.UBAfiuba
FACULTAD DE INGENIERÍA

Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico 1
Algoritmos Greedy

18 de septiembre de 2023

Milena Marchese
100962

Gianni Boccazzi
109304

1. Introducción

El presente trabajo tiene como objetivo el análisis de un problema, en el cual se busca obtener una solución por medio de un algoritmo greedy. Scaloni debe analizar los próximos n rivales de la selección. Cada análisis requiere un tiempo s_i, y a su vez, luego de su análisis, uno de sus n ayudantes debe analizarlo también, los cuales requieren un tiempo a_i, independientemente de quién sea el ayudante. Para que un ayudante pueda comenzar su análisis a un rival, Scaloni **debe haber analizado ese rival primero**. El objetivo es proponer un algoritmo greedy que obtenga la solución óptima, es decir, que Scaloni y sus ayudantes analicen todos los rivales en el menor tiempo posible. La resolución se encuentra en este repositorio [1] de GitHub.

2. Análisis e interpretación del problema

Para interpretar el problema, se propone una situación A. En esta situación, se tienen los siguientes datos:

- Se deben analizar 3 rivales ($\mathbf{n} = 3$), con lo cual, Scaloni tiene 3 ayudantes
- Los tiempos que tarda Scaloni en analizar los rivales son $s_1 = 1$, $s_2 = 3$, $s_3 = 5$
- Los tiempos que tardan los ayudantes en analizar los rivales son $a_1 = 8$, $a_2 = 3$, $a_3 = 1$

Dadas las condiciones mencionadas del problema, una manera de esquematizar la situación podría ser la mostrada en la figura 1.

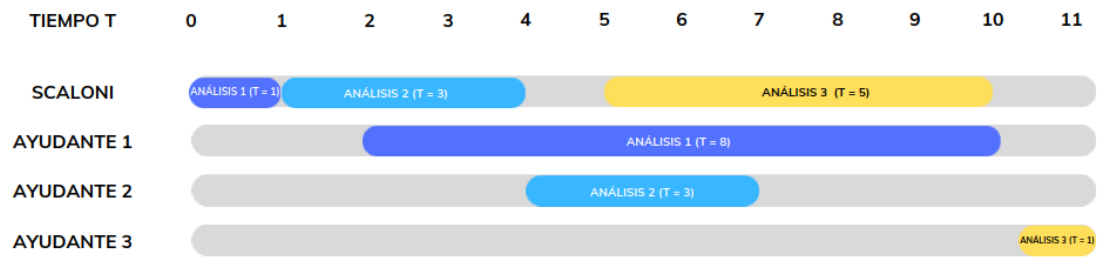


Figura 1: Esquema de solución 1, situación A

Es importante resaltar que cada ayudante puede analizar un problema, ya que siempre habrá la misma cantidad de rivales que de ayudantes. Por ello, se le asigna un solo problema a cada ayudante, y dicho ayudante no puede comenzar su análisis hasta que Scaloni no lo haya analizado. Una vez que Scaloni haya terminado de analizar los rivales, el tiempo total **T** se obtendrá con el último ayudante que finalice su análisis. En este ejemplo, T = 11. Sin embargo, se puede ver claramente que la solución planteada no es la óptima, siendo que, primeramente, Scaloni debería analizar un nuevo rival inmediatamente después de terminar el análisis de un rival previo, y además, un ayudante debería comenzar a analizar el problema justo después que Scaloni finalice su análisis.

De esta forma, se podría suponer que la solución del problema se obtiene si Scaloni analiza primero los rivales en los que le tomará menos tiempo y consecuentemente los que le tomará más. El resultado de la situación A sería T = 10, el cual, por fuerza bruta, se obtiene ese resultado como óptimo.

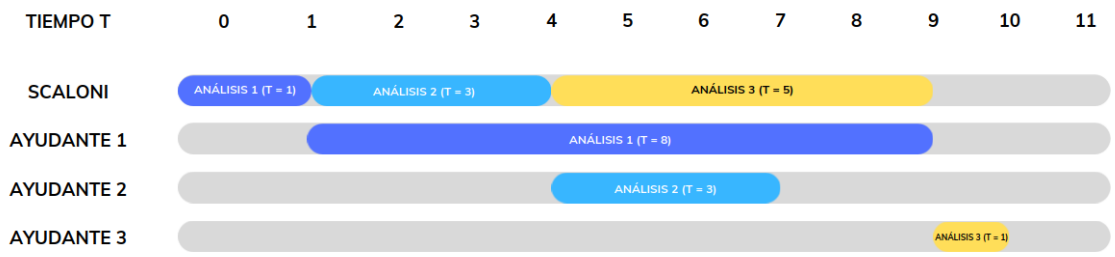


Figura 2: Esquema de solución óptima, situación A

Sin embargo, se puede encontrar un contraejemplo a esta idea. Supongamos ahora que $\boldsymbol{a}_1 = 2$, $\boldsymbol{a}_2 = 8$, $\boldsymbol{a}_3 = 3$ El resultado sería T = 12, lo cual se tiende a pensar que es el resultado óptimo, pero si probamos distintas opciones se llega a una solución mejor, con T = 11.

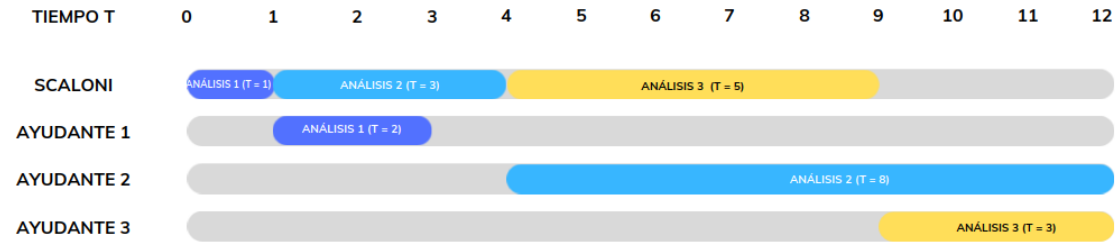


Figura 3: Esquema de solución no óptima, situación B

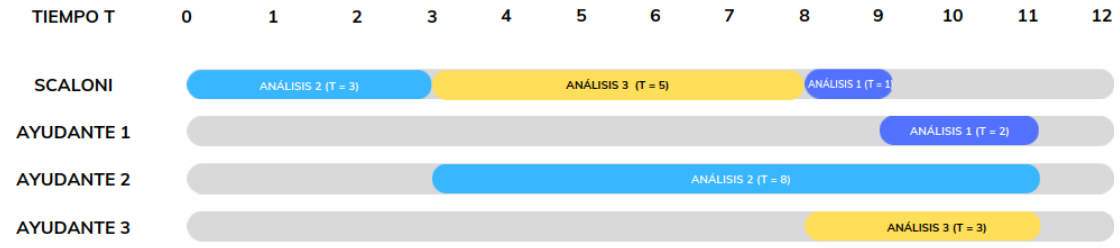


Figura 4: Esquema de solución óptima, situación B

Por lo tanto, la solución planteada, si bien busca una solución óptima local, no lleva a la solución óptima global. Sin embargo, en las dos situaciones, se obtuvo la solución óptima con un mismo patrón. Scaloni y sus ayudantes lograron obtener la solución óptima global cuando el DT de la selección Argentina ordenó la lista de rivales basándose en **el tiempo que tardan sus ayudantes en analizar, de mayor a menor**. Luego, cada ayudante comenzará a analizar un rival inmediatamente después de que Scaloni finalice el análisis del mismo. De esta forma, se obtienen soluciones globales.

3. Algoritmo Greedy para el Análisis de Equipos Rivales

Para abordar esta situación a nivel de código, utilizaremos un arreglo de **n** tuplas, donde cada tupla representará a un rival. En cada tupla, el elemento **0** denotará el tiempo que Scaloni tardará en analizar al rival, mientras que el elemento **1** indicará el tiempo requerido por un ayudante.

Este arreglo de tuplas servirá como base para nuestro algoritmo Greedy, el cual tiene como objetivo optimizar el proceso de análisis de equipos rivales.

A continuación se muestra el código de la solución del problema en Python.

```python
def ayuda_scaloni(arreglo):
    tiempo_scaloni= 0
    tiempo_total = 0
    arreglo_ordenado = sorted(arreglo, key=lambda x: x[1], reverse=True)
    for analisis in arreglo_ordenado:
        tiempo_scaloni += analisis[0]
        tiempo_ayudante = tiempo_scaloni + analisis[1]
        if tiempo_ayudante > tiempo_total:
            tiempo_total = tiempo_ayudante
    return tiempo_total
```

3.1. Análisis temporal del problema

La complejidad temporal de la solución planteada será $\mathcal{O}(n \log n)$ [2] , ya que primeramente se ordena dependiendo el tiempo que tarda cada ayudante, de mayor a menor. Siendo que el problema está planteado en un arreglo de tuplas, a su vez se estarán ordenando los análisis que tiene que hacer Scaloni en su orden ideal para que sea óptimo. El ordenamiento con la función utilizada es $\mathcal{O}(n \log n)$. Luego, se recorre todo el arreglo ordenado y por cada elemento se va sumando en variables el tiempo que va a costar el análisis. Todo lo que se realiza por cada elemento es $\mathcal{O}(1)$. Entonces recorrer todo el arreglo quedaría $\mathcal{O}(n)$ y, quedándonos con la cota más alta, finalmente la complejidad es $\mathcal{O}(n \log n)$.

3.2. Optimalidad

En el problema de Scaloni, que el algoritmo sea óptimo significaría que siempre se obtenga el análisis de los rivales en el menor tiempo posible. Los análisis se ordenan de forma decreciente dependiendo del tiempo de los ayudantes, debido a que el tiempo total de un análisis siempre va a ser un tiempo T = $\boldsymbol{s}_i$ + $\boldsymbol{a}_i$. Por lo tanto, es conveniente que los mayores tiempos de análisis finalicen primero, siendo que el tiempo total va a depender mayormente de los últimos análisis, y cuando estos comienzan. Esto se puede ver claramente en los dos ejemplos de la **situación B**. La primera solución no llega al mejor tiempo posible debido a que $\boldsymbol{a}_2 > \boldsymbol{a}_1$ y además $\boldsymbol{a}_3 > \boldsymbol{a}_1$, y a que siempre su análisis va a comenzar luego de $\boldsymbol{s}_i$.

Se podrían considerar casos bordes para verificar si la solución es válida en cualquier escenario.

Supongamos un arreglo de tuplas donde $\boldsymbol{s}_i > \boldsymbol{a}_i$. Utilizando el algoritmo greedy desarrollado, y esquematizando la situación, el ejemplo resultaría de la siguiente forma.

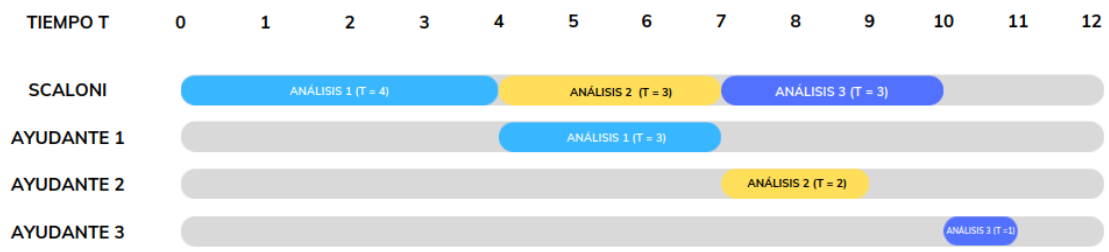


Figura 5: Situación C, $s_i > a_i$.

Como los análisis de Scaloni toman mayor tiempo, es más probable que el tiempo final dependa del último análisis hecho por un ayudante. Por ello, se logra la solución óptima, ya que el último ayudante será el que menos tiempo tome.

Caso contrario, si se propone un esquema donde $a_i > s_i$, el resultado seguiría siendo óptimo y no influiría de ninguna manera, debido a que a_i de mayor tiempo empezarían cuanto antes.

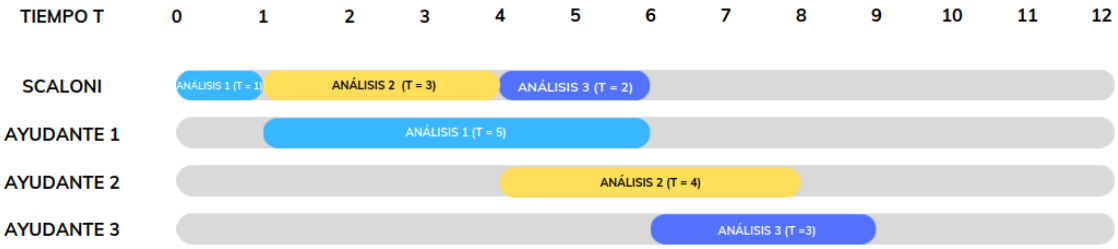


Figura 6: Situación D, $a_i > s_i$.

4. Ejemplos de ejecución y mediciones

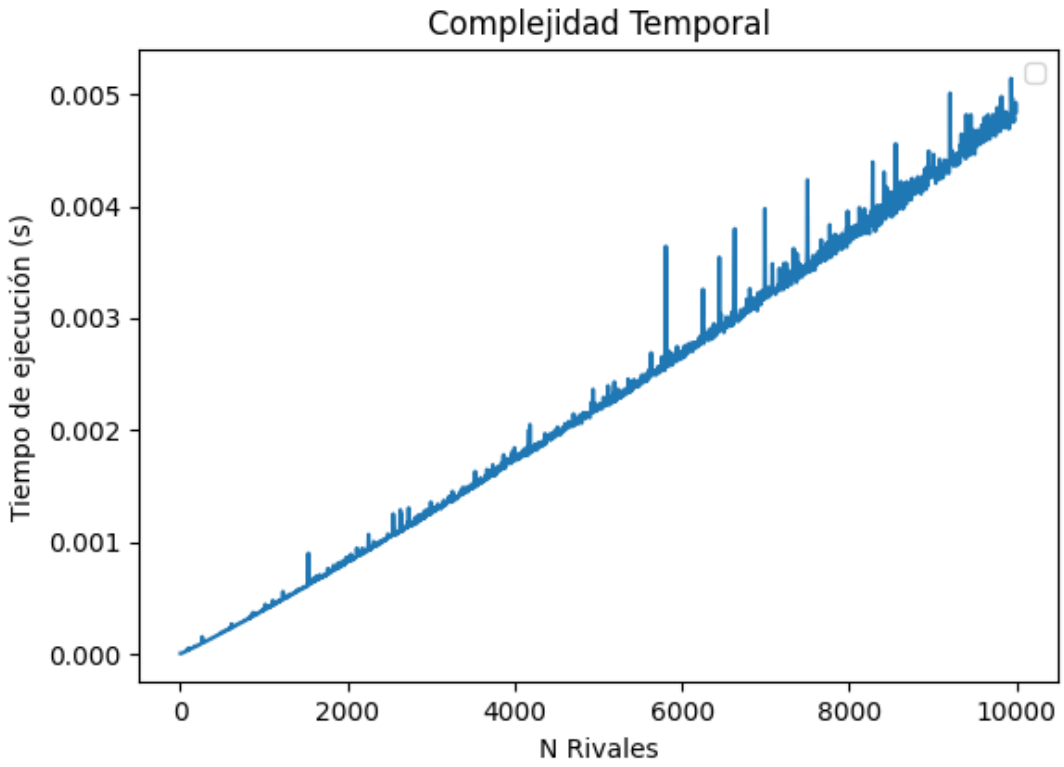
Para los siguientes ejemplos de ejecución, se crearon arreglos de diferentes largos, donde los elementos en cada caso fueron generados por los valores pseudoaleatorios del lenguaje (el módulo `random`). Se realizarán ejemplos de ejecución del algoritmo greedy planteado, y a su vez, se realizarán ejemplos de ejecución ordenando los análisis en orden creciente y decreciente respecto a s_i, y también en orden creciente respecto a a_i.

	Solución óptima	Orden creciente a_i.	Orden decreciente s_i.	Orden creciente s_i.
$\mathbf{n} = 10$	60	68	66	63
$\mathbf{n} = 100$	5795	5894	5870	5861
$\mathbf{n} = 1000$	488676	489673	489596	489068
$\mathbf{n} = 10000$	49830552	49840550	49840396	49840547

Cuadro 1: Tiempo T

Se puede observar claramente la amplia diferencia de eficiencia entre el algoritmo greedy óptimo, con otros algoritmos greedy que no llevan al resultado óptimo global.

Si realizamos un gráfico de tiempo en segundos de lo que tarda el algoritmo con distintas cantidades de rivales (todos con tiempos aleatorios), obtenemos una gráfica aproximada como la siguiente:



Podemos observar que la complejidad temporal sigue la pendiente esperada, como se había previsto en el análisis. Es importante destacar que la variabilidad en los valores de $\boldsymbol{a}_i$ y $\boldsymbol{s}_i$ no tiene impacto en el tiempo de ejecución, ya que el ordenamiento siempre se realiza una sola vez y en ningún caso la complejidad del algoritmo de ordenamiento supera $\mathcal{O}(n \log n)$.

5. Conclusiones

En conclusión encontramos una solución óptima para el problema. Esto es porque partimos sabiendo que Scaloni si o si va a tener que ver todos los videos y tardara lo mismo sea en cualquier orden que los mire, es por eso que comenzamos a paralelizar primero los videos que le van a tomar mas tiempo de ver a sus ayudantes.

Referencias

[1] *Repositorio*. URL: `https://github.com/milemarchese/TDA-buchwald`.

[2] *Sort Function in Python*. URL: `https://www.prepbytes.com/blog/python/sort-function-in-python/`.