.UBAfiuba
FACULTAD DE INGENIERÍA

Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico 2
Programación Dinámica

9 de octubre de 2023

Juan Díaz
103488

Juan Díaz
108183

Juan Muñoz
106699

1. Introducción

En el presente trabajo práctico se realiza un analisis de un algoritmo por *Programación Dinámica* para resolver un problema de maximización.

2. Consideraciones

Un cronograma de entrenamiento indica cuanto esfuerzo demanda el entrenamiento de cada día. Se puede calcular la ganancia de cada día como el mínimo entre el esfuerzo requerido por el entrenamiento para ese día (e_i), y la enegía que tienen los jugadores en ese día (j) o 0 si descansan.

La energía que tienen los jugadores en un día determinado depende de hace cuantos días descansaron por última vez y está dada por la secuencia $\mathcal{S}$:

$$\mathcal{S} = s_1, s_2, ..., s_n$$
$$\text{con } s_1 \geq s_2 \geq ... \geq s_n$$

3. Problema

Dada la secuencia de energía disponible desde el último descanso $\mathcal{S}$, y el esfuerzo/ganancia de cada día, determinar la máxima cantidad de ganancia que se puede obtener de los entrenamientos, considerando posibles descansos.

3.1. Ecuación de Recurrencia

Los jugadores comienzan descansados y sin ninguna ganancia:

$$\mathcal{G}(0,0) = 0$$

Si eligen descansar en el día k, al día siguiente se renueva su energía, por lo que lo mejor será maximizar la ganancia del día anterior:

$$\mathcal{G}(0,k) = \max_{0 \leq i < k} (\mathcal{G}(i, k-1)) \tag{1}$$

Por otro lado, si no descansan la ganancia para ese día depende de su energía y de la ganancia del día anterior:

$$\mathcal{G}(i,k) = \mathcal{G}(i-1, k-1) + \min(s_i, e_k)$$

Lo que nos deja con la siguiente ecuación de recurrencia:

$$\begin{cases} \mathcal{G}(0,0) = 0 \\ \mathcal{G}(0,k) = \max_{0 \leq i < k} (\mathcal{G}(i, k-1)) \\ \mathcal{G}(i,k) = \mathcal{G}(i-1, k-1) + \min(s_i, e_k) \end{cases} \tag{2}$$

donde $\mathcal{G}(0, n+1)$ es la solución al problema.

Resolución

3.2. Algoritmo

Para encontrar $\mathcal{G}(0, n+1)$, proponemos el siguiente algoritmo por *Programación Dinámica*:

Para cada día, considerar la posibilidad de haber descansado por última vez en cualquiera de los días anteriores, y calcular la ganancia correspondiente. Además, considerar que se puede descansar en el mismo día, en cuyo caso la ganancia máxima del día anterior es la ganancia para ese día.

De esta forma podemos calcular las posibles ganancias de cada día de entrenamiento utilizando únicamente las posibles ganancias del día anterior. Como pudieron haber descansado por última vez en cualquiera de los días anteriores, los cuales son a lo sumo n, la complejidad espacial de nuestro algoritmo es $\mathcal{O}(n)$.

3.3. Implementación

```
def maximizar_ganancias(entrenamientos, energias):
    ganancias = [0]

    for dia_actual in range(len(entrenamientos)):
        ganancias.append(max(ganancias))
        for i in range(dia_actual + 1):
            ganancias[i] += min(energias[dia_actual-i], entrenamientos[dia_actual])

    return max(ganancias)
```

La ecuación de recurrencia que corresponde a este algoritmo es:

$$\mathcal{T}(n) = \mathcal{T}(n-1) + \mathcal{O}(n)$$

Esto es porque en cada iteración calculamos el máximo de un arreglo de tamaño n ($\mathcal{O}(n)$), e iteramos por el mismo realizando operaciones de tiempo constante ($\mathcal{O}(n)$).

Aplicando en teorema maestro, o multiplicando por la cantidad de entrenamientos, nos queda que la complejidad es $\mathcal{O}(n^2)$.

3.4. Reconstrucción

Para encontrar el camino que maximiza el esfuerzo alcanza con tener el estado final del arreglo de ganancias propuesto anteriormente, ya que con este se pueden reconstruir todos los estados anteriores del mismo. Considerando un arreglo de ganancias para el día n con un máximo en la posición k se da que:

$$ganancias[k] = \mathcal{G}(0, k+1) + \sum_{i=1}^{n-k} \text{mín}(s_i, e_{k+i})$$

Que es equivalente a decir que el camino cumple con haber descanzado el día k y trabajado todos los días posteriores.

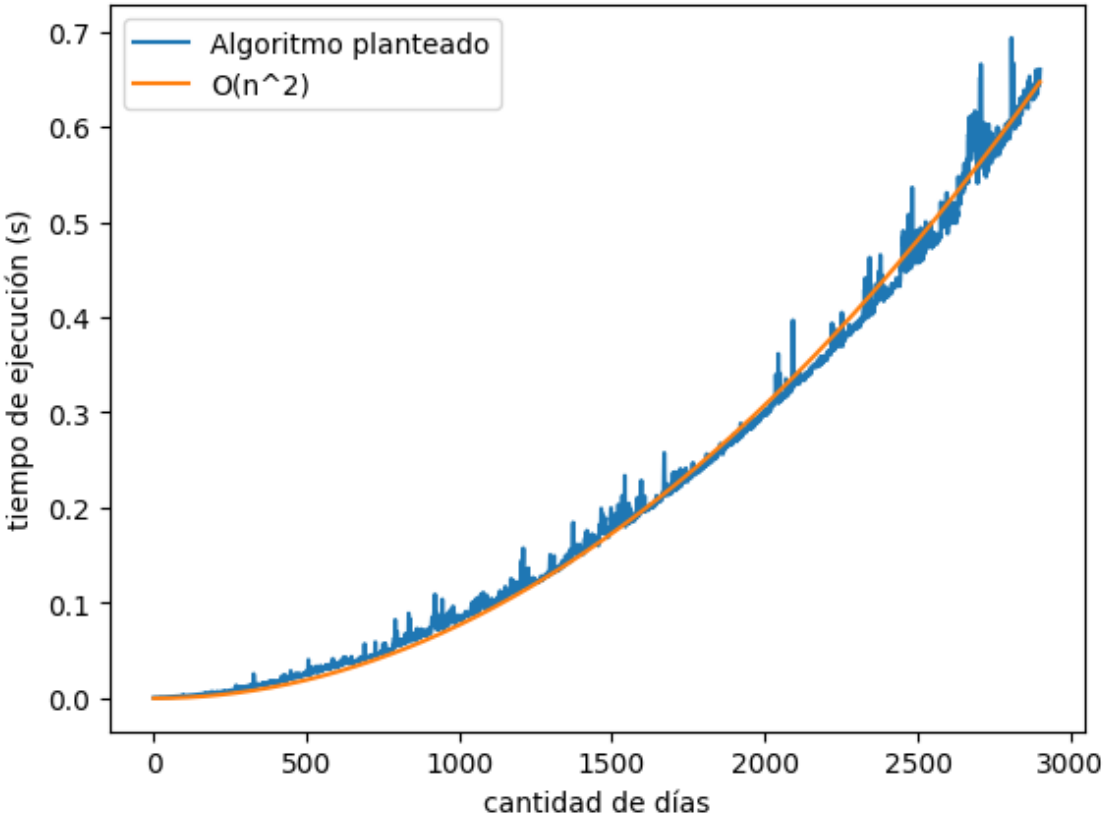
Con esto en mente, alcanza con restarle la sumatoria $\sum_{i=1}^{n-k} \text{mín}(s_i, e_{k+i})$ al elemento en la posición k para recuperar el valor que tenía en el día k.
Generalizando: el valor en el índice j para el día k, con $j \leq k$, puede recuperarse con:

$$ganancias[j] - \sum_{i=1}^{n-k} \text{mín}(s_{k+i-j}, e_{k+i})$$

Una vez reconstruido el arreglo para el día k, y considerando la equación para $\mathcal{G}(0, k)$ (1), tenemos que el problema se reduce a una versión más pequeña de sí mismo.

4. Mediciones

Se realizaron mediciones en base a crear arreglos de diferentes largos con valores de energía consumida por entrenamiento y energías disponibles en el rango de $[1, 100]$ (como en los casos de prueba provistos por la catedra), yendo de 10 en 10 elementos, donde los elementos en cada caso fueron generados por los valores pseudoaleatorios del lenguaje (el módulo `random`).



Como se puede apreciar, el algoritmo tiende a $\mathcal{O}\left(n^2\right)$.

5. Conclusiones

La Programación Dinámica demostró ser una técnica eficiente para resolver el problema de Scaloni, reduciendo a $\mathcal{O}(n^2)$ un problema que parecía tener complejidad temporal exponencial.

No solo eso, sino que también pudimos reconstruir la solución en $\mathcal{O}(n^2)$ -por lo tanto no empeorando la complejidad del algorítmo- y todo esto manteniendo la complejidad espacial $\mathcal{O}(n)$.

Destacamos que nuestro algorimo no considera la condicion inicial de $e_i >= e_{i+1}$, por lo que nuestro programa podria hayar soluciones para casos en los que no este dada esta restricción.