|  | **UNIVERSIDAD DE SANTIAGO DE CHILE**  **FACULTAD DE INGENIERÍA**  **DEPARTAMENTO DE INGENIERÍA INFORMÁTICA** | C:\Users\Lucky\Downloads\Test Lab3\UDS TBLK.png |
| --- | --- | --- |

**Laboratorio 1**

**Análisis de algoritmos y estructura de datos.**

| **Camila Castillo, Manuel Vásquez** |
| --- |
|  |
|  |

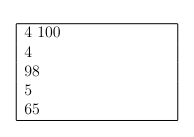
|  | **Profesor:** |
| --- | --- |
|  | Cristían Sepúlveda |
|  |  |
|  |  |
|  |  |

|  | Santiago - Chile |  |
| --- | --- | --- |
|  | 2024 |  |

**Introducción:** En la actualidad el transporte marítimo representa aproximadamente al 80% del comercio internacional. Siendo el transporte marítimo transporte menos costoso que el transporte de aire o tierra.La empresa Maersk Line es pionera en este tipo de transporte, siendo la compañía de transporte de contenedores más grande del mundo, con 80.000 operarios en 130 países. A pesar de su gran expansión por el mundo, Maersk Line no está libre de problemas. Uno de los problemas más cruciales para la compañía es la optimización de los procesos de embarques de cargas. Con el fin de solucionar los tiempos, la empresa requiere la resolución del siguiente problema: ”*Dado un conjunto de cargas de diferentes pesos y un número de contenedores igual al número de cargas, cada uno de ellos con una capacidad de peso máxima fija, se busca minimizar el número número de contenedores utilizados para embarcar todas las cargas”.* Para la resolución de problemas en programación existen diferentes métodos, uno de ellos y en el que se basará este laboratorio es la “enumeración exhaustiva” o “fuerza bruta”. Este método consiste en generar todas las soluciones posibles del problema de forma sistemática, para luego evaluar las soluciones posibles una a una, manteniendo un registro de la mejor encontrada. Cuando la búsqueda finalice, entregar la mejor solución encontrada. Los algoritmos basados en este método funcionan en un tiempo razonable cuando el número de entradas son menores, pero cuando el número de entradas aumenta, el tiempo de ejecución es mayor. (Tello, 2018)

**Objetivos:** Los objetivos serán la creación de un programa en C que resuelva el problema anterior con un algoritmo basado en el método enumeración exhaustiva, para así poder analizar los tiempos que demora el algoritmo con un número de cargas variable. Se espera que entre mayor número de cargas tenga el archivo a ingresar en el código, mayor será el tiempo de ejecución del programa, lo que generará un orden alto de complejidad del algoritmo.

**Método:** Se utilizó para la realización de este laboratorio, un computador con sistema Windows , MinGW (Minimalist GNU For Windows) para la utilización de C en Windows y un editor de texto. Para poder ejecutar correctamente el algoritmo, se necesita que el código proporcionado este en la misma carpeta donde se encuentran los archivos requeridos llamados “cargas\_X.in”, donde X va desde 4 a 20. Sin estos archivos, el código no se podría ejecutar. Se adjunta un manual de instrucciones al final de este informe para la utilización de este programa (Ver apéndice).

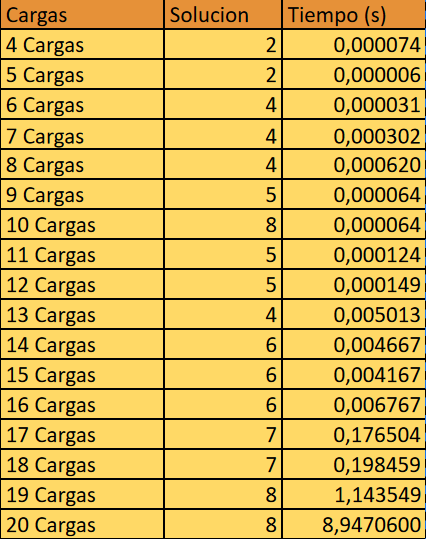
**Procedimiento:** El archivo principal contiene el algoritmo basado en enumeración exhaustiva, el cual requiere de un archivo “cargas\_X.in”, donde X va de 4 al 20. La primera línea del archivo contiene dos números, el primer número representa las cargas totales y el segundo será la capacidad máxima del contenedor. Las demás líneas representan los pesos de cada una de las cargas. El archivo tipo se muestra en la Figura 1. 

**Figura 1**: **Contenido del archivo cargas\_4.in**

*La primera línea contendría el número de 4 cargas y la capacidad máxima del contenedor. Las siguientes líneas corresponden a cargas de 4, 98, 5 y 65 de peso.*

La función “*asignarCargas”,* utiliza la recursión de pila para explorar todas las posibles cargas en los contenedores, se lleva un conteo de “*minContenedores”.* Además, el código proporciona el tiempo de ejecución que toma la función asignar cargas, para dar el número mínimo de contenedores. Algunas dificultades encontradas en la realización del algoritmo fue como representar la carga como un tipo de dato abstracto (TDA), al final se optó por definirse como un struct, donde la carga tendría un peso del tipo entero. Otra dificultad, fue cómo guardar las combinaciones posibles en un tipo de dato y que no hubiera desborde. Por lo que se optó, por el tipo de dato unsigned long long, que guardara la constante 20!, ya que sería el máximo de combinaciones posibles que se podrían hacer con 20 cargas. Por último, con relación al algoritmo, surgieron dificultades a la hora de proponer una solución, ya que se proponían ideas, pero no relacionadas con la enumeración exhaustiva. Finalmente, se logró un algoritmo que trabaja recursivamente, pero a la hora de ejecutar el programa no daba los resultados esperados, por lo que se debió analizar el algoritmo y observar si se le estaba dando los argumentos correctos a las llamadas recursivas, lo cual no era lo correcto, por que se hicieron unas modificaciones a los argumentos y se logró dar con la solución.

**Resultados y análisis:** Los resultados muestran que dependiendo de los archivos ingresados en el programa, se obtienen diferentes resultados. Siendo el número de cargas una variable que aumenta el tiempo de ejecución del programa medido en segundos. Se puede observar que el tiempo de ejecución aumenta a medida que las cargas van en aumento. En el archivo con 17 cargas, se puede notar que el aumento es mayor en comparación a los otros, y se mantiene poco variable hasta llegar al archivo de 19 cargas, donde aumenta a 1 segundo aprox. En el archivo con 20 cargas, ya se puede observar un aumento mucho mayor a los demás, con un tiempo de 8,94 (s), aproximadamente (Ver Figura 2) Los resultados efectivamente demuestran que el orden de complejidad del algoritmo, tiene de base a la función asignarCargas, la cual es una función recursiva dependiente del número de cargas, lo que conlleva a concluir que si se cumple la hipótesis planteada anteriormente, donde el orden del algoritmo que se basa en generar todas las posibles asignaciones de las cargas a contenedores es de un orden alto, en este caso exponencial de O(2nCargas).

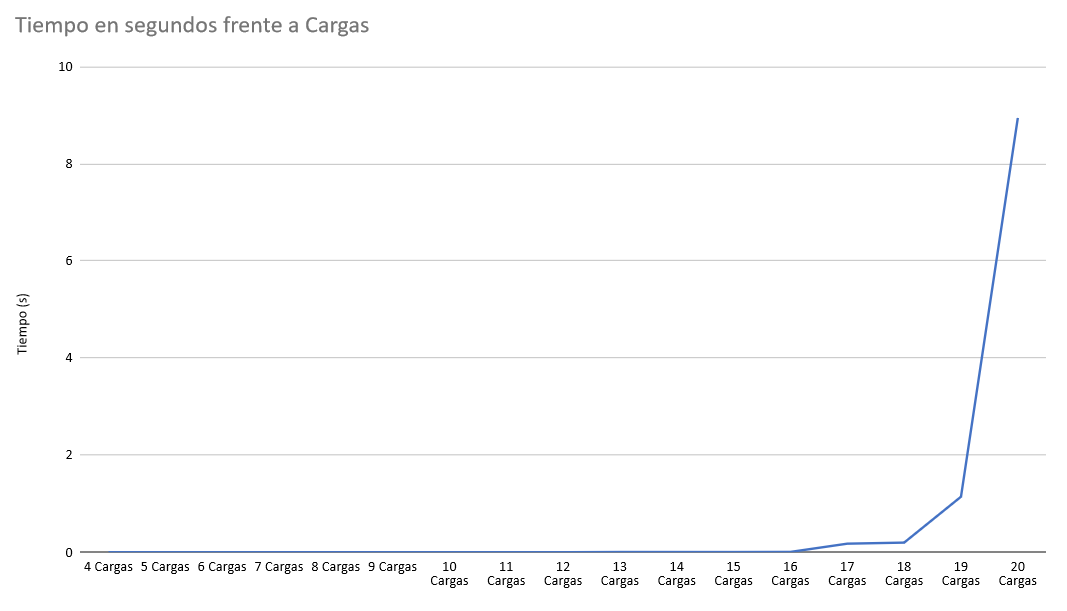


**Figura 2: Valores de los número de cargas, número mínimo de contenedores (solución) y tiempo en segundos.**

*En esta tabla se puede observar tanto la cantidad óptima de contenedores como el tiempo que se demoró el algoritmo en llegar a esta respuesta. Al comparar estos resultados con los entregados, se puede dar cuenta de que el código cumple con lo que se pide satisfactoriamente.*

**Figura 3: Gráfico de tiempo en segundos y número de cargas.**

*Se puede observar que a medida que aumenta el número de cargas aumenta el tiempo de ejecución.*



**Discusión:** Se hace la observación de que a mayor cantidad de cargas que se introduzca al código, mayor será el tiempo de ejecución, ya que serán más las combinaciones posibles. Se puede comprender esto porque el programa está basado en el algoritmo de enumeración exhaustiva o fuerza bruta, el cual genera todas las combinaciones posibles. Se considera que si el algoritmo de base fuera distinto, los tiempos de ejecución cambiarían, quizás en relación a otras variables, pero la enumeración exhaustiva no deja de ser eficaz, por lo que logra el resultado esperado, pero con un costo alto si es que el número de cargas son mayores. También es importante señalar que pasado el máximo de cargas usadas de ejemplo en este trabajo (20 en este caso) se podría esperar que el código no funcione del todo, ya que la cantidad de soluciones posibles serían demasiadas para un procesador convencional y que además existe una constante definida al inicio del código, la cual si el número de cargas es mayor, podría no dar la solución correcta o que tome mucho tiempo generarla. Otro factor a considerar es la potencia del procesador, ya que el equipo donde se ejecute el código también podría afectar en el tiempo de ejecución. Asimismo, el orden de las cargas en los archivos es también otro factor que podría generar una variabilidad en el tiempo de ejecución.

**Conclusiones:** En este informe se detalló cómo se generó la creación de un código en lenguaje de programación C, el cual buscaba optimizar la distribución de cargas en un número de contenedores de capacidad fija para una empresa de transporte marítimo. La resolución del problema debía basarse en la técnica enumeración exhaustiva, para así, después analizar los resultados de los tiempos de ejecución del algoritmo dependiendo de las diversas cargas. Para lograr el objetivo de este laboratorio, se investigó qué es la enumeración exhaustiva para entenderla y luego aplicarla en el código. Habiendo cumplido este objetivo, se pudo dar paso a trabajar alrededor de este sistema de búsqueda de soluciones, luego de implementar unos filtros para todas las soluciones posibles, se llegó a el óptimo buscado. Se cumplieron los objetivos propuestos al principio de este informe satisfactoriamente y también se cumplió la hipótesis propuesta, la cual era que el orden de complejidad sería alto por ser un algoritmo no muy eficiente, pero sí eficaz. Finalmente, se puede considerar que existen diversos factores los cuales pueden afectar al tiempo de ejecución de un programa, más allá de los componentes del equipo, y este va a variar de acuerdo a los algoritmos propuestos por el programador.

**Referencias:**

Tello, E. R. (2018). Algoritmos de búsqueda exhaustiva. *CINVESSTAV-Tamaulipas*. Recuperado de: <https://www.tamps.cinvestav.mx/~ertello/algorithms/sesion07.pdf>

**Apéndice:**

* ***Manual de usuario (Windows/Linux):***

*Como usar:*

1. Se compila el código con en la terminal con la expresión de entrada:

***gcc T1\_A1\_Castillo\_Vasquez.c -o Salida***



Se presiona enter.

1. En la siguiente línea de la terminal cmd, se coloca:

***./Salida cargas\_X.in***

(X sería el número de cargas de 4 a 20)



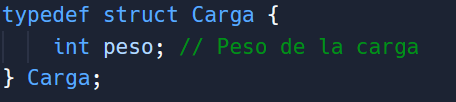
Se presiona enter.

1. En la cmd se podrá ver la salida : “Número mínimo de contenedores: N” (Donde N, sería la solución del respectivo archivo de cargas.

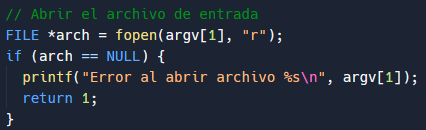


* ***Funciones del código:***

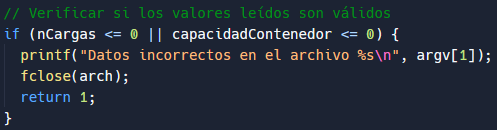
La función “asignarCargas” primero pregunta si se han distribuido todas las cargas, si es ese el caso se actualiza el mínimo de contenedores posibles para el óptimo. Y luego, mediante unos filtros, se decide o no si es posible agregar la carga actual con la que se trabaja al contenedor o no, llamando a la función de forma recursiva con la siguiente carga dependiendo del caso.

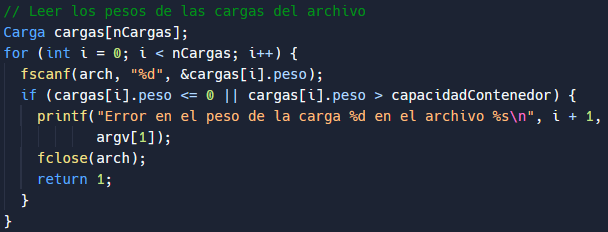
Struct Carga con la cual se trabaja en todo el código. 

* ***Posibles errores:***



1. Dar mal un nombre de archivo o equivocarse en el nombre puede llevar al error a la hora de abrir el archivo, dejando el mensaje de “Error al abrir el archivo X” siendo X el nombre del archivo entregado.



1. A la hora de leer los archivos, si es que el documento tiene un valor de capacidad máxima inferior o igual a cero, conlleva a un error ya que es un caso imposible. Lo mismo pasa si la cantidad de cargas es menor o igual a cero.
2. Cuando se leen los pesos, si cualquiera es igual o menor a cero conlleva a un error ya que no es un peso real para una simulación.

Se recuerda tener instalados los implementos necesarios para la compilación de programas en lenguaje C en equipos y compilar el archivo con el código en el mismo directorio donde se encuentra los archivos cargas\_X.in Si todas estas condiciones se cumplen, el código funcionará de manera correcta y como se espera.