# TRABAJO PRACTICO

# PARQUE NACIONAL

## UNIVERSIDAD

- Universidad Nacional General Sarmiento
- Materia: Programación 3
- Comisión:  01

## ALUMNOS


- Mayra Rossi (35.366.464) Mayra.Rossi03@gmail.com
- Melina Scabini (44.756.058) Melina.Scabini@gmail.com


## PROFESORES

- Patricia Bagnes
- Ignacio Sotelo

## Introducción

El objetivo de este trabajo práctico es implementar una aplicación visual para optimizar la red de senderos en el Parque Nacional Nahuel Huapi, minimizando el impacto ambiental de su construcción. Las estaciones (miradores, refugios, etc.) se representan como nodos de un grafo no dirigido, y los senderos como aristas con pesos (impacto ambiental, enteros en [1,10]). El objetivo es encontrar un Árbol Generador Mínimo (AGM) que conecte todas las estaciones con el menor impacto total, utilizando ambos algoritmos de Prim y Kruskal para comparar su rendimiento.

La aplicación fue diseñada en java, siguiendo un enfoque modular que separa la lógica del juego (paquete Model) de la interfaz gráfica (paquete View), con una capa de control (paquete Controller) que coordina ambas partes mediante el uso del patrón Observer para la comunicación de eventos. La integración de Observers nos permitio que las vistas reaccionen a cambios en el modelo sin acoplamiento directo, mejorando la modularidad y facilitando la extensión del sistema.

En el desarrollo, se hace la carga datos desde archivos JSON, visualiza el grafo en un mapa interactivo con JMapViewer, calcula el AGM, se muestran los senderos seleccionados como camino óptimo y se detalla el impacto total junto con su tiempo de ejecución.

Este documento describe la estructura del proyecto, los desafíos y decisiones tomadas durante el desarrollo basadas en el análisis de complejidad y buenas prácticas, con el propósito de ofrecer una visión clara del proceso llevado a cabo por el equipo.

## Buenas Prácticas

- **Pruebas Unitarias:** Se implementaron pruebas con JUnit para las clases de negocio (NationalParkGraph, NationalParkGraphKruskal, NationalParkGraphPriJsonReader, Station, Trail) Siendo útil para detectar puntos de quiebre cubriendo casos como grafos vacíos, disconexos, archivos JSON malformados, entre otros ejemplos. Esto nos ayudó a asegurar la robustez, dando lugar a la detección y manejo temprano de errores.

- **Modularidad y Organización del Código:** El proyecto está estructurado en paquetes bien definidos como model, view, controller, observer, lo cual refleja un enfoque modular y respetuoso con el principio de separación de responsabilidades. Cada paquete tiene una tarea clara y se encarga de una parte del sistema, lo que facilita la mantenibilidad y la escabilidad del proyecto a medida que crece.

- **Nombres claros:** Se emplean nombres descriptivos para clases y métodos, haciendo que el código sea autoexplicativo y fácil de mantener. Esta práctica, alineada con los principios

de *Clean Code*, reduce la necesidad de comentarios innecesarios y mejora la legibilidad general. También se respeta la **distancia vertical**, agrupando funciones relacionadas y separando conceptos distintos con claridad, lo que facilita seguir el flujo lógico del programa sin necesidad de desplazarse innecesariamente por el archivo.

- **Encapsulado de Componentes para Reutilización:**
  Los elementos visuales, como botones, etiquetas, paneles, se encapsularon en funciones, lo que permitió su reutilización y mantenimiento de forma independiente. Este enfoque también ayuda a mantener los archivos más ordenados, pequeños y comprensibles.

- **Desacoplamiento de Estilos con ColorPalette:**
  El uso del componente ColorPalette permite separar los aspectos visuales del comportamiento de la aplicación. Al centralizar los colores en un único lugar, se logra una mayor coherencia, se facilita la modificación de estilos y se refuerza un diseño desacoplado y escalable.

## Decisiones de Desarrollo

Durante el desarrollo de la aplicación, se tomaron varias decisiones clave para optimizar la implementación, garantizar la modularidad y cumplir con los requisitos del trabajo práctico. A continuación, se detallan las principales:

**Estructura de datos para grafos:** Se implementó una representación basada en listas de adyacencia mediante ArrayList en Java para el grafo, permitiendo acceso eficiente (O(1)) a las estaciones y senderos mediante un índice. Esta decisión equilibra el uso de memoria y la velocidad de acceso, aspecto crítico para la implementación eficiente de los algoritmos de Prim y Kruskal. La clase NationalParkGraph actúa como clase base abstracta que contiene la funcionalidad común entre las implementaciones de los algoritmos.

**Implementación del patrón Observer:** Se desarrolló un sistema de notificación basado en el patrón Observer que permite actualizar automáticamente la vista cuando cambia el estado del modelo. Esto desacopla la lógica de negocio de la presentación, permitiendo que los algoritmos se ejecuten independientemente de cómo se visualicen sus resultados, facilitando futuras extensiones y cambios en la interfaz sin afectar al modelo.

**Implementación de UnionFind:** Para el algoritmo de Kruskal, se desarrolló una clase UnionFind personalizada con las operaciones de union() y find() optimizadas mediante compresión de caminos, reduciendo la complejidad de las operaciones a prácticamente O(1) amortizado. Esto resulta crucial para la eficiencia del algoritmo de Kruskal al trabajar con grafos de mayor tamaño, ya que mejora significativamente la detección de ciclos.

**Cálculo del tiempo de ejecución:** Se empleó System.nanoTime() para medir con precisión los tiempos de ejecución de los algoritmos, permitiendo comparaciones exactas incluso en grafos pequeños donde las diferencias pueden ser del orden de microsegundos. Esta

implementación permite evaluar objetivamente la eficiencia de cada algoritmo en diferentes escenarios.

**Visualización con JMapViewer:** La elección de JMapViewer como biblioteca de mapas responde a su capacidad para representar coordenadas geográficas reales y su integración nativa con Swing. Esto permite visualizar el parque nacional en un contexto geográfico auténtico, mejorando la comprensión espacial de los resultados. Los marcadores y líneas se personalizaron para representar estaciones y senderos de manera intuitiva.

**Estrategia de colores para impacto ambiental:** Se implementó una escala de colores (verde-amarillo-rojo) para representar visualmente el impacto ambiental de cada sendero. Esta decisión facilita la interpretación inmediata de los datos críticos por parte del usuario, sin necesidad de leer valores numéricos. La clase ColorPalette centraliza la gestión de colores como un enum, facilitando la consistencia y posibles cambios en el esquema de colores.

**Separación de algoritmos en clases independientes:** Cada algoritmo (Prim y Kruskal) se implementó en su propia clase heredando de NationalParkGraph. Esta decisión de diseño sigue el principio de responsabilidad única, permitiendo mantener cada implementación enfocada en su propio algoritmo y facilitando la extensión con nuevos algoritmos en el futuro sin modificar el código existente.

**Diseño modular de la interfaz gráfica:** Los componentes visuales se agruparon en métodos específicos (createButtonPanel, createTitleLabel, etc.), aplicando el principio DRY (Don't Repeat Yourself) para facilitar el mantenimiento y las futuras modificaciones. Esta modularidad permite cambiar aspectos específicos de la interfaz sin afectar al resto de componentes.

**Uso de JSON para almacenamiento de datos:** Se seleccionó JSON como formato para los datos de estaciones y senderos debido a su estructura jerárquica natural que permite representar fácilmente la relación entre estaciones y senderos. La biblioteca Gson se utilizó para deserializar los datos directamente a objetos Java, simplificando el proceso de carga y permitiendo una fácil extensión con nuevas propiedades en el futuro.

**Implementación del controlador minimalista:** Se diseñó un controlador ligero que simplemente coordina las acciones entre la vista y el modelo, siguiendo el principio de que el controlador debe contener la menor lógica posible. Esta decisión mejora la testabilidad del sistema y mantiene una clara separación de responsabilidades.

## Análisis de complejidad

**NationalParkGraphKruskal:** La implementación actual tiene una complejidad de O(E log E), donde E es el número de aristas. El algoritmo ordena las aristas por peso y utiliza la estructura UnionFind para detectar ciclos eficientemente.

**NationalParkGraphPrim:** Presenta una complejidad de $O(V^2)$, donde V es el número de vértices. En cada iteración se busca la arista de menor peso que conecte un vértice ya incluido con uno no incluido.

## Comparativa de Rendimiento

Las pruebas realizadas con el conjunto de datos del Parque Nacional Nahuel Huapi muestran que:

Para grafos pequeños: Ambos algoritmos tienen un rendimiento similar, con tiempos de ejecución en el orden de microsegundos.

Para grafos medianos: Kruskal tiende a ser más eficiente debido a su implementación optimizada con UnionFind.

Consistencia de resultados: Ambos algoritmos generan el mismo árbol mínimo con un impacto ambiental total idéntico, validando la correcta implementación de los mismos.

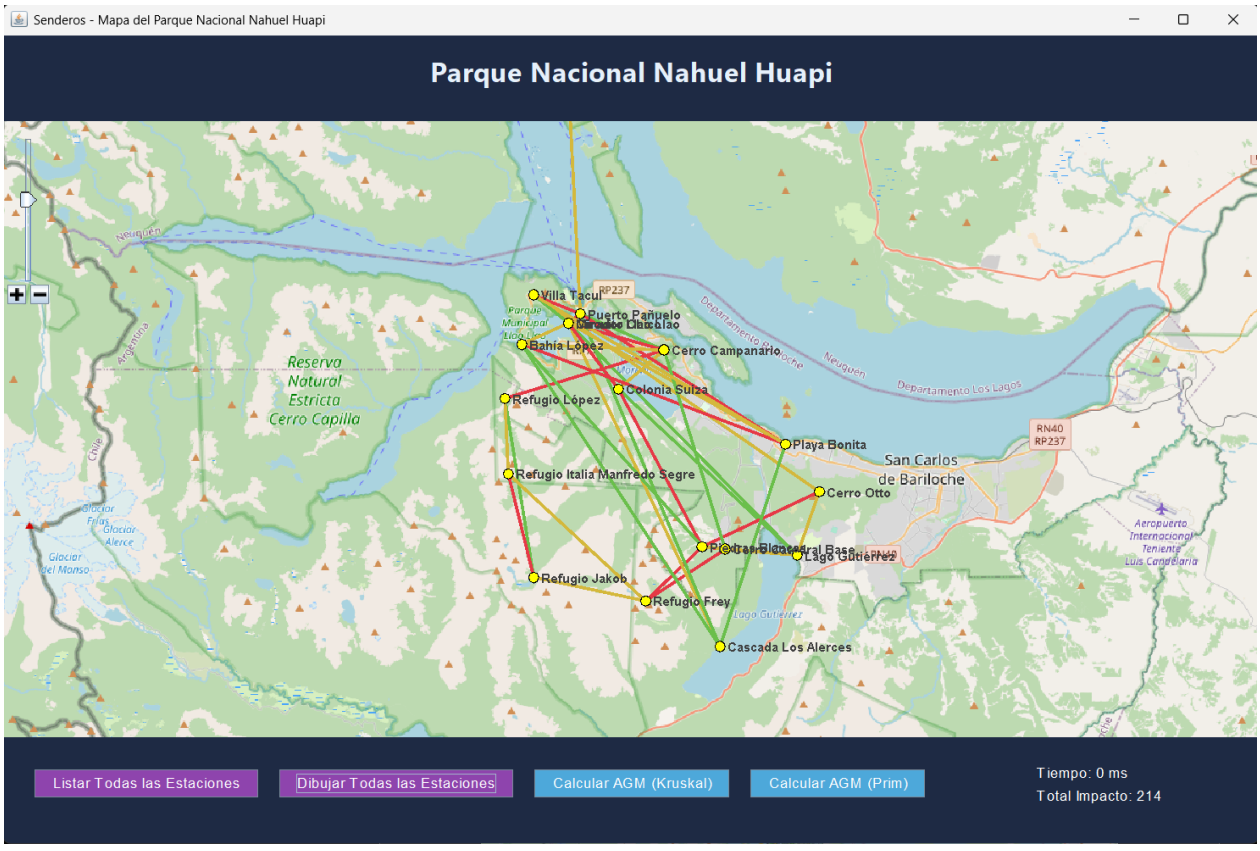
## Funcionalidades opcionales

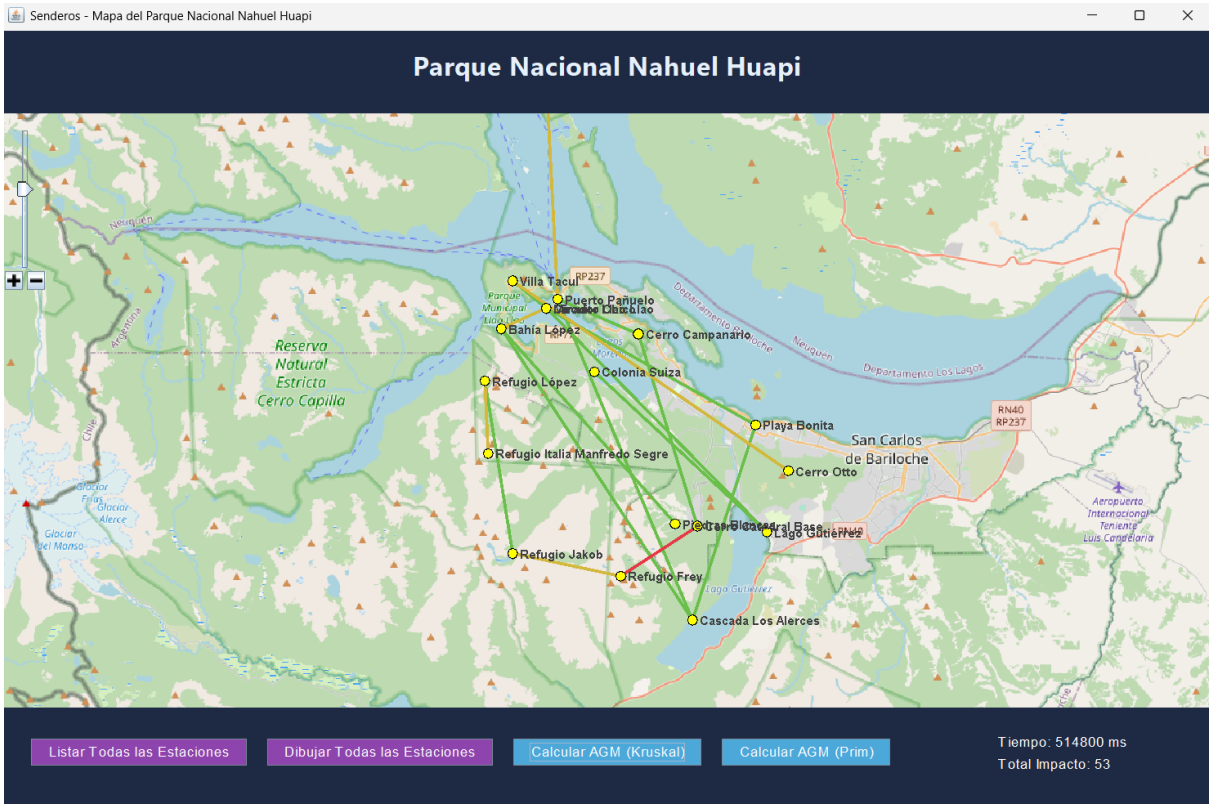
1. **Implementación y Comparación de Algoritmos de Prim y Kruskal:**
   Se implementaron los algoritmos de Prim y Kruskal para calcular el Árbol Generador Mínimo (AGM). Se midieron sus tiempos de ejecución usando System.nanoTime() mostrando los resultados en la interfaz. Esto permite comparar su rendimiento.

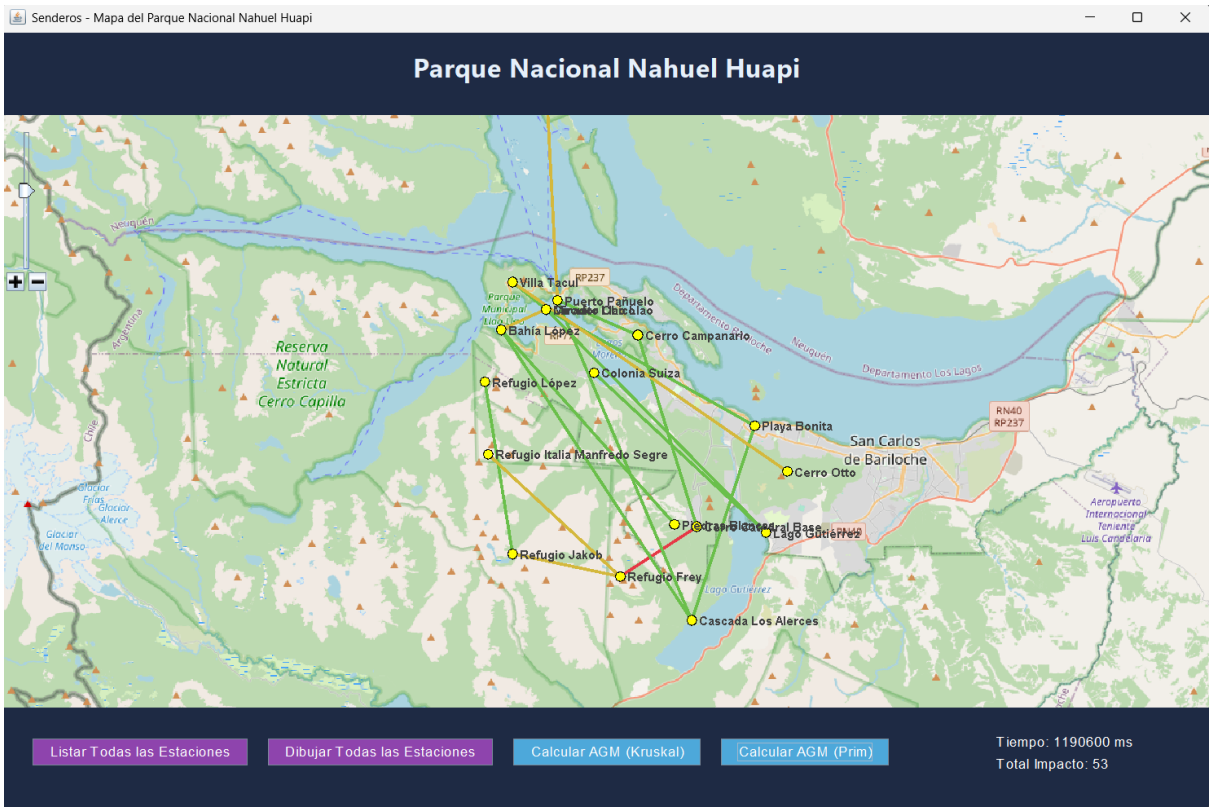
2. **Lectura de Datos desde Archivos:**
   La clase JsonReader carga estaciones y senderos desde nationalParkStations.json usando Gson.

3. **Visualización de Senderos por Impacto Ambiental**:
   En NationalParkMap.drawTrails, los senderos se renderizan con colores según su impacto: verde ([1-3]), amarillo ([4-6]), rojo ([7-10]), definidos en ColorPalette. La actualización dinámica vía NationalParkObserver mejora la usabilidad.

## Vistas




Senderos - Mapa del Parque Nacional Nahuel Huapi
Parque Nacional Nahuel Huapi
Reserva Natural Estricta Cerro Capilla
Villa Tacul
Puerto Panuelo
Bahía López
Cerro Campanario
Refugio López
Colonia Suiza
Playa Bonita
San Carlos de Bariloche
Refugio Italia Manfredo Segre
Cerro Otto
Refugio Jakob
Refugio Frey
Cascada Los Alerces
Lago Gutiérrez
Listar Todas las Estaciones
Dibujar Todas las Estaciones
Calcular AGM (Kruskal)
Calcular AGM (Prim)
Tiempo: 514800 ms
Total Impacto: 53


Senderos - Mapa del Parque Nacional Nahuel Huapi
Parque Nacional Nahuel Huapi
Reserva Natural Estricta Cerro Capilla
Villa Tacul
Puerto Panuelo
Bahía López
Cerro Campanario
Refugio López
Colonia Suiza
Playa Bonita
San Carlos de Bariloche
Refugio Italia Manfredo Segre
Cerro Otto
Refugio Jakob
Refugio Frey
Cascada Los Alerces
Lago Gutiérrez
Listar Todas las Estaciones
Dibujar Todas las Estaciones
Calcular AGM (Kruskal)
Calcular AGM (Prim)
Tiempo: 1190600 ms
Total Impacto: 53

## Ubicación del programa

- https://github.com/ScabiniMelina/NationalPark