



UNIVERSIDAD
DE SANTIAGO
DE CHILE

Departamento de Ingeniería en Informática

Análisis de Algoritmos y Estructuras de Datos

Laboratorio 1

Pseudocódigo y Formula de Complejidad

Álvaro Valenzuela

Índice

1. Introducción	2
2. Método	2
3. Resultados y Análisis	3
4. Discusión	3
5. Conclusiones	4
Referencias	5
A. Apéndice	6

1. Introducción

En este laboratorio se abordará un problema de optimización conocido como "Problema de la Mochila" en el cual tenemos analizar y diseñar un algoritmo que pueda resolverlo. En esta oportunidad, el problema debe ser resuelto con utilizando el enfoque de enumeración exhaustiva. Esto quiere decir que el algoritmo debe encontrar todas las posibles soluciones para determinar cual es el más óptimo.

El problema de la mochila corresponde a un problema de complejidad **NP-Completo**. Recordemos que, los problemas de complejidad **P** corresponden a todos los que se pueden resolver en tiempo polinómico (eficiente). Por otro lado, existen los problemas de tipo **NP** que solo pueden ser resueltos en tiempo exponencial. También existe un conjunto llamado **NP-Hard** que se refiere a los problemas al menos son igual o mas difíciles que el problema mas complejo en **NP** y que no necesariamente pertenecen a este conjunto. Esto significa que no todos los resultados quizás puedan ser verificados en tiempo polinómico. Por ultimo, esta el conjunto **NP-Completo** que contiene los problemas catalogados como **NP-Hard** y **NP**, y en caso de ser resueltos, esto se haría en tiempo polinómico.

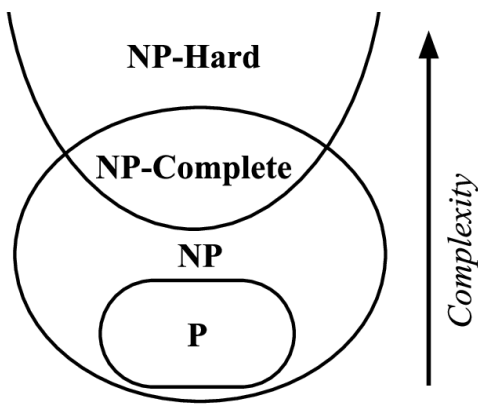


Figura 1: Diagrama complejidad algorítmica

En cuanto a la estrategia para el desarrollo de este laboratorio, utilizaremos un conjunto de archivos de distinto tamaño en donde cada uno de ellos contendrá distintos valores y ponderaciones. Con esto podremos analizar los resultados obtenidos.

2. Método

El laboratorio fue desarrollado en C utilizando un equipo Macbook Pro con un procesador 2,6 GHz Intel Core i7 de seis núcleos y 16 GB de ram.

En cuanto a código, Se disponibilizaron un conjunto de archivos en donde en su nombre se señala la cantidad de registros que se contiene y además la restricción de la ponderación con la cual debemos trabajar. Sobre estos, la estrategia utilizada para encontrar todas las posibles soluciones de cada uno de estos archivos fue la recursión natural. En cada una de las iteraciones se fue completando un arreglo binario en el cual el 1 nos indica tomar uno de estos valores y el 0 descartarlo. Luego, teniendo en cuenta la restricción, en cada una de las iteraciones se fue almacenando la suma de los valores y reemplazando en caso de que la suma de la iteración actual fuese mayor a la anterior.

3. Resultados y Análisis

Dado que la metodología a utilizar para este experimento es de enumeración exhaustiva, es decir, probar cada una de las combinaciones para cada una de las posiciones de un arreglo, lo esperable es que los resultados sean de orden factorial. Esto porque cada vez que se tiene un nuevo elemento en la función, debemos de multiplicar dicho elemento con todas las posibilidades existentes. Es decir, debemos encontrar todas las permutaciones posibles en nuestro arreglo[2]. Lo anterior se puede representar como $\mathcal{O}(n!)$. Para conocer en donde se sitúa este orden de complejidad en comparación a otros, podemos revisar el gráfico número 4 en el apéndice de este documento.

Luego de realizar los experimentos, se obtuvieron los siguientes resultados:

Archivo	Resultado	Tiempo
knapPI_6_500	1461	0.005s
knapPI_8_500	1271	0.006s
knapPI_10_500	2046	0.020s
knapPI_12_500	1952	0.088s
knapPI_14_500	1682	0.432s
knapPI_16_500	2144	2.210s
knapPI_18_500	1956	10.768s
knapPI_20_500	2268	45.102s
knapPI_24_1000	10627	13:11.69s

Cuadro 1: Resultado de ejecuciones

Al observar los resultados, podemos analizar que en los archivos en donde la cantidad de elementos excede de los 16, el tiempo de ejecución se eleva sustancialmente. Es más, el archivo que tiene 28 elementos alcanzó un tiempo de ejecución superior a las 3 horas. Esto último causó que se detuviese el análisis de los demás archivos producto a los largos tiempos de ejecución.

En el apéndice de este documento se encuentra la gráfica número 3 en donde podremos ver los tiempos de ejecución.

4. Discusión

Sobre las limitaciones de la enumeración exhaustiva, la más evidente es que es imposible analizar los archivos que contienen grandes cantidades de elementos para nuestra función. Recordemos que para este experimento, se disponibilizaron 32 archivos pero solamente se pudieron analizar 9 de ellos (28%). Por otro lado, debido a la complejidad algorítmica en la cual se sitúa esta problemática, se observó que incluso con pequeños incrementos de elementos, el tiempo de ejecución subía considerablemente.

Dadas las complejidades mencionadas, la pregunta que surge es investigar si existen otras metodologías para resolver esta problemática. Estudios sugieren que si bien el método de fuerza bruta es el más "óptimo" en cuanto a tiempo de ejecución, también es posible resolverlo utilizando las siguientes estrategias[1]:

1. Branch and bound
2. Dynamic programming
3. Genetic programming

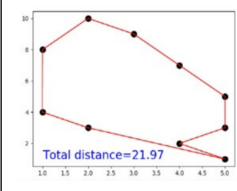
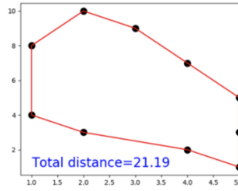
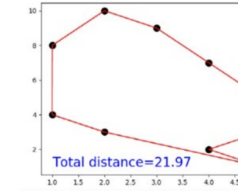
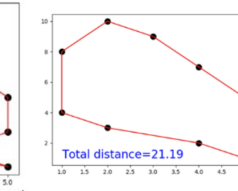
	greedy method	branch and bound method	dynamic programming method	genetic algorithm
Length	21.97	21.19	21.19	21.19
Time	0.00016	13.16	0.025	0.021
Fig				

Figura 2: Comparación de algoritmos [3]

En este experimento podemos corroborar que los tiempos de ejecución utilizando la estrategia de fuerza bruta fueron considerablemente mas cortos que los demás.

5. Conclusiones

En este laboratorio se desarrolló una solución para el problema de la mochila utilizando la técnica de fuerza bruta. Teóricamente, este es un problema de tipo NP-Completo dado que no se puede conocer su solución en tiempo polinómico ni tampoco su solución.

Lo anterior fue corroborado con los resultados obtenidos contrastando estos con sus tiempos de ejecución correspondiente. Se observó que a partir del archivo que contenía 28 elementos, el costo de ejecución era sumamente alto dado que el anterior tomó mas de 3 horas en hacerlo.

Con lo anterior y en base a comparativas con otros estudios podemos concluir que efectivamente este es un problema que no posee una resolución optima, es decir, en tiempo polinómico.

Referencias

- [1] Pedram Ataei. <https://towardsdatascience.com/how-to-solve-the-traveling-salesman-problem-a-c> June 2020. (Accessed on 11/02/2022).
- [2] Jared Nielsen. Big o factorial time complexity — jarednielsen.com. <https://jarednielsen.com/big-o-factorial-time-complexity/>, April 2020. (Accessed on 11/02/2022).
- [3] JianChen Zhang. pdf. <https://iopscience.iop.org/article/10.1088/1742-6596/2083/3/032007/pdf>, 2021. (Accessed on 11/02/2022).

A. Apéndice

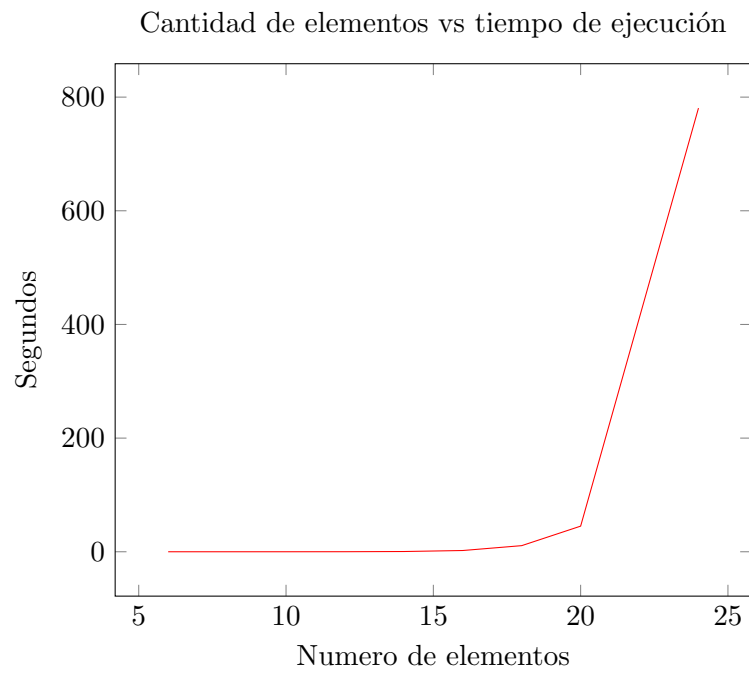


Figura 3: Gráfico de tiempos de ejecución

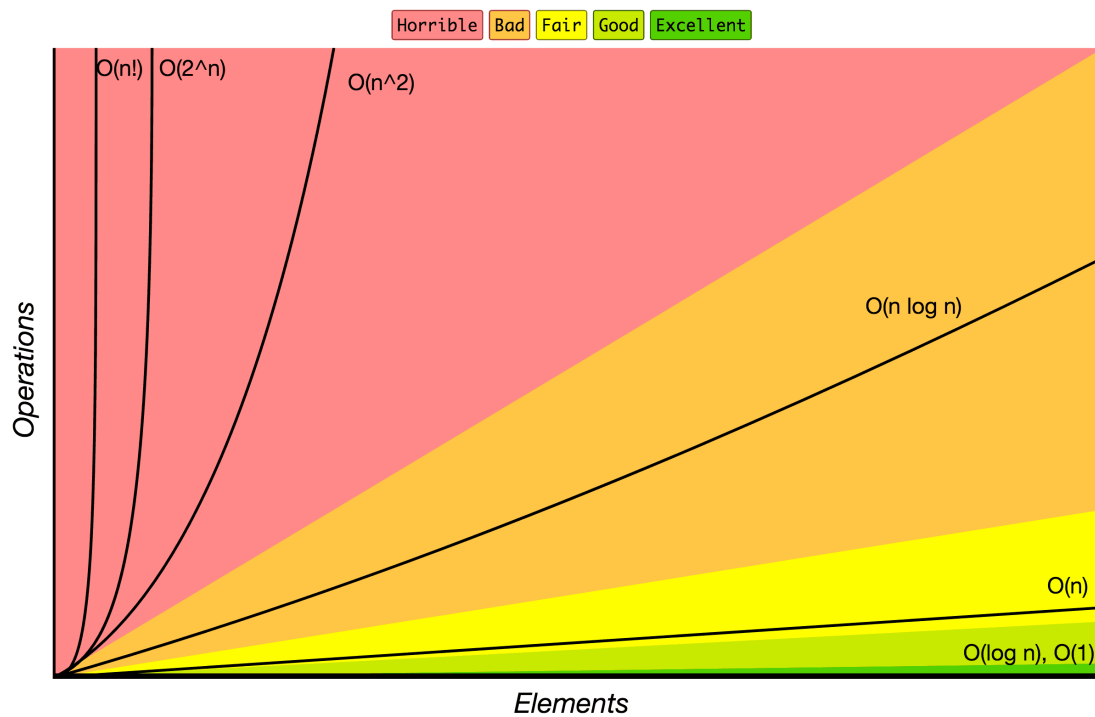


Figura 4: Gráfica comparativa de complejidad algorítmica