.UBAfiuba
FACULTAD DE INGENIERÍA

Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico 1
Algoritmos Greedy

30 de septiembre de 2023

Tomas Caporaletti
108598

Helen Chen
110195

Lucas Garcia Campos
110099

1. Introduccion

El problema que se nos plantea es que el tiempo total para poder analizar todos los siguientes n rivales del **CAMPEÓN DEL MUNDO** sea el más óptimo. A su vez, se solicita que sea hallado con un algoritmo Greedy. Para tener en mente, los algoritmos Greedy siguen una regla sencilla que les permiten obtener un *óptimo local* según el estado actual del programa, y poder llegar a un *óptimo general* juntando los locales. Pero como todo, pueden tener sus ventajas y desventajas, como por ejemplo que no siempre dan el resultado óptimo, o que demostrar que el resultado es óptimo es difícil. Por otro lado, son intuitivos de pensar y fácil de entender, y suelen ser rápidos. Lo que en reglas generales suele indicar que el algoritmo es Greedy es el uso de colas de prioridad como el *heap* u ordenamientos (recalcar que un algoritmo puede hacer uso de heaps u ordenamientos y no ser Greedy).

Dicho esto, empezamos a plantear posibles soluciones para el problema. Lo primero que se nos vino a la idea es hacer uso de algún tipo de ordenamiento, ya sea ordenando por los tiempos de Scaloni o los ayudantes, en relación a cuánto tardaría cada uno. Lo que nos ayudó a volcarnos por el lado de ordenar por los tiempos de los ayudantes fue el siguiente:

- El tiempo total que tarda Scaloni siempre va a ser el mismo, no importa cómo ordenemos los videos a ver. Bien como se dice, *."El orden de los factores no altera el producto"*.
- Por más que Scaloni termine de ver el último video del último rival, va a quedar que después un ayudante analice el video, por lo que es más importante el tiempo que va a tardar este último ayudante que el que va a tardar Scaloni.

Ahora bien, ya tenemos definido por donde queremos encarar el problema, pero todavía falta definir en qué orden queremos que los videos se visualicen dependiendo de los ayudantes, si los más rápidos primero o viceversa. Acá entra en juego un factor muy importante a tener en cuenta: **los ayudantes analizan los videos inmediatamente termina Scaloni de ver el video, y el análisis de cada ayudante es independiente a los otros**. Esto quiere decir que si Scaloni terminó un video, inmediatamente uno de los ayudantes se pondrá a analizarlo. Y si Scaloni termina otro video, otro ayudante podrá empezar a analizar ese video, no importa si el anterior terminó de realizar su análisis o no. Esto nos ha llevado a tomar la decisión de ordenar por el tiempo de los ayudantes del que más tarde al que menos, por los siguientes puntos:

- Los ayudantes pueden analizar un video independientemente de si el anterior haya terminado o no su análisis.
- Si para el último video queda el ayudante que mas tarda, el tiempo total no seria el optimo sino todo lo contrario, ya que se tardaría el tiempo total de Scaloni más lo que tarde este último.
- Por ese motivo, conviene que los ayudantes que más tarden estén al principio, ya que tienen tiempo hasta que Scaloni termine todos los videos para terminar. Y los ayudantes que menos tardan, estarán al final, de modo que si Scaloni termina, los que les falten sabemos que son los más rápidos y terminan lo antes posible.

2. Algoritmo para encontrar el tiempo óptimo

A continuación se detallan el código y los pasos que se siguieron para llevar a cabo el algoritmo planteado.

2.1. Obtener el tiempo

Una vez que se obtuvieron los datos, se los ordena según el tiempo de los ayudantes de mayor a menor, y luego podemos conseguir el tiempo total que se tardará en analizar todos los rivales:

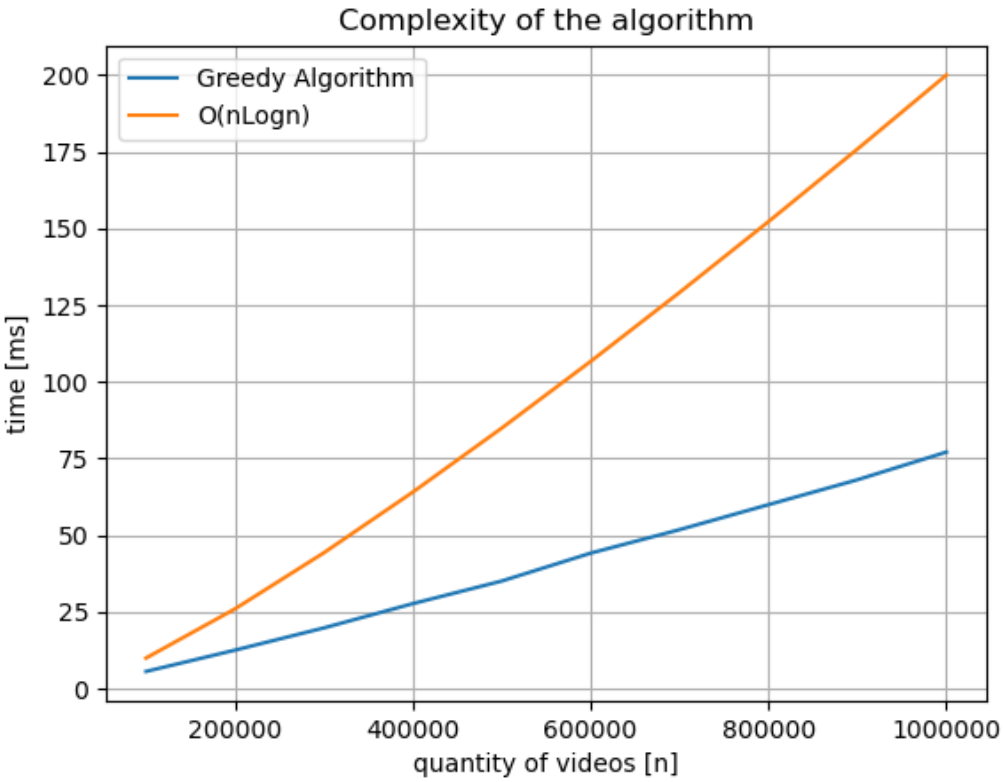
```
import input

def main(data = None):
    params = []
    if not data:
        params = input.ReadInputs()
    else:
        params = data
    params.sort(key = lambda item: 1/item[1])

    total, longest, actual = 0, 0, 0
    for t in params:
        total += t[0]
        actual = total + t[1]
        if actual > longest:
            longest = actual
    print("It took: " , longest , "hs")

if __name__ == "__main__":
    main()
```

La complejidad del algoritmo para ordenar todos los datos es $\mathcal{O}(n \log n)$, ya que el método *sort* de Python tiene esa complejidad (complejidad del método *sort*) y la función auxiliar que usa como *key* para ordenar es $\mathcal{O}(1)$. Luego, recorrer todo los datos ya ordenados e ir procesando la información es $\mathcal{O}(n)$, ya que solo recorre el arreglo y va sumando los valores correspondientes a las variables declaradas, y eso es $\mathcal{O}(1)$. La complejidad final es $\mathcal{O}(n \log n + n)$.

La variable *total* se utiliza para ir contabilizando el tiempo que tarda Scaloni, sumando el tiempo del video que está analizando en el momento más todos los ya analizados. A su vez, tenemos la variable *actual*, que se encarga de guardar el tiempo que va a requerir el video que se está analizando actualmente, que es la suma del tiempo que se lleva en total más la que vaya a tardar el ayudante. Y por último tenemos la variable *longest*, que se encarga de almacenar el video que vaya a tardar más en analizarse. Es necesaria ya que nada nos garantiza que el último ayudante vaya a ser el que más influencia en el tiempo total de los análisis de los videos. Un ejemplo para mostrar esto es ir a la exageración: Que el primer video de todos, el ayudante tarde 5 meses. Por más que el último tarde 2 hs, la duración total va a ser lo que tarde el analisis que más se aleje de los ya terminados, tanto por Scaloni y los demás ayudantes.

3. Mediciones

Se realizaron mediciones en base a crear pruebas de distintos tamaños y tomar su tiempo de ejecución individualmente, y en base a los datos recolectados hacer el gráfico. Los datos fueron desde 100000 hasta 1000000 elementos, los cuales los tiempos tanto de Scaloni como los ayudantes para cada uno de los rivales fue un número aleatorio desde 1000 hasta 100000.



Como se puede apreciar, el algoritmo Greedy efectivamente fue veloz, tal como se los suele caracterizar. A su vez, se puede ver que se asemeja a la complejidad indicada, ya que la tendencia es aproximadamente lineal en términos de $\mathcal{O}(n \log n)$.

4. Conclusiones

En resumen, hemos logrado optimizar significativamente el tiempo requerido para analizar todos los rivales mediante la implementación de un algoritmo Greedy. Este enfoque nos ha permitido descomponer el problema general en partes más manejables, lo que ha simplificado su resolución gradual y ha reducido considerablemente la complejidad computacional. Como resultado, hemos evitado la necesidad de analizar todos los posibles escenarios exhaustivamente.

Además, hemos respaldado con mediciones empíricas que confirman la eficiencia y la complejidad esperada. Estos resultados refuerzan la validez y la utilidad de nuestra estrategia para abordar problemas complejos de manera efectiva y eficiente.

Finalmente, valoramos la oportunidad de colaborar en esta tarea crucial y estamos determinados a contribuir al éxito continuo de la selección **CAMPEONA DEL MUNDO**.

5. Anexo

5.1. Introduccion (reentrega)

...

Dicho esto, empezamos a plantear posibles soluciones para el problema. Lo primero que se nos vino a la mente es la idea de hacer uso de algún tipo de ordenamiento, ya sea ordenando por los tiempos de Scaloni o los ayudantes, en relación a cuánto tardaría cada uno. Lo que nos ayudó a volcarnos por el lado de ordenar por los tiempos de los ayudantes fue el siguiente:

- El tiempo total que tarda Scaloni siempre va a ser el mismo, no importa cómo ordenemos los videos a ver. Sea s_i el tiempo que tarda Scaloni para analizar el rival r_i y haciendo uso de la propiedad conmutativa de la suma se demuestra:

$$\sum_{i=1}^{n} s_i = s_1 + s_2 + ... + s_n = s_n + s_{n-1} + s_{n-2} + ... + s_1 = \sum_{i=1}^{n} s_i$$

- Por más que Scaloni termine de ver el último video del último rival, va a quedar que después un ayudante analice el video, por lo que es más importante el tiempo que va a tardar este último ayudante que el que va a tardar Scaloni.

Ahora bien, ya tenemos definido por donde queremos encarar el problema, pero todavía falta definir en qué orden queremos que los videos se visualicen dependiendo de los ayudantes, si los más rápidos primero o viceversa. Acá entra en juego un factor muy importante a tener en cuenta: **los ayudantes analizan los videos inmediatamente termina Scaloni de ver el video, y el análisis de cada ayudante es independiente a los otros**. Esto quiere decir que si Scaloni termina un video, inmediatamente uno de los ayudantes se pondrá a analizarlo. Y si Scaloni termina otro video, otro ayudante podrá empezar a analizar ese video, no importa si el anterior terminó de realizar su análisis o no. Esto nos ha llevado a tomar la decisión de ordenar por el tiempo de los ayudantes del que más tarde al que menos, por los siguientes puntos:

- Los ayudantes pueden analizar un video independientemente de si el anterior haya terminado o no su análisis.
- Sea t_i el tiempo total tomado para analizar al rival r_i, s_i el tiempo que tarda Scaloni y a_i el tiempo que tarda el ayudante en analizar ese video:

$$\sum_{i=1}^{n} t_i = \sum_{i=1}^{n} s_i + a_i = s_1 + a_1 + s_2 + a_2 + ... + s_n + a_n$$

  Suponiendo por ejemplo que el análisis hecho por a_n es el que más tarda, y teniendo en cuenta que cada ayudante es independiente al resto, el tiempo total para analizar todos los videos va a ser

$$t_{total} = t_{scaloni} + a_n$$

  donde $t_{scaloni}$ es el tiempo total de Scaloni en analizar todos los rivales. Entonces si el ayudante a_n es el que más tarda, el tiempo no será el óptimo sino todo lo contrario, será el peor de todos ya que el que más tarda es el último en empezar a analizar a su correspondiente rival.
- Por ese motivo, conviene que los ayudantes que más tarden estén al principio, ya que tienen tiempo hasta que Scaloni termine todos los videos para terminar. Y los ayudantes que menos tardan, estarán al final, de modo que si Scaloni termina, los que les falten sabemos que son los más rápidos y terminan lo antes posible, acortando el valor de a_n.

5.2. Algoritmo para encontrar el tiempo óptimo (reentrega)

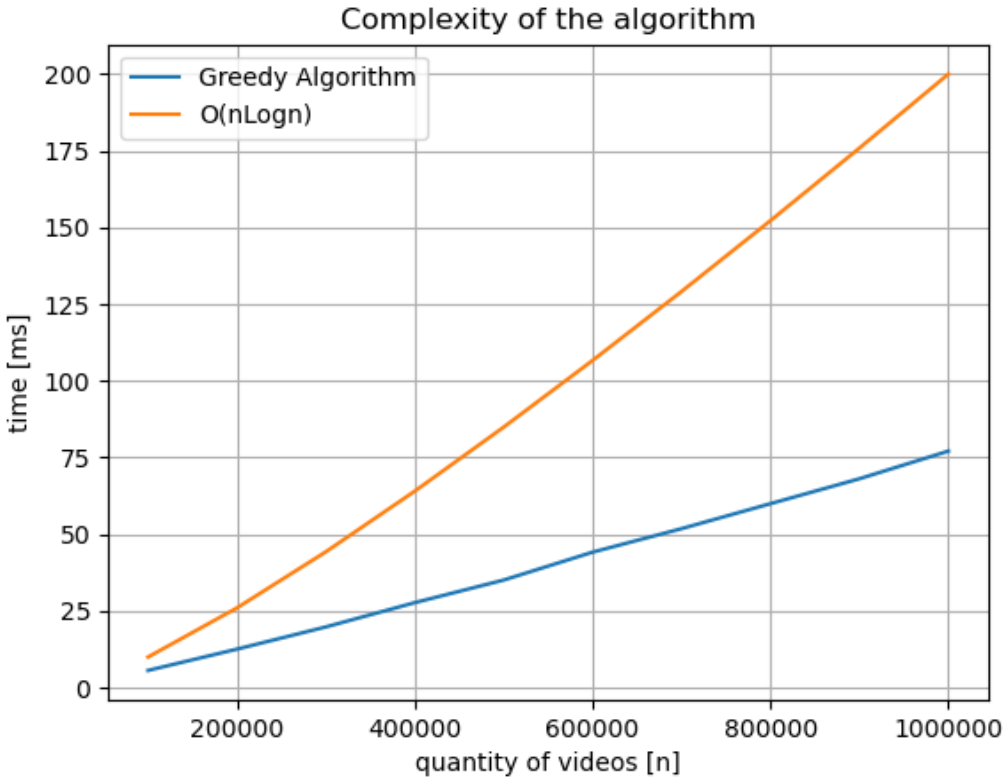
...[1]

La complejidad del algoritmo para ordenar todos los datos es $\mathcal{O}(n \log n)$, ya que el método *sort* de Python tiene esa complejidad (complejidad del método *sort*) y la función auxiliar que usa como *key* para ordenar es $\mathcal{O}(1)$. Luego, recorrer todo los datos ya ordenados e ir procesando la información es $\mathcal{O}(n)$, ya que solo recorre el arreglo y va sumando los valores correspondientes a las variables declaradas en $\mathcal{O}(1)$. Por lo tanto, la complejidad final del algoritmo propuesto es $\mathcal{O}(n \log n)$.

...

5.3. Mediciones (reentrega)

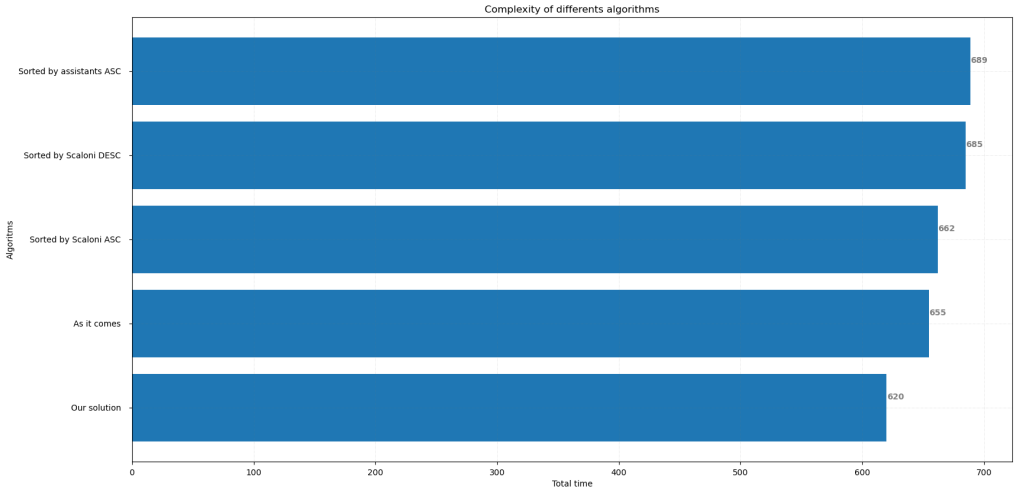
Se realizaron mediciones en base a crear pruebas de distintos tamaños y tomar su tiempo de ejecución individualmente, y en base a los datos recolectados hacer el gráfico. Los datos fueron desde 100.000 hasta 1.000.000 elementos, los cuales los tiempos tanto de Scaloni como los ayudantes para cada uno de los rivales fue un número aleatorio desde 1.000 hasta 100.000.



Como se puede apreciar, la complejidad del algoritmo se asemeja a la de $\mathcal{O}(n \log n)$, bien como se analizó anteriormente. El hecho de que varíe la pendiente de las funciones no se debe dar mucha importancia, sino que lo importante es que en términos de "linealidad."mbas se asemejan.

Por otro lado, se han hecho mediciones de otros algoritmos para poder comparar con el propuesto por nosotros:

[1] Aclaración: se ha suprimido el código donde se obtenían los datos en la parte del informe original.



- `Our solution`: algoritmo greedy propuesto por nosotros.
- `As it comes`: algoritmo que analiza los rivales en el orden en el que llegan.
- `Sorted by Scalini ASC`: algoritmo que ordena los análisis en función de lo que tarde Scaloni en orden ascendente.
- `Sorted by Scalini DESC`: algoritmo que ordena los análisis en función de lo que tarde Scaloni en orden descendente.
- `Sorted by assistants ASC`: algoritmo que ordena los análisis en función de lo que tarden los ayudantes en orden ascendente.

Como se puede observar, el algoritmo que hace que el tiempo total requerido para que se analicen todos los rivales sea el más grande es el que ordena los análisis en función de los ayudantes de forma ascendente. Como se explicó anteriormente, esto es así porque al estar el ayudante que más vaya a tardar en el último análisis, el tiempo total de todo será el tiempo total de Scaloni en analizar los videos mas lo que vaya a tardar este último. Justamente todo lo contrario a nuestro algoritmo propuesto, que lo inserta al principio para que mientras Scaloni y el resto de ayudantes terminan de hacer sus análisis, sus tiempos se superpondran con lo que tarde el ayudante y tenga posibilidades de terminar antes que el resto de los análisis están listos. Por otra parte, el hecho de ordenar los videos segun lo que tarde Scaloni, ya sea de forma ascendente o descendente, no implica que el tiempo total se acorte, ya que de eso depende exclusivamente los ayudantes en ver cuando se terminara el ultimo analisis y donde se sitúan los que más tarden. Y por último, analizar los rivales tal cual llegan, es más un factor de suerte o aleatoriedad ya que el ayudante que más tarda puede ser tanto el primero como el último.