



# Tecnológico de Monterrey

## Portada

Implementación de métodos computacionales  
(Gpo 820)

Reporte Final situación problema #3

Profesores:

Román Martínez Martínez  
Fernando Mendívil Borquez

Alumno:

Diego Samperio Arce A01662935

31 de mayo del 2024

# Índice

<b>Portada.....</b>	<b>1</b>
<b>Índice.....</b>	<b>2</b>
<b>Glosario de funciones usadas en el código.....</b>	<b>4</b>
<b>Diseño de estructura de datos.....</b>	<b>6</b>
<b>Diseño conceptual del autómata.....</b>	<b>6</b>
Explicación gráfica del autómata en el que me base.....	7
Explicación gráfica del código.....	8
<b>Mutabilidad de los datos.....</b>	<b>8</b>
<b>Cómo se paralelizaron los procesos.....</b>	<b>9</b>
<b>Ventajas y/o desventajas de realizarlo en Clojure.....</b>	<b>9</b>
<b>Código en clojure.....</b>	<b>9</b>
<b>Estrategia de testing.....</b>	<b>21</b>
<b>Evidencia de testing.....</b>	<b>23</b>
Foto 1.....	23
Foto 2.....	23
Foto 3.....	24
Foto 4.....	24
Foto 5.....	24
Foto 6.....	24
Archivos de llegada de coches por crucero.....	25
Ejemplos de archivos de coches.....	26
Crucero 9.....	26
Crucero 18.....	26
Crucero 29.....	26
Código de estos.....	26
Crucero 9.....	26
Crucero 18.....	27
Crucero 29.....	28
Archivos de lógica de los cruceros.....	29
Ejemplos de archivos de formato de los cruceros.....	30
Crucero 9.....	30
Crucero 18.....	30
Crucero 29.....	30
Código de estos.....	31
Crucero 9.....	31
Crucero 18.....	31
Crucero 29.....	31
Archivo de captura de resultados por crucero.....	32
Ejemplos de archivos de formato de los cruceros.....	33
Crucero 9.....	33
Crucero 18.....	33

Crucero 29.....	33
Código de estos.....	33
Crucero 9.....	33
Crucero 18.....	33
Crucero 29.....	34
<b>Explicaciones de funcionamiento del código.....</b>	<b>35</b>
Funcionamiento de cómo se accede a los cruceros con base en su id.....	35
Funcionamiento de cómo se procesan los cruceros y se muestran estadísticas.....	36
Funcionamiento de promedios y tiempos muertos.....	37
Lectura de archivos.....	38
Procesamiento de archivos.....	38
Tiempo del programa.....	39
<b>Experiencia de aprendizaje (Conclusión).....</b>	<b>39</b>

## Glosario de funciones usadas en el código

**->:** Macro de encadenamiento, permite aplicar una serie de funciones a un valor inicial. Se usa en el código para encadenar la línea en la función de limpiar línea del código.

**trim:** Elimina los espacios en blanco al inicio y al final de una cadena. Se usa de igual manera en la función de limpiar línea para quitar espacios que no deberían de estar en los archivos que se leen.

**split:** Divide una cadena en una secuencia de subcadenas utilizando un delimitador. Se usa en el código para dividir las secuencias fijándose en los espacios que tienen entre cada uno.

**max:** Devuelve el valor máximo de los argumentos proporcionados. Se usa en el código para verificar que cumple con el número requerido en los índices. Se usa en la función de parsear línea.

**min:** Devuelve el valor mínimo de los argumentos proporcionados. Se usa en el código para saber hasta qué número se debe de realizar en la función de procesar-crucero.

**mapcat:** Combina map y concat, aplicando una función a una colección y concatenando los resultados. Esta función la descubrí y es una forma eficiente de realizar un map y concatenar los resultados, se ocupa en la función de procesar líneas y procesar líneas sentido mapeando lo que se requiere y luego haciéndole un concat

**remove:** Elimina elementos de una colección que satisfacen un predicado. Se usa igual en las mismas funciones que el mapcat, pero este se ocupa para revisar si un valor es nil, si es así se elimina para que no se tome en cuenta en los cálculos y promedios.

**atom:** Proporciona un contenedor mutable para datos, que se pueden actualizar de manera segura. Estos se ocupan ya que variables normales no me dejaba modificarlas, pero estas necesitaban estar cambiando en las funciones de lógica para ir actualizando valores, así que investigué que se podría utilizar y encontré que esta era una buena opción.

**let:** Define variables locales. Como ya vimos en clase, esta función especial se ocupa para definir variables, de esto hice uso para ahorrarme mucho código, sin quitar la ventaja de la recursividad.

**fnil:** Devuelve una función que llama a la función dada, sustituyendo argumentos nil por valores predeterminados. Este se ocupó sobre todo en mi función de tiempos muertos, en el que iba viendo si alguno era nil, para modificar su valor y así ir actualizando el valor de tiempos muertos.

**swap!:** Aplica una función a un valor contenido en un atom y actualiza el atom con el nuevo valor. Este se ocupó en las funciones de lógica para ir cambiando los valores de los átomos

**update-in:** Actualiza un valor en una estructura de datos anidada, aplicando una función al valor actual. Esta se uso en mi función de verificar tiempos muertos y la de actualizar tiempos, este lo ocupo para ver si existe o no algo en un ruta y así modificar o no su valor.

**@:** Desreferencia un atom, ref, agent o var. Este se ocupaba de la mano del swap! en las funciones de lógica para desreferenciar y modificar los valores de los átomos.

**constantly:** Devuelve una función que siempre retorna el mismo valor. Este lo ocupe para definir como siempre el valor inicial del tiempo como cero para cada vez que se reinicia la simulación.

**conj:** Añade un elemento a una colección. Es una función muy útil que me ayuda a agregar valores a un lugar en específico.

**recur:** Llamada recursiva. Esta es una función muy útil en clojure para realizar recursividad de una manera muy fácil sin la necesidad de hacer algo explícito.

**rest:** Devuelve todos los elementos de una colección excepto el primero.

**reduce:** Reduce una colección a un solo valor aplicando una función acumulativa. Esta se ocupa en mi función del promedio general en la que va haciéndolo hasta que quede un valor y se obtenga el promedio de los datos obtenidos.

**throw:** Lanza una excepción. Lo uso para cachar errores en caso de que sucedan y se muestre un mensaje explicando que ocurrió.

**into {}:** Into se ocupa para transferir los elementos de una colección a otra.

En este caso en un mapa que se representa con unas "{ }".

**flatten:** Aplana una colección anidada. Este lo ocupo para que las listas imbricadas se puedan leer mucho más rápido y fácil todo los valores.

**vals:** Devuelve una secuencia de los valores de un mapa. Esta función la ocupa donde realice el mapa para poder acceder a los valores obtenidos de los mismos.

**prn:** Imprime un argumento, seguido de una nueva línea. Esto solo lo ocupo para segmentar en mis archivos de resultados de una buena manera todos los resultados y sea más legible.

**sort-by:** Ordena una colección utilizando una función clave. Esta función me ayudó para ordenar las estadísticas de una buena manera y mostrarlas luego en el código.

**doall:** Realiza una evaluación estricta de una secuencia. El doall lo ocupo para que me realizará de todos los archivos la función del paralelismo.

**pmap:** map paralelo; aplica un map a una colección en paralelo. Esta es la función que nos enseñaron a usar y la clave de esta evidencia que se usa para paralelizar el proceso del código y realice de manera paralela la lectura de todos los cruceros.

**{: keys }:** Extrae los valores asociados a las claves especificadas en un mapa. Este lo ocupe para que accediera a la información específica del crucero que haya solicitado el usuario.

**[& \_]:** Permite que la función reciba cualquier número de argumentos. Este lo ocupe para realizar la llamada al main y que no me apareciera ninguna advertencia.

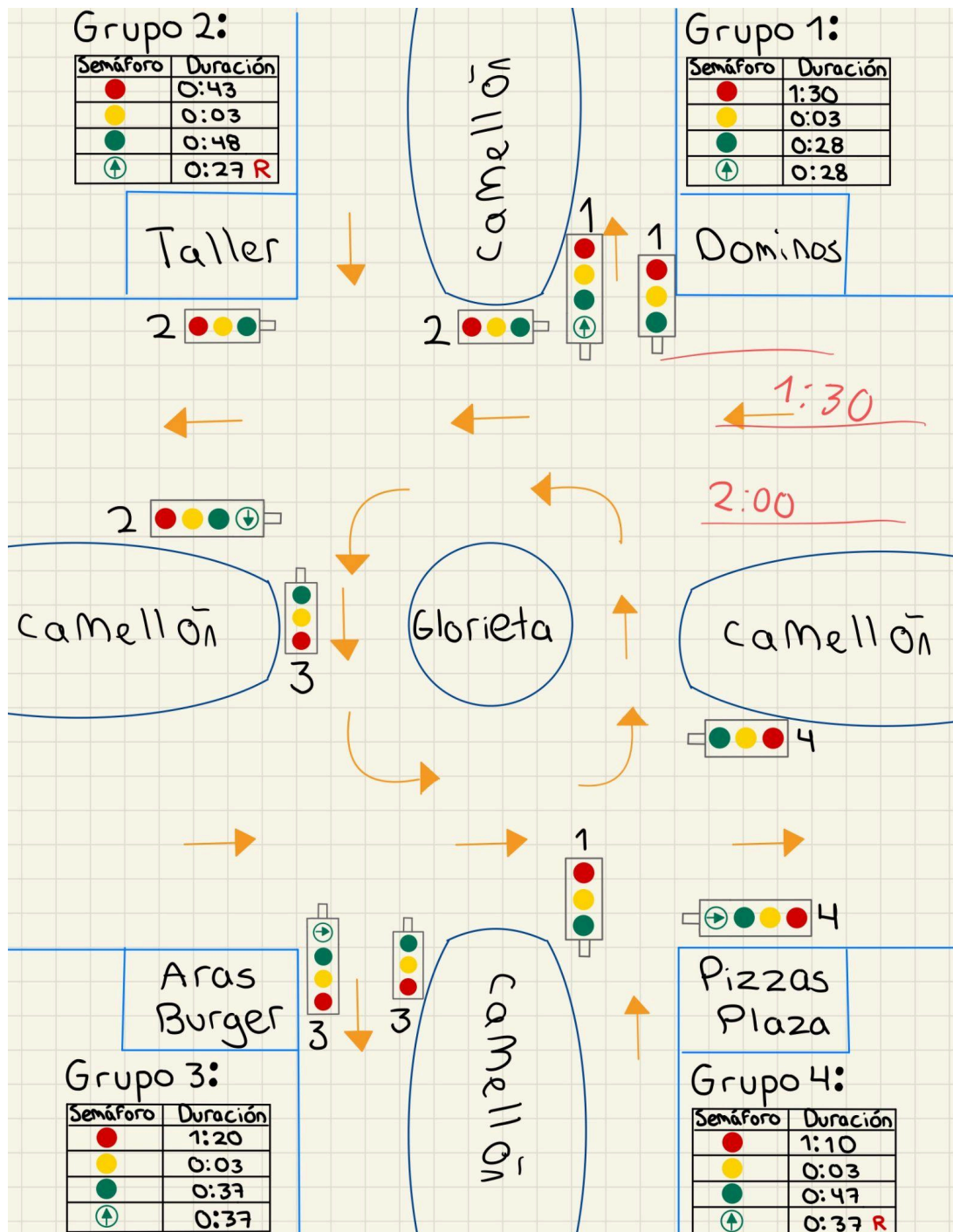
## Diseño de estructura de datos

El diseño de las estructuras de datos en el programa que realicé, se basa en el uso de mapas y átomos para almacenar y actualizar de manera concurrente los resultados de la simulación de tráfico. Se utilizan funciones de parseo y procesamiento de líneas para leer y analizar los datos de los archivos de entrada, sin que aparezcan errores y que si se tomen como números, mientras que los átomos permiten hacer uso de la mutabilidad de las variables. La paralelización se logra mediante la función **pmap**, que se encarga de distribuir el procesamiento de los 40 cruceros de manera concurrente. pmap toma una función (en este caso, procesar-crucero) y una colección de elementos (los identificadores de los cruceros) y aplica la función a cada elemento en paralelo, aprovechando múltiples núcleos de la CPU. Esto permite que el procesamiento de los datos de cada crucero se realice de forma simultánea, reduciendo significativamente el tiempo total de ejecución del programa.

## Diseño conceptual del autómata

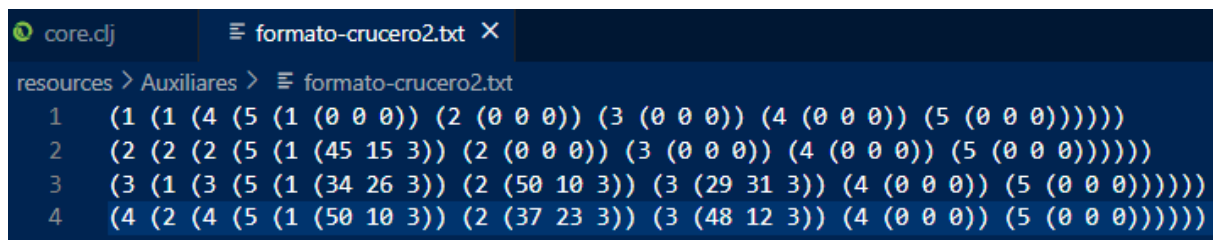
Para el diseño de autómata me basé en parte en el que ya había diseñado en una de las primeras entregas del semestre, que era el funcionamiento de uno de los cruceros donde vivo, en este observé cómo era que se cambiaban los estados entre los diferentes sentidos que existían, en este observe que existían principalmente dos estados, o los sentidos inician con el semáforo rojo, o inician con el estado de verde, y este ciclo se va repitiendo continuamente, en mi código segmenté la lógica por carril ya que pueden existir carriles que tienen un estado de dar la vuelta en lugar de continuar derecho y esos tienen un tiempo diferente al resto, pero se va repitiendo continuamente el ciclo de rojo-verde o verde-rojo según sea el caso, en este caso opté por qué el identificador 1 son los que inician con la luz roja y los que tienen el identificador 2 inician con verde.

## Explicación gráfica del autómata en el que me base



Este fue el cruce del que me base para el diseño del autómata viendo tiempos de duración y como se sincronizaban entre cada uno de estos y porque había tal cantidad de semáforos en cada sentido.

## Explicación gráfica del código



```
core.clj  formato-crucero2.txt X
resources > Auxiliares > formato-crucero2.txt
1  (1 (1 (4 (5 (1 (0 0 0)) (2 (0 0 0)) (3 (0 0 0)) (4 (0 0 0)) (5 (0 0 0))))))
2  (2 (2 (2 (5 (1 (45 15 3)) (2 (0 0 0)) (3 (0 0 0)) (4 (0 0 0)) (5 (0 0 0))))))
3  (3 (1 (3 (5 (1 (34 26 3)) (2 (50 10 3)) (3 (29 31 3)) (4 (0 0 0)) (5 (0 0 0))))))
4  (4 (2 (4 (5 (1 (50 10 3)) (2 (37 23 3)) (3 (48 12 3)) (4 (0 0 0)) (5 (0 0 0))))))
```

Este es un ejemplo de uno de los archivos de lógica que tengo para un crucero, como se ve tenemos 4 listas, cada una tiene un identificador estos simbolizan los 4 puntos cardinales, iniciando por Norte, seguido por Oeste, seguido de Sur y por último Este. Luego de este bien un número que va del 1 al 2, esto simboliza el estado del autómata que tendrá ese sentido, luego viene la cantidad de semáforos que hay en el mismo, luego el siguiente número simboliza la cantidad de carriles que hay como máximo y ya después varias listas explicando que tiempos tiene el carril 1,2,3,4 y 5. Los que tienen en sus 3 estados 0 significa que no existe ese carril, y si los 5 carriles de un sentido tiene todos en 0 significa que ese sentido no existe y no se tomará en cuenta para el promedio.

## Mutabilidad de los datos

El programa gestiona la mutabilidad de los datos utilizando átomos de Clojure, que permiten cambios seguros y coordinados a los datos compartidos en un entorno concurrente. Los átomos proporcionan operaciones atómicas de actualización, garantizando que las modificaciones a los datos compartidos sean consistentes y evitando condiciones de carrera. En el programa, se emplean átomos para almacenar los resultados totales, los tiempos muertos, y otros contadores relacionados con el tráfico en los cruceros.

En cuanto a la portabilidad del programa a cruceros con diferentes cantidades de semáforos y luces, el programa está diseñado de manera flexible para leer y procesar datos de entrada que describen la configuración específica de cada crucero. Por lo tanto, no se necesitan cambios en el código.

Para evitar los problemas comunes de la paralelización como condiciones de carrera, exclusión mutua y deadlocks, utilicé átomos en lugar de estructuras de datos mutables convencionales. Los átomos de Clojure garantizan que las actualizaciones sean atómicas y serializadas, es decir, cada cambio es visible para todas las threads de manera secuencial y consistente. Aparte de que gracias a usar la función de `pmap` ayuda a manejar la paralelización sin requerir bloqueos explícitos ni mecanismos de exclusión mutua evitando así estos problemas. Esto se debe a que `pmap` distribuye las tareas entre los núcleos de CPU de manera controlada,



asegurando que cada thread trabaje de manera independiente sobre su propia porción de datos sin interferencias.

## Cómo se paralelizaron los procesos

Para paralelizar los procesos y hacer más eficiente la ejecución del simulador, se utilizó la función `pmap` de Clojure. Esta función permite aplicar una función de procesamiento a cada elemento de una colección en paralelo, distribuyendo las tareas entre múltiples núcleos de CPU. En este caso, `pmap` se emplea para procesar simultáneamente los datos de 40 cruceros diferentes, cada uno identificado por un número del 1 al 40. La función `procesar-crucero` se aplica a cada identificador de crucero, lo que permite que el procesamiento de datos, la simulación de tráfico y el cálculo de estadísticas se realicen concurrentemente.

## Ventajas y/o desventajas de realizarlo en Clojure

Desarrollar el simulador en un lenguaje funcional como Clojure ofrece varias ventajas significativas. La inmutabilidad por defecto evita problemas comunes de concurrencia, como las condiciones de carrera, simplificando el razonamiento sobre el estado del programa y mejorando la seguridad en entornos concurrentes. Además, me permitió reducir en gran medida el código que ya tenía implementado de la anterior evidencia de racket, pudiendo utilizar funciones como `reduce`, `pmap`, `nth`, el uso de átomos, etc... que hizo que acceda a los elementos de una forma mucho más rápida y clara, que me ayudaron para manipular colecciones y paralelizar las tareas de manera clara y eficiente. Algo importante es que la integración con librerías de Java permite aprovechar una amplia variedad de bibliotecas y herramientas maduras, mejorando y facilitando mucho todo.

La desventaja que puedo mencionar es que no vi que la comunidad sea tan grande y tuve poco tiempo de aprendizaje.

## Código en clojure

```
(ns prueba.core
  (:require [clojure.java.io :as io]
            [clojure.string :as str])
  (:gen-class))

(defn limpiar-linea [linea]; Define la función limpiar-linea que toma
una línea como entrada
```

```
(-> linea; Usa el threading macro (->) para encadenar funciones
  (str/replace #"[]" "" ); Reemplaza todos los paréntesis en la
línea con una cadena vacía
  (str/trim)); Elimina los espacios en blanco iniciales y finales
de la línea
```

```
(defn parsear-linea [linea]; Define la función parsear-linea que toma
una línea como entrada
  (if (empty? (limpiar-linea linea)); Si la línea limpiada está vacía
    []; Retorna una lista vacía
    (map #(Integer/parseInt %) (str/split (limpiar-linea linea)
#"\s+")))); De lo contrario, divide la línea limpiada en tokens usando
espacios en blanco como delimitador y conierte en enteros
```

```
(defn leer-archivo [ruta]; Define la función leer-archivo que toma la
ruta de un archivo como entrada
  (try; Intenta ejecutar el siguiente bloque de código
    (with-open [rdr (io/reader ruta)]; Abre un lector para el archivo
especificado por la ruta
      (doall (map parsear-linea (line-seq rdr)))); Lee todas las líneas
del archivo, parsea cada línea y retorna una lista de las líneas
parseadas
    (catch Exception e; Si ocurre una excepción durante la lectura del
archivo
      (println (str "Error al leer el archivo " ruta ": " (.getMessage
e)))); Imprime un mensaje de error junto con la ruta del archivo y el
mensaje de la excepción
    [])); Retorna una lista vacía en caso de error
```

```
(defn procesar-lineas-sentido-helper [linea indices crucero-id
linea-num]
  (if (>= (count linea) (apply max indices)) ; Verifica si la línea es
suficientemente larga para los índices
    (map #(nth linea %) indices) ; Si es suficientemente larga, extrae
los valores en los índices especificados
    (do ; Si la línea es demasiado corta
      (println (str "Línea demasiado corta
(procesar-lineas-sentido-helper) en el crucero " crucero-id " en la
línea " linea-num ": " linea " con indices: " indices)) ; Imprime un
mensaje de error
      (println (str "Longitud de la línea: " (count linea))) ; Imprime
la longitud de la línea
      nil))) ; Retorna nil
```

```
(defn procesar-lineas-sentido [coches crucero-id]
  (mapcat (fn [linea-num linea]
```

```

        (if (empty? linea) ; Verifica si la línea está vacía
            (do
                (println (str "Línea vacía
(procesar-lineas-sentido-helper) en el crucero " crucero-id " en la
línea " linea-num ": " linea)) ; Imprime un mensaje de error
                [])) ; Retorna una lista vacía
            (remove nil?
                [(procesar-lineas-sentido-helper linea [1 2 3 4]
crucero-id linea-num) ; Procesa la línea con diferentes conjuntos de
índices
                (procesar-lineas-sentido-helper linea [2 3 4 5]
crucero-id linea-num)
                (procesar-lineas-sentido-helper linea [3 4 5 6]
crucero-id linea-num)
                (procesar-lineas-sentido-helper linea [4 5 6 7]
crucero-id linea-num)
                (procesar-lineas-sentido-helper linea [5 6 7 8]
crucero-id linea-num)])))
        (range) coches)) ; Aplica la función a cada línea en 'coches'

(defn procesar-lineas-helper [linea indices crucero-id linea-num]
    (if (>= (count linea) (apply max indices)) ; Verifica si la línea es
suficientemente larga para los índices
        (map #(nth linea %) indices) ; Si es suficientemente larga, extrae
los valores en los índices especificados
        (do ; Si la línea es demasiado corta
            (println (str "Línea demasiado corta (procesar-lineas-helper) en
el crucero " crucero-id " en la línea " linea-num ": " linea " con
índices: " indices)) ; Imprime un mensaje de error
            (println (str "Longitud de la línea: " (count linea))) ; Imprime
la longitud de la línea
            nil))) ; Retorna nil

(defn procesar-lineas [logicas crucero-id]
    (mapcat (fn [linea-num linea]
        (if (empty? linea) ; Verifica si la línea está vacía
            (do
                (println (str "Línea vacía (procesar-lineas-helper) en
el crucero " crucero-id " en la línea " linea-num ": " linea)) ; Imprime
un mensaje de error
                [])) ; Retorna una lista vacía
            (remove nil?
                [(procesar-lineas-helper linea [8 9 10] crucero-id
linea-num) ; Procesa la línea con diferentes conjuntos de índices
                (procesar-lineas-helper linea [9 10 11]
crucero-id linea-num)

```

```

                                (procesar-lineas-helper linea [10 11 12]
crucero-id linea-num)
                                (procesar-lineas-helper linea [11 12 13]
crucero-id linea-num)
                                (procesar-lineas-helper linea [12 13 14]
crucero-id linea-num)))))
    (range) logicas)) ; Aplica la función a cada línea en
'logicas'

(def resultados-totales (atom {})) ; Define un atom para almacenar los
resultados totales
(def veces-total (atom 1)) ; Define un atom para contar el número total
de veces
(def tiempos-muertos (atom {})) ; Define un atom para almacenar los
tiempos muertos por crucero
(def total-tiempos (atom {})) ; Define un atom para almacenar el total
de tiempos por crucero
(def total-llamadas (atom {})) ; Define un atom para contar el número
total de llamadas por crucero
(def tiempos-muertos-detallado (atom {})) ; Define un atom para
almacenar tiempos muertos detallados por semáforo y carril
(def tiempos-detallados (atom {})) ; Define un atom para almacenar
tiempos detallados por crucero
(def tiempo-muerto-acumulado (atom {})) ; Define un atom para almacenar
el tiempo muerto acumulado por crucero

(defn verificar-tiempos-muertos [tiempo-actual tiempo-anterior semaforo
carril crucero-id funcionamiento]; Define la función
verificar-tiempos-muertos con parámetros tiempo-actual, tiempo-anterior,
semaforo, carril, crucero-id y funcionamiento
    (let [tiempo-limite (+ (first funcionamiento) (second
funcionamiento))]; Calcula el tiempo límite como la suma de los primeros
dos elementos de funcionamiento
        (if (> (- tiempo-anterior tiempo-actual) tiempo-limite); Si la
diferencia entre tiempo-anterior y tiempo-actual es mayor que el
tiempo-limite
            (do
                (swap! tiempos-muertos update crucero-id (fn nil inc 0));
Incrementa el contador de tiempos muertos para el crucero-id en el atom
tiempos-muertos
                (swap! tiempos-muertos-detallado update-in [crucero-id semaforo
carril] (fn nil inc 0)); Incrementa el contador de tiempos muertos
detallados para el crucero-id, semaforo y carril en el atom
tiempos-muertos-detallado
                (swap! tiempo-muerto-acumulado update crucero-id (fn nil + 0) (-
tiempo-anterior tiempo-actual))); Actualiza el tiempo muerto acumulado

```

para el crucero-id en el atom tiempo-muerto-acumulado  
(@tiempos-muertos crucero-id)))) ; Retorna el valor actual del  
tiempo muerto para el crucero-id

(defn convertir-a-positivo [valor]; Define la función  
convertir-a-positivo con el parámetro valor  
(if (neg? valor); Si el valor es negativo  
(\* -1 valor); Retorna el valor multiplicado por -1 (lo convierte en  
positivo)  
valor)) ; De lo contrario, retorna el valor tal como está

(defn calcular-tiempo-espera [tiempo-llegada tiempo-ciclo-rojo  
coches-pasados tiempo-restV tiempo-espR]; Define la función  
calcular-tiempo-espera con los parámetros tiempo-llegada,  
tiempo-ciclo-rojo, coches-pasados, tiempo-restV y tiempo-espR  
(let [tiempo-espera-rojo (- tiempo-ciclo-rojo tiempo-llegada); Calcula  
el tiempo de espera en rojo como la diferencia entre tiempo-ciclo-rojo y  
tiempo-llegada

tiempo-espera-total (convertir-a-positivo (+ tiempo-espera-rojo  
tiempo-espR))]; Calcula el tiempo de espera total como la suma del  
tiempo-espera-rojo y tiempo-espR, convertido a positivo

(cond  
(and (= coches-pasados 0) (<= tiempo-restV 0) (<= tiempo-llegada  
tiempo-ciclo-rojo)); Si no han pasado coches, el tiempo restante en  
verde es menor o igual a 0 y tiempo-llegada es menor o igual a  
tiempo-ciclo-rojo

(convertir-a-positivo (+ tiempo-ciclo-rojo (\* 2 coches-pasados)));  
Retorna el tiempo-ciclo-rojo multiplicado por 2 veces el número de  
coches pasados, convertido a positivo

(= coches-pasados 0); Si no han pasado coches  
tiempo-espera-total ;Retorna el tiempo de espera total

:else;Sino

(convertir-a-positivo (+ tiempo-espera-total (\* 2  
coches-pasados))))); Retorna la suma del tiempo de espera total y 2  
veces el número de coches pasados, convertido a positivo

(defn iterar-coches [coches-pasados resultados tiempo-restV  
funcionamiento tiempo-espR tiempo-espera-total semaforo carril  
crucero-id tiempo-anterior]

; Define la función iterar-coches con varios parámetros:  
coches-pasados, resultados, tiempo-restV, funcionamiento, tiempo-espR,  
tiempo-espera-total, semaforo, carril, crucero-id y tiempo-anterior

(if (empty? coches-pasados) ; Verifica si la lista de coches-pasados  
está vacía

```

    resultados ; Si está vacía, retorna los resultados
    (let [tiempo-llegada (first coches-pasados) ; Obtiene el tiempo de
llegada del primer coche en la lista
        tiempo-ciclo-completo (+ (first funcionamiento) (second
funcionamiento) (nth funcionamiento 2)) ; Calcula el tiempo de ciclo
completo sumando los primeros tres valores de funcionamiento
        tiempo-ciclo-rojo (first funcionamiento) ; Obtiene el tiempo
de ciclo rojo del primer valor de funcionamiento
        tiempo-actualizado (if (or (<= tiempo-restV 0)
                                   (< (* @veces-total
tiempo-ciclo-completo)
                                   (+ tiempo-llegada
tiempo-espera-total))))
        tiempo-ciclo-completo tiempo-ciclo-rojo)
    ; Determina el tiempo actualizado basado en las condiciones del
tiempo-restV y tiempo-llegada
    nuevo-tiempo-restV (if (<= tiempo-restV 0)
                           (+ (second funcionamiento) (nth
funcionamiento 2))
                           (- tiempo-restV 2)) ; Calcula el nuevo
tiempo restante en verde
    tiempo-espera (calcular-tiempo-espera tiempo-llegada
tiempo-actualizado (count resultados) tiempo-restV tiempo-espR)] ;
Calcula el tiempo de espera usando la función calcular-tiempo-espera
    (verificar-tiempos-muertos tiempo-espera tiempo-anterior semaforo
carril crucero-id funcionamiento) ; Verifica y actualiza los tiempos
muertos si es necesario
    (recur (rest coches-pasados) ; Llama recursivamente a
iterar-coches con los parámetros actualizados
           (conj resultados tiempo-espera) ; Añade el tiempo de espera
a los resultados
           nuevo-tiempo-restV funcionamiento tiempo-espR
tiempo-espera-total semaforo carril crucero-id tiempo-espera)))) ; Pasa
el tiempo de espera actual como tiempo-anterior en la siguiente llamada
recursiva

(defn Tiempos1 [funcionamiento tiempos semaforo carril crucero-id];
Define la función Tiempos1 con los parámetros funcionamiento, tiempos,
semaforo, carril y crucero-id
    (if (or (< (count funcionamiento) 3) (empty? tiempos)) ; Verifica si
la longitud de funcionamiento es menor a 3 o si tiempos está vacío
        '() ; Si alguna de las condiciones se cumple, retorna una lista
vacía
        (let [resultados (iterar-coches tiempos '() (+ (second
funcionamiento) (nth funcionamiento 2)) funcionamiento 0 0 semaforo
carril crucero-id 0)]

```

```

    ; Si las condiciones no se cumplen, llama a iterar-coches con los
    parámetros correspondientes y almacena el resultado en resultados
    (reset! veces-total 1) ; Reinicia el contador de veces-total a 1
    (swap! tiempos-muertos update crucero-id (constantly 0)) ;
    Reinicia el contador de tiempos-muertos para el crucero-id a 0
    (swap! resultados-totales update crucero-id (fnil conj []))
    resultados) ; Actualiza los resultados-totales con los nuevos resultados
    para el crucero-id
    resultados))) ; Retorna los resultados

```

(defn actualizar-tiempos [tiempos semaforo carril crucero-id]; Define la función actualizar-tiempos con los parámetros tiempos, semaforo, carril y crucero-id

```

    (if (not (or (empty? tiempos) (empty? (@resultados-totales
    crucero-id)))); Verifica si tiempos no está vacío y si
    resultados-totales para el crucero-id no está vacío
    (do
        (swap! total-tiempos update crucero-id (fnil + 0) (apply +
    tiempos)) ; Actualiza el total-tiempos para el crucero-id sumando los
    tiempos
        (swap! total-llamadas update crucero-id (fnil inc 0)) ; Incrementa
    el contador de total-llamadas para el crucero-id
        (swap! tiempos-detallados update-in [crucero-id semaforo carril]
    (fnil conj []) tiempos)) ; Actualiza tiempos-detallados con los nuevos
    tiempos para el crucero-id, semaforo y carril
        (println (str "No existe ese carril o sentido así que no se tomara
    en cuenta para el promedio en el crucero " crucero-id)))) ; Si alguna de
    las condiciones no se cumple, imprime un mensaje de error

```

(defn procesar-carril [llegadas logica semaforo carril crucero-id]; Define la función procesar-carril con los parámetros llegadas, logica, semaforo, carril y crucero-id

```

    (let [tiempos (Tiempos1 logica llegadas semaforo carril crucero-id)];
    Llama a Tiempos1 con los parámetros correspondientes y almacena el
    resultado en tiempos
        (actualizar-tiempos tiempos semaforo carril crucero-id) ; Llama a
    actualizar-tiempos con los parámetros correspondientes
    tiempos)) ; Retorna los tiempos

```

(defn leer-archivos [crucero-id]; Define la función leer-archivos con el parámetro crucero-id

```

    (let [archivo-port1 (str
    "C:\\Users\\diego\\IMECO\\Clojure\\prueba\\resources\\Auxiliares\\format
    o-crucero" crucero-id ".txt"); Construye la ruta del archivo de formato
    usando crucero-id
        archivo-port2 (str

```

```
"C:\\Users\\diego\\IMECO\\Clojure\\prueba\\resources\\Auxiliares\\coches-  
-crucero" crucero-id ".txt"]); Construye la ruta del archivo de coches  
usando crucero-id
```

```
(if (and (.exists (io/file archivo-port1)) (.exists (io/file  
archivo-port2))); Verifica si ambos archivos existen  
(let [logicas (leer-archivo archivo-port1); Lee el archivo de  
formato y almacena el contenido en logicas  
coches (leer-archivo archivo-port2)]; Lee el archivo de  
coches y almacena el contenido en coches  
[logicas coches]); Retorna una lista con logicas y coches  
(do  
(println (str "Alguno de los archivos del crucero " crucero-id "  
no existe."))); Si alguno de los archivos no existe, imprime un mensaje  
de error  
[[[] []]])) ; Retorna una lista con dos listas vacías
```

```
(defn mostrar-promedio-general [crucero-id]; Define la función  
mostrar-promedio-general con el parámetro crucero-id  
(let [total-carros (reduce + (map count (@resultados-totales  
crucero-id))); Calcula el total de carros sumando las longitudes de las  
listas en resultados-totales para el crucero-id  
total-tiempo (reduce + (map #(reduce + %) (@resultados-totales  
crucero-id))); Calcula el total de tiempo sumando todos los tiempos en  
resultados-totales para el crucero-id  
(if (zero? total-carros); Verifica si el total de carros es cero  
(throw (ArithmeticException. "No se puede dividir por cero: No hay  
carros"))); Si es cero, lanza una excepción de división por cero  
(/ total-tiempo total-carros)))] ; Si no es cero, calcula y  
retorna el promedio dividiendo total-tiempo por total-carros
```

```
(defn calcular-promedio-por-semaforo-carril [tiempos-detallados  
crucero-id]; Define la función calcular-promedio-por-semaforo-carril con  
los parámetros tiempos-detallados y crucero-id  
(into {} ; Convierte el resultado de la siguiente expresión en un mapa  
(for [[sem carriles] (get tiempos-detallados crucero-id)] ;  
Itera sobre cada par [sem carriles] en el mapa para el crucero-id en  
tiempos-detallados  
[sem (into {} ; Convierte el resultado en un mapa  
(for [[carril tiempos] carriles]; va iterando con  
los parametros de carril y sus tiempos  
[carril (if (seq tiempos); Checa si no esta vacio  
(/ (reduce + (flatten tiempos)) (count  
(flatten tiempos)))]); Calcula el promedio dividiendo la suma de tiempos  
aplanados por la cantidad de tiempos aplanados  
0)))])); Si esta vacio asigna el  
valor 0
```



(defn contar-tiempos-muertos-totales [tiempos-muertos-detallado crucero-id]; Define la función contar-tiempos-muertos-totales con los parámetros tiempos-muertos-detallado y crucero-id

(reduce + 0; Suma todos los valores en el resultado de la siguiente expresión, comenzando desde 0

(for [\_ carriles] (get tiempos-muertos-detallado crucero-id)); Itera sobre cada par [sem carriles] en el mapa para el crucero-id en tiempos-muertos-detallado

(reduce + (vals carriles)))); Suma todos los valores de carriles (tiempos muertos por carril) y retorna la suma total

(defn contar-tiempo-muerto-acumulado [tiempo-muerto-acumulado crucero-id]; Define la función contar-tiempo-muerto-acumulado con los parámetros tiempo-muerto-acumulado y crucero-id

(get tiempo-muerto-acumulado crucero-id 0)) ; Obtiene el tiempo muerto acumulado para el crucero-id, retornando 0 si no existe

(defn guardar-tiempos-en-archivo [ruta tiempos-detallados crucero-id]; Define la función guardar-tiempos-en-archivo con los parámetros ruta, tiempos-detallados y crucero-id

(let [contenido (with-out-str (prn (get tiempos-detallados crucero-id)))); Cambia los tiempos para el crucero-id a una cadena de texto y la almacena en contenido

(spit ruta contenido));y lo escribe donde corresponde

(defn procesar-crucero [crucero-id] ; Define la función procesar-crucero con el parámetro crucero-id

(let [[logicas coches] (leer-archivos crucero-id)]; Lee los archivos de lógica y coches para el crucero-id

resultados-logicas (procesar-lineas logicas crucero-id); Procesa las líneas de lógica y almacena los resultados en resultados-logicas

resultados-coches (procesar-lineas-sentido coches crucero-id); Procesa las líneas de coches y almacena los resultados en resultados-coches

archivo-tiempos (str "C:\\Users\\diego\\IMECO\\Clojure\\prueba\\resources\\Auxiliares\\tiempos\_de\_llegada-crucero" crucero-id ".txt"); Construye la ruta del archivo de tiempos de llegada usando crucero-id

(doseq [i (range (min (count resultados-logicas) (count resultados-coches)))); Itera sobre el rango desde 0 hasta el menor valor entre la longitud de resultados-logicas y resultados-coches

(let [logica (nth resultados-logicas i); Obtiene la lógica en la posición i de resultados-logicas

coche (nth resultados-coches i)]; Obtiene el coche en la posición i de resultados-coches

```
(procesar-carril coche logica (int (/ i 5)) (mod i 5)
crucero-id)); Procesa el carril con los parámetros coche, logica,
semaforo, carril y crucero-id
```

```
(guardar-tiempos-en-archivo archivo-tiempos @tiempos-detallados
crucero-id);guarda los tiempos en el archivo correspondiente
{:crucero-id crucero-id, :promedio (mostrar-promedio-general
crucero-id), :tiempo-muerto (contar-tiempo-muerto-acumulado
@tiempo-muerto-acumulado crucero-id))};y devuelve uno mapeo con los
dellates solicitados
```

```
(defn contar-coches-en-archivo [ruta] ; Define la función
contar-coches-en-archivo con el parámetro ruta
```

```
(try
  (with-open [rdr (io/reader ruta)]; intenta abrir el archivo
  especificado por la ruta
    (reduce + 0 (map count (map parsear-linea (line-seq rdr))))); Lee
todas las líneas del archivo, parsea cada línea, cuenta los elementos en
cada línea, y suma estas cuentas usando reduce, comenzando desde 0
  (catch Exception e
    (println (str "Error al leer el archivo " ruta ": " (.getMessage
e)))) ; Si sucede un error, imprime un mensaje de error junto con la ruta
del archivo y el mensaje de la excepción
    0)));y devuelve 0
```

```
(defn mostrar-estadisticas [crucero-id]; Define la función
mostrar-estadisticas con el parámetro crucero-id
```

```
(let [ruta-coches (str
"C:\\Users\\diego\\IMECO\\Clojure\\prueba\\resources\\Auxiliares\\coches
-crucero" crucero-id ".txt"); Construye la ruta del archivo de coches
usando crucero-id
```

```
total-coches (contar-coches-en-archivo ruta-coches)]; Cuenta el
total de coches en el archivo especificado por ruta-coches y almacena el
resultado en total-coches
```

```
;realiza las impresiones de cada estadistica del crucero
```

```
(println "Cantidad de coches que pasaron en el crucero:"
total-coches)
```

```
(println "Cantidad de coches que pasaron por semaforo y carril:"
(get @tiempos-detallados crucero-id))
```

```
(println "Cantidad total de veces en el crucero que un semaforo
estuvo en verde y no paso ningun vehiculo:"
```

```
(contar-tiempos-muertos-totales @tiempos-muertos-detallado crucero-id))
```

```
(println "Veces de tiempo muerto por semaforo y carril:" (get
@tiempos-muertos-detallado crucero-id))
```

```
(println "Tiempo promedio total de los coches en el crucero:"
(mostrar-promedio-general crucero-id))
```

```

    (println "Tiempo promedio por semaforo y carril:"
(calcular-promedio-por-semaforo-carril @tiempos-detallados
crucero-id))))

(defn solicitar-crucero-id []; Define la función solicitar-crucero-id
  (println "\n---Estadísticas de Crucero Especifico---\n")
  (print "Si quieres ver la estadística de un crucero, pon su
identificador (0 para salir):"); Solicita al usuario que ingrese el
identificador de un crucero
  (Integer. (read-line))) ; Lee la entrada del usuario, la convierte a
entero y la retorna

(defn convertir-tiempo [nano-segundos]
  ; Define la función convertir-tiempo con el parámetro nano-segundos
  (let [segundos (/ nano-segundos 1e9)]
    ; Convierte los nanosegundos a segundos dividiendo por 1e9 (mil
millones)
    (format "%.5f segundos" segundos))) ; Formatea los segundos a 5
decimales y añade la unidad "segundos"

(defn main []; Define la función principal main
  (let [cruceros-ids (range 1 41)]; Crea una lista de identificadores de
cruceros del 1 al 40
    tiempo-inicial (System/nanoTime); Obtiene el tiempo inicial en
nanosegundos
    tiempos-promedios (doall (pmap procesar-crucero cruceros-ids));
Procesa todos los cruceros en paralelo usando pmap y almacena los
resultados en tiempos-promedios
    tiempo-final (System/nanoTime); Obtiene el tiempo final en
nanosegundos
    tiempos-promedios-ordenados (sort-by :promedio
tiempos-promedios); Ordena los tiempos-promedios por el campo :promedio
    tiempos-muertos-ordenados (sort-by :tiempo-muerto
tiempos-promedios); Ordena los tiempos-promedios por el campo
:tiempo-muerto
    tiempo-total (- tiempo-final tiempo-inicial)]; Calcula el tiempo
total de ejecución restando tiempo-inicial de tiempo-final
    (println "\n---Estadísticas Generales---\n"); Imprime el encabezado
de estadísticas generales
    (println "Tiempo promedio de espera en la totalidad de los vehiculos
de la simulacion:"
      (/ (reduce + (map :promedio tiempos-promedios)) (count
tiempos-promedios))); Calcula e imprime el tiempo promedio de espera de
todos los vehículos en la simulación
    (println "\nTop 4 cruceros con mayor tiempo de espera promedio:");
Imprime el encabezado para el top 4 de cruceros con mayor tiempo de

```

```

espera promedio
    (doseq [{:keys [crucero-id promedio]} (take-last 4
tiempos-promedios-ordenados)]; Itera sobre los últimos 4 elementos de
tiempos-promedios-ordenados
        (println (str "Crucero " crucero-id ": " promedio))); Imprime el
identificador y el promedio de tiempo de espera para cada crucero

    (println "\nTop 4 cruceros con menor tiempo de espera promedio:");
Imprime el encabezado para el top 4 de cruceros con menor tiempo de
espera promedio
    (doseq [{:keys [crucero-id promedio]} (take 4
tiempos-promedios-ordenados)]; Itera sobre los primeros 4 elementos de
tiempos-promedios-ordenados
        (println (str "Crucero " crucero-id ": " promedio))); Imprime el
identificador y el promedio de tiempo de espera para cada crucero

    (println "\nTop 4 cruceros con mayor cantidad de tiempo muerto
acumulado:"); Imprime el encabezado para el top 4 de cruceros con mayor
cantidad de tiempo muerto acumulado
    (doseq [{:keys [crucero-id tiempo-muerto]} (take-last 4
tiempos-muertos-ordenados)]; Itera sobre los últimos 4 elementos de
tiempos-muertos-ordenados
        (println (str "Crucero " crucero-id ": " tiempo-muerto "
segundos"))); Imprime el identificador y el tiempo muerto acumulado para
cada crucero

    (println "\nTop 4 cruceros con menor cantidad de tiempo muerto
acumulado:"); Imprime el encabezado para el top 4 de cruceros con menor
cantidad de tiempo muerto acumulado
    (doseq [{:keys [crucero-id tiempo-muerto]} (take 4
tiempos-muertos-ordenados)]; Itera sobre los primeros 4 elementos de
tiempos-muertos-ordenados
        (println (str "Crucero " crucero-id ": " tiempo-muerto "
segundos"))); Imprime el identificador y el tiempo muerto acumulado para
cada crucero

    (println "\nTiempo que se tardo el programa: " (convertir-tiempo
tiempo-total)); Imprime el tiempo total de ejecución del programa
convertido a segundos

    (loop []; Inicia un bucle para que el usuario pueda pedir varias
veces
        (let [crucero-id (solicitar-crucero-id)]; Solicita al usuario un
identificador de crucero
            (if (not= crucero-id 0); Si el identificador no es 0
                (do

```

```
(mostrar-estadisticas crucero-id); Muestra las estadísticas
para el crucero-id
(recur)); Repite el bucle
(println "Saliendo del programa..."))))))); Si el identificador
es 0, imprime un mensaje de salida del programa

(defn -main [& _] ;LLama a main
  (main))
```

## Estrategia de testing

La estrategia de testing que se llevará a cabo será ejecutar el programa para ver que las estadísticas generales si funcionen correctamente, para comprobar esto se puede pedir al programa que te muestre los resultados en cuestión y comprobar que las estadísticas tengan sentido y sean acordes, además de ver los archivos de resultados y ver que también tengan una relación, cosa que se muestra a continuación y si tiene sentido.



# Evidencia de testing

## Foto 1

```
259 (let [crucero-id (solicitar-crucero-id)]; Solicita al usuario un identificador de
260 (if (not= crucero-id 0); Si el identificador no es 0

PROBLEMS  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\diego\IMECO\Clojure\prueba> lein run

---Estadisticas Generales---

Tiempo promedio de espera en la totalidad de los vehiculos de la simulacion: 181387/3200

Top 4 cruceros con mayor tiempo de espera promedio:
Crucero 25: 723/10
Crucero 33: 1469/20
Crucero 24: 3109/40
Crucero 5: 1591/20

Top 4 cruceros con menor tiempo de espera promedio:
Crucero 1: 1467/80
Crucero 15: 1823/80
Crucero 37: 3001/80
Crucero 40: 1513/40
```

## Foto 2

```
231 tiempo-inicial (System/nanoTime); Obtiene el tiempo inicial en nanosegundos

PROBLEMS  DEBUG CONSOLE  TERMINAL  PORTS

Top 4 cruceros con mayor cantidad de tiempo muerto acumulado:
Crucero 34: 160 segundos
Crucero 35: 162 segundos
Crucero 29: 167 segundos
Crucero 37: 205 segundos

Top 4 cruceros con menor cantidad de tiempo muerto acumulado:
Crucero 3: 0 segundos
Crucero 5: 0 segundos
Crucero 6: 0 segundos
Crucero 8: 0 segundos

Tiempo que se tardo el programa: 0.10419 segundos

---Estadisticas de Crucero Especifico---

Si quieres ver la estadistica de un crucero, pon su identificador (0 para salir):

```

## Foto 3

```
251 tiempoInicial (System.nanoTime()); Obtiene el tiempo inicial en nanosegundos

PROBLEMS  DEBUG CONSOLE  TERMINAL  PORTS  java + - [ ] [ ] ... ^ x

---Estadísticas de Crucero Especifico---

Si quieres ver la estadística de un crucero, pon su identificador (0 para salir):
1
Cantidad de coches que pasaron en el crucero: 862
Cantidad de coches que pasaron por semáforo y carril: {0 {0 [(28 14 8 1)], 1 [(6 10 20 22)], 2 [(7 4 8 18)], 3 [(18 36 29 21)], 4 [(55 49 34 27)]}, 1 {0 [(31 16 9 2)], 1 [(9 7 1 8 21)], 2 [(14 7 5 16)], 3 [(24 9 32 24)], 4 [(29 25 10 3)]}, 2 {0 [(27 13 7 0)], 1 [(7 11 21 23)], 2 [(8 5 9 19)], 3 [(17 35 28 20)], 4 [(52 48 33 26)]}, 3 {0 [(24 10 4 7)], 1 [(10 14 24 26)], 2 [(7 8 12 22)], 3 [(17 32 25 17)], 4 [(22 18 3 30)]}}
Cantidad total de veces en el crucero que un semáforo estuvo en verde y no paso ningún vehículo: 2
Veces de tiempo muerto por semáforo y carril: {0 {3 1}, 2 {3 1}}
Tiempo promedio total de los coches en el crucero: 1467/80
Tiempo promedio por semáforo y carril: {0 {0 51/4, 1 29/2, 2 37/4, 3 26, 4 165/4}, 1 {0 29/2, 1 55/4, 2 21/2, 3 89/4, 4 67/4}, 2 {0 47/4, 1 31/2, 2 41/4, 3 25, 4 159/4}, 3 {0 45/4, 1 37/2, 2 49/4, 3 91/4, 4 73/4}}

---Estadísticas de Crucero Especifico---

Si quieres ver la estadística de un crucero, pon su identificador (0 para salir):
[ ]
```

## Foto 4

```
PROBLEMS  DEBUG CONSOLE  TERMINAL  PORTS  java + - [ ] [ ] ... ^ x

---Estadísticas de Crucero Especifico---

Si quieres ver la estadística de un crucero, pon su identificador (0 para salir):
3
Cantidad de coches que pasaron en el crucero: 540
Cantidad de coches que pasaron por semáforo y carril: {0 {0 [(53 39 25 11)], 1 [(67 53 39 25)], 2 [(76 62 48 34)], 3 [(88 74 60 46)], 4 [(100 86 72 58)]}, 1 {0 [(65 49 33 17)], 1 [(81 65 49 33)], 2 [(92 76 60 44)], 3 [(106 90 74 58)], 4 [(120 104 88 72)]}, 2 {0 [(70 52 34 16)], 1 [(88 70 52 34)], 2 [(101 83 65 47)], 3 [(117 99 81 63)], 4 [(133 115 97 79)]}, 3 {0 [(63 47 31 15)], 1 [(16 21 5 11)], 2 [(69 53 37 21)], 3 [(51 35 19 3)], 4 [(58 42 26 10)]}}
Cantidad total de veces en el crucero que un semáforo estuvo en verde y no paso ningún vehículo: 0
Veces de tiempo muerto por semáforo y carril: nil
Tiempo promedio total de los coches en el crucero: 4621/80
Tiempo promedio por semáforo y carril: {0 {0 32, 1 46, 2 55, 3 67, 4 79}, 1 {0 41, 1 57, 2 68, 3 82, 4 96}, 2 {0 43, 1 61, 2 74, 3 90, 4 106}, 3 {0 39, 1 53/4, 2 45, 3 27, 4 34}}

---Estadísticas de Crucero Especifico---

Si quieres ver la estadística de un crucero, pon su identificador (0 para salir):
[ ]
```

## Foto 5

```
PROBLEMS  DEBUG CONSOLE  TERMINAL  PORTS  java + - [ ] [ ] ... ^ x

---Estadísticas de Crucero Especifico---

Si quieres ver la estadística de un crucero, pon su identificador (0 para salir):
37
Cantidad de coches que pasaron en el crucero: 850
Cantidad de coches que pasaron por semáforo y carril: {0 {0 [(55 41 27 13)], 1 [(69 55 41 27)], 2 [(78 64 50 36)], 3 [(63 49 35 21)], 4 [(41 27 13 60)]}, 1 {0 [(54 40 26 12)], 1 [(68 54 40 26)], 2 [(77 63 49 35)], 3 [(54 40 26 12)], 4 [(21 7 73 59)]}, 2 {0 [(9 53 35 17)], 1 [(24 6 20 2)], 2 [(70 52 34 16)], 3 [(84 66 48 30)], 4 [(72 54 36 18)]}, 3 {0 [(54 40 26 12)], 1 [(34 20 6 8)], 2 [(45 31 17 3)], 3 [(52 38 24 10)], 4 [(39 25 11 59)]}}
Cantidad total de veces en el crucero que un semáforo estuvo en verde y no paso ningún vehículo: 4
Veces de tiempo muerto por semáforo y carril: {0 {4 1}, 1 {4 1}, 2 {0 1}, 3 {4 1}}
Tiempo promedio total de los coches en el crucero: 3001/80
Tiempo promedio por semáforo y carril: {0 {0 34, 1 48, 2 57, 3 42, 4 141/4}, 1 {0 33, 1 47, 2 56, 3 33, 4 40}, 2 {0 57/2, 1 13, 2 43, 3 57, 4 45}, 3 {0 33, 1 17, 2 24, 3 31, 4 67/2}}

---Estadísticas de Crucero Especifico---

Si quieres ver la estadística de un crucero, pon su identificador (0 para salir):
[ ]
```

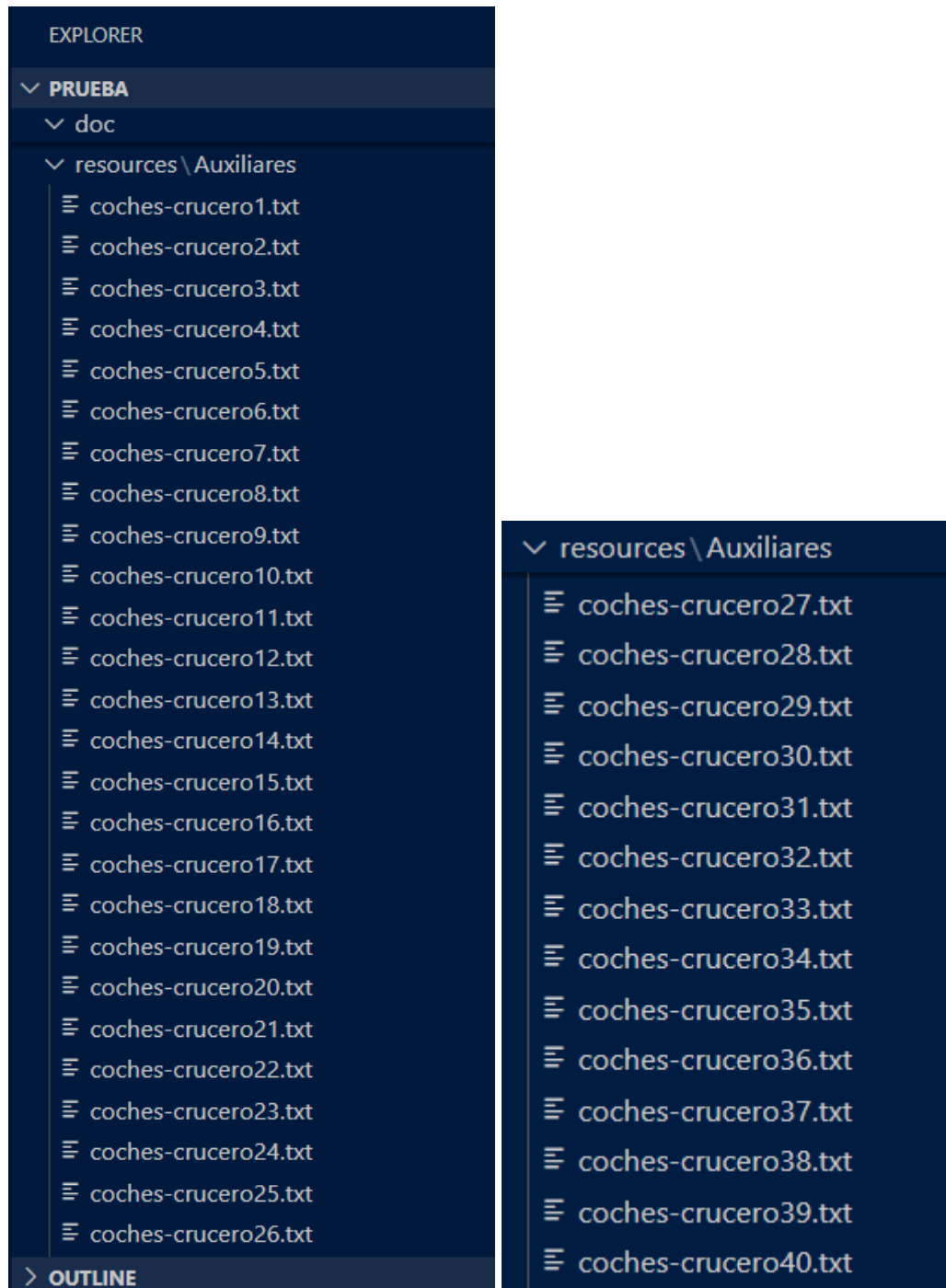
## Foto 6

```
---Estadísticas de Crucero Especifico---

Si quieres ver la estadística de un crucero, pon su identificador (0 para salir):
0
Saliendo del programa...
PS C:\Users\diego\IMECO\Clojure\prueba> [ ]
```



## Archivos de llegada de coches por crucero



Estos archivos son en los que está la información de cada uno de los 40 cruceros, estos como se puede ver, se llaman igual, pero al final tienen un identificador distinto, por lo que en el código se utilizará ese último número para acceder a la información correspondiente, aquí están algunos ejemplos de cómo se ven los archivos de coches.

# Ejemplos de archivos de coches

## Crucero 9

```
core.dj  coches-crucero9.txt
resources > Auxiliares > coches-crucero9.txt
1 (1(5 21 37 53 69 85 101 117 133 149 165 181 197 213 229 245 261 277 293 309 325 341 357 373 389 405 421 437 453 469 485 501 517 533 549 565 581 597 613 629 645 661 677 693 709 725 741 757 773 789 805 821 837 853 869 885 901 917 933 949 965 981 997)))(2(1 17 33 49 65 81 97 113 129 145 161 177 193 209 225 241 257 273 289 305 321 337 353 369 385 401 417 433 449 465 481 497 513 529 545 561 577 593 609 625 641 657 673 689 705 721 737 753 769 785 801 817 833 849 865 881 897 913 929 945 961 977 993)))(3(2 19 36 53 70 87 104 121 138 155 172 189 206 223 240 257 274 291 308 325 342 359 376 393 410 427 444 461 478 495 512 529 546 563 580 597 614 631 648 665 682 699 716 733 750 767 784 801 818 835 852 869 886 903 920 937 954 971 988)))(4(6 24 42 60 78 96 114 132 150 168 186 204 222 240 258 276 294 312 330 348 366 384 402 420 438 456 474 492 510 528 546 564 582 600 618 636 654 672 690 708 726 744 762 780 798 816 834 852 870 888 906 924 942 960 978 996)))
```

## Crucero 18

```
core.dj  coches-crucero9.txt  coches-crucero18.txt
resources > Auxiliares > coches-crucero18.txt
1 (1(1(4 18 32 46 60 74 88 102 116 130 144 158 172 186 200 214 228 242 256 270 284 298 312 326 340 354 368 382 396 410 424 438 452 466 480 494 508 522 536 550 564 578 592 606 620 634 648 662 676 690 704 718 732 746 760 774 788 802 816 830 844 858 872 886 900 914 928 942 956 970 984 998)))(2(1 21 41 61 81 101 121 141 161 181 201 221 241 261 281 301 321 341 361 381 401 421 441 461 481 501 521 541 561 581 601 621 641 661 681 701 721 741 761 781 801 821 841 861 881 901 921 941 961 981 999)))(3(1 17 33 49 65 81 97 113 129 145 161 177 193 209 225 241 257 273 289 305 321 337 353 369 385 401 417 433 449 465 481 497 513 529 545 561 577 593 609 625 641 657 673 689 705 721 737 753 769 785 801 817 833 849 865 881 897 913 929 945 961 977 993)))(4(3 13 23 33 43 53 63 73 83 93 103 113 123 133 143 153 163 173 183 193 203 213 223 233 243 253 263 273 283 293 303 313 323 333 343 353 363 373 383 393 403 413 423 433 443 453 463 473 483 493 503 513 523 533 543 553 563 573 583 593 603 613 623 633 643 653 663 673 683 693 703 713 723 733 743 753 763 773 783 793 803 813 823 833 843 853 863 873 883 893 903 913 923 933 943 953 963 973 983 993)))
```

## Crucero 29

```
core.dj  coches-crucero9.txt  coches-crucero29.txt
resources > Auxiliares > coches-crucero29.txt
1 (1(1(3 17 31 45 59 73 87 101 115 129 143 157 171 185 199 213 227 241 255 269 283 297 311 325 339 353 367 381 395 409 423 437 451 465 479 493 507 521 535 549 563 577 591 605 619 633 647 661 675 689 703 717 731 745 759 773 787 801 815 829 843 857 871 885 899 913 927 941 955 969 983 997)))(2(1 22 43 64 85 106 127 148 169 190 211 232 253 274 295 316 337 358 379 400 421 442 463 484 505 526 547 568 589 610 631 652 673 694 715 736 757 778 799 820 841 862 883 904 925 946 967 988 999)))(3(1 17 33 49 65 81 97 113 129 145 161 177 193 209 225 241 257 273 289 305 321 337 353 369 385 401 417 433 449 465 481 497 513 529 545 561 577 593 609 625 641 657 673 689 705 721 737 753 769 785 801 817 833 849 865 881 897 913 929 945 961 977 993)))(4(1 17 33 49 65 81 97 113 129 145 161 177 193 209 225 241 257 273 289 305 321 337 353 369 385 401 417 433 449 465 481 497 513 529 545 561 577 593 609 625 641 657 673 689 705 721 737 753 769 785 801 817 833 849 865 881 897 913 929 945 961 977 993)))(5(6 26 46 66 86 106 126 146 166 186 206 226 246 266 286 306 326 346 366 386 406 426 446 466 486 506 526 546 566 586 606 626 646 666 686 706 726 746 766 786 806 826 846 866 886 906 926 946 966 986 999)))
```

# Código de estos

## Crucero 9

```
(1(5 21 37 53 69 85 101 117 133 149 165 181 197 213 229 245 261 277 293
309 325 341 357 373 389 405 421 437 453 469 485 501 517 533 549 565 581
597 613 629 645 661 677 693 709 725 741 757 773 789 805 821 837 853 869
885 901 917 933 949 965 981 997)))(2(14 26 38 50 62 74 86 98 110 122 134
146 158 170 182 194 206 218 230 242 254 266 278 290 302 314 326 338 350
362 374 386 398 410 422 434 446 458 470 482 494 506 518 530 542 554 566
578 590 602 614 626 638 650 662 674 686 698 710 722 734 746 758 770 782
794 806 818 830 842 854 866 878 890 902 914 926 938 950 962 974 986
998)))(3(2 19 36 53 70 87 104 121 138 155 172 189 206 223 240 257 274 291 308
325 342 359 376 393 410 427 444 461 478 495 512 529 546 563 580 597 614
631 648 665 682 699 716 733 750 767 784 801 818 835 852 869 886 903 920
937 954 971 988)))(4(6 24 42 60 78 96 114 132 150 168 186 204 222 240 258 276 294 312 330 348 366 384 402 420 438 456 474 492 510 528 546 564 582 600 618 636 654 672 690 708 726 744 762 780 798 816 834 852 870 888 906 924 942 960 978 996)))
```

343 353 363 373 383 393 403 413 423 433 443 453 463 473 483 493 503 513  
523 533 543 553 563 573 583 593 603 613 623 633 643 653 663 673 683 693  
703 713 723 733 743 753 763 773 783 793 803 813 823 833 843 853 863 873  
883 893 903 913 923 933 943 953 963 973 983 993)())  
(4(6 24 42 60 78 96 114 132 150 168 186 204 222 240 258 276 294 312 330  
348 366 384 402 420 438 456 474 492 510 528 546 564 582 600 618 636 654  
672 690 708 726 744 762 780 798 816 834 852 870 888 906 924 942 960 978  
996)(4 18 32 46 60 74 88 102 116 130 144 158 172 186 200 214 228 242 256  
270 284 298 312 326 340 354 368 382 396 410 424 438 452 466 480 494 508  
522 536 550 564 578 592 606 620 634 648 662 676 690 704 718 732 746 760  
774 788 802 816 830 844 858 872 886 900 914 928 942 956 970 984 998)(7  
23 39 55 71 87 103 119 135 151 167 183 199 215 231 247 263 279 295 311  
327 343 359 375 391 407 423 439 455 471 487 503 519 535 551 567 583 599  
615 631 647 663 679 695 711 727 743 759 775 791 807 823 839 855 871 887  
903 919 935 951 967 983 999)())

## Crucero 18

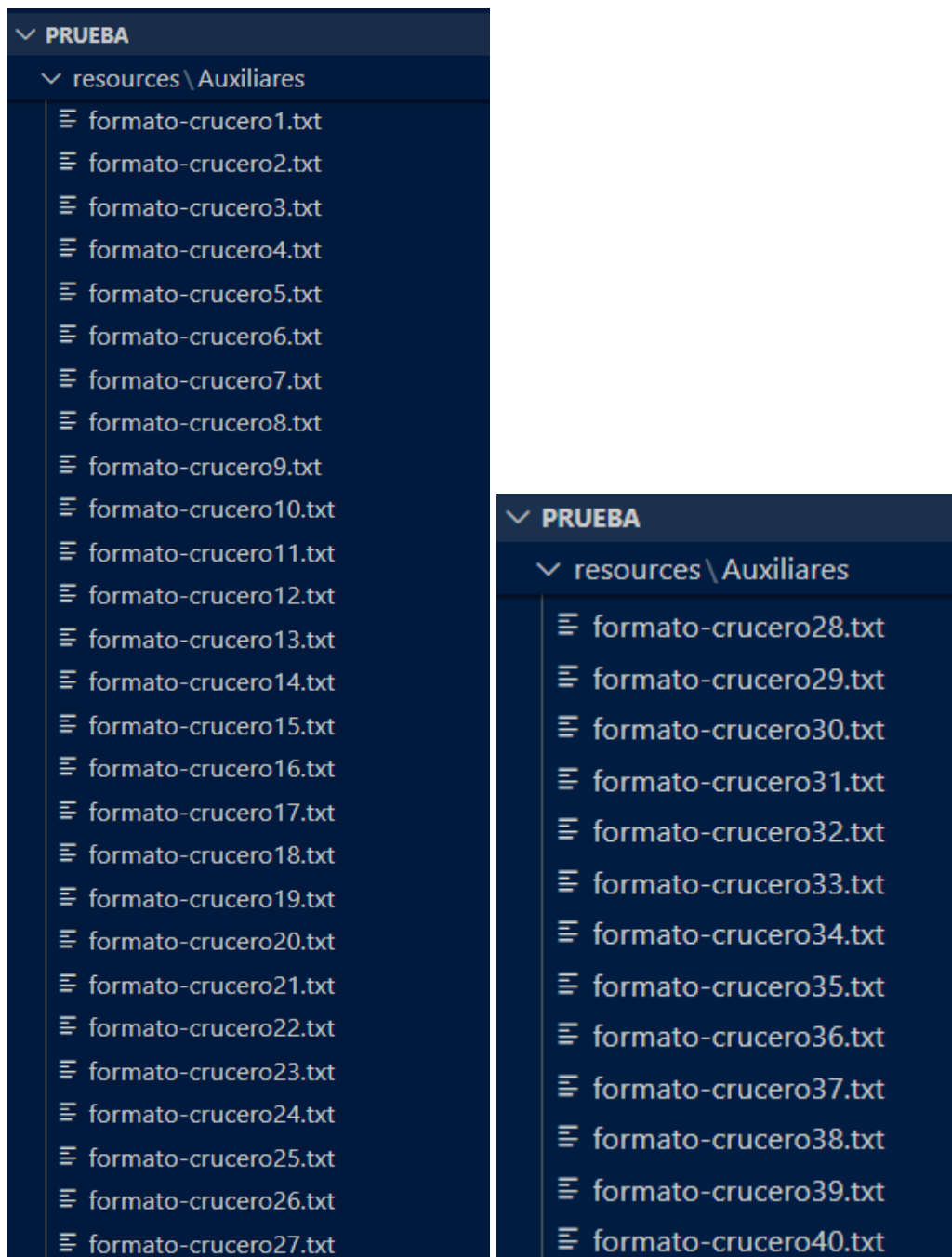
(1())()(4 18 32 46 60 74 88 102 116 130 144 158 172 186 200 214 228 242  
256 270 284 298 312 326 340 354 368 382 396 410 424 438 452 466 480 494  
508 522 536 550 564 578 592 606 620 634 648 662 676 690 704 718 732 746  
760 774 788 802 816 830 844 858 872 886 900 914 928 942 956 970 984  
998)(6 20 34 48 62 76 90 104 118 132 146 160 174 188 202 216 230 244 258  
272 286 300 314 328 342 356 370 384 398 412 426 440 454 468 482 496 510  
524 538 552 566 580 594 608 622 636 650 664 678 692 706 720 734 748 762  
776 790 804 818 832 846 860 874 888 902 916 930 944 958 972 986 1000))  
(2(1 21 41 61 81 101 121 141 161 181 201 221 241 261 281 301 321 341 361  
381 401 421 441 461 481 501 521 541 561 581 601 621 641 661 681 701 721  
741 761 781 801 821 841 861 881 901 921 941 961 981)(3 15 27 39 51 63 75  
87 99 111 123 135 147 159 171 183 195 207 219 231 243 255 267 279 291  
303 315 327 339 351 363 375 387 399 411 423 435 447 459 471 483 495 507  
519 531 543 555 567 579 591 603 615 627 639 651 663 675 687 699 711 723  
735 747 759 771 783 795 807 819 831 843 855 867 879 891 903 915 927 939  
951 963 975 987 999)(5 25 45 65 85 105 125 145 165 185 205 225 245 265  
285 305 325 345 365 385 405 425 445 465 485 505 525 545 565 585 605 625  
645 665 685 705 725 745 765 785 805 825 845 865 885 905 925 945 965  
985)())()  
(3())()(2 19 36 53 70 87 104 121 138 155 172 189 206 223 240 257 274  
291 308 325 342 359 376 393 410 427 444 461 478 495 512 529 546 563 580  
597 614 631 648 665 682 699 716 733 750 767 784 801 818 835 852 869 886  
903 920 937 954 971 988 1005))  
(4(3 13 23 33 43 53 63 73 83 93 103 113 123 133 143 153 163 173 183 193  
203 213 223 233 243 253 263 273 283 293 303 313 323 333 343 353 363 373  
383 393 403 413 423 433 443 453 463 473 483 493 503 513 523 533 543 553  
563 573 583 593 603 613 623 633 643 653 663 673 683 693 703 713 723 733  
743 753 763 773 783 793 803 813 823 833 843 853 863 873 883 893 903 913

923 933 943 953 963 973 983 993)(4 24 44 64 84 104 124 144 164 184 204  
224 244 264 284 304 324 344 364 384 404 424 444 464 484 504 524 544 564  
584 604 624 644 664 684 704 724 744 764 784 804 824 844 864 884 904 924  
944 964 984 1004)())

## Crucero 29

(1())(3 17 31 45 59 73 87 101 115 129 143 157 171 185 199 213 227 241  
255 269 283 297 311 325 339 353 367 381 395 409 423 437 451 465 479 493  
507 521 535 549 563 577 591 605 619 633 647 661 675 689 703 717 731 745  
759 773 787 801 815 829 843 857 871 885 899 913 927 941 955 969 983  
997)(4 20 36 52 68 84 100 116 132 148 164 180 196 212 228 244 260 276  
292 308 324 340 356 372 388 404 420 436 452 468 484 500 516 532 548 564  
580 596 612 628 644 660 676 692 708 724 740 756 772 788 804 820 836 852  
868 884 900 916 932 948 964 980 996))  
(2(1 22 43 64 85 106 127 148 169 190 211 232 253 274 295 316 337 358 379  
400 421 442 463 484 505 526 547 568 589 610 631 652 673 694 715 736 757  
778 799 820 841 862 883 904 925 946 967 988)(2 19 36 53 70 87 104 121  
138 155 172 189 206 223 240 257 274 291 308 325 342 359 376 393 410 427  
444 461 478 495 512 529 546 563 580 597 614 631 648 665 682 699 716 733  
750 767 784 801 818 835 852 869 886 903 920 937 954 971 988)())  
(3(3 19 35 51 67 83 99 115 131 147 163 179 195 211 227 243 259 275 291  
307 323 339 355 371 387 403 419 435 451 467 483 499 515 531 547 563 579  
595 611 627 643 659 675 691 707 723 739 755 771 787 803 819 835 851 867  
883 899 915 931 947 963 979 995)(5 25 45 65 85 105 125 145 165 185 205  
225 245 265 285 305 325 345 365 385 405 425 445 465 485 505 525 545 565  
585 605 625 645 665 685 705 725 745 765 785 805 825 845 865 885 905 925  
945 965 985 1005)())  
(4())(6 26 46 66 86 106 126 146 166 186 206 226 246 266 286 306 326  
346 366 386 406 426 446 466 486 506 526 546 566 586 606 626 646 666 686  
706 726 746 766 786 806 826 846 866 886 906 926 946 966 986 1006))

## Archivos de lógica de los cruceros



Estos archivos son los que contienen la lógica de cada uno de estos cruceros, en este cuando hay carriles que no existen, se puede representar con todo 0, en los tiempos de semáforo rojo, verde y amarillo o simplemente con una lista vacía, depende solo de gustos, pero funciona exactamente igual, esta información es la que se accede en el código para comprobar si los coches que llegan si pueden pasar al momento o si tienen que esperar.

## Ejemplos de archivos de formato de los cruceros

### Crucero 9

```
core.cj  coches-crucero9.txt  coches-crucero29.txt  formato-crucero29.txt  formato-crucero9.txt X
resources > Auxiliares > formato-crucero9.txt
1  (1 (1 (2 (5 (1 (38 22 3)) (2 (0 0 0)) (3 (0 0 0)) (4 (50 10 3)) (5 (0 0 0))))))
2  (2 (1 (2 (5 (1 (0 0 0)) (2 (0 0 0)) (3 (0 0 0)) (4 (0 0 0)) (5 (42 18 3))))))
3  (3 (1 (4 (5 (1 (45 15 3)) (2 (35 25 3)) (3 (40 20 3)) (4 (0 0 0)) (5 (0 0 0))))))
4  (4 (1 (4 (5 (1 (30 30 3)) (2 (50 10 3)) (3 (37 23 3)) (4 (0 0 0)) (5 (48 12 3))))))
```

### Crucero 18

```
core.cj  coches-crucero9.txt  coches-crucero29.txt  formato-crucero29.txt  formato-crucero18.txt X
resources > Auxiliares > formato-crucero18.txt
1  (1 (1 (2 (5 (1 (0 0 0)) (2 (0 0 0)) (3 (0 0 0)) (4 (50 10 3)) (5 (0 0 0))))))
2  (2 (1 (2 (5 (1 (35 25 3)) (2 (45 15 3)) (3 (40 20 3)) (4 (0 0 0)) (5 (0 0 0))))))
3  (3 (1 (2 (5 (1 (0 0 0)) (2 (0 0 0)) (3 (0 0 0)) (4 (0 0 0)) (5 (42 18 3))))))
4  (4 (1 (2 (5 (1 (30 30 3)) (2 (0 0 0)) (3 (50 10 3)) (4 (40 20 3)) (5 (0 0 0))))))
```

### Crucero 29

```
core.cj  coches-crucero9.txt  coches-crucero29.txt  formato-crucero29.txt X
resources > Auxiliares > formato-crucero29.txt
1  (1 (1 (2 (5 (1 (0 0 0)) (2 (0 0 0)) (3 (33 29 3)) (4 (32 30 3)) (5 (0 0 0))))))
2  (2 (1 (2 (5 (1 (34 28 3)) (2 (33 29 3)) (3 (0 0 0)) (4 (0 0 0)) (5 (0 0 0))))))
3  (3 (1 (2 (5 (1 (35 27 3)) (2 (0 0 0)) (3 (0 0 0)) (4 (32 30 3)) (5 (31 31 3))))))
4  (4 (1 (2 (5 (1 (0 0 0)) (2 (0 0 0)) (3 (0 0 0)) (4 (36 26 3)) (5 (37 25 3))))))
```

## Código de estos

### Crucero 9

(1 (1 (2 (5 (1 (38 22 3)) (2 (0 0 0)) (3 (0 0 0)) (4 (50 10 3)) (5 (0 0 0)))))  
(2 (1 (2 (5 (1 (0 0 0)) (2 (0 0 0)) (3 (0 0 0)) (4 (0 0 0)) (5 (42 18 3)))))  
(3 (1 (4 (5 (1 (45 15 3)) (2 (35 25 3)) (3 (40 20 3)) (4 (0 0 0)) (5 (0 0 0)))))  
(4 (1 (4 (5 (1 (30 30 3)) (2 (50 10 3)) (3 (37 23 3)) (4 (0 0 0)) (5 (48 12 3)))))

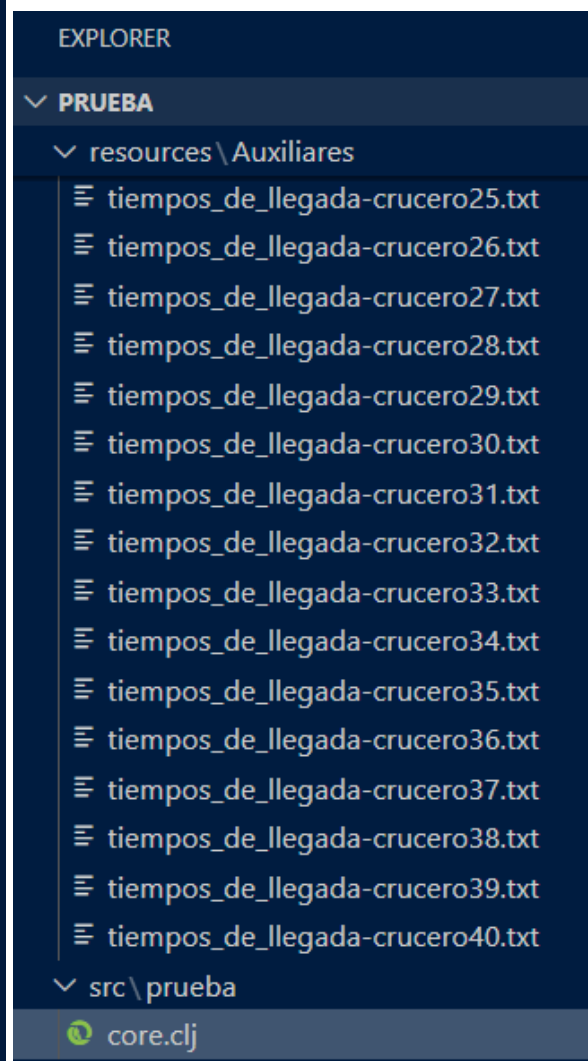
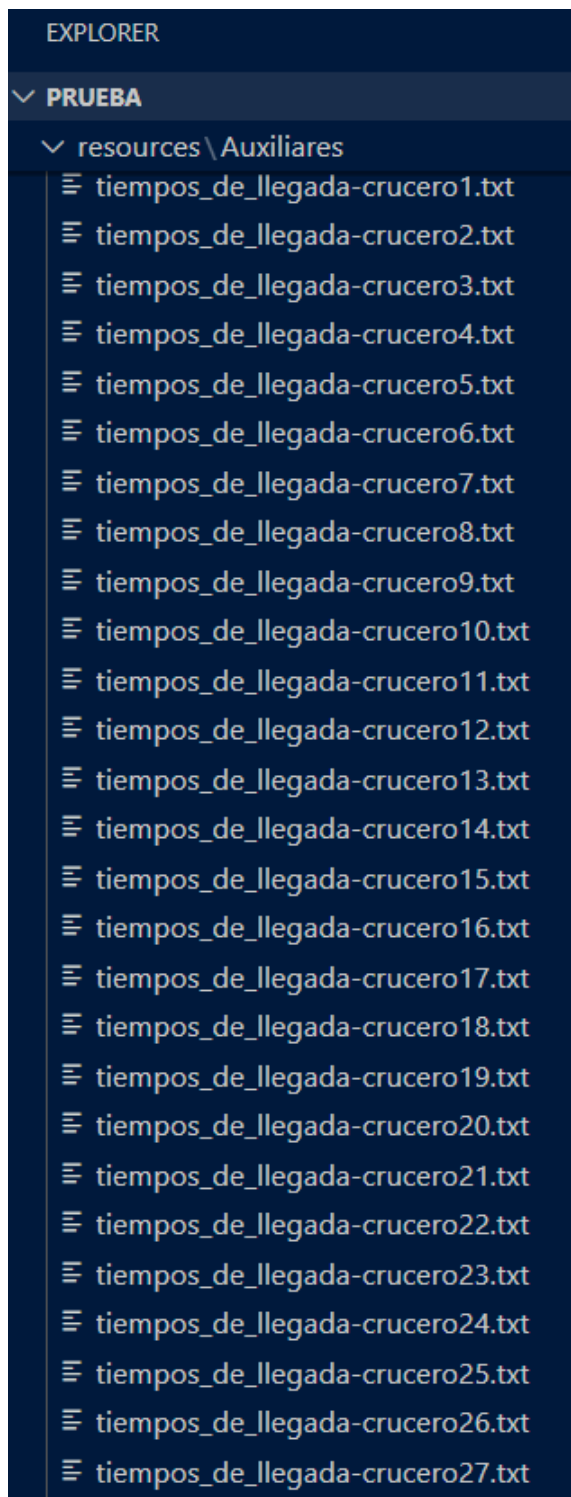
### Crucero 18

(1 (1 (2 (5 (1 (0 0 0)) (2 (0 0 0)) (3 (0 0 0)) (4 (50 10 3)) (5 (0 0 0)))))  
(2 (1 (2 (5 (1 (35 25 3)) (2 (45 15 3)) (3 (40 20 3)) (4 (0 0 0)) (5 (0 0 0)))))  
(3 (1 (2 (5 (1 (0 0 0)) (2 (0 0 0)) (3 (0 0 0)) (4 (0 0 0)) (5 (42 18 3)))))  
(4 (1 (2 (5 (1 (30 30 3)) (2 (0 0 0)) (3 (50 10 3)) (4 (40 20 3)) (5 (0 0 0)))))

### Crucero 29

(1 (1 (2 (5 (1 (0 0 0)) (2 (0 0 0)) (3 (33 29 3)) (4 (32 30 3)) (5 (0 0 0)))))  
(2 (1 (2 (5 (1 (34 28 3)) (2 (33 29 3)) (3 (0 0 0)) (4 (0 0 0)) (5 (0 0 0)))))  
(3 (1 (2 (5 (1 (35 27 3)) (2 (0 0 0)) (3 (0 0 0)) (4 (32 30 3)) (5 (31 31 3)))))  
(4 (1 (2 (5 (1 (0 0 0)) (2 (0 0 0)) (3 (0 0 0)) (4 (36 26 3)) (5 (37 25 3)))))

## Archivo de captura de resultados por crucero





## Ejemplos de archivos de formato de los cruceros

### Crucero 9

```
core.cj  tiempos_de_llegada-crucero9.txt X
resources > Auxiliares > tiempos_de_llegada-crucero9.txt
1 {0 {0 [(73 55 37 19)], 1 [(91 73 55 37)], 2 [(104 86 68 50)], 3 [(120 102 84 66)], 4 [(136 118 100 82)]}, 1 {0 [(69 51 33 15)], 1 [(87 69 51 33)], 2 [(100 82 64 46)], 3 [(116 98 80 62)],
```

### Crucero 18

```
core.cj  tiempos_de_llegada-crucero9.txt  tiempos_de_llegada-crucero18.txt X
resources > Auxiliares > tiempos_de_llegada-crucero18.txt
1 {0 {0 [(64 48 32 16)], 1 [(80 64 48 32)], 2 [(91 75 59 43)], 3 [(105 89 73 57)], 4 [(119 103 87 71)]}, 1 {0 [(25 63 41 19)], 1 [(44 22 18 4)], 2 [(106 84 62 40)], 3 [(101 79 57 35)], 4 [(116 98 80 62)],
```

### Crucero 29

```
core.cj  tiempos_de_llegada-crucero9.txt  tiempos_de_llegada-crucero18.txt  tiempos_de_llegada-crucero29.txt X
resources > Auxiliares > tiempos_de_llegada-crucero29.txt
1 {0 {0 [(29 13 4 12)], 1 [(40 24 8 28)], 2 [(27 11 57 41)], 3 [(42 26 10 27)], 4 [(83 67 51 35)]}, 1 {0 [(27 66 43 20)], 1 [(47 24 33 10)], 2 [(98 75 52 29)], 3 [(144 121 98 75)], 4 [(16 9 1 1)],
```

## Código de estos

### Crucero 9

```
{0 {0 [(73 55 37 19)], 1 [(91 73 55 37)], 2 [(104 86 68 50)], 3 [(120 102 84 66)], 4 [(136 118 100 82)]}, 1 {0 [(69 51 33 15)], 1 [(87 69 51 33)], 2 [(100 82 64 46)], 3 [(116 98 80 62)], 4 [(132 114 96 78)]}, 2 {0 [(14 55 36 17)], 1 [(30 11 20 1)], 2 [(79 60 41 22)], 3 [(81 62 43 24)], 4 [(81 62 43 24)]}, 3 {0 [(22 62 42 22)], 1 [(39 19 12 8)], 2 [(104 84 64 44)], 3 [(95 75 55 35)], 4 [(93 73 53 33)]}}
```

### Crucero 18

```
{0 {0 [(73 55 37 19)], 1 [(91 73 55 37)], 2 [(104 86 68 50)], 3 [(120 102 84 66)], 4 [(136 118 100 82)]}, 1 {0 [(69 51 33 15)], 1 [(87 69 51 33)], 2 [(100 82 64 46)], 3 [(116 98 80 62)], 4 [(132 114 96 78)]}, 2 {0 [(14 55 36 17)], 1 [(30 11 20 1)], 2 [(79 60 41 22)], 3 [(81 62 43 24)], 4 [(81 62 43 24)]}, 3 {0 [(22 62 42 22)], 1 [(39 19 12 8)], 2 [(104 84 64 44)], 3 [(95 75 55 35)], 4 [(93 73 53 33)]}}
```

## Crucero 29

{0 {0 [(29 13 4 12)], 1 [(40 24 8 28)], 2 [(27 11 57 41)], 3 [(42 26 10 27)], 4 [(83 67 51 35)]}, 1 {0 [(27 66 43 20)], 1 [(47 24 33 10)], 2 [(98 75 52 29)], 3 [(144 121 98 75)], 4 [(163 140 117 94)]}, 2 {0 [(64 46 28 10)], 1 [(50 32 14 32)], 2 [(39 21 3 47)], 3 [(56 38 20 2)], 4 [(99 81 63 45)]}, 3 {0 [(27 5 12 10)], 1 [(78 56 34 12)], 2 [(87 65 43 21)], 3 [(85 63 41 19)], 4 [(107 85 63 41)]}}

# Explicaciones de funcionamiento del código

## Funcionamiento de cómo se accede a los cruceros con base en su id

```
(defn main []; Define la función principal main
  (let [cruceros-ids (range 1 41); Crea una lista de identificadores de cruceros del 1 al 40
        tiempo-inicial (System/nanoTime); Obtiene el tiempo inicial en nanosegundos
        tiempos-promedios (doall (pmap procesar-crucero cruceros-ids)); Procesa todos los cruceros en paralelo usando pmap
        tiempo-final (System/nanoTime); Obtiene el tiempo final en nanosegundos
        tiempos-promedios-ordenados (sort-by :promedio tiempos-promedios); Ordena los tiempos-promedios por el campo :promedio
        tiempos-muertos-ordenados (sort-by :tiempo-muerto tiempos-promedios); Ordena los tiempos-promedios por el campo :tiempo-muerto
        tiempo-total (- tiempo-final tiempo-inicial)]; Calcula el tiempo total de ejecución restando tiempo-inicial de tiempo-final
    (println "\n---Estadísticas Generales---\n"); Imprime el encabezado de estadísticas generales
    (println "Tiempo promedio de espera en la totalidad de los vehículos de la simulación:"
      (/ (reduce + (map :promedio tiempos-promedios)) (count tiempos-promedios))); Calcula e imprime el tiempo promedio
    (println "\nTop 4 cruceros con mayor tiempo de espera promedio:"); Imprime el encabezado para el top 4 de cruceros con mayor tiempo de espera promedio
    (doseq [{:keys [crucero-id promedio]} (take-last 4 tiempos-promedios-ordenados)]; Itera sobre los últimos 4 elementos de tiempos-promedios-ordenados
      (println (str "Crucero " crucero-id ": " promedio))); Imprime el identificador y el promedio de tiempo de espera promedio
    (println "\nTop 4 cruceros con menor tiempo de espera promedio:"); Imprime el encabezado para el top 4 de cruceros con menor tiempo de espera promedio
    (doseq [{:keys [crucero-id promedio]} (take 4 tiempos-promedios-ordenados)]; Itera sobre los primeros 4 elementos de tiempos-promedios-ordenados
      (println (str "Crucero " crucero-id ": " promedio))); Imprime el identificador y el promedio de tiempo de espera promedio
    (println "\nTop 4 cruceros con mayor cantidad de tiempo muerto acumulado:"); Imprime el encabezado para el top 4 de cruceros con mayor tiempo muerto acumulado
    (doseq [{:keys [crucero-id tiempo-muerto]} (take-last 4 tiempos-muertos-ordenados)]; Itera sobre los últimos 4 elementos de tiempos-muertos-ordenados
      (println (str "Crucero " crucero-id ": " tiempo-muerto " segundos"))); Imprime el identificador y el tiempo muerto acumulado
    (println "\nTop 4 cruceros con menor cantidad de tiempo muerto acumulado:"); Imprime el encabezado para el top 4 de cruceros con menor tiempo muerto acumulado
    (doseq [{:keys [crucero-id tiempo-muerto]} (take 4 tiempos-muertos-ordenados)]; Itera sobre los primeros 4 elementos de tiempos-muertos-ordenados
      (println (str "Crucero " crucero-id ": " tiempo-muerto " segundos"))); Imprime el identificador y el tiempo muerto acumulado

  (loop []; Inicia un bucle para que el usuario pueda pedir varias veces
    (let [crucero-id (solicitar-crucero-id)]; Solicita al usuario un identificador de crucero
      (if (not= crucero-id 0); Si el identificador no es 0
        (do
          (mostrar-estadisticas crucero-id); Muestra las estadísticas para el crucero-id
          (recur)); Repite el bucle
        (println "Saliendo del programa...")))); Si el identificador es 0, imprime un mensaje de salida del programa
```

En esta foto se puede observar como en el main es donde creo los identificadores para mis cruceros, desde el crucero 1 hasta el 40, que como mostré es la cantidad de cruceros que tengo con sus tiempos de llegada, lógica y resultados, luego en este por medio de la herramienta de keys se revisa cuál coincide con la entrada puesta por el usuario y así mostrar sus estadísticas relevantes.

Y se crea un loop para solicitar al usuario el id del crucero que quiera ver sus estadísticas hasta que ponga 0 para terminar el programa.

Algo importante a notar aquí es que como se puede ver, en esta funciona es la que se hace uso de pmap para que haga todo este proceso de sacar tiempos de llegada y estadísticas de un crucero, pero de manera paralela para el rango que se le dió que en este caso es 40, por lo que procesa de manera paralela, sacando estadísticas, tiempos muertos y tiempos de llegada de manera paralela.

## Funcionamiento de cómo se procesan los cruceros y se muestran estadísticas

```
(defn procesar-crucero [crucero-id] ; Define la función procesar-crucero con el parámetro crucero-id
  (let [[logicas coches] (leer-archivos crucero-id); Lee los archivos de lógica y coches para el crucero-id
        resultados-logicas (procesar-lineas logicas crucero-id); Procesa las líneas de lógica y almacena los resultados en res
        resultados-coches (procesar-lineas-sentido coches crucero-id); Procesa las líneas de coches y almacena los resultados
        archivo-tiempos (str "C:\\Users\\diego\\IMECO\\Clojure\\prueba\\resources\\Auxiliares\\tiempos_de_llegada-crucero" crucero-id ".txt");
        (doseq [i (range (min (count resultados-logicas) (count resultados-coches)))] ; Itera sobre el rango desde 0 hasta el menor
          (let [logica (nth resultados-logicas i); Obtiene la lógica en la posición i de resultados-logicas
                coche (nth resultados-coches i)]; Obtiene el coche en la posición i de resultados-coches
            (procesar-carril coche logica (int (/ i 5)) (mod i 5) crucero-id))); Procesa el carril con los parámetros coche, logica y crucero-id

        (guardar-tiempos-en-archivo archivo-tiempos @tiempos-detallados crucero-id);guarda los tiempos en el archivo correspondiente
        {:crucero-id crucero-id, :promedio (mostrar-promedio-general crucero-id), :tiempo-muerto (contar-tiempo-muerto-acumulado @tiempos-detallados crucero-id)}))

(defn contar-coches-en-archivo [ruta] ; Define la función contar-coches-en-archivo con el parámetro ruta
  (try
    (with-open [rdr (io/reader ruta)]; intenta abrir el archivo especificado por la ruta
      (reduce + 0 (map count (map parsear-linea (line-seq rdr))))); Lee todas las líneas del archivo, parsea cada línea, cuenta y suma
    (catch Exception e
      (println (str "Error al leer el archivo " ruta ": " (.getMessage e))) ; Si sucede un error, imprime un mensaje de error
      0)));y devuelve 0

(defn mostrar-estadisticas [crucero-id]; Define la función mostrar-estadisticas con el parámetro crucero-id
  (let [ruta-coches (str "C:\\Users\\diego\\IMECO\\Clojure\\prueba\\resources\\Auxiliares\\coches-crucero" crucero-id ".txt");
        total-coches (contar-coches-en-archivo ruta-coches)]; Cuenta el total de coches en el de tiempos de llegada y muestra
    (println "Cantidad de coches que pasaron en el crucero:" total-coches)
    (println "Cantidad de coches que pasaron por semaforo y carril:" (get @tiempos-detallados crucero-id))
    (println "Cantidad total de veces en el crucero que un semaforo estuvo en verde y no paso ningun vehiculo:" (contar-tiempo-verde @tiempos-detallados crucero-id))
    (println "Veces de tiempo muerto por semaforo y carril:" (get @tiempos-muertos-detallado crucero-id))
    (println "Tiempo promedio total de los coches en el crucero:" (mostrar-promedio-general crucero-id))
    (println "Tiempo promedio por semaforo y carril:" (calcular-promedio-por-semaforo-carril @tiempos-detallados crucero-id)))

(defn solicitar-crucero-id []; Define la función solicitar-crucero-id
  (println "\n---Estadísticas de Crucero Especifico---\n")
  (println "Si quieres ver la estadística de un crucero, pon su identificador (0 para salir:"); Solicita al usuario que ingrese un número
  (Integer. (read-line))) ; Lee la entrada del usuario, la convierte a entero y la retorna
```

En esta función de procesar se ve como se toma el id del coche como parámetro para acceder al crucero correspondiente y también en la función de estadística se muestra cómo se accede al archivo de coches que son los tiempos en los que llegaron los coches, pero teniendo como parámetro el id que se proporcionó por el usuario y así acceder al correcto y comprobar cuántos coches tenía.

Por último en solicitar-crucero es donde se le pregunta al usuario que crucero quiere ver sus estadísticas.

## Funcionamiento de promedios y tiempos muertos

```
(defn mostrar-promedio-general [crucero-id]; Define la función mostrar-promedio-general con el parámetro crucero-id
  (let [total-carros (reduce + (map count (@resultados-totales crucero-id))); Calcula el total de carros sumando los resultados de la siguiente expresión
        total-tiempo (reduce + (map #(reduce + %) (@resultados-totales crucero-id))); Calcula el total de tiempos muertos sumando los resultados de la siguiente expresión
        (if (zero? total-carros); Verifica si el total de carros es cero
            (throw (ArithmeticException. "No se puede dividir por cero: No hay carros")); Si es cero, lanza una excepción
            (/ total-tiempo total-carros)))] ; Si no es cero, calcula y retorna el promedio dividiendo total-tiempo por total-carros)

(defn calcular-promedio-por-semaforo-carril [tiempos-detallados crucero-id]; Define la función calcular-promedio-por-semaforo-carril
  (into {} ; Convierte el resultado de la siguiente expresión en un mapa
    (for [[sem carriles] (get tiempos-detallados crucero-id)] ; Itera sobre cada par [sem carriles] en el mapa de tiempos detallados
      [sem (into {} ; Convierte el resultado en un mapa
        (for [[carril tiempos] carriles]; va iterando con los parámetros de carril y sus tiempos
          [carril (if (seq tiempos); Checa si no está vacío
            (/ (reduce + (flatten tiempos)) (count (flatten tiempos))); Calcula el promedio de tiempos muertos por carril
            0)))])))] ; Si está vacío asigna el valor 0

(defn contar-tiempos-muertos-totales [tiempos-muertos-detallado crucero-id]; Define la función contar-tiempos-muertos-totales
  (reduce + 0; Suma todos los valores en el resultado de la siguiente expresión, comenzando desde 0
    (for [_ carriles] (get tiempos-muertos-detallado crucero-id)]; Itera sobre cada par [sem carriles] en el mapa de tiempos muertos detallados
      (reduce + (vals carriles)))] ; Suma todos los valores de carriles (tiempos muertos por carril) y retorna el total

(defn contar-tiempo-muerto-acumulado [tiempo-muerto-acumulado crucero-id]; Define la función contar-tiempo-muerto-acumulado
  (get tiempo-muerto-acumulado crucero-id 0) ; Obtiene el tiempo muerto acumulado para el crucero-id, retornando 0 si no está presente

(defn guardar-tiempos-en-archivo [ruta tiempos-detallados crucero-id]; Define la función guardar-tiempos-en-archivo
  (let [contenido (with-out-str (prn (get tiempos-detallados crucero-id)))] ; Cambia los tiempos para el crucero-id a una cadena de texto
    (spit ruta contenido)); y lo escribe donde corresponde
```

En estas funciones se ve como se muestra la estadística del promedio en concreto que solicite el usuario para ver qué promedio de tiempos tuvieron los coches que quiso ver, al igual que su promedio por cada uno de los carriles y veces que su semáforo estaba en verde y no pasó ningún coche.

Al igual que con base en la estadística de tiempos muertos al inicio del programa se compara todos los tiempos muertos de los cruceros y muestra el top 10% según las estadísticas solicitadas, y la última función que se observa es la manera en la que se guardan los archivos en su archivo de tiempos de llegada correspondiente, iterando sobre el id del crucero.

# Lectura de archivos

```
6 (defn limpiar-linea [linea]; Define la función limpiar-linea que toma una línea como entrada
7   (-> linea; Usa el threading macro (->) para encadenar funciones
8     (str/replace #"[]" "")); Reemplaza todos los paréntesis en la línea con una cadena vacía
9     (str/trim)); Elimina los espacios en blanco iniciales y finales de la línea
10
11 (defn parsear-linea [linea]; Define la función parsear-linea que toma una línea como entrada
12   (if (empty? (limpiar-linea linea)); Si la línea limpiada está vacía
13     []; Retorna una lista vacía
14     (map #(Integer/parseInt %) (str/split (limpiar-linea linea) #" "))); De lo contrario, divide la línea limpiada en tokens usando espacios en t
15
16 (defn leer-archivo [ruta]; Define la función leer-archivo que toma la ruta de un archivo como entrada
17   (try; Intenta ejecutar el siguiente bloque de código
18     (with-open [rdr (io/reader ruta)]; Abre un lector para el archivo especificado por la ruta
19       (doall (map parsear-linea (line-seq rdr)))); Lee todas las líneas del archivo, parsea cada línea y retorna una lista de las líneas parseadas
20     (catch Exception e; Si ocurre una excepción durante la lectura del archivo
21       (println (str "Error al leer el archivo " ruta ": " (.getMessage e)))); Imprime un mensaje de error junto con la ruta del archivo y el mensaje
22       [])); Retorna una lista vacía en caso de error
23
```

En esta parte del código se crearon unas funciones para ajustar los archivos de coches y de lógica de manera que se pueda acceder a la información de manera correcta, eliminando los parámetros no requeridos, paréntesis innecesarios y convirtiendo los valores a enteros, para después leer los archivos.

# Procesamiento de archivos

```
(defn procesar-lineas-sentido-helper [linea indices crucero-id linea-num]
  (if (>= (count linea) (apply max indices)) ; Verifica si la línea es suficientemente larga para los índices
    (map #(nth linea %) indices) ; Si es suficientemente larga, extrae los valores en los índices especificados
    (do ; Si la línea es demasiado corta
      (println (str "Línea demasiado corta (procesar-lineas-sentido-helper) en el crucero " crucero-id " en la línea " linea-num ": " linea " con índices: " indices)); Imprime un men
      (println (str "Longitud de la línea: " (count linea))); Imprime la longitud de la línea
      nil)); Retorna nil

  (defn procesar-lineas-sentido [coches crucero-id]
    (mapcat (fn [linea-num linea]
      (if (empty? linea) ; Verifica si la línea está vacía
        (do
          (println (str "Línea vacía (procesar-lineas-sentido-helper) en el crucero " crucero-id " en la línea " linea-num ": " linea)); Imprime un mensaje de error
          []); Retorna una lista vacía
        (remove nil?
          [(procesar-lineas-sentido-helper linea [1 2 3 4] crucero-id linea-num) ; Procesa la línea con diferentes conjuntos de índices
            (procesar-lineas-sentido-helper linea [2 3 4 5] crucero-id linea-num)
            (procesar-lineas-sentido-helper linea [3 4 5 6] crucero-id linea-num)
            (procesar-lineas-sentido-helper linea [4 5 6 7] crucero-id linea-num)
            (procesar-lineas-sentido-helper linea [5 6 7 8] crucero-id linea-num))]))
      (range) coches)); Aplica la función a cada línea en 'coches'

  (defn procesar-lineas-helper [linea indices crucero-id linea-num]
    (if (>= (count linea) (apply max indices)) ; Verifica si la línea es suficientemente larga para los índices
      (map #(nth linea %) indices) ; Si es suficientemente larga, extrae los valores en los índices especificados
      (do ; Si la línea es demasiado corta
        (println (str "Línea demasiado corta (procesar-lineas-helper) en el crucero " crucero-id " en la línea " linea-num ": " linea " con índices: " indices)); Imprime un mensaje de
        (println (str "Longitud de la línea: " (count linea))); Imprime la longitud de la línea
        nil)); Retorna nil

    (defn procesar-lineas [logicas crucero-id]
      (mapcat (fn [linea-num linea]
        (if (empty? linea) ; Verifica si la línea está vacía
          (do
            (println (str "Línea vacía (procesar-lineas-helper) en el crucero " crucero-id " en la línea " linea-num ": " linea)); Imprime un mensaje de error
            []); Retorna una lista vacía
          (remove nil?
            [(procesar-lineas-helper linea [8 9 10] crucero-id linea-num) ; Procesa la línea con diferentes conjuntos de índices
              (procesar-lineas-helper linea [9 10 11] crucero-id linea-num)
              (procesar-lineas-helper linea [10 11 12] crucero-id linea-num)
              (procesar-lineas-helper linea [11 12 13] crucero-id linea-num)
              (procesar-lineas-helper linea [12 13 14] crucero-id linea-num))]))
            (range) logicas)); Aplica la función a cada línea en 'logicas'
```

En estas funciones es donde se procesan las líneas tanto de los archivos de tiempos que llegaron los coches, como los archivos de lógica, en estos se segmenta la información principal de todos los tiempos de llegada en 4 elementos, esto es para optimizar el pmap como nos habían enseñado en clase para que no existan tantos hilos y la eficiencia del código se mucho mayor y saque verdadero provecho de la función paralela de pmap.

## Tiempo del programa

```
(defn convertir-tiempo [nano-segundos]
  ; Define la función convertir-tiempo con el parámetro nano-segundos
  (let [segundos (/ nano-segundos 1e9)]
    ; Convierte los nanosegundos a segundos dividiendo por 1e9 (mil millones)
    (format "%.5f segundos" segundos))) ; Formatea los segundos a 5 decimales y añade la unidad "segundos"
```

Esta es la función que se encarga de convertir el tiempo y obtener cuanto tiempo se tarda el programa en acabar la ejecución del mismo.

```
(println (str "Crucero " crucero-id " : tiempo-muerto " segundos ")); Imprime el identificador y el tiempo muerto acumulado para cada crucero

(println "\nTiempo que se tardó el programa: " (convertir-tiempo tiempo-total)); Imprime el tiempo total de ejecución del programa convertido a segundos

(loop []); Inicia un bucle para que el usuario pueda pedir varias veces
```

Luego en la función main se llama a la función de convertir-tiempo para que nos muestre cuánto tiempo tardó el programa en procesar los 40 cruceros de manera paralela.

## Experiencia de aprendizaje (Conclusión)

La verdad al momento de ver que para el último periodo íbamos a tener que pasar el programa de la evidencia dos que ya se me había hecho súper compleja, pero ahora en un nuevo lenguaje, en este caso Clojure, y que además se tenía que paralelizar cosa para la que tendríamos que aprender un nuevo paradigma de programación, si estaba muy preocupado, ya que era primero pasar el código a Clojure, luego adaptarlo y mejorarlo con las recomendaciones que nos dieron en la evidencia 2, luego adaptar todo el código para que funcionará de manera paralela y procesara más de un crucero, en este caso 40 que decidí hacer, no sabía como se lograría, pero la verdad si vi que Clojure tenía muchas cosas igual y/o similares a racket, así que si bien fue algo difícil pasar el código, no fue nada del otro mundo, el detalle en mi código es que aparte de que tuve que adaptar mi código luego tuve que cambiar toda mi lógica anterior ya que antes creaba 40 archivos por cada crucero, y si quería hacer esto de manera paralela no hubiera sido nada factible, así que tuve que rediseñar el código, pero gracias a esto en lugar de que mi código fuera de 600 líneas, terminó siendo de 260, así que lo reduje a más de la mitad, cosa que facilitó mucho más todo y terminé haciendo uso de pmap para poner en práctica el paradigma paralelo de una manera muy sencilla y de manera optima gracias a todo lo que fui aprendiendo a lo largo de este bloque y materia, así que me voy muy contento y feliz de que aparte de haber aprendido un nuevo lenguaje de programación, pude aprender uno muy útil y un nuevo paradigma.