



INFO 1126 - Programación 3

Evaluación 2 (35%)

Proyecto: Sistema Logístico Autónomo con Drones

Mayo 2025

Problemática

Correos Chile ha decidido implementar una red de drones autónomos para mejorar su cobertura y reducir los tiempos de entrega. Esta decisión busca superar las limitaciones del transporte terrestre, afectado por congestión y rutas ineficientes.

Como no existe una infraestructura aérea preexistente, el sistema debe ser diseñado completamente desde cero. Este sistema debe contemplar:

- **Centros de distribución (Almacenamiento):** Puntos donde los drones recogen paquetes.
- **Estaciones de carga:** Nodos estratégicos que los drones deben visitar para recargar si exceden su autonomía.
- **Destinos de entrega (Clientes):** Nodos dinámicos con prioridad asignada. Su ubicación y prioridad pueden variar.
- **Rutas seguras:** Las rutas deben ser viables considerando el consumo energético. Si el trayecto excede el límite de autonomía, se debe forzar la visita a estaciones de recarga.
- **Registro de rutas:** El sistema debe registrar la frecuencia de uso de rutas y nodos, para análisis posterior.
- **Selección de rutas:** A partir del registro anterior se debe crear una heurística basada en frecuencia que permita reutilizar rutas recurrentes a nodos más visitados, esto permitirá replicar recorridos ya realizados.






Objetivo

Diseñar e implementar una simulación logística autónoma para drones, asegurando autonomía energética, optimización de rutas, conectividad, análisis de datos y visualización completa del sistema.

Para esto se ha dispuesto un video en youtube a modo de guía

<https://youtu.be/AXj14zeKqTI>

Parámetros de Simulación

- **Cantidad máxima de nodos:** hasta 150
- **Roles de nodos:**
 -  Almacenamiento: 20%
 -  Recarga: 20%
 -  Cliente: 60%
- **Autonomía máxima del dron:** 50 unidades de costo (suma de pesos de aristas)
- **Si una ruta supera el límite de batería, se fuerza el paso por nodos de recarga.**
- **Todas las rutas se generan sobre grafos conectados.**

Metas Funcionales

1. Gestión dinámica de rutas

- Crear rutas entre cualquier centro de almacenamiento y cliente.
- Considerar estaciones de recarga si la energía del dron no alcanza.
- Solo usar algoritmos BFS,DFS y Topological Sort para búsqueda de caminos.

2. Simulación funcional

- Simulación inicial con 15 nodos, 20 aristas y 10 órdenes.
- Soporta hasta 150 nodos y 300 aristas y 500 órdenes como máximo.
- Optimizar uso de memoria y estructura de datos (AVL, mapas, grafos).



3. Análisis estadístico

- Registrar cada ruta usada en un AVL.
- Determinar las rutas más utilizadas.
- Registrar frecuencia de nodos destino y origen.

4. Garantía de conectividad

- Todos los grafos generados son conexos.
- Se evita la generación de nodos aislados.

5. Visualización y Dashboard

La interfaz principal se desarrolla mediante **Streamlit (Recomendado)**, y permite visualizar y operar el sistema en 5 pestañas organizadas funcionalmente. A continuación, se describe cada una:




Pestaña 1: Run Simulation

Propósito:

Permitir la configuración e inicio de la simulación con parámetros personalizables.

Componentes:




- **Slider**: número de nodos (**n_nodes**, entre 10 y 150).
- **Slider**: número de aristas (**m_edges**, entre 10 y 300).
- **Slider**: número de órdenes (**n_orders**, entre 10 y 300).
- **Campo informativo**: Texto informativo indicando la cantidad de Nodos cliente, almacenamiento, abastecimiento y su porcentaje.
- **Botón**:  **Start Simulation (Inicia la simulación)**

Validaciones:



- El número de aristas debe permitir un grafo conexo ($\geq n_{\text{nodes}} - 1$)
- El valor de nodos no debe superar 150.
- Si se hace clic sin modificar parámetros, se mantiene la configuración por defecto(ítem de componentes).

Interacciones esperadas:

- Al presionar el botón, se genera un grafo aleatorio con roles asignados proporcionalmente:
 -  20% almacenamiento
 -  20% recarga
 -  60% clientes

* Nota del docente:

Esta pestaña sirve como punto de entrada para validar la generación del entorno base y comprobar la conectividad y escalabilidad. (Manejo de parámetros)





Pestaña 2: Explore Network

Propósito:

Visualizar la red de transporte y calcular rutas entre nodos, considerando recarga por batería.

Componentes:

- **Gráfico:** red de nodos coloreados por tipo (`matplotlib`)
- **Selectbox:** nodo origen
- **Selectbox:** nodo destino
- **Botón:**  **Calculate Route (Calculará la ruta entre 2 nodos)**
- **TextBox:** muestra un mensaje de la ruta encontrada (lista de nodos + costo, ejemplo : Path: W → I → N → R | Cost: 25)
 - Luego de calcular una ruta debe dar la opción de completar la ruta **Complete Delivery and Create Order** .
- **Leyenda:** colores según tipo de nodo

Validaciones:



- Ambos nodos deben existir en el grafo.
- Si no hay ruta posible con la batería actual, se busca obligatoriamente una alternativa que pase por nodos de recarga.

Interacciones esperadas:

- El grafo muestra rutas visualmente en color rojo.
- Al seleccionar origen y destino y presionar el botón, se calcula la ruta más corta permitida, utilizando **BFS, DFS o Topological Sort (Recomendado)** modificado con batería límite (50).
- Se muestra el camino recorrido y el costo.

* Nota docente:

Evaluar la integración entre lógica (batería, nodos) y visualización. Es una parte central de la evaluación funcional.



Pestaña 3: Clients & Orders

Propósito:

Listar los clientes activos y los pedidos generados, mostrando atributos relevantes.

Componentes:

- **Subsección:** lista de **clientes** (**st.json**)
 - Cliente: ID, nombre, tipo, total de pedidos
- **Subsección:** lista de **órdenes** (**st.json**)
 - Orden: ID, cliente asociado, cliente ID, origen, destino, status, fecha de creación, prioridad, fecha entrega, costo total.

Validaciones:

- Se muestran sólo si hay simulación activa.



Interacciones esperadas:

- Al iniciar una simulación, esta sección se autoactualiza con los datos generados.

*** Nota docente:**

Es útil para verificar que el sistema esté registrando correctamente los pedidos y asociándose a clientes.



Pestaña 4: Route Analytics

Propósito:

Visualizar las rutas más frecuentes utilizadas, registradas en una estructura AVL.

Componentes:

- **Lista:** rutas más frecuentes (clave = camino, valor = frecuencia).
- **Gráfico:** visualización del árbol AVL con etiquetas "A → B → C\nFreq: X" usando `networkx`.

Validaciones:

- Solo disponible si se han generado órdenes y rutas.
- El AVL debe reflejar el uso repetido de rutas idénticas (por ejemplo, múltiples pedidos de un mismo cliente).

Interacciones esperadas:

- Las rutas se ordenan por recorrido (orden lexicográfico).
- Se visualiza gráficamente la estructura del AVL para apoyar aprendizaje en estructuras balanceadas.

*Nota docente:

Parte esencial de la evaluación por uso de TDA AVL. Refuerza estructuras y visualización de árboles.



Pestaña 5: General Statistics

Propósito:

Entregar una vista global del sistema en funcionamiento, con gráficas visuales.

Componentes:

- **Gráfico de barras:** comparación entre número de nodos clientes más visitados, nodos almacenamiento más visitados y nodos de abastecimiento más visitados.
- **Gráfico de torta:** proporción entre nodos por rol.

Validaciones:

- Se requiere simulación activa para mostrar los datos.

Interacciones esperadas:

- Se actualiza automáticamente luego de iniciar una simulación.



Estructura de Clases y Módulos sugerida

Clase / Módulo	Función	Ubicación
Graph, Vertex, Edge	Modela el grafo base (nodos, conexiones)	model/
Simulation	Controlador principal de la simulación	sim/
SimulationInitializer	Generación de grafos conectados y roles	sim/
Route, Order, Client	Representan las entidades del sistema	domain/
AVL	Almacena rutas más frecuentes	tda/
Map (hash map propio)	Acceso O(1) a clientes y órdenes	tda/
NetworkXAdapter	Adaptador visual de grafos para matplotlib	visual/
AVLVisualizer	Dibuja gráficamente el árbol AVL	visual/
dashboard.py	Interfaz principal (pestañas Streamlit)	visual/



El proyecto debe enfocarse en una **ubicación jerárquica**, **dependencia lógica** y **modularidad**, como si construyéramos un **diagrama de arquitectura lógica por capas (layered architecture)**.

```
+-----+
|      Interfaz Visual      | ← GUI / Dashboard
+-----+
|      Lógica de Aplicación | ← Coordina simulaciones, entrega, análisis
+-----+
|      Dominio del Problema | ← Modelos: Pedido, Cliente, Ruta
+-----+
|      Infraestructura TDA  | ← AVL, Diccionario
+-----+
|      Estructuras Base     | ← Graph, Vertex, Edge, Inicializador
+-----+
```

```
proyecto/
├── domain
│   ├── client.py      # ← Clase Client
│   ├── order.py       # ← Clase Order
│   └── route.py       # ← Clase Route
├── model
│   ├── graph.py       # ← TDA del grafo base
│   ├── vertex.py      # ← Clase Graph
│   ├── edge.py        # ← Clase Vertex
│   └── __init__.py    # ← Clase Edge
├── sim
│   ├── init_simulation.py # ← Lógica de simulación y orquestación
│   └── simulation.py   # ← Generador de grafos conectados
│                       # ← Clase Simulation (núcleo principal)
├── tda
│   ├── avl.py         # ← Tipos de Datos Abstractos personalizados
│   └── hasp_map.py    # ← AVL funcional con key-value
│                       # ← Hash map personalizado (tipo dict)
└── visual
    ├── dashboard.py   # ← Visualización Streamlit y adaptadores gráficos
    ├── networkx_adapter.py # ← Interfaz Streamlit con pestañas
    └── avl_visualizer.py # ← Adaptador de grafo → matplotlib
                        # ← Visualización de árbol AVL como gráfico
```



Criterios de Evaluación	Descripción	Excelente (10 pts)	Bueno (8 pts)	Suficiente (4 pts)	Insuficiente (1 pt)	Peso (%)
Simulación funcional con recarga automática	Sistema de simulación completo con lógica de recarga autónoma	Funciona perfectamente sin errores	Funciona con pequeños errores	Funciona parcialmente con fallos evidentes	No funciona o tiene errores críticos	25%
Correcto uso de estructuras TDA	Implementación adecuada de AVL y Hashmap según requerimientos	Ambas estructuras implementadas óptimamente	Una estructura óptima y otra funcional	Estructuras básicas pero con problemas	Estructuras mal implementadas	10%
Dashboard modular con navegación clara	Interfaz organizada en pestañas/secciones intuitivas	Navegación excelente con diseño profesional	Navegación clara con pequeños detalles	Navegación funcional pero mejorable	Interfaz confusa o difícil de usar	15%
Rutas que respetan autonomía de drones	Planificación considera límite de batería y puntos de recarga	Rutas dentro de autonomía	Rutas generalmente válidas (ocasionales fallos)	Rutas a veces exceden autonomía	Rutas no consideran autonomía	15%
Presentación Oral	Claridad y organización en la presentación, descripción de la problemática, diagramas, solución, demostración	Excelente presentación oral; describe la problemática y solución de manera clara, responde	Buena presentación describe la problemática y solución de manera clara, aunque con detalles menores;	Presentación suficiente pero con falta de claridad y organización; la descripción de la problemática y solución es incompleta; manejo básico	Presentación pobremente organizada, falta de claridad y omisión de varios elementos clave; manejo ineficiente de preguntas;	25%



	de código y manejo de preguntas.	adecuadamente las preguntas y cumple con el tiempo de 15 minutos.	responde adecuadamente las pregunta ;Excede el tiempo definido.	de preguntas ; Excede el tiempo definido.	Excede el tiempo definido.	
Código modular y documentado	Organización del código y calidad de documentación	Código ejemplar con documentación completa	Código organizado y bien documentado	Código aceptable con documentación mínima	Código desorganizado sin documentar	5%
Evaluación del Diagrama de Clases	Correcta representación de las clases, relaciones, herencia y otros elementos orientados a objetos en un diagrama de clases.	Diagrama de clases claro, completo y correctamente estructurado, mostrando todas las relaciones adecuadas .	Diagrama de clases funcional y correcto, con algunos detalles por mejorar.	Diagrama de clases con errores o ausencias en algunas relaciones.	Diagrama de clases incorrecto o incompleto, faltando elementos clave.	5%



¿Qué debo entregar?

Debes preparar una presentación **PPT en grupos de 3-5** a presentar en la **semana 13(09/06/2025-13/06/205)**.

Grupo de 1 persona singular debe entregar el proyecto la semana 12(02/06/2025 - 06/06/2025).
la cual debe contener:

- 1) Presentación del grupo y tema (título del problema, nombre de los integrantes, fecha y logo del departamento).
- 2) Diagrama de clases de la problemática.
- 3) Presentación de la solución en código explicando el flujo principal de la solución.
- 4) La presentación de la interfaz visual del problema.
- 5) La demostración de la funcionalidad total del programa.

La presentación tiene un tiempo máximo de 15 minutos y luego una tanda de preguntas de 5 minutos.

Fecha por confirmar en base al número de grupos a conformarse.