.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

Índice

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 definir los días en que los jugadores deban entrenarse y los días que les convenga descansar de tal forma de tener la mayor ganancia posible. 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:

- Scaloni definió un cronograma de entrenamiento.
- El entrenamiento del día i demanda una cantidad de esfuerzo e_i.
- El entrenamiento que corresponde al día i es inamovible.
- La cantidad de energía que tienen disponible para cada día va disminuyendo a medida que pasan los entrenamientos.
- La distribución de energía en cada día sigue la siguiente secuencia: $s_1 \geq s_2 \geq ... \geq s_n$. De esta manera s_i es la energía disponible para el i-ésimo día de entrenamiento consecutiva.
- El entrenador tiene la opción de otorgarles un día de descanso, lo que resulta en la renovación de la energía de los jugadores (es decir, el próximo entrenamiento comenzaría nuevamente con la energía s_1, seguida de s_2, y así sucesivamente).
- Si se opta por el descanso, el entrenamiento programado para ese día no se lleva a cabo, y en consecuencia, no se obtiene ninguna ganancia.
- Si el nivel de esfuerzo e_i excede la energía disponible s_i en un día determinado, la ganancia resultante del entrenamiento es igual a la energía disponible. Caso contrario, la ganancia del entrenamiento se define por el nivel de esfuerzo realizado.
- Dada la secuencia de energía disponible desde el último descanso $s_1, s_2, ..., s_n$ y el esfuerzo/ganancia de cada día e_i, determinar la máxima cantidad de ganancia que se puede obtener de los entrenamientos, considerando posibles descansos.

2. Análisis de diferentes soluciones propuestas

Vamos a proponer dos escenarios posibles para poder analizar sus soluciones. Estos dos escenarios son de la forma:

Primero analizamos el primer caso de 3 días ya que es el proporcionado por la cátedra.

$$n = 3, \quad S = (1, 5, 4), \quad E = (10, 2, 2)$$

En este notamos que la ganancia de un día i puede tomar 2 valores posibles:

- Suma entre la ganancia del día anterior y $min(e_i, s_d)$, con d cantidad de días seguidos entrenando
- Suma entre la ganancia de dos días antes y $min(e_i, s_1)$

Esto debido a que observamos una relación con el problema planteado de Juan el Vago, en el que buscábamos la mayor ganancia, pero sin trabajar dos días seguidos.

Para esta instancia, supusimos erroneamente, que la ganancia máxima para un día i iba a estar dada si en cada día de entrenamiento se maximizaba su ganancia. De esta forma, cada vez que se necesite la ganancia máxima de un día o más anteriores, esta ya iba a estar calculada anteriormente. Para poder calcular esta propuesta propusimos el siguiente algoritmo: $g(i,d) = maxg(n-1) + min(e_i, s_d), g(n-2) + min(e_i, s_1)$,

De esta manera, este algoritmo daba la respuesta esperada que es 7 en el primer caso de 3 días.

El inconveniente en este algortimo es que al querer maximizar cada día solo comparando si el día anterior hubiera convenido descansar o no, no se tiene en consideración el hecho de que hay escenarios donde el descanso es conveniente en un día $j < i - 1$ y solo se hace evidente en el día i. Esto se debe a que la ganancia máxima en un i-ésimo día es esencialmente la suma del mínimo entre el esfuerzo requerido y la energía disponible.

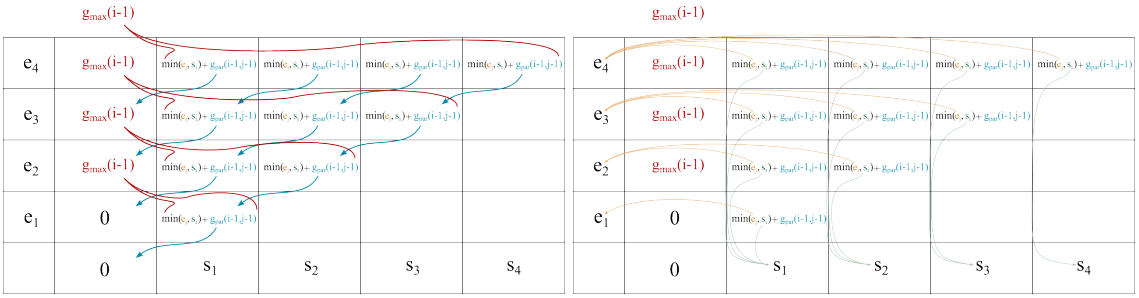
Dado que esta ganancia está condicionada por los niveles de energía de cada día, en ocasiones es más conveniente que la ganancia relativa de un día no sea óptima individualmente, sino que en conjunto ofrezca la mayor ganancia total. El algoritmo propuesto no aborda adecuadamente esta complejidad, ya que solo considera esta condición con respecto a los dos días anteriores, lo que limita la capacidad de adaptación a situaciones donde la estrategia de descanso óptima se extiende en un período más largo.

Al darnos cuenta de este problema, procedimos a utilizar una planilla de cálculo para ayudarnos a resolverlo. Arreglamos E verticalmente y S horizontalmente en un cuadro de doble entrada. Luego, cada casilla (i, j) era la ganancia parcial que podríamos obtener si ese fuera el i-ésimo día de entrenamiento consecutivo para el día j del plan de entrenamiento. Notamos que la ganancia máxima hasta el entrenamiento j era el máximo de las ganancias parciales de la misma fila. Decidimos entonces colocar a la izquierda una columna auxiliar que contenía la ganancia maxima de la fila inferior. Ver resultado en la figura 1.

		22				
3	10	15	17	22	21	16
2	8	7	12	11	15	
1	3	4	3	7		
0	4	0	4			
i↑	e↑					
	s→	0	10	10	10	1
		i→	0	1	2	3

Figura 1: Caso 2 planilla

Luego, logramos abstraer las funciones, mostradas en la figuras 2, 2 y 2.



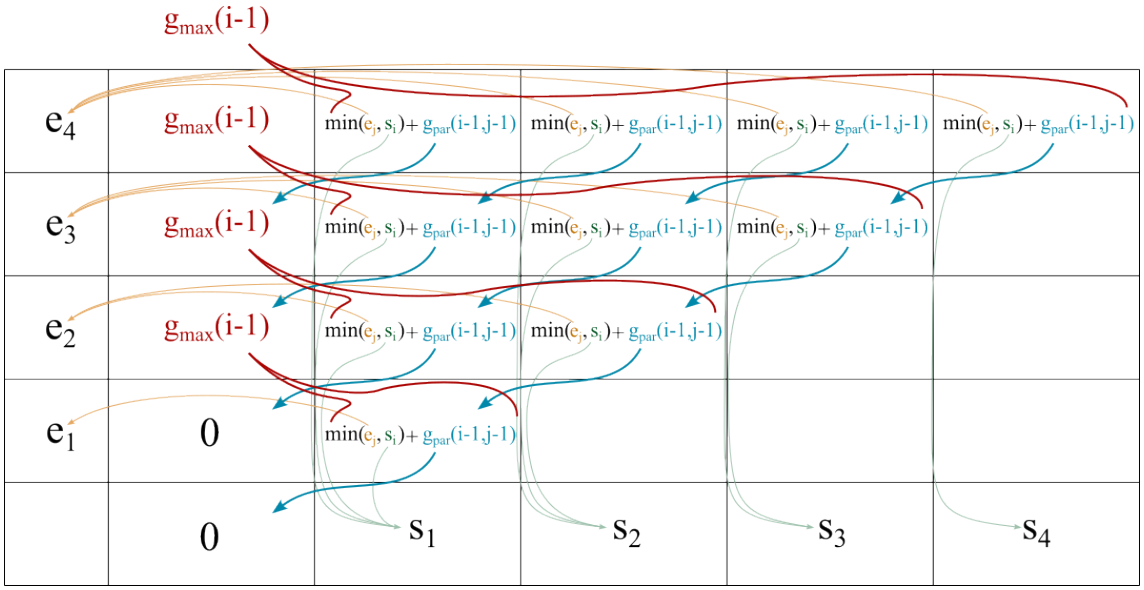


Figura 2: Formulas en planilla

El algoritmo presentado tiene una complejidad temporal de $O(n^2)$. Esto se debe a que, para obtener la solución óptima, requerimos analizar cada día de entrenamiento, con todas sus variantes de energía disponible por día. Técnicamente lo que hacemos es recorrer una matriz de $n*n$, siendo n la cantidad de días a entrenar (igual cantidad que los valores que toman las energías). Ahora, para cada día, calculamos el ganancia parcial respecto a los días anteriores, es decir,

3. Posibles algoritmos

Posterior a nuestro análisis, propusimos dos algoritmos:

3.1. Ordenando por a_i de forma creciente

Este algorítmo sigue las siguientes indicaciones:

- Ordenar los compilados de menor a mayor tiempo de análisis por parte de los ayudantes.
- Cada vez que termina Scaloni de ver un compilado, inmediatamente un ayudante es asignado a visualizarlo.

3.2. Ordenando por a_i de forma decreciente

Este algoritmo sigue las siguientes indicaciones:

- Ordenar los compilados de mayor a menor tiempo de análisis por parte de los ayudantes.
- Cada vez que termina Scaloni de ver un compilado, inmediatamente un ayudante es asignado a visualizarlo.

Comparemos cómo se desempeñan nuestros dos algorítmos en cada escenario propuesto:

3.3. Resultados obtenidos del Caso 1

3.4. Resultados obtenidos del Caso 2

Como se puede observar, para ambos casos el ordenamiento decreciente resulta en un tiempo total menor.

4. Demostración

Procederemos a demostrar que ordenar los compilados por a_i de forma decreciente minimiza el tiempo total.

Llamamos C al conjunto ordenado de n compilados, s_i el tiempo que le tardar visualizar a Scaloni el i-ésimo compilado, y a_i lo que le tarda a un ayudante. Además, al tiempo que tarda Scaloni analizar los compilados hasta el k-ésimo lo representaremos como S_k.

$$S_k = \sum_{i=1}^{k} s_i$$

Luego, el tiempo total que se tarda en analizar los compilados hasta el k-ésimo, en el orden en el que se los presenta, sigue la siguiente formula:

$$T_k = \begin{cases} s_1 + a_1 & k = 1 \\ \text{máx}\,(T_{k-1}, S_k + a_k) & k > 1 \end{cases}$$

Ordenar C para minimizar T_n requiere minimizar $\text{máx}\,(T_{n-1}, S_n + a_n)$. Como S_n es independiente del orden de C, $S_n + a_n$ se minimiza colocando último al compilado con el tiempo del ayudante más pequeño. Luego, T_{n-1} se minimiza de la misma manera, resultando en un algoritmo recursivo. Solo nos falta justificar por qué el orden que minimiza $S_n + a_n$ también minimiza $\text{máx}\,(T_{n-1}, S_n + a_n)$. Haremos esto partiendo de la asumpción de que se respeta el orden propuesto y de que modificarlo resulta en un T_n más grande.

- Si $\text{máx}\,(T_{n-1}, S_n + a_n) = S_n + a_n$ ($\implies S_n + a_n \geq T_{n-1}$)
  Es el caso más trivial, ya que minimizar $S_n + a_n$ también minimiza $\text{máx}\,(T_{n-1}, S_n + a_n)$.
- Si $\text{máx}\,(T_{n-1}, S_n + a_n) = T_{n-1}$ ($\implies T_{n-1} \geq S_n + a_n$)
  Este caso implica que existe un compilado j tal que $a_j \geq \sum_{i=j+1}^{n} s_i + a_n \implies a_j > a_n$. Colocar a este compilado último causaría que $T'_n = S_n + a_j$, y

$$T'_n = S_n + a_j, \quad T_n = S_j + a_j$$

$$\begin{gathered} S_n > S_j \\ S_n + a_j > S_n + a_j \\ T'_n > T_n \end{gathered}$$

Esto demuestra que el orden de C que minimiza T_n es con los tiempos a_i decrecientes.

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.

```
class Compilado:
    def __init__(self, scaloni, ayudante):
        self.tiempo_scaloni = scaloni
        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.

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

- 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 algorítmo, a_n es la duración del compilado más corto.

4.1. Cota de complejidad temporal

El algoritmo presentado tiene una complejidad temporal de $O(n^2)$. Esto se debe a que, para obtener la solución óptima, requerimos analizar cada día de entrenamiento, dependiendo de la energía disponible para ese día y sus variantes. Técnicamente lo que hacemos es recorrer la mitad de una matriz de $n*n$, siendo n la cantidad de días a entrenar (igual cantidad que los valores que toman las energías). Ahora, para cada día, calculamos el resultado parcial respecto a los días anteriores, es decir, observamos el máximo

5. Mediciones

Para las mediciones, creamos escenarios de ejemplo de forma aleatoria. Cada escenario consta de una lista de esfuerzos de cada día de entrenamiento y otra con las energías de cada día de entrenamiento consecutivo. Decidmos que el los valores de esfuerzos y energías sean valor enteros 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 esfuerzos y energías.

Notamos que los procesos de fondo del sistema nos generaba distorciones en los tiempos de ejecución de una misma muestra. Atacamos este problema realizando varias mediciones para el mismo escenario de esfuerzos y energías, tomando el promedio de los tiempos obtenidos.

Con el objetivo de comparar los tiempos de ejecución de nuestro algoritmo con la complejidad teórica, optamos por realizar tanto un análisis de regresión cuadrática con un regresión exponencial que se ajustara a nuestros datos. Para evaluar qué curva se ajusta mejor, usamos la raíz del error cuadrático medio (RMSE). Realizamos este análisis en un intervalo con tamaños hasta 1'000. Ver figuras 3.

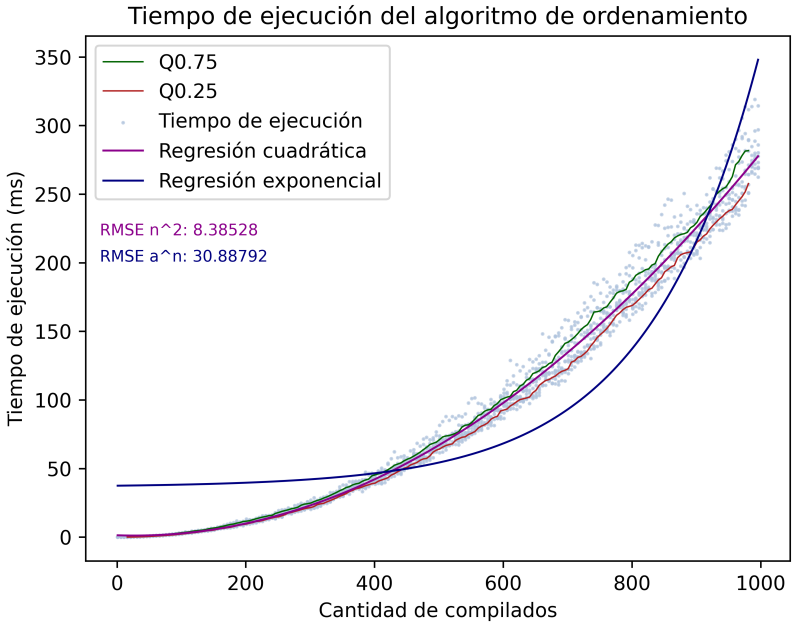


Figura 3: Tendencia de la complejidad algoritmica.

De esta manera, mediante el gráfico y la validación a través del error cuadrático medio, hemos comprobado que la curva cuadrática se adapta de manera más precisa a nuestros datos.

De esta forma pudimos comprobar empíricamente que la complejidad tiende a $O(n^2)$.

Nótese que graficamos también dos curvas que demarcan una estimación de los cuantiles verticales 0,25 y 0,75. Esta estimación se realizó calculando los cuantiles para un grupo pequeño centrado en cada punto. Estas curvas nos ayudan a dimensionar cómo la varianza del tiempo de ordenamiento crece con el aumento del tamaño de la información de entrada.

Adicionalmente, graficamos la densidad de los tiempos que también brinda una visualización de cómo aumenta la varianza con el aumento del tamaño de los datos de entrada. Ver figura 4.

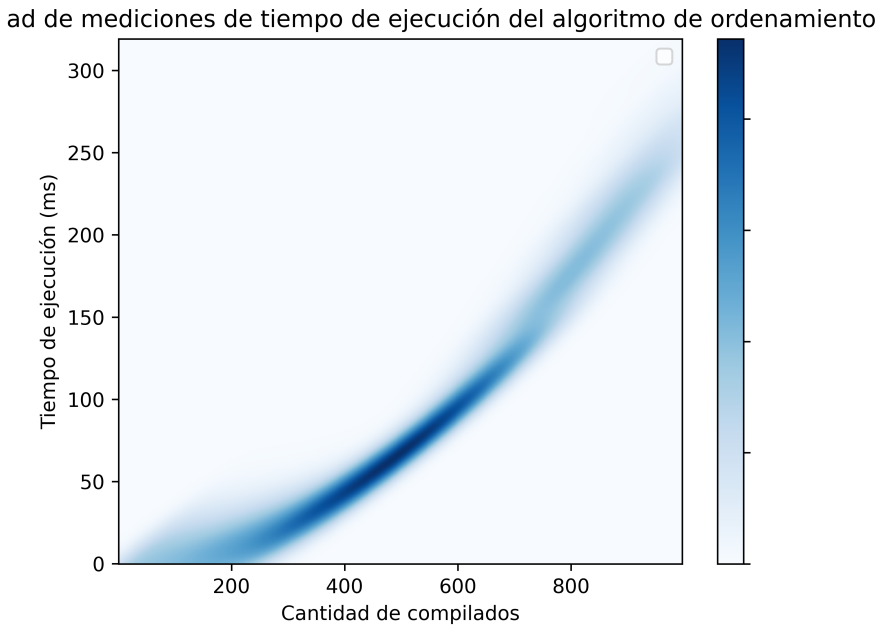


Figura 4: Densidad de mediciones.

6. 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 compilados y permite que el tiempo invertido por Scaloni y sus asistentes se administre de manera óptima.