DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

ANÁLISIS DE ALGORITMOS Y ESTRUCTURA DE DATOS

Simulación de procesos de embarque de cargas

Profesor: Cristián Sepúlveda S.
Alumno: Rodrigo Pereira Yánez

20 de diciembre de 2024

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Índice

Introducción.

La teoría de colas constituye una disciplina fundamental dentro de la investigación de operaciones, diseñada para modelar y analizar sistemas en los que se forman líneas de espera. Su aplicación permite predecir y optimizar aspectos clave como la longitud de las colas y los tiempos de espera, lo que resulta esencial para la toma de decisiones en una amplia gama de sectores industriales y de servicios. Desde telecomunicaciones hasta la gestión de proyectos, esta teoría proporciona herramientas valiosas para diseñar sistemas más eficientes.

En el ámbito del transporte marítimo, que representa aproximadamente el 80 % del comercio internacional según la UNCTAD (2020), la optimización de los procesos de embarque es crítica para garantizar la competitividad y sostenibilidad de las operaciones. Maersk Line, la compañía líder en transporte de contenedores, enfrenta desafíos operativos significativos relacionados con la preparación y embarque de carga, como la gestión eficiente del peso y la funcionalidad de las unidades de refrigeración. Entre estos desafíos, uno de los más relevantes es agilizar los procesos de embarque para minimizar tiempos y costos asociados.

Este informe aborda el problema de simular el comportamiento de una cola de embarque en operaciones marítimas. Utilizando estructura de datos lineales, se diseñará un programa en lenguaje C que permita modelar el sistema, evaluar distintas configuraciones y analizar métricas clave como los tiempos de espera promedio, la cantidad de cargas embarcadas y no embarcadas, el momento de embarque entre otros datos. Además, se calculará la complejidad computacional de los algoritmos empleados, garantizando así la aplicación de soluciones robustas y escalables.

Método.

Para el desarrollo de este laboratorio se cuenta con la información entregada en el enunciado del problema. En este documento se detalla el contexto del problema y cuál es el objetivo principal

- “Se le solicita a usted la construcción de un programa en lenguaje C que simule el comportamiento de una cola de embarque.”.

Con esto en mente, se detalla lo siguiente:

1. Equipos y materiales
    a. Equipo de desarrollo:
        i. Computadoras con sistema operativo Linux (PopOs) para implementar, ejecutar y analizar el programa.

b. Herramientas de desarrollo:
    i. Un entorno de desarrollo integrado (IDE) como Visual Studio Code y/o CLion configurado para trabajar con lenguaje C.
    ii. Compiladores de C como GCC (GNU Compiler Collection).
    iii. Terminal de Linux.
c. Materiales adicionales:
    i. Documentación de la materia vista en clases y laboratorios sobre estructura de datos lineales y teoría de colas.

2. Procedimientos
    a. Comprensión del problema:
        i. Se estudió el enunciado, el cual detalla el objetivo principal: construir un programa en C que simule el comportamiento de una cola de embarque.
    b. Diseño de estructura de datos:
        i. Se implementó una estructura de datos tipo cola basada en listas enlazadas para modelar la fila de cargas a embarcar.
        ii. Cada nodo de la cola representa una carga, definida mediante una estructura con atributos como número de cargas, datos asociados y un puntero al siguiente nodo.
    c. Desarrollo de algoritmos:
        i. Se elaboraron algoritmos en pseudocódigo para las siguientes operaciones principales:
            - Creación de colas vacías.
            - Inserción de cargas en la cola al momento de su llegada.
            - Simulación del proceso de embarque mediante la función Simula(), que considera parámetros como tiempo total de simulación, intervalo entre llegadas y probabilidad de embarque.
            - Gestión del proceso de embarque en períodos activados aleatoriamente mediante la función embarcar().
            - Cálculo de métricas como tiempos promedio de espera y número de cargas embarcadas.
            - Impresión en consola del estado de la cola y los resultados finales de la simulación.
    d. Implementación en lenguaje C:
        i. Se tradujeron los algoritmos diseñados a código C, asegurando que todas las operaciones se ejecutaran sobre la estructura de datos definida.
        ii. Se incluyó una función embarcar() que utiliza un generador de números aleatorios para determinar los períodos de embarque.
        iii. Los parámetros de entrada del programa incluyen tiempo de simulación, intervalo entre llegadas y probabilidad de activación del embarque.
    e. Ejecución y pruebas:

   i. Se probaron múltiples configuraciones de los parámetros para evaluar el comportamiento del sistema.
   ii. Los datos de salida se registraron y analizaron para evaluar la eficiencia de las operaciones.
f. Procesamiento de datos:
   i. Los resultados obtenidos se procesaron para calcular:
      - Tiempos de activación del embarque.
      - Tiempos promedio de permanencia de las cargas.
      - Cantidad total de cargas embarcadas y no embarcadas.
      - Estadísticas de tiempos del sistema.
g. Análisis de complejidad temporal:
   i. Se determinó la complejidad de las operaciones realizadas utilizando la notación O grande (Landau).
   ii. Esto permitió evaluar la eficiencia en términos de tiempo de procesamiento del algoritmo.

3. Métricas clave
   a. Tiempo promedio de permanencia en la cola:
      i. Calculado sumando los tiempos individuales y dividiendo por el número total de cargas embarcadas.
   b. Cantidad de cargas embarcadas y no embarcadas

Con este enfoque, se estableció una metodología estructura que combina diseño, implementación y análisis para abordar el problema planteado.

Algoritmos propuestos.

A continuación se presentan todos los algoritmos implementados para poder resolver la problemática de embarque de cargas.

Estructura de datos: datos

`REGISTRO datos`
`    NUM tiempo_carga`
`    NUM tiempo_embarque`
`    NUM tiempo_permanencia`
`    NUM tiempo_activacion`

Estructura de datos: Carga

`REGISTRO carga`
`    NUM num_carga`
`    datos* datos_carga`
`    carga *siguiente`

Estructura de datos: Cola

```
REGISTRO cola
    carga* frente
    carga* final
```

Operación TDA Cola: crea_cola() crea y retorna una cola vacía

Orden de complejidad: O(1)

```
crea_cola(): cola*
    cola* c ← (cola*)malloc(sizeof(cola))
    c→frente ← NULL
    c→final ← NULL
    RETURN c
```

Operación TDA Cola: es_cola_vacia(c) determina si la cola c está vacía

Orden de complejidad: O(1)

```
es_cola_vacia(cola* c): NUM
    IF (c→frente = NULL) THEN
        RETURN 1
    ELSE
        RETURN 0
```

Operación TDA Cola: encolar(c, datos_carga, num_carga) agrega un elemento a la cola c

Orden de complejidad: O(1)

```
encolar(cola* c, datos* datos_carga, NUM num_carga)
    carga* n ← (carga*)malloc(sizeof(carga))
    n→datos_carga ← datos_carga
    n→num_carga ← num_carga
    n→siguiente ← NULL
    IF (c→frente = NULL) THEN
        c→frente ← n
        c→final ← n
    ELSE
        c→final→siguiente ← n
        c→final ← n
```

Operación TDA Cola: desencolar(c) elimina el primer elemento de la cola c

orden de complejidad: O(1)

```
desencolar(cola* c)
    IF (es_cola_vacia(c) = 0) THEN
        carga* ptr_aux ← c→frente
        c→frente ← c→frente→siguiente
        liberar(ptr_aux)
     ELSE
        c→final ← NULL
```

Operación TDA Cola: frente(c) retorna el primer elemento de la cola c

Orden de complejidad: O(1)

```
frente(cola* c): carga*
    IF (es_cola_vacia(c) = 0) THEN
        RETURN c→frente
    RETURN NULL
```

Operación TDA Cola: final(c) retorna el último elemento de la cola c

Orden de complejidad: O(1)

```
final(cola* c): carga*
    IF (es_cola_vacia(c) = 0) THEN
        RETURN c→final
    RETURN NULL
```

Operación TDA Cola: imprime_cola(c) imprime los elementos de la cola c

Orden de complejidad: O(n)

```
imprime_cola(cola* c)
     IF (es_cola_vacia(c) = 0) THEN
         carga* ptr ← c→frente
         WHILE (ptr <> NULL) DO
         WRITE("[ptr→num_carga|ptr→datos_carga→tiempo_carga]")
         IF (ptr→siguiente <> NULL) THEN
             WRITE(" → ")
              ptr ← ptr→siguiente
         WRITE("")
      ELSE
         WRITE("[]")
```

Operación TDA Cola: contar_cola(c) cuenta los elementos de la cola c

Orden de complejidad: O(n)

```
contar_cola(cola* c): NUM
    NUM count ← 0
```

```
    IF (es_cola_vacia(c) = 0) THEN
        carga* ptr ← c→frente
        WHILE (ptr <> NULL) DO
            count = count + 1
            ptr ← ptr→siguiente
    RETURN count
```

Operación Simulación: embarcar(probabilidad) simula el embarque de un pasajero

Orden de complejidad: O(1)

```
embarcar(NUM probabilidad): NUM
    NUM aleatorio ← 1.0 * rand() / RAND_MAX
    IF (aleatorio < probabilidad) THEN
        RETURN 1
    ELSE
        RETURN 0
```

Operación Simulación: simular(cola_espera, cola_embarque, tiempo_simulacion, tiempo_intervalo, probabilidad_embarque) simula el proceso de embarque

*Orden de complejidad: O(n*m)*

```
simular(cola* cola_espera, cola* cola_embarque, NUM
tiempo_simulacion, NUM tiempo_intervalo, NUM
probabilidad_embarque)
    NUM count_tiempo ← 0
    NUM num_carga ← 1
    NUM proceso_embarque ← 0
    NUM tiempo_activacion ← 0
    WHILE (count_tiempo <= tiempo_simulacion) DO
        WRITE("--------------------------------")
        WRITE("*** Tiempo de simulación: count_tiempo ***")
        WRITE("--------------------------------")
            IF (count_tiempo % tiempo_intervalo = 0) THEN
                datos* datos_carga ← (datos*)malloc(sizeof(datos))
                datos_carga→tiempo_carga ← count_tiempo
                datos_carga→tiempo_embarque ← 0
                datos_carga→tiempo_permanencia ← 0
                datos_carga→tiempo_activacion ← 0
                encolar(cola_espera, datos_carga, num_carga)
                num_carga = num_carga + 1
            NUM embarque ← embarcar(probabilidad_embarque)
            IF (embarque = 1 || proceso_embarque = 1) THEN
                IF (not(es_cola_vacia(cola_espera))) THEN
                    WRITE("*Proceso de embarque activo ")
                    IF (not(proceso_embarque)) THEN
                        tiempo_activacion ← count_tiempo
                    WRITE("desde tiempo tiempo_activacion...")
```

```
                    proceso_embarque ← 1
                    carga* carga_actual ← frente(cola_espera)
                    IF (carga_actual→datos_carga→tiempo_carga <
tiempo_activacion) THEN
                        carga_actual→datos_carga→tiempo_embarque
← count_tiempo

carga_actual→datos_carga→tiempo_permanencia ← count_tiempo
                        carga_actual→datos_carga→tiempo_carga

carga_actual→datos_carga→tiempo_activacion ← tiempo_activacion
                        encolar(cola_embarque,
carga_actual→datos_carga, carga_actual→num_carga)
                        desencolar(cola_espera)
                    ELSE
                        proceso_embarque ← 0
                ELSE
                    proceso_embarque ← 0
            WRITE("Cola de espera")
            imprime_cola(cola_espera)
            WRITE("Cola de embarque")
            imprime_cola(cola_embarque)
            count_tiempo = count_tiempo + 1
```

Operación Simulación: imprimir_informacion(cola_embarque, cola_espera) imprime la información de la simulación

Orden de complejidad: O(n)

```
imprimir_informacion(cola* cola_embarque, cola* cola_espera)
    WRITE("RESUMEN DE LA SIMULACIÓN:")
    WRITE(" -Tiempos de activación de periodos de embarque: ")
    imprimir_tiempo_activacion(cola_embarque)
    WRITE(" -El momento de embarque de cada carga es: ")
    imprimir_momento_embarque(cola_embarque, cola_espera)
    WRITE(" -El tiempo de permanencia de cada carga es: ")
    imprimir_tiempo_permanencia(cola_embarque, cola_espera)
    WRITE(" -Tiempo promedio de permanencia de las cargas
embarcadas: promedio_tiempo_permanencia(cola_embarque) unidades de
tiempo")
    WRITE(" -Número de cargas embarcadas:
contar_cola(cola_embarque)")
    WRITE(" -Número de cargas no embarcadas:
contar_cola(cola_espera)")
```

Operación Simulación: imprimir_tiempo_activacion(cola_embarque) imprime el tiempo de activación de los periodos de embarque

Orden de complejidad: O(n)

```
imprimir_tiempo_activacion(cola* cola_embarque)
    IF (es_cola_vacia(cola_embarque) = 0) THEN
        carga* ptr ← cola_embarque→frente
        NUM tiempo_activacion ← 0
        WHILE (ptr <> NULL) DO
            IF (ptr→datos_carga→tiempo_activacion <>
tiempo_activacion) THEN
                WRITE("ptr→datos_carga→tiempo_activacion ")
                tiempo_activacion ←
ptr→datos_carga→tiempo_activacion
            ptr ← ptr→siguiente
        WRITE("")
    ELSE
        WRITE("0")
```

Operación Simulación: imprimir_momento_embarque(cola_embarque, cola_espera) imprime el momento de embarque de cada carga

Orden de complejidad: O(n)

```
imprimir_momento_embarque(cola* cola_embarque, cola* cola_espera)
    IF (es_cola_vacia(cola_embarque) = 0) THEN
        carga* ptr ← cola_embarque→frente
        WHILE (ptr <> NULL) DO
            WRITE("c ptr→num_carga:
ptr→datos_carga→tiempo_embarque, ")
            ptr ← ptr→siguiente
    IF (es_cola_vacia(cola_espera) = 0) THEN
        carga* ptr ← cola_espera→frente
        WHILE (ptr <> NULL) DO
            WRITE("c ptr→num_carga:-1")
            IF (ptr→siguiente <> NULL) THEN
                WRITE(" ,")
            ptr ← ptr→siguiente
```

Operación Simulación: imprimir_tiempo_permanencia(cola_embarque, cola_espera) imprime el tiempo de permanencia de cada carga

Orden de complejidad: O(n)

```
imprimir_tiempo_permanencia(cola* cola_embarque, cola*
cola_espera)
    IF (es_cola_vacia(cola_embarque) = 0) THEN
        carga* ptr ← cola_embarque→frente
        WHILE (ptr <> NULL) DO
            WRITE("c ptr→num_carga:
ptr→datos_carga→tiempo_permanencia, ")
            ptr ← ptr→siguiente
    IF (es_cola_vacia(cola_espera) = 0) THEN
```

```
        carga* ptr ← cola_espera→frente
        WHILE (ptr <> NULL) DO
            WRITE("c ptr→num_carga:-1")
            IF (ptr→siguiente <> NULL) THEN
                WRITE(" ,")
            ptr ← ptr→siguiente
```

Operación Simulación: promedio_tiempo_permanencia(cola_embarque) calcula el promedio del tiempo de permanencia de las cargas embarcadas

Orden de complejidad: O(n)

```
promedio_tiempo_permanencia(cola* cola_embarque): NUM
    NUM count ← 0
    NUM sum ← 0
    IF (es_cola_vacia(cola_embarque) = 0) THEN
           carga* ptr ← cola_embarque→frente
           WHILE (ptr <> NULL) DO
            sum ← sum + ptr→datos_carga→tiempo_permanencia
            count = count + 1
            ptr ← ptr→siguiente
    RETURN (NUM)sum/count
```

Cálculo de complejidad.

Basado en el código anteriormente entregado en pseudocódigo, a modo de resumen las complejidades de cada función son las siguientes:

Resumen de complejidad por función	
Función	**Orden de Complejidad (Notación Big O)**
`crea_cola(): cola*`	O(1)
`es_cola_vacia(cola* c): NUM`	O(1)
`encolar(cola* c, datos* datos_carga, NUM num_carga)`	O(1)
desencolar(cola* c)	O(1)

frente(cola* c): carga*	O(1)
final(cola* c): carga*	O(1)
imprime_cola(cola* c)	O(n)
contar_cola(cola* c): NUM	O(n)
embarcar(NUM probabilidad): NUM	O(1)
simular(cola* cola_espera, cola* cola_embarque, NUM tiempo_simulacion, NUM tiempo_intervalo, NUM probabilidad_embarque)	O(n*m)
imprimir_informacion(cola* cola_embarque, cola* cola_espera)	O(n)
imprimir_tiempo_activacion(cola* cola_embarque)	O(n)
imprimir_momento_embarque(cola* cola_embarque, cola* cola_espera)	O(n)
imprimir_tiempo_permanencia(cola* cola_embarque, cola* cola_espera)	O(n)
promedio_tiempo_permanencia(cola* cola_embarque): NUM	O(n)

La función principal main() solo llama a las funciones para crear las colas, ejecutar la simulación y mostrar la información, sin tener un impacto significativo en la complejidad.

El factor más costoso es la función simular(), que tiene una complejidad O(n * m), donde n es el tamaño máximo de las colas y m es el tiempo de simulación.

Por lo tanto, la complejidad total del algoritmo es:

$$O(n * m)\ ;\ n = \ tamaño\ cola,\ m = \ tiempo\ de\ simulacion$$

Además, las funciones de impresión y cálculo como imprimir_informacion, imprimir_tiempo_activacion, y promedio_tiempo_permanencia también contribuyen con un O(n) adicional. Sin embargo, estos se ejecutan solo una vez al final de la simulación, por lo que la complejidad dominante sigue siendo O(n * m).

Conclusiones.

En este trabajo se ha investigado la optimización del proceso de embarque de cargas en un puerto marítimo, modelado como una cola de espera en la que las cargas llegan periódicamente y deben ser embarcadas según una probabilidad determinada como

parámetro de entrada. El objetivo principal fue simular el comportamiento de esta cola de espera y analizar el desempeño del proceso, considerando factores como el tiempo de simulación, los intervalos entre llegadas de cargas y la probabilidad de embarque.

Los hallazgos principales respecto a la simulación del proceso de embarque son los siguientes:

1. Eficiencia en la gestión de cargas: La eficiencia de este proceso está estrechamente vinculada a la probabilidad de embarque, ya que esta es la que determina si las cargas pueden ser embarcadas. A mayor probabilidad de embarque, el tiempo de espera se reduce considerablemente.
2. Complejidad del algoritmo: El orden de complejidad del algoritmo depende del tamaño de la cola y del tiempo de simulación, puesto que la función simular() es la que asume la mayor carga de trabajo. El orden de complejidad es O(n*m), donde n es el número de cargas y m es el tiempo de simulación.

En resumen, el análisis de la simulación de la cola de embarque ha demostrado que, mediante un control adecuado de parámetros como la probabilidad de embarque, la periodicidad de llegada de las cargas y el tiempo de simulación, los tiempos de procesamiento de cargas pueden ser optimizados. Siendo estos tres factores los que condicionan la eficiencia del proceso de embarque en un puerto marítimo.

Apéndice

Para el empleo del programa, se deben seguir los siguientes pasos:

1. Abrir la terminal en la carpeta donde están los archivos .c
2. ejecutar el comando de compilado
    a. gcc T2_coordinacion_Rodrigo_Pereira.c tarea2_implementacion.c -o tarea2_simulacion
3. Ejecutar el comando para iniciar el programa
    a. ./tarea2_simulacion <tiempo_simulacion> <intervalo_llegada> <probabilidad de embarque>
    b. Donde:
        - <tiempo_simulacion>: Entero que indica el tiempo de ejecución de la simulación
        - <intervalo_llegada>: Entero que indica el intervalo de tiempo entre las llegadas de las cargas.
        - <probabilidad de embarque>: Decimal que indica la probabilidad de que se inicie el proceso de embarque, puede ser un decimal entre 0 y 1, con un valor después del punto.

Posibles errores:

1. El número a ingresar como probabilidad debe estar entre 0 y 1. Es un decimal que debe ser separado con un “.” no con “,”. Si no, tomara como probabilidad el primer número antes de la coma.
2. Si no ingresa los parámetros de entrada al ingresar al programa, tendrá un error con la siguiente leyenda.
    a. Uso: ./tarea2_simulacion <tiempo_simulacion> <intervalo_llegada> <probabilidad de embarque>

Principales funcionalidades del programa:

1. Permite simular un proceso de carga de un puerto, con base en 3 parámetros: Tiempo de simulación, intervalo de llegada y probabilidad de embarque.
2. Imprime por cada unidad de tiempo el estado de la cola de espera y la de embarque.
3. Al final de la simulación entrega un resumen con los siguientes datos.
    a. Tiempo de activación de los periodos de embarque.
    b. Tiempo donde cada carga fue embarcada.
    c. Tiempo de permanencia de cada carga en la cola de espera.
    d. Tiempo promedio de permanencia de todas las cargas que fueron embarcadas.
    e. Número de cargas embarcadas.
    f. Número de cargas no embarcadas.

Referencias.

1. Análisis de Algoritmos y Estructura de datos (13107) 2-2024. [Material] extraído desde Campus Virtual de la asignatura de Análisis de algoritmos y estructura de datos.
2. Cálculo de BIG O: https://www.geeksforgeeks.org/analysis-algorithms-big-o-analysis/