Teoría de Algoritmos I

Curso de verano 2024

Trabajo Práctico 1: Algoritmos Greedy

Alumno	Padrón	E-mail
Juan Ignacio Pérez Di Chiazza	109.887	jperezd@fi.uba.ar
Graciela Rodríguez	92.299	gmrodriguez@fi.uba.ar
Joaquín Parodi	100.752	jparodi@fi.uba.ar
Máximo Utrera	109.651	mutrera@fi.uba.ar

Índice

Introducción

Tenemos que ayudar a Scaloni a analizar los próximos n rivales de la selección campeona del mundo. Como técnico perfeccionista, quiere analizar videos de cada uno de los rivales. Recibió un compilado por cada rival, y necesita hacer un análisis muy detallado, lo cual le implica tomar apuntes, analizar tácticas, ver cuándo hay que hacerle un masaje a Messi, etc... Para que el análisis sea detallado, cada compilado no lo revisa únicamente él, sino también un ayudante.

El análisis del rival i le toma s_i de tiempo a Scaloni, y luego a_i al ayudante (independientemente de cuál ayudante lo vea). Lo bueno, es que después de los grandes logros obtenidos, Scaloni cuenta con n ayudantes (es decir, la misma cantidad que rivales), que pueden ver los videos completamente en paralelo. Siempre los ayudantes podrán ver los videos después que Scaloni haya terminado de verlo y analizarlo como corresponde (esto no lo delega). Cuando llega la hora que un ayudante lo vea, puede ser cualquiera, pero sólo uno lo verá (no hay ganancia en que dos lo vean).

El DT necesita que los rivales estén todos con sus correspondientes análisis lo antes posible, y por eso te pide que lo ayudes. Dice que confía en vos. Sabe que no lo vas a dejar tirado.

Consigna

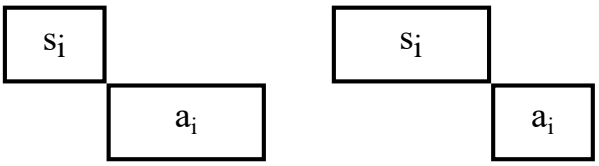
1. Hacer un análisis del problema, y proponer un algoritmo greedy que obtenga la solución óptima al problema planteado: dado los valores de n, los s_i y a_i, determinar un orden en el que Scaloni debe ver los videos tal que todos los análisis estén listos lo antes posible (es decir, en el mínimo tiempo necesario). Explicar detalladamente por qué el algoritmo planteado obtiene siempre la solución óptima.
2. Escribir el algoritmo planteado. Describir y justificar la complejidad de dicho algoritmo. Analizar si (y cómo) afecta la variabilidad de los valores de a_i y s_i a los tiempos y optimalidad del algoritmo planteado.
3. Realizar ejemplos de ejecución para encontrar soluciones y corroborar lo encontrado. Adicionalmente, el curso proveerá con algunos casos particulares que deben cumplirse su optimalidad también.
4. De las pruebas anteriores, hacer también mediciones de tiempos para corroborar la complejidad teórica indicada. Realizar gráficos correspondientes.
5. Agregar cualquier conclusión que parezca relevante.

1. Análisis del problema

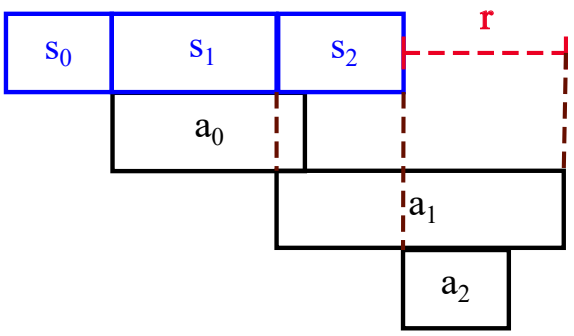
- *Cantidad de rivales = Cantidad de videos = Cantidad de ayudantes* $= n$ (Se considera que cada ayudante analiza un solo video.)
- *Demora de Scaloni en analizar el video* $i = s_i$
- *Demora de un ayudante en analizar el video* $i = a_i$
- Un ayudante únicamente puede analizar un video inmediatamente después que lo haya terminado de analizar Scaloni.
- Scaloni mira todos los videos seguidos y solo puede mirar un video a la vez.
- El tiempo que demora un ayudante en analizar video es independiente del tiempo que le demora a Scaloni.

Posibilidades:

$s_i < a_i$ $\qquad$ $s_i > a_i$

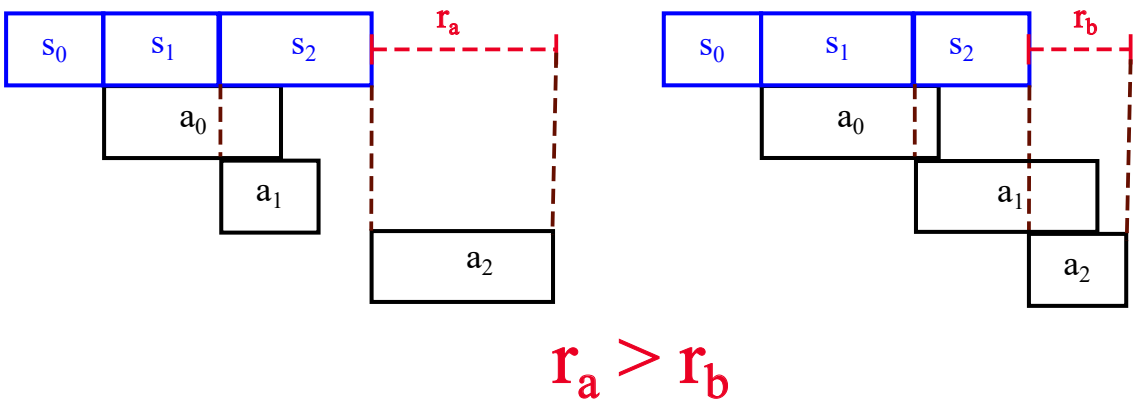


Scaloni debe mirar todos los videos en serie, ya que no puede analizar 2 al mismo tiempo. Por lo tanto, el tiempo total que demora Scaloni es fijo sin importar el orden es: $S = \sum_0^n s_i$. Como los ayudantes pueden ver los videos de forma independiente, pero solo cuando Scaloni haya finalizado cada uno, esto significa que alterar el orden de los videos varía el momento de inicio del ayudante en el tiempo. El análisis está completo cuando todos los ayudantes hayan terminado, por lo que variar el orden de los ayudantes hace que se modifique el final del análisis. Entonces para optimizar el tiempo total de análisis, y partiendo de que el tiempo de Scaloni es fijo, lo que importa es el tiempo que demoran los ayudantes en terminar de ver los videos luego de que Scaloni haya finalizado el último. Por lo tanto, ese tiempo extra r (ver figura) debería ser lo más pequeño posible.



Se sabe que el último ayudante empieza cuando Scaloni termina el último video, por lo tanto su tiempo de análisis a_i va a estar contenido completamente en r. Esto significa que si se quiere reducir r, entonces no se puede poner el mayor tiempo al final. A raíz de esto, se intuye razonablemente que poniendo los a_i más chicos al final se reduce el tiempo total de análisis. Resta demostrar si esto se cumple siempre.

Cuando se analizan por ejemplo tres videos, se tienen los siguientes 3 casos posibles. Hay que hallar cuál es el que demora menos tiempo total.



Caso 1: Si $a_i < a_{i+1}$ entonces $r = a_{i+1}$ (sin importar el tamaño de los s_i). Como se quiere minimizar r, lo que convendria es ordenarlos al revés, por lo tanto el más grande no puede ir al final.

Caso 2: Si $a_i > a_{i+1}$ entonces hay que analizar qué pasa si:

- $s_{i+1} + a_{i+1} < a_i$ entonces $r = a_i - s_{i+1}$ (sigue conveniendo que el ayudante que más demore se analice antes, porque de intercambiarlos se demoraría más tiempo).
- $s_{i+1} + a_{i+1} > a_i$ entonces $r = a_{i+1}$

Caso 3: Si $a_i = a_{i+1}$. En tal caso no importará como queden ordenados, mientras que se siga la logica de ordenar de mayor a menor por tiempo de ayudantes. Esto es porque la suma de los tiempos de scaloni será la misma independientemente de cual se tome en cuenta, y por lo tanto $r = a_i = a_{i+1}$

2. Planteo del algoritmo

2.0. Algoritmo no optimo considerado

En principio se considero un algoritmo que ordene los videos de mayor a menor por s_i, el problema de este es que se llega al optimo solamente cuando los ayudantes tardan mas o lo mismo que Scaloni y no en otro caso, por lo que fue descartado.

Esto se puede ver en el siguiente contraejemplo:

```
# (Scaloni, ayudante)
EJ1: con a_i >= s_i = [ (10, 20), (40,80) ]

Ordenando por tiempo de finalizacion de Scaloni de mayor a menor..

[ (40, 80), (10, 20) ]

se obtiene un tiempo minimo de 120 y es el optimo



EJ2: con a_i independiente de s_i = [ (10, 20), (40,10) ]

Ahora el resultado luego de ordenar es..

[ (40, 10), (10, 20) ] # notar como el ayudante que mas tarda queda en el final

y el tiempo minimo que se obtiene de este algoritmo es 70 cuando el optimo seria 60
```

2.1. Algoritmo

El algoritmo propuesto a partir del el analisis previo consiste en ordenar la lista de videos de mayor a menor por el tiempo que tardan los ayudantes, ya que ese resultó ser el orden óptimo en el que Scaloni debe ver los videos. Posteriormente se inicializan dos variables, una para el tiempo de finalización de Scaloni, y otra para indicar el maximo tiempo de finalización de los ayudantes obtenido hasta el momento. En este punto solo queda calcular, de manera greedy, el tiempo de finalizacion (tanto de Scaloni como del ayudante para el video i), para esto se recorre la lista de videos de izquierda a derecha, aprovechando el orden establecido en la misma. Si el tiempo calculado de finalizacion para el ayudante i es mas alto que el obtenido hasta entonces, se reemplaza la variable asociada al máximo tiempo obtenido.

Una vez analizados los videos se tiene una lista del orden óptimo y el tiempo que se tardaría hasta que estén todos los análisis completos.

2.2. Por qué es greedy

El algoritmo es greedy ya que para obtener el orden óptimo se queda siempre con el de a_i más grande disponible, y luego al calcular el tiempo óptimo se calculan los nuevos tiempos de ayudante y si se encuentra un tiempo más grande que el que se tenía, se decide quedarse con este.

2.3. Por qué es óptimo

Demostracion por inversiones

Siguiendo los pasos de Kleinberg y Tardos [1], pero adaptado al algoritmo propuesto.

La primera observacion es que no hay tiempo ocioso. Decimos que existe una solucion optima para el problema sin tiempo ocioso. Decimos que la lista de videos tiene una inversión si el video i con el tiempo de ayudante a_i es ubicado antes que el video j, con tiempo de ayudante $a_j > a_i$. De esto también se puede deducir que todas las listas sin inversiones, sin tiempo ocioso, y con los mismos tiempos para cada ayudante, tienen el mismo tiempo máximo de finalización, que corresponde al tiempo del último ayudante + $\sum s_i$. Luego, queremos demostrar que hay una forma óptima de ver los videos, en la cual no hay tiempo ocioso ni inversiones. Para esto, supongamos que tenemos la lista A con una inversión, es decir, tenemos $[i, j]$, $a_i < a_j$, invertimos ambos y ahora queremos demostrar que dicha inversión, que genera la nueva lista $\bar{A}$, no aumenta el tiempo máximo en que se termina de ver los videos.

Definimos la lista A (con inversiones), que tiene tiempo de finalización adicional al de Scaloni como $x = \max a - \sum s_i$, $\bar{x} = \max \bar{a} - \sum s_i$. Para demostrar que $x \geq \bar{x}$, hay que demostrar que $\max a > \max \bar{a}$, porque $\sum s_i$ es igual para ambos. Para esto demostramos todos los casos.

- Si $s_j > s_i \rightarrow \max a = s_j + s_i + a_j$, $\max \bar{a} = s_j + a_i \rightarrow s_j + s_i + a_j > s_j + a_i \rightarrow s_i + a_j > a_i$ (cierto porque $s_i > 0$, $a_j > a_i$)
- Si $s_j < s_i \rightarrow \max a = s_j + s_i + a_j$, $\max \bar{a} = s_j + s_i + a_i \rightarrow a_j > a_i$
- Si $s_j = s_i \rightarrow \max a = s_j + s_i + a_j$, $\max \bar{a} = s_j + a_j \rightarrow s_j + s_i + a_j > s_j + a_j$ (cierto porque $s_i > 0$)

Por lo tanto, $x \geq \bar{x}$. Consecuentemente, queda demostrado que la inversión no conlleva a un aumento del tiempo máximo que se tarda en ver los videos, y se demuestra la optimalidad del algoritmo propuesto.

2.4. Complejidad

La complejidad temporal del algoritmo es $O(n.log(n))$ por el ordenamiento inicial más $O(n)$ por recorrer la lista de videos. Es decir, es $O(n.log(n))$.

La complejidad espacial del algoritmo depende del tipo de ordenamiento que se use, en este caso es $O(n)$ porque no es *in place* pero si lo fuera sería $O(1)$ a cambio de modificar la lista original.

2.5. Código

```python
def obtener_orden_y_tiempo_optimo_de_videos(tiempos: list[tuple[int, int]]):
    orden_optimo = sorted(tiempos, key=lambda tiempo: tiempo[1], reverse=True)

    finalizacion_anterior_scaloni = 0
    finalizacion_anterior_ayudante = 0

    for tiempo_scaloni, tiempo_ayudante in orden_optimo:
        finalizacion_scaloni = finalizacion_anterior_scaloni + tiempo_scaloni
        finalizacion_ayudante = finalizacion_scaloni + tiempo_ayudante

        finalizacion_anterior_scaloni = finalizacion_scaloni

        if finalizacion_anterior_ayudante < finalizacion_ayudante:
            finalizacion_anterior_ayudante = finalizacion_ayudante

    return orden_optimo, finalizacion_anterior_ayudante
```

3. Ejemplos de ejecución

Se utiliza la siguiente convencion para describir los tiempos del video i: (Si, Ai), por ejemplo para 2 videos distintos sería: $[(S_1, A_1), (S_2, A_2)]$.

3.1. 3 elementos

```
1
2 Tiempos: [ (3, 3), (5, 1), (1, 8) ]
3
4
5 La ejecucion de este algoritmo con esta entrada da como resultado el
6 orden de videos y tiempo:
7
8 Orden optimo: [ (1, 8), (3, 3), (5, 1) ] # notar que el orden es de mayor a menor por
      tiempo de ayudante
9 Tiempo optimo: 10
10
11 Y son optimos
```

3.2. 10 elementos

```
1
2 Tiempos: [(1, 3), (5, 1), (4, 8), (4, 3), (1, 5), (2, 9), (2, 2), (2, 4), (1, 6), (6, 5)]
3
4 La ejecucion de este algoritmo con esta entrada da como resultado el
5 orden de videos y tiempo:
6
7 Orden = [(2, 9), (4, 8), (1, 6), (1, 5), (6, 5), (2, 4), (1, 3), (4, 3), (2, 2), (5, 1)]
8 Tiempo = 29
9
10 Y son optimos
```

3.3. Contraejemplo de ordenamiento por s_i

En este ejemplo se ve el resultado de la ejecucion del algoritmo propuesto con la entrada que se uso de contraejemplo para otro algoritmo considerado en la seccion 2.0. **Algoritmo no optimo considerado**.

```
1
2 Tiempos: [ (10, 20), (40,10) ]
3
4 La ejecucion de este algoritmo con esta entrada da como resultado el
5 orden de videos y tiempo:
6
7 Orden = [ (10, 20), (40, 10) ]
8 Tiempo = 60 # notar que el otro algoritmo mencionado dio un tiempo de 70
9
10 Y son optimos.
```

4. Mediciones de tiempo

4.1. Cómo afecta la cantidad de videos n al tiempo de ejecución

En el siguiente benchmark se analiza el tiempo del algoritmo en funcion de el tamaño n de la entrada. La complejidad teorica del algoritmo esta explicitada en la seccion 2.4. **Complejidad**.

Para el benchmark se toman diferentes largos desde 0 hasta 100.000 con un salto de 500, es decir se mide el tiempo con el largo 0, luego el largo 500, el 1000 y asi sucesivamente.

Los valores s_i y a_i son randomizados entre 1 y 1000.

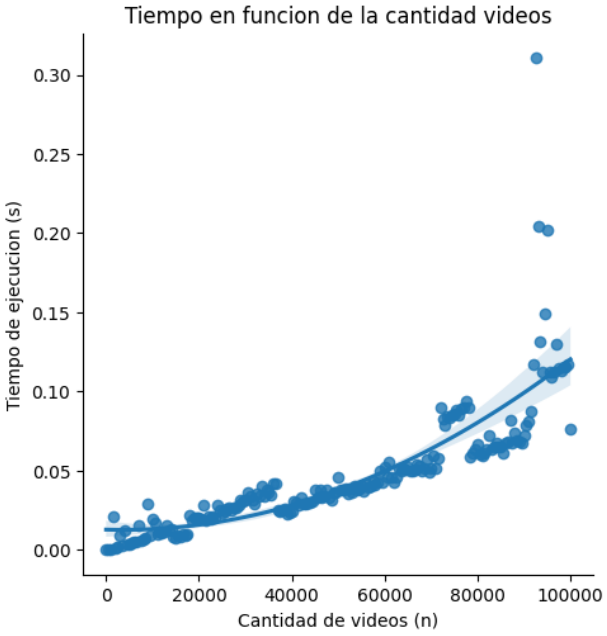


Figura 1: Se ve que a medida que la cantidad de videos aumenta el grafico del tiempo del algoritmo se asemeja a la funcion nlog(n) confirmando la complejidad teorica propuesta.

4.2. Cómo afecta la variabilidad de s_i y a_i al tiempo de ejecución

A continuacion se ve un segundo benchmark, esta vez analizando los cambios en el tiempo del algoritmo al variar los tiempos que tardan Scaloni y sus ayudantes en terminar un video en particular.

Para esto s_i y a_i son randomizados con un limite maximo que va de 0 a 10.000 con un salto de 100, por lo que se toma el tiempo que tarda el algoritmo con los valores siendo randomizados entre 0 y 100, luego 0 y 200 y asi sucesivamente, luego a partir de estas mediciones se hace el grafico de la figura 2.

La cantidad de videos se fija en 1000 para todas las mediciones.

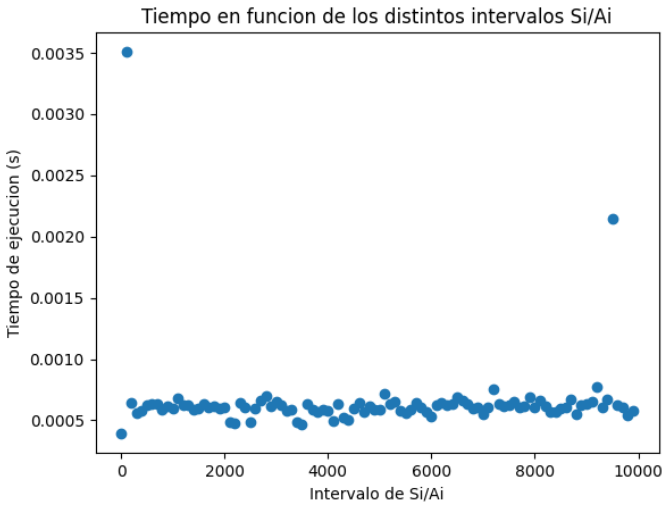


Figura 2: Se ve que el tiempo del algoritmo es independiente de la variabilidad de s_i y a_i.

5. Conclusiones

Se logra ver el análisis completo de un algoritmo óptimo que utiliza la técnica de diseño "Greedy", incluyendo su optimalidad, con la demostracion por inversiones e ilustrada con ejemplos de ejecución. Su complejidad teórica está justificada empíricamente a través de mediciones de tiempos mediante benchmarks representados mediante graficos acordes.

Referencias

[1] Éva Kleinberg, Jon y Tardos. *Algorithm Design.* Pearson/Addison-Wesley, 2006, 2005.