.UBAfiuba
FACULTAD DE INGENIERÍA

Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico 1

Fecha de entrega: 18/09/2023

- Gauler, Ian Benjamin, Padrón: 109437
- Jauregui, Melina Belén, Padrón: 109524
- Tourne Passarino, Patricio, Padrón: 108725

Contents

1 Introducción

En el presente informe, se tiene como objetivo llevar a cabo un análisis del diseño implementado para abordar la problemática planteada por Scaloni, que consiste en la optimización máxima del tiempo total empleado en la realización de un análisis exhaustivo de sus rivales. A lo largo de este, se expondrán los criterios empleados y se proporcionará su justificación.

1.1 Información de la problemática

El enunciado propuesto, nos da la siguiente información:

- Cada compilado lo debe analizar Scaloni y alguno de sus ayudantes.
- El análisis del rival i le toma s_i tiempo a Scaloni y luego a_i tiempo al ayudante.
- Scaloni cuenta con n ayudantes, siendo n la cantidad de rivales a analizar. Además, cada ayudante puede ver los videos completamente en paralelo a Scaloni y a los respectivos ayudantes.
- Al momento en que Scaloni haya terminado de analizar el i ésimo video, comenzará inmediatamente algún ayudante a analizarlo, para no desperdiciar ningún segundo.
- Sólo un ayudante verá el video, dado que no aporta mayor ganancia que dos ayudantes lo vean.

2 Análisis de diferentes soluciones propuestas

Vamos a proponer dos escenarios posibles para poder analizar sus soluciones.

Estos dos escenarios los vamos a representar mediante un gráfico cada uno:

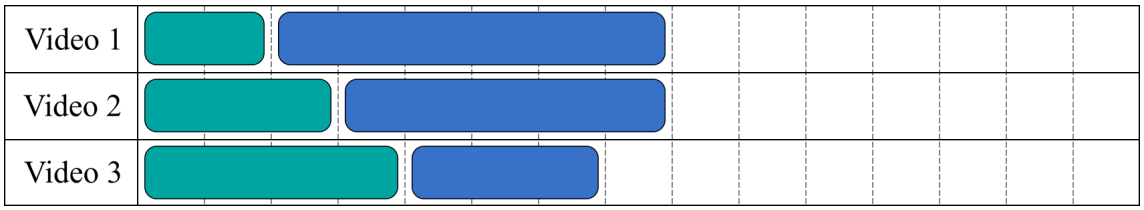


Figure 1: El tiempo que le tarda a Scaloni ver cada video es relativamente corto comparado con sus ayudantes.

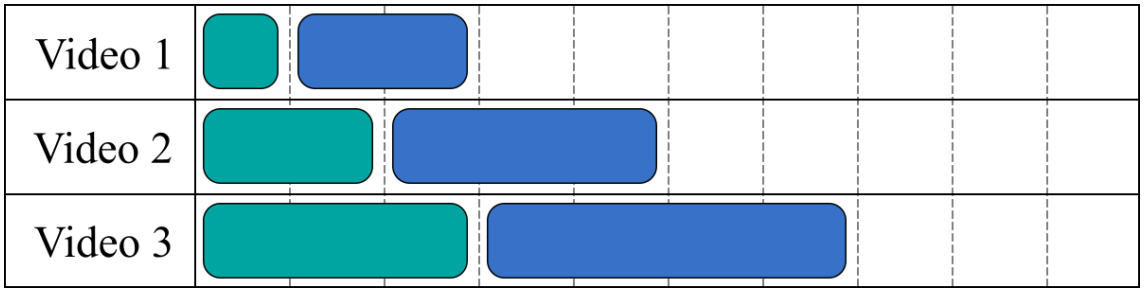


Figure 2: El tiempo que le tarda a los ayudantes ver cada video es proporcional a lo que le tarda a Scaloni.

En primer lugar, analizando ambos casos pudimos observar las siguientes características:

- El orden en que Scaloni visualiza los compilados no incide en el tiempo que le toma a él en finalizar la revisión de los mismos.

- Minimamente, el tiempo final va a ser $\sum_{k=1}^{n} s_k + a_n$, $siendo a_n el ayudante asignado a ver el último video. Este caso es el mí$

  Tomando esto en cuenta, los algorítmos propuestos a continuación ignoran el tiempo que le tarda al DT ver cada compilado.

  Para poder resolverlos se propusieron los siguientes algorítmos:

2.1 Solución 1

- Ordenar los compilados de menor a mayor tiempo de análisis por parte de los ayudantes.
- Asignarle a Scaloni el compilado de menor tiempo de análisis por parte de los ayudantes, y a medida que termina un compilado, que analice los demás en orden creciente.
- Asignarle a cada ayudante el compilado que acaba de terminar Scaloni.

2.2 Solución 2 - La Solución Óptima

- Los asistentes realizan el análisis de cada uno de los compilados asignados en paralelo. Por lo tanto, el tiempo que se invierte en la revisión de un compilado específico de máxima duración, puede ser aprovechado de manera tal que este sea visto mientras Scaloni se dedica a la revisión de otros compilados. De esta forma, nos aseguramos que se minimice el tiempo que suman los ayudantes en la resvisión total.
- También en esta solución tiene en consideración el análisis 2 descripto previamente, ya que en el mejor de los casos, la $\sum_{k=1}^{n} s_k + a_n$ $termina siendo la mínima ya que, por la forma del ordenamiento del algorít$

Para ello propusimos el siguiente algoritmo:

En primer lugar, hemos definido una clase llamada `Compilado` para modelar el compilado de cada oponente, con los atributos `tiempo_scaloni` y `tiempo_ayudante`, que almacenan el tiempo que le lleva analizarlo a Scaloni y a algún ayudante, respectivamente.

```
1 class Compilado:
2     def __init__(self, scaloni, ayudante):
3         self.tiempo_scaloni = scaloni
4         self.tiempo_ayudante = ayudante
```

De esta forma, y teniendo en cuenta los criterios previamente detallados, definimos la función `compilados_ordenados_de_forma_optima` que recibe como parámetro un arreglo con elementos de la clase `Compilado`. Esta ordena el arreglo en función del tiempo requerido por los asistentes para visualizar cada compilado, en orden descendente.

```
1 def compilados_ordenados_de_forma_optima(compilados):
2     return sorted(compilados, key=lambda compilado: compilado.tiempo_ayudante,
      reverse=True)
```

2.3 Cota de complejidad temporal

El algoritmo presentado tiene una complejidad temporal de $O(n \log n)$. Esto se debe a que seguimos los siguientes pasos: Ordenamos los compilados según el tiempo de análisis de los ayudantes. Para ello utilizamos sorted de la librería de python, que usa por detrás el algoritmo de Timsort, teniendo este una complejidad de $O(n \log n)$.

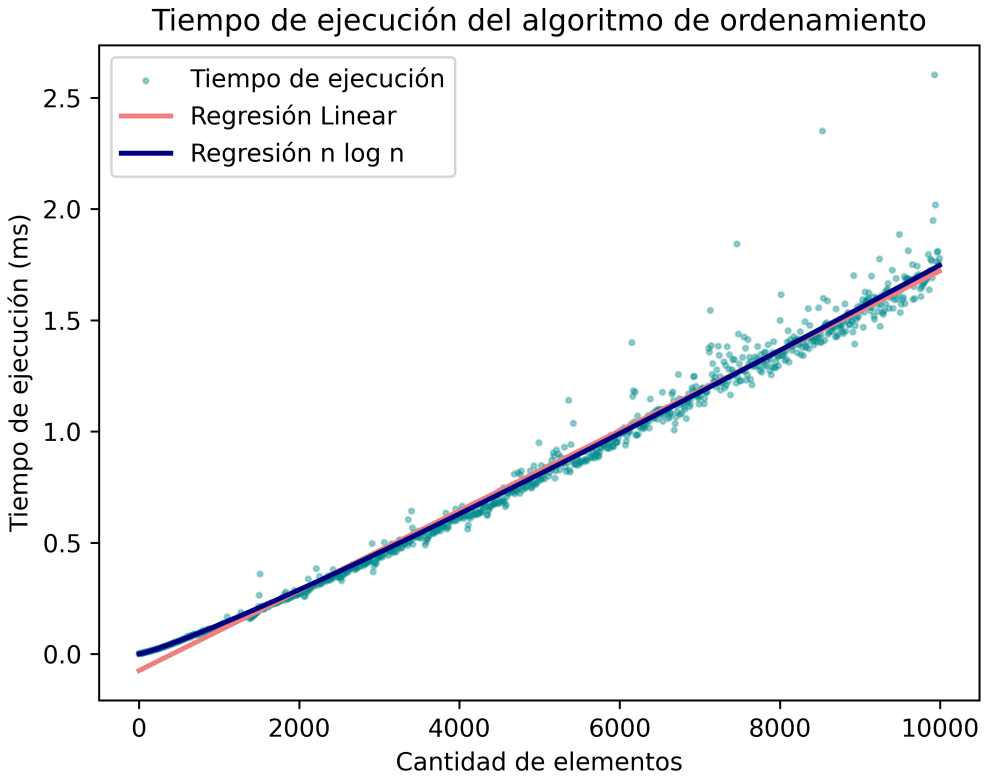
3 Mediciones

Para las mediciones, creamos listas de compilados de ejemplo de forma aleatoria. Decidimos que el tiempo de análisis de un compilado fuera un valor aleatorio entre 1 y 99'999, basándonos en los datos de ejemplo provistos por la cátedra. Medimos el tiempo de ejecución de nuestro algoritmo para ciertas cantidades de compilados.

Calculamos una regresión lineal y una regresión lineal logaritmica que se ajuste a nuestros datos y graficamos sus curvas. Ahora, debemos comprobar cuál de las dos curvas se ajusta mejor a nuestros datos. Para ello, usamos la raíz del error cuadrático medio, que nos da una idea de cuán cerca están los valores predichos de los valores reales.

Cuando calculamos el error cuadrático medio para ambas curvas, observamos que el error era prácticamente similar. Por lo que decidimos hacer zoom a los gráficos para observar cómo se estaban comportando los datos, y pudimos observar que en valores pequeños de muestras, la curva lineal logaritmica se ajustaba mejor a los datos. Sin embargo, para valores mayores, las diferencias de tiempos fueron tendiendo a una forma lineal, y no solamente eso, sino que la curva lineal se ajustaba mejor a los datos que la logaritmica. Por lo que decidimos incrementar el número de muestras para cantidades de compilados menores, y disminuir el número de muestras para cantidades de compilados mayores. De esta forma, pudimos observar, y mediante el error cuadrático medio, comprobar que la curva lineal logaritmica se ajustaba mejor a nuestros datos.

o usamos un rango continuo porque el tiempo de ejecución para pequeñas cantidades de compilados reflejaba mejor la complejidad lineal logaritmica. Por lo que al usar un rango continuo, observamos que



De esta forma pudimos comprobar empíricamente que la complejidad tiende a $O(n \log n)$.

4 Conclusiones

Finalmente, consideramos que la solución óptima para abordar la problemática de Scaloni sería que éste analizara los compilados en función del tiempo requerido por los asistentes para visualizar cada uno, organizándolos en orden descendente. Esta estrategia permitiría resolver el problema con una complejidad algorítmica de orden $O(n \log n)$. Este enfoque garantiza la máxima eficiencia en la visualización de los videos y permite que el tiempo invertido por Scaloni y sus asistentes se administre de manera óptima.