.UBAfiuba
FACULTAD DE INGENIERÍA

Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico 1
Algoritmos Greedy

24 de enero de 2024

Martin Tomas
100835

Adriana Macarena Iglesias Tripodi
103384

Ezequiel Lassalle
105801

Cecilia Caffaratti
72629

1. Introducción

En este trabajo practico tuvimos que solucionar un problema propuesto por la cátedra mediante el planteo y el uso de algoritmos greedy. El problema nos propone que Scaloni, analizando los futuros rivales de la selección argentina, necesita encontrar una forma de analizar el juego de todos ellos y para esto necesita ver una serie de grabaciones de partidos. Luego de analizarlos, el tiene un conjunto de ayudantes que también deben analizarlos. Hay que tener en cuenta que solo podrán hacerlo una vez que Scaloni haya terminado cada vídeo.

El problema nos propone encontrar la manera más eficiente en la que los vídeos podrían ser vistos por Scaloni, buscando que estos sean analizados lo antes posible. Hay que considerar que todos los ayudantes tardarán lo mismo en analizar los vídeos y que, como se tiene la misma cantidad de ayudantes que rivales, siempre habrá un ayudante disponible para comenzar con el vídeo una vez que Scaloni haya terminado con él.

Los algoritmos greedy son algoritmos que intentan resolver un problema realizando operaciones siempre entre la posición actual y su consiguiente, con la intención de obtener óptimos locales. Este tipo de algoritmos trabaja de forma rápida y sencilla y bajo la suposición de que al ir obteniendo óptimos locales se llegará a un óptimo global o por lo menos a una buena aproximación de este.

Resolvimos el problema inspirándonos en varios de los problemas planteados en las clases de algoritmos greedy.

Resolución

2. Análisis del problema

A lo largo de la resolución del trabajo fuimos pensando varios algoritmos que nos parecían capaces de solucionar el problema.

Siempre intentando de proponer un algoritmo que sea capaz de calcular cuanto va a ser el menor tiempo posible que Scaloni y sus ayudantes tarden en ver y analizar los videos.

Comenzamos pensando que la solución del problema podía llegar a ser parecida a la solución planteada en clase para el problema de la mochila, pensando que dicha solución dependería de todos los datos dados. Entonces, pensamos que podríamos resolverlo ordenando la lista según un valor obtenido mediante la división entre el tiempo que tarda Scaloni y el tiempo que tarda cada ayudante en ver el mismo video. Sin embargo, al darnos cuenta que Scaloni debe analizar siempre todos los partidos grabados de sus rivales concluimos que sin importar con cual comience el tiempo en el que este estará realizando la tarea es el mismo. Y que, como tiene la misma cantidad de ayudantes que de rivales (y por lo tanto videos a analizar), el máximo tiempo que tarde el conjunto de ayudantes en analizar los videos será el de mayor duración de estos. La respuesta al problema no iba por ahí.

Entonces, con estas ideas en mente, pensamos que si se veían los videos en el orden que más tiempo le costaban a los ayudantes en analizar entonces se llegaría a la solución buscada. Esto se planteó esperando que la solución sea, a excepciones también contempladas en el algoritmo, la sumatoria del tiempo que tarda Scaloni en ver la totalidad de los video con el tiempo que tardan los ayudantes en ver el último de estos, que será el de menor duración.

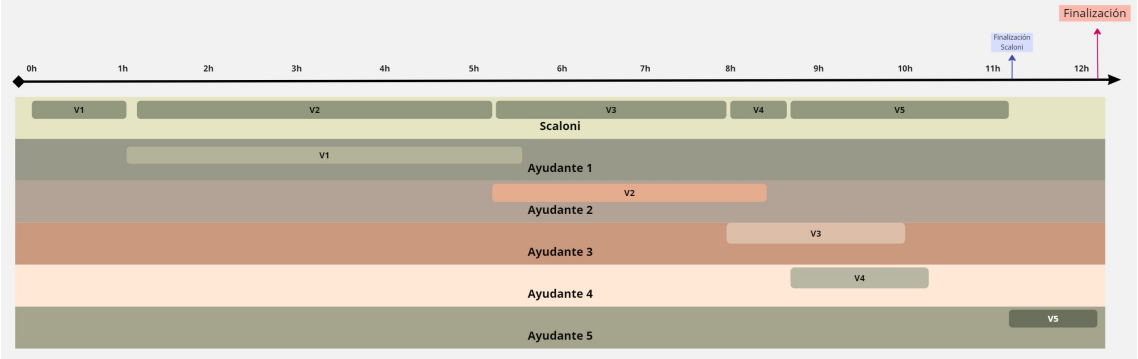


Figura 1: Ejemplo de Comportamiento normal, esperado, de la duración del análisis de los partidos

Las excepciones son todos los casos en que la sumatoria del tiempo ocupado por Scaloni hasta finalizar con el video que comienza a ver el ayudante y la duración del análisis que hace sobre este dicho ayudante superen al tiempo total de Scaloni en conjunto con el video de menor duración para los ayudantes.

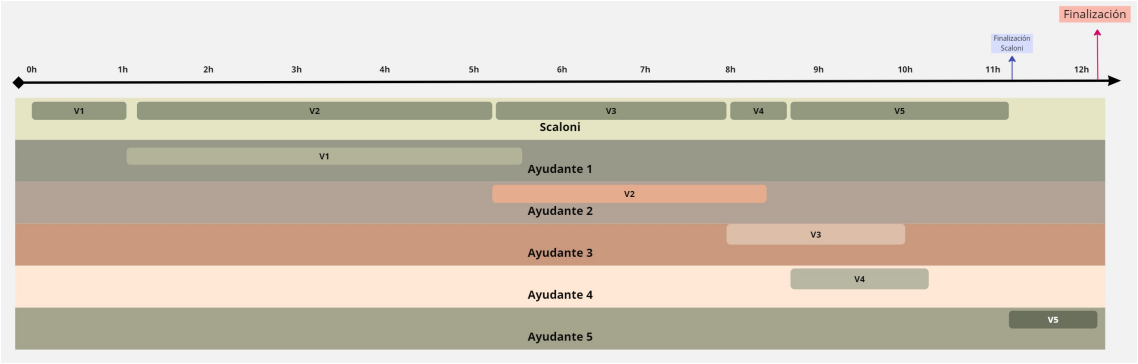


Figura 2: Ejemplo de un comportamiento excepcional de la duración del análisis de los partidos

Consecuentemente se plantea un algoritmo que evalué, de manera greedy, cual va a ser el menor tiempo en que tarde el equipo en analizar todos los videos, teniendo en cuenta estos problemas.

El algoritmo primero ordena, de forma decreciente, según la duración del análisis de los ayudantes para cada rival. No se tiene en cuenta un criterio de orden para Scaloni porque, como bien se estableció antes, independiente de como este lo realice el costo temporal será siempre el mismo (la sumatoria del tiempo que tarde en finalizar cada video). Además, como tenemos la misma cantidad de ayudantes disponibles que rivales, el tiempo de análisis de los ayudantes es independiente entre sí. Pero, como Scaloni debe realizar el análisis primero, el tiempo final no es independiente de la tardanza causada por Scaloni y consecuentemente convendrá ver último el video de menor duración para los ayudantes.

Luego, nuestro algoritmo itera sobre la lista y va comparando el tiempo acumulado que Scaloni tardo en ver los vídeos hasta llegar el vídeo actual(contándolo a este también) y le suma el tiempo que el ayudante tarda en analizarlo. Vamos guardando este tiempo en una variable llamada tiempoMaximo y si en alguna de las iteraciones el tiempo resulta mayor que el anterior entonces elegimos quedarnos con este nuevo tiempo, siendo este el nuevo óptimo local. Es así como llegamos al mínimo tiempo en el que el eqquipo va a tardar en analizar toda la lista.

Por lo tanto el algoritmo ordena la lista de videos en función de un criterio y luego toma decisiones locales en cada paso (eligiendo un óptimo local) esperando llegar así al óptimo global, la menor duración para el análisis de todos los rivales.

2.1. Algoritmo y Complejidad

A continuación se muestra el algoritmo principal que utilizamos para solucionar nuestro problema.

```
1
2       # Complejidad O(nlogn)
3       tiempoScaloniAyudantes.sort(key=lambda tup: tup[1], reverse=True)
4
5       tiempoEnFinalizarAyudante = []
6       tiempoAcumuladoScaloni = 0
7       tiempoMaximo = 0
8
9       # Complejidad O(n)
10      for tiempo in tiempoScaloniAyudantes:
11          tiempoScaloni = tiempo[0]
12          tiempoAyudante = tiempo[1]
13
14          tiempoAcumuladoScaloni += tiempoScaloni
15
16          tiempoMaximo = max(tiempoAyudante + tiempoAcumuladoScaloni, tiempoMaximo)
```

La complejidad del algoritmo propuesto para encontrar el máximo es $\mathcal{O}(n \log n)$. Esto es debido a que el ordenamiento de la lista inicial utiliza el método sort que tiene esa complejidad. Esta es la de mayor peso sobre todo el algoritmos debido a que luego la lista en el *"for"* es recorrida de manera ordenada donde se itera cada elemento una vez, con complejidad $\mathcal{O}(n)$ y la complejidad de la selección del máximo es $\mathcal{O}(1)$ por ser una comparación lógica simple. Entonces:

$$\mathcal{O}(f(n)) = \mathcal{O}(n \log(n)) + \mathcal{O}(n) + \mathcal{O}(1) \approx \mathcal{O}(n \log(n))$$

Los valores de las duraciones de los análisis de Scaloni y sus ayudantes no parece cambiar ni el tiempo ni si se logra la obtención del óptimo global en nuestro algoritmo. Esto es consecuencia de que, como previamente establecido, estamos ordenando la lista de mayor a menor y recorriendo los elementos de esta comparando si la nueva duración es peor o no que la de nuestro óptimo local, sea o no continuo iterando hasta llegar al final de la lista, cambiado mi óptimo local solo cuando dicha duración sea mayor.

Por otro lado la cantidad de videos a analizar si tiene un efecto directo en el costo temporal del algoritmo, es el n de la complejidad previamente establecida.

3. Ejemplos y gráficos

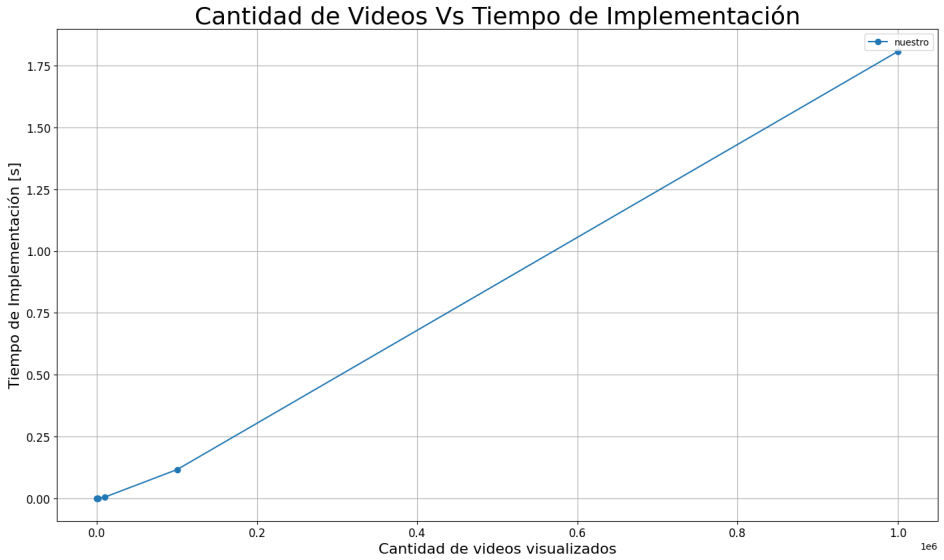
Tal como piden los puntos 4 y 5 de las consignas realizamos una extensa serie de pruebas para verificar que la complejidad algorítmica que habíamos propuesto era la correcta .

Como se puede ver en el codigo:

```
x=np.arange(0,100000,1000)
y=[]

for n in x:

    star=[]
    en=[]
    for i in range(1, 5):

        videosAMirarScaloni = [random.uniform(1, 300) for _ in range(n)]
        videosAMirarAyudantes = [random.uniform(1, 400) for _ in range(n)]
        star.append(time.perf_counter())
        calcularTiempoMaximo(
            n,
            videosAMirarScaloni,
            videosAMirarAyudantes,
            debug=False
        )
    en.append(time.perf_counter())
    start=np.mean(star)
    end=np.mean(en)

    y.append(end-start)
```

Vamos iterando de 1000 en 1000 hasta llegar a 100000 vídeos, realizando en cada iteración 5 pruebas siempre con números generados aleatoriamente y guardando un promedio los tiempos que tarda el algoritmo en ejecutarse en un vector y. Entendimos que una aproximación así era lo suficientemente buena en cantidad de vídeos y de pasos en los que incrementan como para que se vea la tendencia real del algoritmo.Los sobresaltos que toma el gráfico en algunos valores claramente son por que la generación de los valores de los vídeos es aleatoria. Los resultados fueron los siguientes:


Cantidad de Videos Vs Tiempo de Implementación
nuestro
Tiempo de Implementación [s]
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0
20000
40000
60000
80000
100000
Cantidad de videos visualizados


Cantidad de Videos Vs Tiempo de Implementación
nuestro
Tiempo de Implementación [s]
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
0.0
0.2
0.4
0.6
0.8
1.0
1e6
Cantidad de videos visualizados


Cantidad de Videos Vs Tiempo de Implementación
nuestro
Tiempo de Implementación [s]
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0
20000
40000
60000
80000
100000
Cantidad de videos visualizados

Como se puede apreciar, la complejidad mostrada en los gráficos es acorde a la que habíamos planteado anteriormente. Los gráficos muestran una clara tendencia $\mathcal{O}(n \log n)$.

4. Conclusiones

Como conclusión principal del trabajo podemos establecer que el algoritmo greedy utilizado alcanza el máximo global de una forma rápida y eficiente (tardando alrededor de 2 segundos cuando se visualizan un millón de videos). Además de ser un algoritmo sencillo, no solo de unas pocas liñas de código sino que también simple para la comprensión. Por lo tanto, se pcree que para resolver problemas de este tipo, Por lo tanto para problemas de optimización o en general cuando se quiera buscar una buena aproximación rápidamente a un problema donde se pueden ir seleccionando valores en cada paso se cree que se podrían utilizar algoritmos similares al obtenido.

El tema nos resulto muy interesante y la idea de ir resolviendo un problema buscando paso a paso el siguiente óptimo local y así intentar encontrar el óptimo global es muy práctica y eficiente.