |V| veces y cada extracción, donde utilizamos la función “eliminarMin()”, toma un tiempo de O(log|V|).
  Entonces el conjunto tiene una complejidad de O(|V| logV)
- Dentro del ciclo for se recorren las aristas adyacentes al vértice actual, esto se hace |E|, ya que en total se recorren todas las aristas del grafo. Cada vez que se encuentra un vecino con menor ponderación se actualiza la distancia y se llama a la función “decrementarClave()”, la cual busca el vértice de forma lineal, haciendo que tenga un costo de O(|V|). Ademas, puede haber hasta |E| llamadas (una llamada por arista), haciendo que el costo total de esta parte sea de $O(|E| \cdot |V|)$.
- Sumando todas las partes la **complejidad total** del algoritmo es:
  **$O(|E| \cdot |V| + |V| \log |V|)$**

***Resultados y conclusiones***

El programa construye correctamente el grafo de aldeas y, al aplicar Prim, obtiene el camino mínimo total que las palomas deben recorrer para entregar los mensajes.

Asimismo, mediante la función “distanciaTotal()”, se obtiene la suma de las ponderaciones (distancias) del árbol de expansión mínima, lo cual representa el recorrido más corto que conecta todas las aldeas sin redundancias.

Finalmente, para concluir, es posible afirmar que la combinación de las estructuras “Grafo”, “Vertice” y “ColaPrioridad” junto con el algoritmo de Prim permite construir una

solución eficiente para el envío de mensajes entre aldeas, minimizando la distancia total recorrida.