

# Informe Técnico Detallado – Taller 3:

## Maniobras de Trenes

---

Nombre: Gustavo Restrepo (2380618) – Santiago Velazquez (2380378)

Curso: Programación Funcional

Fecha: 3 de mayo de 2025

Lenguaje: Scala

Archivos involucrados: BuscarLista.scala, ManiobrasTrenesTest.scala

### 1. Introducción

### 2. Descripción del Problema

Dado un conjunto de vagones con un orden inicial, se deben realizar maniobras para llegar a un orden final utilizando:

- Una vía de entrada (donde llegan los vagones),
- Una vía auxiliar (donde se pueden estacionar temporalmente),
- Una vía de salida (donde deben quedar en el orden deseado).

La solución consiste en verificar si dicha transformación es posible, y en caso afirmativo, realizarla paso a paso.

### 3. Estructura General del Código

#### 3.1 Función Principal: buscarLista

Ubicada en el archivo BuscarLista.scala, esta función explora recursivamente todas las posibles maniobras para alcanzar el estado deseado.

```
def buscarLista(  
  entrada: List[Int],
```

```

    auxiliar: List[Int],
    salida: List[Int],
    finalDeseado: List[Int],
    visitados: Set[(List[Int], List[Int], List[Int])]
  ): Boolean = {

```

#### 4. Lógica de Exploración

Condición base:

```
if (salida == finalDeseado) true
```

Evita estados repetidos:

```
else if (visitados.contains((entrada, auxiliar, salida))) false
```

Movimientos posibles:

```

val nuevosMovimientos = for {
  nuevoEstado <- List(
    if (entrada.nonEmpty)
      Some((entrada.tail, entrada.head :: auxiliar, salida))
    else None,
    if (entrada.nonEmpty)
      Some((entrada.tail, auxiliar, salida :+ entrada.head))
    else None,
    if (auxiliar.nonEmpty)
      Some((entrada, auxiliar.tail, salida :+ auxiliar.head))
    else None
  ).flatten
} yield nuevoEstado

```

Recursión:

```
nuevosMovimientos.exists {  
    case (e, a, s) => buscarLista(e, a, s, finalDeseado, visitados + ((entrada, auxiliar, salida)))  
}
```

## 5. Función de Prueba: testBuscarLista

Ubicada en ManiobrasTrenesTest.scala, permite ejecutar casos de prueba fácilmente:


```
def testBuscarLista(entrada: List[Int], finalDeseado: List[Int]): Unit = {  
    println(s"Probando: entrada = $entrada, final deseado = $finalDeseado")  
    val resultado = buscarLista(entrada, Nil, Nil, finalDeseado, Set())  
    println(s"Resultado: $resultado\n")  
}
```


## 6. Casos de Prueba Incluidos

Ejemplos de pruebas:

```
testBuscarLista(List(1, 2, 3), List(3, 2, 1)) // Posible  
testBuscarLista(List(1, 2, 3), List(2, 1, 3)) // Posible  
testBuscarLista(List(1, 2, 3), List(2, 3, 1)) // Imposible  
testBuscarLista(List(1, 2, 3, 4), List(4, 3, 2, 1)) // Posible
```


## 7. Cumplimiento de la Rúbrica del Taller


Uso de listas inmutables: 


Recursividad: 

Uso de expresión for: 

Corrección de la lógica: 

Variedad en los casos de prueba: 

Comentarios y legibilidad del código: 

Estructura funcional pura: 

## **8. Conclusión**

La solución presentada implementa correctamente un simulador funcional de maniobras ferroviarias, haciendo uso exclusivo de programación funcional en Scala. El código es limpio, modular, y con buena cobertura de pruebas. La expresión `for` está utilizada correctamente para construir alternativas de movimientos, lo cual cumple un requerimiento clave del enunciado.