Universidad Nacional de La Matanza
Departamento de Ingeniería e Investigaciones Tecnológicas

Programación Avanzada

Grupo beta-1

Trabajo Práctico N°2 : Logística de Robots

Integrantes

DNI	Nombre	Apellido	Email
43.903.622	Juan Ignacio	Bernardez	jbernardez@alumno.unlam.edu.ar
43.013.625	Tomás Agustín	Branchesi	tbranchesi @alumno.unlam.edu.ar
41.744.079	Lautaro Nahuel	Casas	lcasas@alumno.unlam.edu.ar
43.873.225	Agustin Ignacio	Garcia Riveros	agarciariveros@alumno.unlam.edu.ar
44.690.247	Federico Ariel	Martucci	fmartucci@alumno.unlam.edu.ar

Contenido

Introducción

El presente trabajo práctico tiene como objetivo desarrollar un sistema logístico autónomo compuesto por robots, cofres y robopuertos, que interactúan en un entorno simulado. La finalidad es modelar y validar el comportamiento de estos elementos, asegurando que las condiciones del sistema permitan el cumplimiento de las tareas logísticas asignadas.
Dentro del sistema, los **robots logísticos** se encargan de transportar recursos entre distintos **cofres**, los cuales pueden ser de distintos tipos como provisiones, almacenamiento, solicitudes, entre otros. Para funcionar correctamente, los robots deben permanecer dentro del rango de cobertura de al menos un **robopuerto**, que les brinda la posibilidad de recargarse y mantener su operatividad.
La información sobre el entorno se carga desde un archivo JSON estructurado, el cual define la ubicación y configuración de robopuertos, cofres y robots. A partir de esta información, el sistema realiza distintas validaciones para asegurar que:

- Todos los cofres estén cubiertos por al menos un robopuerto.
- Las solicitudes puedan ser atendidas por al menos un proveedor compatible.
- Los robots estén ubicados dentro de zonas de cobertura válidas.

A su vez, el sistema incluye algoritmos de planificación de caminos para que los robots puedan desplazarse de manera eficiente, y estructuras de datos diseñadas para permitir la extensión y flexibilidad del sistema.
A lo largo del informe se detallarán las decisiones de diseño tomadas, la forma en que se modelaron los distintos componentes y las validaciones implementadas para garantizar el correcto funcionamiento del sistema logístico.

Descripción General del Sistema

El sistema simula una red logística compuesta por:

- Cofres, que representan unidades logísticas con distintas funciones (almacenamiento, provisión, solicitud, etc.).
- Robopuertos, que definen áreas de cobertura donde los robots pueden operar y recargarse.
- Robots logísticos, encargados de transportar ítems entre cofres dentro del área de cobertura.

Toda la configuración inicial se especifica en un archivo JSON, que contiene la ubicación y atributos de todos los elementos. Esta información es leída mediante la clase GestorArchivos, la cual:

- Carga y valida los robopuertos utilizando RobopuertoFactory.
- Carga y valida los cofres con CofreFactory, aplicando validaciones de cobertura y factibilidad de solicitudes. Además valida que no exista ubicaciones duplicadas entre cofres y/o robopuertos.
- Crea los robots, asegurándose de que su ubicación inicial esté dentro del área de cobertura.

Una vez construido el entorno, el sistema verifica que:

- Todos los cofres estén dentro de la conectividad definida por los robopuertos (ValidadorConectividad).
- Toda solicitud de ítems tenga al menos un proveedor posible (ValidadorFactibilidad).

Durante la simulación, se utiliza la clase BuscadorCaminos para calcular el camino más corto entre dos ubicaciones, implementando el algoritmo de Dijkstra. Este camino es utilizado por los robots para optimizar sus desplazamientos, teniendo en cuenta restricciones como cobertura y recarga.
Además, el sistema cuenta con un componente de recolección de métricas (MetricsCollector), que permite al final de la ejecución evaluar el rendimiento logístico: cantidad de transportes realizados, distancia total recorrida y recargas efectuadas.
Toda la lógica se organiza en un diseño modular, facilitando la extensión y mantenibilidad del sistema.

Arquitectura del Sistema

El sistema esta organizado en un conjunto de paquetes que separan las distintas responsabilidades del sistema:

- **main.java.coloniaDeRobots:** Contiene las clases base del modelo, como RobotLogistico, Robopuerto, Cofre, Ubicacion, e interfaces o clases auxiliares como ElementoLogistico.
- **main.java.coloniaDeRobots.cofres**: Agrupa los diferentes tipos de cofres, como CofreSolicitud, CofreProvisionActiva, CofreIntermedio, etc., cada uno con su lógica específica.
- **main.java.coloniaDeRobots.util:** Incluye herramientas de validación búsqueda de caminos y recolección de métricas.
- **main.java.logistica.factory:** Implementa el patron Factory, el cual es responsable de instanciar los objetos del modelo a partir de estructuras
- **main.java.logistica.io:** Se encarga de la carga del sistema desde el archivo de configuración, validación de datos y construcción de objetos.

Jerarquía de clases

- ElementoLogistico: clase base para todos los elementos ubicables dentro del sistema (cofres y robopuertos).
  - Cofre: implementa operaciones generales como obtener ubicación y manipular ítems.
    - Subclases: CofreSolicitud, CofreIntermedio, CofreAlmacenamiento, CofreProvisionActiva, CofreProvisionPasiva.
  - Robopuerto: define una ubicación fija con un alcance, desde donde operan los robots y se recargan.

- RobotLogistico: representa un robot que se mueve entre cofres, con batería limitada y capacidad de carga.

Relaciones clave

- Cada RobotLogístico conoce la red de cofres y robopuertos, lo que le permite planificar su movimiento.
- Los Cofres pueden tener relaciones de demanda o provisión de Items.
- Los Robopuertos determinan la cobertura espacial, afectando dónde puede operar un robot.
- Ubicación encapsula coordenadas cartesianas y operaciones de distancia.

Planificación de Rutas

El sistema implementa el algoritmo de Dijkstra para calcular el camino más corto entre dos ubicaciones logísticas (cofres o robopuertos). Esta funcionalidad es clave para la operación de los robots, ya que determina la secuencia óptima de movimientos considerando la distancia y la cobertura de recarga.

Para ejecutar el algoritmo de Dijkstra, primero se construye un grafo ponderado donde:

- Los **nodos** representan instancias de ElementoLogistico, es decir, cofres o robopuertos.
- Las **aristas** se construyen con dos criterios:
  - Entre **cofres y robopuertos** cuando el robopuerto cubre al cofre.
  - Entre **robopuertos**, si la distancia entre ellos es menor o igual a la suma de sus alcances.
- El **peso** de cada arista corresponde a la **distancia euclídea** entre las ubicaciones.

La construcción del grafo se realiza en el método calcularCaminoMasCorto() de la clase BuscadorCaminos, utilizando un diccionario de diccionarios: (Map<ElementoLogistico, Map<ElementoLogistico, Double>>).

Una vez definido el grafo:

- Se inicializan las distancias mínimas desde el origen a cada nodo como ∞, salvo el nodo de origen (0).
- Se utiliza una **cola de prioridad** para elegir en cada iteración el nodo con menor distancia acumulada.
- Se actualizan las distancias y las precedencias de los nodos recorridos.
- Se detiene cuando se alcanza el destino.

El resultado es una instancia de CaminoEsperado, que contiene:

- La lista de nodos (ElementoLogistico) que forman el camino.
- La distancia acumulada a cada nodo.
- La tabla de precedencia de nodos.
- Las estaciones de carga (Robopuerto) encontradas a lo largo del camino.

Lo que se logra con este enfoque es evitar caminos inviables por falta de cobertura, el calculo de manera dinámica para el recorrido mas eficiente ante distintas configuraciones del entorno logístico y priorizar aquellos trayectos que incluyan los puntos de recarga

Simulación y Métricas

Una vez validada la red logística, el sistema se inicializa con cofres accesibles, robopuertos y robots, todos representados en un grafo con pesos definidos por la distancia euclídea entre ubicaciones. La simulación consiste en ciclos donde los robots logísticos intentan cumplir misiones de transporte de ítems desde cofres proveedores hacia cofres solicitantes o intermedios.

La planificación del recorrido se realiza a través del método calcularCaminoMasCorto() de la clase BuscadorCaminos, que aplica el algoritmo de Dijkstra sobre el grafo de elementos logísticos. Además también se almacena en la clase CaminoEsperado las estaciones de carga (Robopuertos) que forman parte del camino más corto, lo que permite que el robot las utilice si su batería no alcanza para llegar al destino final directamente.

Por otra parte, cada robot planifica sus acciones basándose en su capacidad de batería y en el consumo por unidad de distancia (controlado por un factor configurable). Si el camino hasta el destino excede su batería actual, el algoritmo prioriza caminos que incluyan robopuertos intermedios como puntos de recarga.
Este comportamiento se refleja directamente en la lista de estacionDeCarga del objeto CaminoEsperado, la cual el robot analiza para recargar antes de continuar la misión.

Por último, la clase MetricsCollector acumula en tiempo real información relevante sobre el desempeño de la simulación:

- totalTransportes: contador de misiones de transporte completadas.
- distanciaTotal: distancia total recorrida por todos los robots.
- recargas: número de recargas realizadas por los robots.

Al finalizar la simulación, se imprime un resumen con estos datos, que permite evaluar el desempeño general del sistema logístico, considerando eficiencia y nivel de actividad.

Complejidad Computacional

1. Carga y Validación Inicial (GestorArchivos)

1. Lectura del JSON:
Recorre todas las entradas (robopuertos, cofres, robots) con Jackson ⇒ O(N)
2. Instanciación con Factory:
Llamadas a fábricas (CofreFactory, RobotFactory, RobopuertoFactory) ⇒ O(N)
3. Detección de ubicaciones duplicadas:
Inserción de cada ubicación en un HashSet ⇒ O(N) promedio
4. Validación de cobertura (ValidadorConectividad):
Construcción del grafo: $O(C \cdot P + P^2)$ donde C = número de cofres, P = número de robopuertos
Búsqueda de componentes (BFS): $O((C+P) + (C \cdot P + P^2))$
5. Validación de factibilidad (ValidadorFactibilidad):
Para cada solicitud (S) comprobar proveedores en C cofres ⇒ $O(S \cdot C)$
Complejidad total de carga y validación: $O(N + C \cdot P + P^2 + S \cdot C)$**

2. Planificación de Rutas (BuscadorCaminos – Dijkstra)

1. Inicialización:
Construcción de listas de adyacencia y distancias iniciales ⇒ O(V + E)
2. Extracción y relajación:
Operaciones en cola de prioridad: $O((V + E) \cdot \log V)$
Complejidad de Dijkstra: $O((V + E) \cdot \log V) \approx O((C \cdot P + P^2) \cdot \log(C + P))$**

3. Complejidad Combinada en un Ciclo de Simulación

En T ciclos y R robots, cada uno replaneando en el peor caso:
$O(T \cdot [C + R \cdot ((C \cdot P + P^2) \cdot \log(C + P))])$

4. Conclusión

Este análisis de complejidad demuestra que la fase de carga y validación inicial funciona en tiempo lineal respecto a la cantidad de elementos y relaciones definidas, mientras que la planificación de rutas, a través de Dijkstra optimizado, domina el coste en grafos grandes con complejidad $O((C \cdot P + P^2) \log(C + P))$. En la práctica, el sistema mantiene un rendimiento escalable para configuraciones moderadas, y la combinación de ambos procesos garantiza una simulación eficiente y predecible.

Conclusiones

En este proyecto hemos implementado un sistema logístico completo que cumple de manera rigurosa con las exigencias de la consigna: gestión de cinco tipos de cofres (activos, pasivos, solicitud, intermedio y almacenamiento), definición de robopuertos como zonas operativas y estaciones de recarga, y diseño de robots capaces de planificar rutas óptimas mediante Dijkstra y administrar su energía de forma autónoma. La separación de responsabilidades en módulos bien definidos como carga y validación de configuración, dominio de cofres y robots, planificación de rutas y orquestación de la simulación aseguró un alto grado de cohesión y bajo acoplamiento, facilitando la mantenibilidad y futura expansión del sistema.

El uso de patrones de diseño clave ha sido determinante para la robustez y flexibilidad del código. El Factory Method y el Strategy permitieron crear y extender fácilmente nuevos tipos de cofres y comportamientos de parseo. El Template Method garantizó que cada subtipo de cofre ejecute su acción específica respetando un flujo común. Gracias a validaciones exhaustivas en la carga inicial (duplicados, cobertura, factibilidad), el sistema evita errores de configuración y detecta de manera clara tanto el estado estable como las situaciones inviables.

Finalmente, el registro de métricas, es decir, número de transportes, distancia total recorrida, recargas y ciclos, evidencia la eficiencia del sistema. Para futuras versiones, proponemos incorporar heurísticas de búsqueda (como A*) para mejorar la planificación en grafos muy grandes y una interfaz visual que permita seguir en tiempo real el desplazamiento de los robots y el estado de los cofres.