Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico 1
Algoritmos Greedy

18 de septiembre de 2023

Nombre	Padron
Buono, Fernando	105935
Katta, Gabriel	103523
Jarmolinski, Arian	94727

1. Análisis del Problema

La problemática a ser estudiada en el siguiente trabajo consiste en un problema de **scheduling** en donde el Director Técnico de la Selección Argentina debe analizar a n rivales con la ayuda de sus n asistentes. Cada vídeo le toma a los asistentes un tiempo a_i y al director técnico un tiempo s_i. Además, los vídeos deben siempre ser **vistos por Scaloni antes de que los asistentes puedan en paralelo verlos**. Se busca un algoritmo que le ayude a Scaloni a organizarse con sus asistentes y terminar de ver los n vídeos de sus rivales en el tiempo óptimo.

1.1. Consideraciones Iniciales

Se empezó el análisis usando como punto de referencia el panorama que tendría Scaloni al terminar de ver un vídeo y acto seguido encargar a su asistente a empezar a verlo mientras que el arranca con el siguiente rival. En ese momento se analizaron los siguientes incógnitas:

- Una vez que termine el asistente ¿Ese sera el tiempo máximo transcurrido?
- ¿Existe algún otro asistente que aun no termine de analizar su vídeo?
- ¿Que importancia tiene el tiempo de Scaloni?
- ¿Que importancia tiene el tiempo de los asistentes?

Teniendo en consideración estas preguntas se llegó a una regla sencilla típica de un algoritmo greedy, la cual consiste en que Scaloni se haga la pregunta de que si la tanda de análisis de vídeos actual, es decir, el acumulado de los vídeos que ha visto más lo que tardara el asistente, terminará después de alguno de los asistentes que sigue trabajando en segundo plano.

1.2. Primera Iteración del Algoritmo

Como respuesta a esta problemática planteamos el siguiente algoritmo base:

```
def analizar_rivales(videos):
  scaloni_total = 0
  t_total = 0
  for video in videos:
    scaloni_total += video[0]
    t_total = max(t_total, scaloni_total + video[1])
  return t_total
```

Sin embargo, esta no es la solución óptima ya que es altamente susceptible al orden en que llegan los vídeos.

Análisis de la Solución

2. Ordenamiento Óptimo

Como se menciono anteriormente, ya contamos con la regla sencilla de nuestro algoritmo greedy, sin embargo, este aun necesita un ordenamiento que prepare el dataset para maximizar la eficacia de dicha regla, para esto se plantearon diversos ordenamientos para llegar a la solución óptima.

```
def analizar_rivales(videos, ordenamiento):
  ordenamiento(videos)
  s_total = 0
  t_total = 0
  for video in videos:
    s_total += video[0]
    t_total = max(t_total, s_total + video[1])
  return t_total
```

Nota: Recordemos que los videos tienen un formato (s_i, a_i)

Vale la pena destacar que para el ordenamiento estamos usando la función built-in de Python, la cual según la documentación en el peor de los casos añade tiene un costo de complejidad de $O(n \cdot log(n))$.

2.1. Ordenar por Tiempos de Scaloni

Lo mas intuitivo al inicio del trabajo practico, fue simplemente analizar primero los vídeos que le llevaran mas tiempo a Scaloni.

```
def orden_scaloni(videos):
   videos.sort(key = lambda x: x[0])
```

Sin embargo, durante el análisis nos dimos cuenta de lo siguiente: El acumulado de tiempo que tarda Scaloni con sus vídeos **siempre formara parte de cualquier solución, y siempre sera el mismo**. Esto se debe a que el enunciado indica que el debe ver todos los vídeos y los asistentes solo pueden ver vídeos que el ya haya visto. Por ende, usar esto como criterio de ordenamiento no tiene ningún impacto positivo sobre la solución final.

2.2. Ordenar por la Suma de los Tiempos

Luego, se probo tomando en cuenta la suma total tanto de Scaloni como los Asistentes para cada uno de los vídeos, de esta manera el vídeo que consuma la mayor cantidad de tiempo sean los primeros que sean vistos por Scaloni.

```
def orden_suma(videos):
   videos.sort(key = lambda x: x[0] + x[1], reverse=True)
```

2.3. Ordenar por la relación entre los Tiempos

Tomando un poco de inspiración del problema de la mochila se pensó que una relación entre el tiempo de Scaloni y el del Asistente (s_i/a_i) podría optimizar el orden de llegada de los vídeos, especialmente para aquellos casos en que la diferencia de magnitud sea notable.

```
def orden_relacion_si_ai(videos):
   videos.sort(key = lambda x: (x[0]/x[1]))
```

2.4. Ordenar por los Tiempos de los Asistentes

Por ultimo pero no menos importante, simplificamos un poco el análisis y se decidió ordenar en orden descendente los vídeos por el tiempo que le tome al asistente. Esta solución fue la que

nos pareció mejor, ya que le da completa prioridad a los asistentes en lugar de a Scaloni, el cual ya planteamos que el orden en que ve los vídeos no es relevante.

```
def orden_asistentes_desc(videos):
    videos.sort(key = lambda x: (x[1]), reverse = True)
```

Nota: Como se puede notar, los últimos 3 ordenamientos tienen de alguna manera en consideración a los tiempos de los asistentes, sin embargo, no podemos aun inferir cual de estos nos resulta en el óptimo sin hacer análisis de performance y optimalidad.

3. Análisis de Complejidad

El algoritmo greedy propuesto ya cumple con sus dos ingredientes principales:

- Un ordenamiento de los datos.
- Una regla simple que permite calcular un máximo local.

Podemos ahora proceder con el análisis de complejidad resultante tras haber armado nuestro algoritmo:

```
def analizar_rivales(videos, ordenamiento):
  ordenamiento(videos)
  s_total = 0
  t_total = 0
  for video in videos:
    s_total += video[0]
    t_total = max(t_total, s_total + video[1])
  return t_total
```

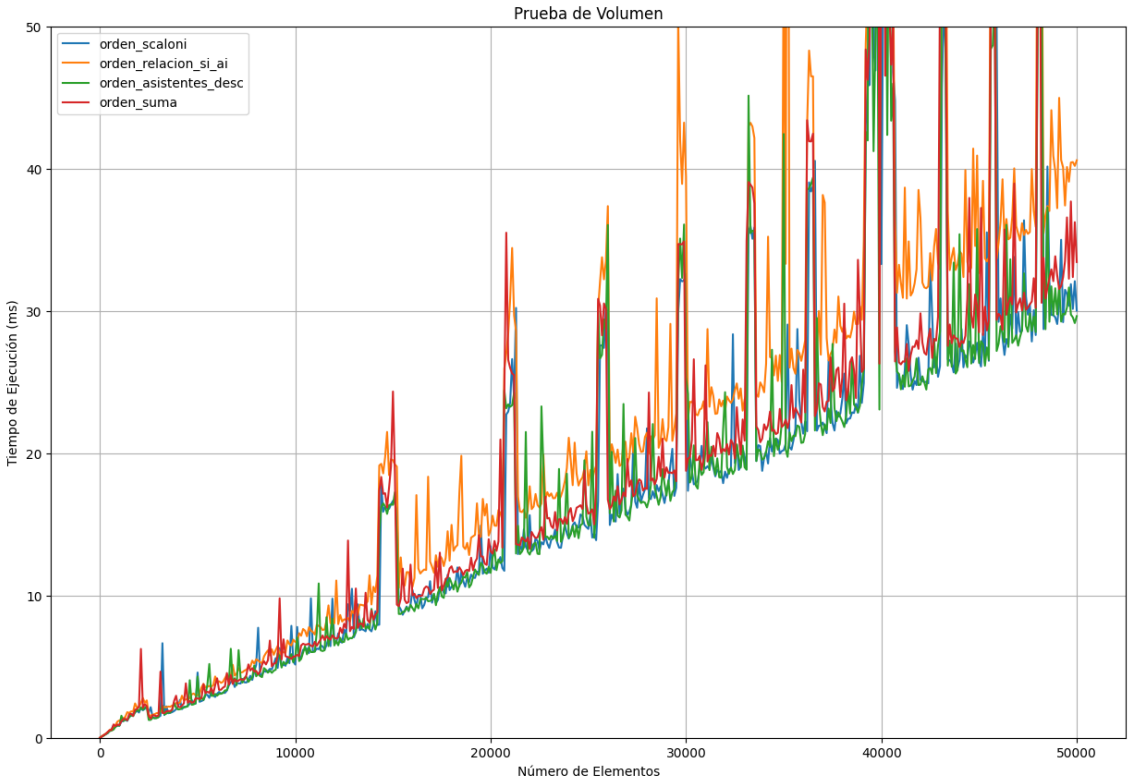
- La primera operación realizada por el algoritmo es aquella de ordenamiento, como se menciono previamente, la librería estándar de Python provee el método ***sort*** el cual tiene una complejidad algorítmica de $O(n \cdot log(n))$.
- Luego de dicho ordenamiento, simplemente se declaran variables, lo cual conocemos que es $O(1)$.
- Acto seguido se empieza a recorrer cada uno de los vídeos, este proceso completo se puede asegurar que contiene una complejidad algorítmica $O(n)$.
- Por ultimo, ahora dentro del ciclo, solo se hacen operaciones $O(1)$ como son la suma y el máximo.

Luego de listar todas las operaciones del algoritmo, podemos concluir que su complejidad algorítmica final es de: $O(n \cdot log(n))$. Esto porque el ordenamiento es la operación de mayor complejidad y podemos obviar el recorrido lineal de los vídeos.

3.1. Análisis de Tiempos de Ejecución

Para corroborar que la complejidad analizada es correcta, se hicieron pruebas de volumen y se midieron los tiempos de ejecución para cada uno de los distintos ordenamientos propuestos en las secciones anteriores.

Se puede observar en el gráfico de los tiempos de ejecución se ilustra la tendencia de como crecerán los tiempos a medida que sigamos aumentando el dataset, y esta es acorde con el análisis teórico.



Además, vale la pena comentar que aquellos algoritmos de ordenamiento que no incluyen una operación matemática en su criterio de orden (Scaloni y Asistentes Descendientes) son aquellos que se desempeñan mejor mientras que aquellos que si (Suma y Relación) empiezan a deteriorarse a medida que el numero de vídeos incrementa por su procesamiento extra.

4. Optimalidad de la Solución

El análisis de desempeño temporal de los algoritmos propuestos no proporciona toda la información necesaria para tener una decisión firme de cual es el óptimo, es por esto que es necesario someter el algoritmo propuesto a una serie de pruebas que demuestren su efectividad con distintos tipos y volumen de datos.

4.1. Casos de Prueba

En el GITHUB de la entrega se puede observar los diversos casos de prueba a los que fueron sometidos los algoritmos propuestos. Con el objetivo de mantener la simpleza del presente documento, optamos por describir un resumen de lo que fue realizado:

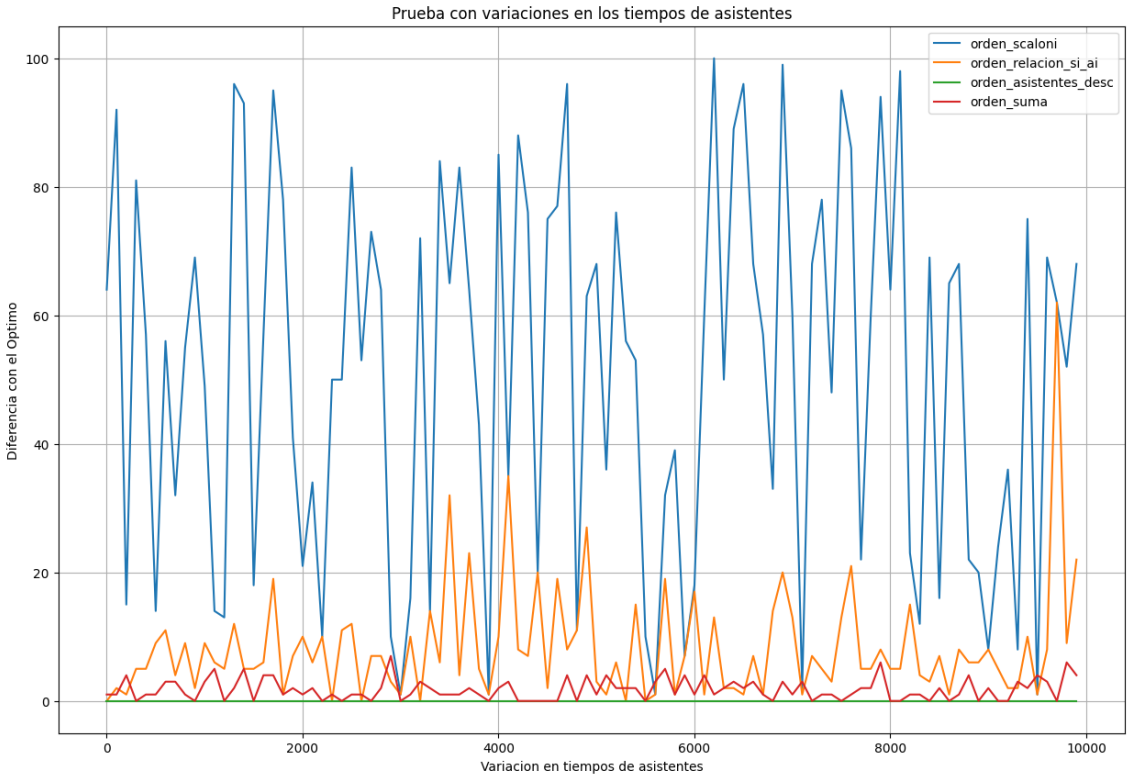
Usando los casos de prueba proporcionados por la cátedra, observamos que el algoritmo que cumplió con todos los casos de optimalidad fue el de ***Ordenar por los tiempos de los asistentes de manera descendiente***. Es por esta razón que al momento de testear los casos de variabilidad de datos usamos dicho algoritmo como el punto de comparación para todos.

4.2. Variabilidad de Datos

El objetivo de esta prueba era observar como se comportaban los algoritmos propuestos cuando los datos (s_i, a_i) sufrían cambios en su magnitud con respecto al otro. Esto con la finalidad de alcanzar algún caso borde que sirva de contra ejemplo para las soluciones propuestas.

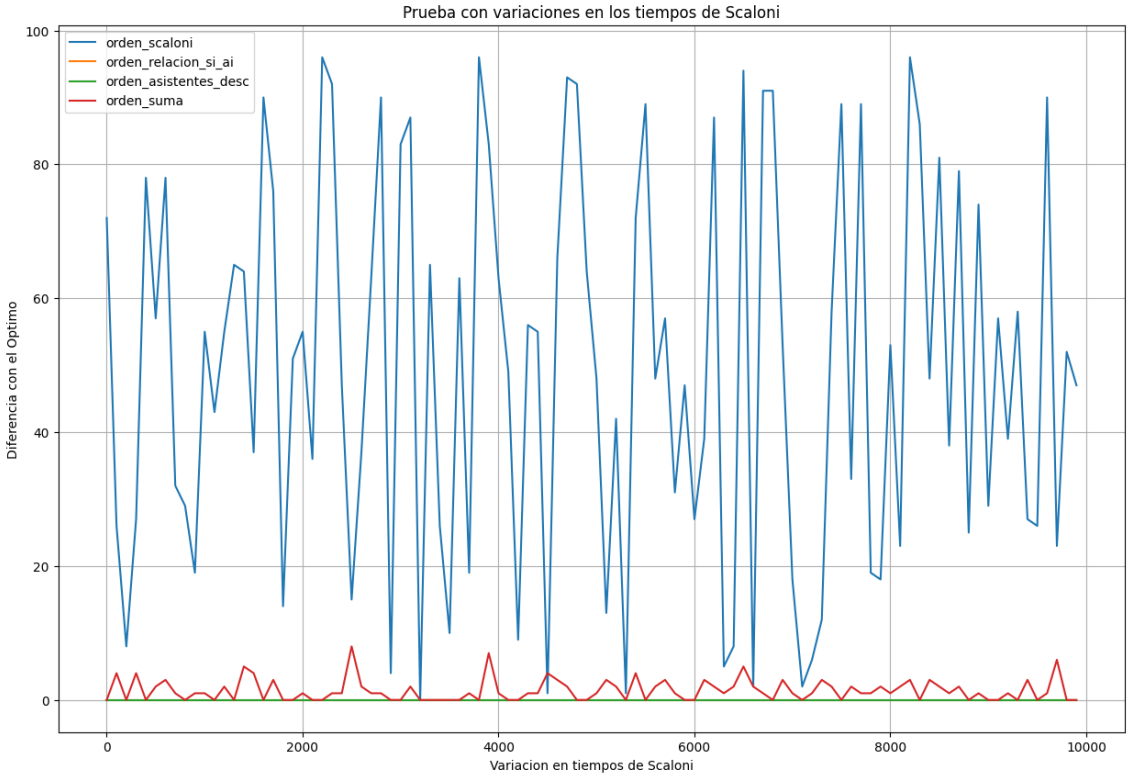
Como fue mencionado previamente, se utilizo como baseline el algoritmo que ordena en orden descendiente los tiempos de los asistentes ya que fue el que cumplió con los casos de optimalidad de la cátedra.

- Variación en la magnitud de los tiempos de los Asistentes



NOTE Podemos observar que el algoritmo que ordena por los tiempos de Scaloni es el que mayor diferencia con el pseudo-óptimo tiene, mientras que el de la suma esta bastante cerca de performar igual.

- Variación en la magnitud de los tiempos de Scaloni



NOTE En este caso observamos que el algoritmo que ordena por la relación performa de manera idéntica al pseudo-óptimo, esto es porque al estar ordenado de manera ascendente y los tiempos de Scaloni ser tan grandes, de primero se evaluaran aquellos cuyo denominador (tiempo asistente) sea el mayor.

De estos gráficos podemos inferir que la solución que consideramos como pseudo-optimo performa mejor que todos los otros propuestos al momento de toparse con estos casos borde, esto es porque si alguno de los otros algoritmos cubriera un caso borde mejor que el pseudo-optimo el gráfico tendría valores negativos en el eje Y.

5. Conclusiones

Tras todo el análisis realizado en las secciones anteriores, podemos concluir que el algoritmo greedy óptimo para esta problemática es el siguiente:

```
def analizar_rivales(videos):
  videos.sort(key = lambda x: (x[1]), reverse = True)
  s_total = 0
  t_total = 0
  for video in videos:
    s_total += video[0]
    t_total = max(t_total, s_total + video[1])
  return t_total
```

De esta manera, se analizarían primero todos aquellos vídeos que los asistentes demorarían mucho mas en terminar, minimizando así tiempos de espera por asistentes en segundo plano.

Además, se puede afirmar que **aquellas soluciones que no tomen en cuenta los tiempos de los asistentes son las que peor se desempeñarían**, por el simple hecho de que los tiempos de Scaloni siempre van a formar parte de la solución final, sin importar en que orden este vea los vídeos. Por ende lo que es **variable y optimizable en este problema son los tiempos de los asistentes** y como estos se solapan entre si.