.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 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 consecutivo.
- 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, se pide 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:

- Primer escenario:

$$n = 3, \quad E = \boxed{10}\boxed{2}\boxed{2} \qquad S = \boxed{1}\boxed{5}\boxed{4}$$

- Segundo escenario:

$$n = 4, \quad E = \boxed{5}\boxed{3}\boxed{8}\boxed{10} \qquad S = \boxed{10}\boxed{1}\boxed{1}\boxed{1}$$

Notación: j es la cantidad de días consecutivos de entrenamiento. De esta manera s_j representa la energía disponible para cierto día.

Primero procederemos a analizar el primer caso ya que es el proporcionado por la cátedra. En este, notamos que la ganancia de un día i puede tomar 2 valores posibles:

- Suma entre la ganancia del día anterior y el mín (e_i, s_j)

- Suma entre la ganancia de dos días anteriores y el mín (e_i, s_1). Esta opción se basa en la suposición de que se realizó un descanso en el día $i-1$.

Esto se debe a que hemos observado una relación con el problema de Juan el Vago. En este problema, se cuenta con un arreglo de ganancias diarias, y Juan busca maximizar su ganancia sin trabajar dos días seguidos. Para lograr esto, se itera sobre el arreglo observando los días actuales y los dos días anteriores para calcular la ganancia máxima acumulada. Si Juan elige trabajar el día i, su ganancia sería la suma entre la ganancia del día actual (g_i) y la ganancia de dos días atrás (g_{i-2}). En caso contrario, si decide no trabajar, su ganancia sería igual a la ganancia del día anterior (g_{i-1}). De estas dos opciones, se elige la que maximiza la ganancia para el día i.

Esta estrategia nos recuerda al enfoque que estamos explorando en este algoritmo, donde se busca maximizar la ganancia acumulada eligiendo si conviene o no descansar en el día actual, considerando la energía disponible y el esfuerzo requerido en cada día.

Relacionando el problema de Scaloni con el problema de Juan el Vago, supusimos erroneamente que para obtener la ganancia máxima hasta el día $k, \forall k \in [2, n]$ bastaría con calcular la ganancia máxima de los dos días anteriores. La función de recursión propuesta fue la siguiente:

$$g(i, j) = \text{máx}\left\{g(n-1) + \text{mín}\left(e_i, s_j\right), g(n-2) + \text{mín}\left(e_i, s_1\right)\right\}$$

En relación al caso de tres días que estábamos analizando para ese entonces, cabe destacar que este algoritmo proporcionaba la respuesta esperada, que en este contexto era igual a siete.

El inconveniente de 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 $k < i-1$ y solo se hace evidente recién en el día i.

Dado que la ganancia está condicionada por los niveles de energía de cada día consecutivo, en ocasiones es más conveniente que la ganancia relativa de un día no sea óptima individualmente, sino que en conjunto ofrezca la mayor ganancia posible. El algoritmo propuesto inicialmente 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.

Esto se puede observar en el segundo escenario propuesto. El algoritmo incial procede a solucionarlo de la siguiente manera:

i	días trabajo consec.	ganancia máxima
1	1	4
2	2	7
3	3	15
4	1	17

Por lo tanto, el resultado obtenido es 17 descansando en el día 3, mientras que el resultado óptimo es 22, descansando en el día 2. Este es un claro contraejemplo que ilustra que la decisión óptima de descansar en el día 2 no es tomada en consideración ya que no puede deducirse manteniendo registro de únicamente las ganancias de los dos días anteriores.

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 de fila i y columna j es la ganancia parcial que podríamos obtener si ese fuera el j-ésimo día de entrenamiento consecutivo para el día i del plan de entrenamiento. Notamos que la ganancia máxima hasta el entrenamiento i 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
		j→	0	1	2	3

Figura 1: Caso 2 planilla

Luego, logramos abstraer las funciones, mostradas en la figuras 2, 3 y 4.

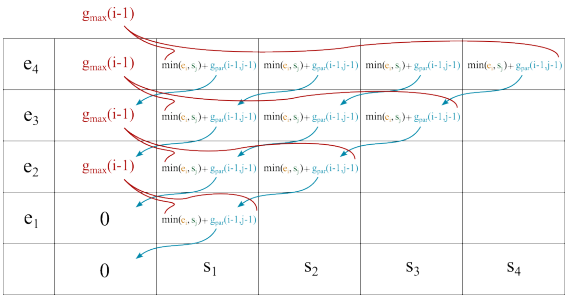

Figura 2: Formulas en planilla a

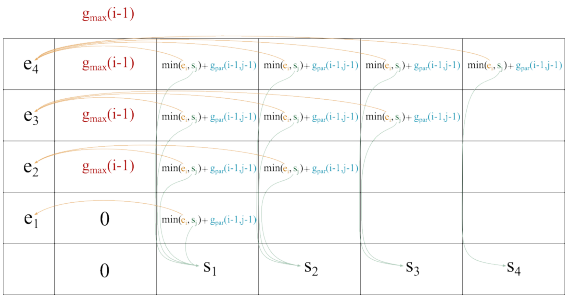

Figura 3: Formulas en planilla b

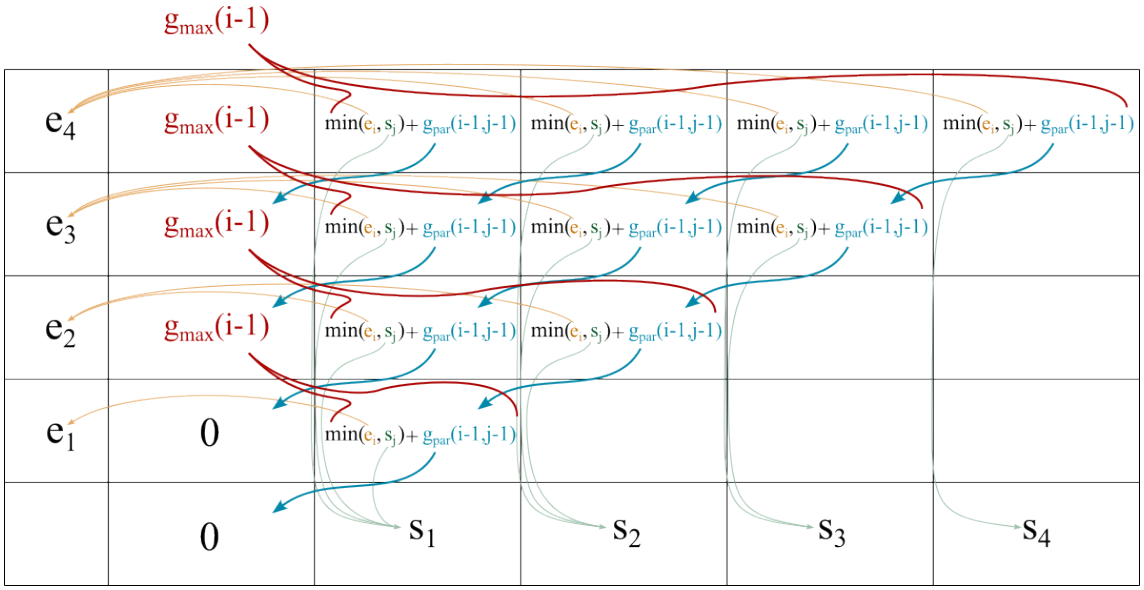

Figura 4: Formulas en planilla

Basándonos en esta planilla, pudimos abstraer la solución.

3. Solución final

A continuación procederemos a describir el algoritmo solución desarrollado con Programación Dinámica.

3.1. Función de recurrencia

Sea s_j la energía restante para j días consecutivos de entrenamiento y e_i la cantidad de esfuerzo del entrenamiento i.

El problema nos da dos arreglos de tamaño n:

$$S = (s_1, s_2, \ldots, s_n) \quad E = (e_1, e_2, \ldots, e_n)$$

Definimos la función $g(i, j)$ como la ganancia obtenida del i-ésimo entrenamiento, dado que estamos en el día j de entrenamiento consecutivo.

$$g(i, j) = \text{mín}\,(e_i, s_j)$$

Definimos la función $g_{par}(i, j)$ como la ganancia parcial acumulada hasta el i-ésimo entrenamiento dado que estamos en el día j de entrenamiento consecutivo.

$$\forall j \leq i,\ g_{par}(i, j) = \begin{cases} 0 & i = 0 \\ g(i, j) & i = 1 \\ g(i, j) + g_{max}(i - 2) & i > 1 \wedge j = 1 \\ g(i, j) + g_{par}(i - 1, j - 1) & i > 1 \wedge j > 1 \end{cases}$$

Finalmente, definimos la función $g_{max}(i)$ como la ganancia máxima obtenida hasta el i-ésimo entrenamiento, que es la solución al problema de la consigna.

$$g_{max}(i) = \text{máx}\,\{g_{par}(i, j)\}_{j \leq i}$$

3.2. Solución iterativa

Como podemos observar en la sección anterior, la solución es recursiva. Implementarla de esta forma sin optimizaciones requeriría mucho recálculo. Esto puede ser solucionado con memoization y la implemantación más simple es de forma iterativa y bottom-up.

Definimos la función `optimizar_entrenamiento` que recibe como parámetro `n`, un arreglo con los esfuerzos requeridos en los entrenamientos y otro con las energías disponibles por días de entrenamiento consecutivo. Devuelve la ganancia máxima y un array con el plan de entrenamiento óptimo.

```
1 def optimizar_entrenamiento(n, esfuerzos, energias):
2     ganancias_parciales = _ganancias_parciales(n, esfuerzos, energias)
3     ganancia_maxima = _ganancia_maxima(ganancias_parciales)
4     plan_entrenamiento_optimo = _plan_entrenamiento_optimo_2(ganancias_parciales)
5     return ganancia_maxima, plan_entrenamiento_optimo
```

La función `_ganancias_parciales(n, esfuerzos, energias)` genera una lista de longitud n donde la casilla i es una lista de longitud i con las ganancias parciales de cada día de entrenamiento consecutivo para el entrenamiento i. Se construye bottom-up, es decir, primero se calculan las ganancias parciales para el entrenamiento 1, luego para el 2 y así sucesivamente hasta el n.

Luego, `_ganancia_maxima(ganancias_parciales)` devuelve el máximo de las ganancias parciales de la última fila.

Finalmente, `_plan_entrenamiento_optimo(ganancias_parciales)` reconstruye el plan de entrenamiento óptimo a partir de la lista de ganancias parciales. Para esto, recorre la lista de ganancias parciales desde el final al principio. Parte de la ganancia parcial máxima de la última fila y baja en diagonal hasta la primera columna. Esta se refiere a entrenar ese día siendo el primer día de entrenamiento consecutivo, lo que implica que el día anterior se descanzó. Por esto, se salta la linea anterior y sigue con la anterior de la anterior, partiendo de nuevo desde la ganancia parcial máxima y bajando en diagonal. Repite estos pasos hasta llegar a la primera fila.

3.3. Cota de complejidad temporal

Para obtener la solución óptima, requerimos analizar cada día de entrenamiento, y por cada uno calculamos su ganancia respecto a cada nivel de energía disponible. Esto implica que, para cada día i se hacen como máximo $n-1$ operaciones $O(1)$ y una operación con complejidad $O(n)$ (por buscar la máxima ganancia parcial del día $i-2$). Esto implica que por cada día se hacen operaciones de $(n-1)O(1) + O(n) = nO(1) + O(1) + O(n) = O(n)$ y como se recorren todos los días, finalmente el algoritmo presentado tiene una complejidad temporal $\mathrm{O}(n^2)$.

4. 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 5.

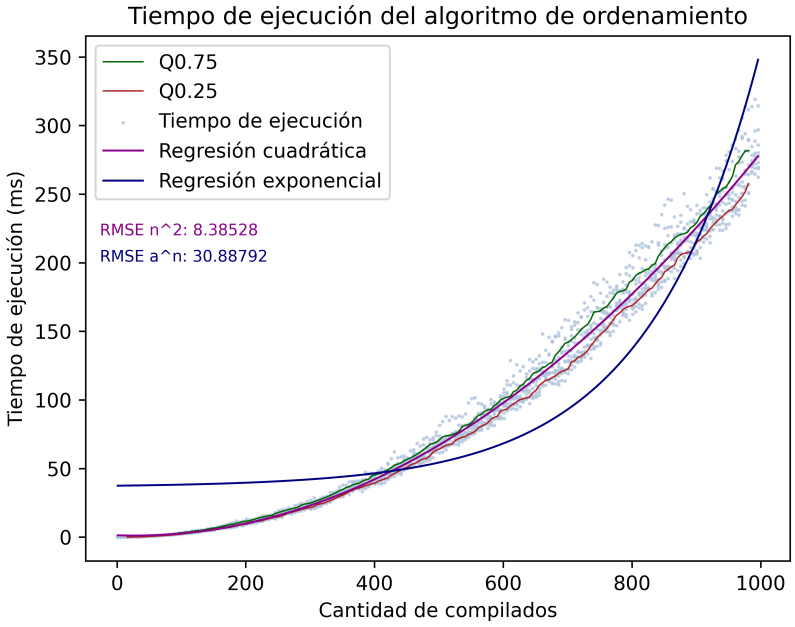

Figura 5: Tendencia de la complejidad algoritmica.

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 6.

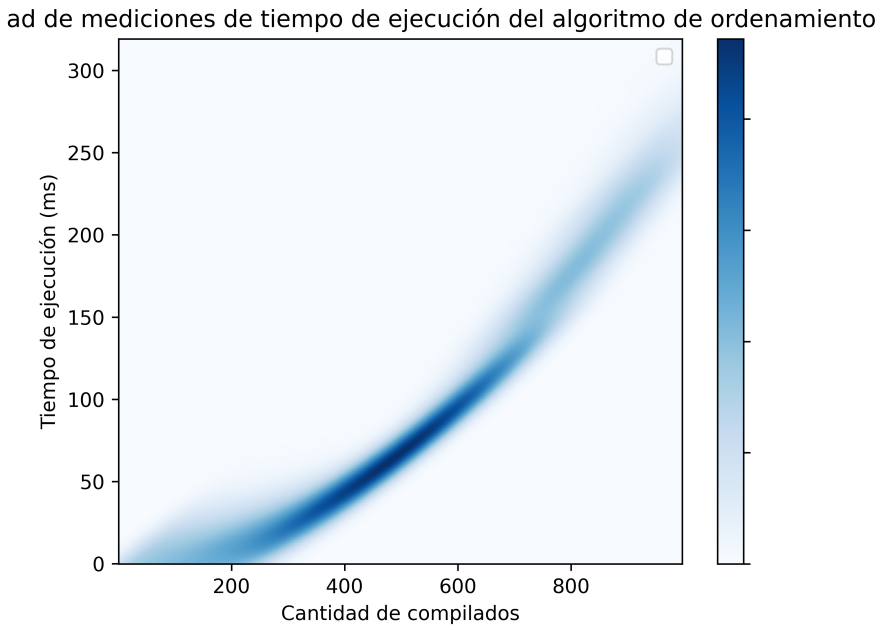

Figura 6: Densidad de mediciones.

5. Conclusiones

Finalmente, consideramos que la solución óptima para abordar la problemática de Scaloni aplicando la técnica de diseño de programación dinámica debe considerar todas las ganancias parciales hasta el día n. Utilizando memoization, la complejidad del algoritmo terminó siendo $O(n)$. A través de esta estrategia, se logra maximizar la ganancia total, teniendo en cuenta las limitaciones de energía y los esfuerzos requeridos en cada día.