.UBAfiuba
FACULTAD DE INGENIERÍA

Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico 2
Programación Dinámica

10 de octubre de 2023

Nombre	Padrón
Buono, Fernando	103523
Katta, Gabriel	105935
Jarmolinski, Arian	94727

1. Análisis del Problema

La problemática a ser estudiada en el siguiente trabajo consiste en buscar la ganancia óptima de un cronograma de entrenamiento para Scaloni de n días, tal que para cada día i el esfuerzo que requiere hacer ese entrenamiento es e_i. Estos entrenamientos que tienen asociado el esfuerzo e_i para cada día i están bien definidos y son inamovibles.
La ganancia que se obtiene por hacer el entrenamiento un día de entrenamiento es justamente e_i
Si la cantidad de energía que se tiene para un día i es $j<e_i$, entonces la ganancia que se obtiene para ese día es justamente j.
Los jugadores disponen de una cantidad de energía que tienen disponible para cada día va disminuyendo a medida que pasan los entrenamientos. Empiezan con energía inicial s_1, luego para el segundo entrenamiento disponen s_2, y así sucesivamente, tal que $s_1 \geq s_2 \geq ... s_n$. Scaloni puede decidir dejarlos descansar un día, haciendo que la energía "se renueve", es decir, el próximo entrenamiento lo harían con energía s_1, y ese día no habría ganancia.

1.1. Análisis de Subproblemas

- Caso Base: Entrenar 0 días nos da una ganancia de 0.
- Caso 1 Día: Existe la posibilidad de entrenar o no un día. Si se conviene entrenar la ganancia sera el mínimo entre el esfuerzo del entrenamiento y la energía a usarse.
- Caso 2 Días: Existe la posibilidad de entrenar el segundo día, para esto hay que tener en cuenta lo siguiente:
  - Si se descanso el día anterior, en este caso la ganancia seria equivalente a el mínimo entre el esfuerzo del día actual y la energía máxima disponible mas la ganancia del caso base que seria 0 (ya que nos salteamos el día 1).
  - Si se entreno el día anterior, la ganancia en este caso es equivalente a la ganancia del día anterior mas el mínimo entre el esfuerzo y energía del día actual.
- Caso 3 Días: Similar al caso anterior, sin embargo si se descanso el día 2, se debe tomar en cuenta la solución del día 1 y no el caso base, las otras consideraciones siguen manteniéndose.

1.2. Ecuación de Recurrencia

Para resolver el problema planteado, es necesario determinar si en un día particular i se maximiza la ganancia descansando o entrenando. Este ecuación de recurrencia contempla ambas situaciones y se queda con el optimo.

Con esto podemos plantear la siguiente ecuación de recurrencia:

- $Gan(i,j) = max(Gan_{Entrenando}, Gan_{Descansando})$
- $Gan_{Entrenando} = Gan(i-1, j-1) + min(s_j, e_i)$
- $Gan_{Descansando} = Gan(i-2, j-2) + min(s_1, e_i)$

Análisis de la Solución

1.3. Algoritmo Propuesto

```python
def calcular_ganancia_maxima(esfuerzos, energias, dias):
    ganancia = [[0] * (dias) for _ in range(dias)]

    for i in range(1,dias):
        for j in range(1,dias):
          ganancia_entrenando = ganancia[i-1][j-1] + min(energias[j], esfuerzos[i])
          ganancia_descansando = 0 if (i < 2 or j < 2) else ganancia[i-2][j-2] +
    min(energias[1], esfuerzos[i])
          ganancia[i][j] = max(ganancia_entrenando, ganancia_descansando)
          ganancia[i][1] = max(ganancia[i][1], ganancia_descansando)

    return ganancia, max(ganancia[dias-1])
```

1.4. Reconstrucción de la Solución

En conjunto con el algoritmo propuesto, anexamos el código encargado de imprimir el itinerario como respuesta, basándose en la matriz resultado de la solución.

```python
def plan_entrenamiento(entrenamiento):
    esfuerzos, energias = entrenamiento[0], entrenamiento[1]
    dias = len(energias)
    matriz_ganancia, ganancia_maxima = calcular_ganancia_maxima(esfuerzos, energias
    , dias)
    plan = []

    i, j = dias-1, np.argmax(matriz_ganancia[dias-1])

    while i > 0:
        plan.append("Entrenar")
        if j != 1:
            i -= 1
            j -= 1
        else:
            i -= 2
            j = np.argmax(matriz_ganancia[i])
            if i >= 0:
              plan.append("Descansar")

    return ganancia_maxima, plan[::-1]
```

Nota de color: El ciclo para la reconstrucción tiene como cota superior $O(n^2)$, siendo n los días de entrenamiento.

1.5. Almacenamiento de Soluciones Previas

La estructura elegida para el almacenamiento de soluciones es una matriz de tamaño $n \times n$, donde n es la cantidad de días a ser analizados. En cada posición de la matriz se guarda su respectiva ganancia.

Cada fila de la matriz representa un día del cronograma, mientras que cada columna de la matriz representa la energía que está siendo utilizada (es un elemento del arreglo de energías dado por parámetro). De esta manera, en la columna 1 se guarda la ganancia al descansar el día anterior, es decir, la ganancia al contar con energía restaurada a su máximo nivel.

0	350	342	356	365	364	380	302	327	267	300
0	301	315	324	324	342	279	309	243	272	254
0	282	291	291	309	246	286	226	259	244	244
0	230	242	268	206	248	201	221	221	221	221
0	181	219	165	208	163	196	181	181	181	179
0	158	116	167	123	158	158	158	158	158	158
0	57	118	82	118	118	118	118	116	112	96
0	99	63	99	99	99	99	99	99	86	80
0	2	38	38	38	38	38	38	25	19	15
0	36	36	36	36	36	36	23	17	13	10
0	0	0	0	0	0	0	0	0	0	0

En la figura se muestra una matriz de ejemplo generada por el algoritmo.

NOTA Dentro del código se eligió como índice inicial el 1 en lugar de 0 para estar alineados al acceder a los índices de las listas tanto de esfuerzos como de energías. Esto se refleja en los rangos de los ciclos del algoritmo.

1.6. Implementación de la Ecuación de Recurrencia

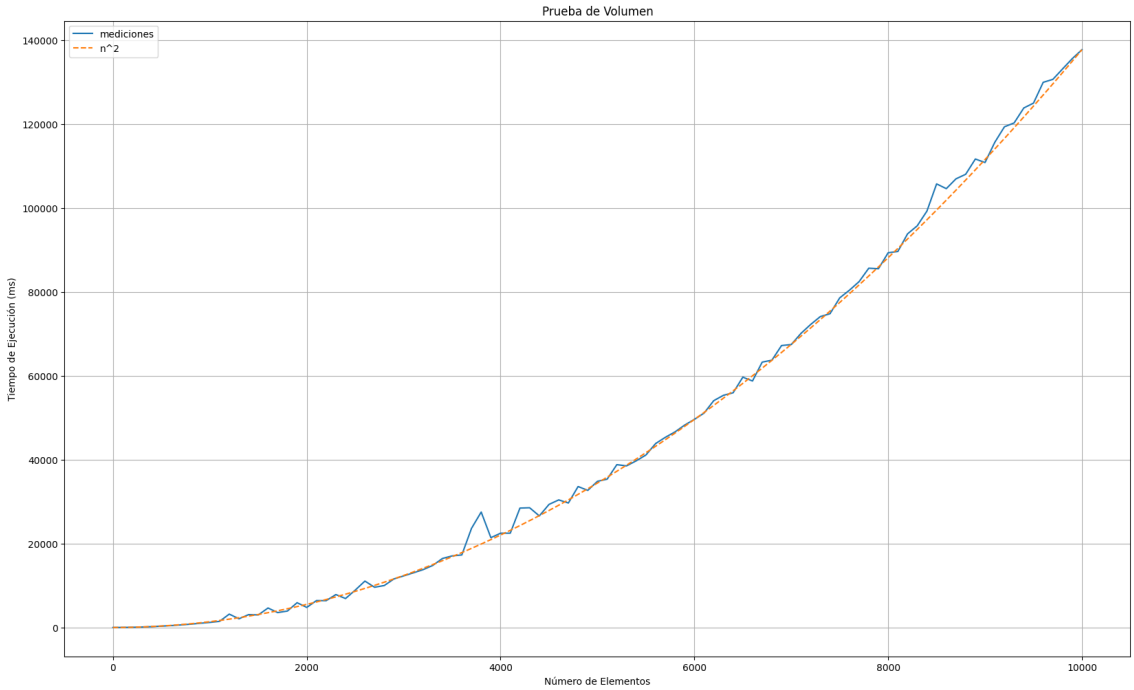
Para cada posición (i, j) de la matriz, se obtiene la ganancia óptima por haber entrenado y se almacena en la posición $(i, 1)$ la ganancia por haber descansado.

1.7. Complejidad Algorítmica

La complejidad del algoritmo propuesto es de $O(n^2)$, donde n es la cantidad de días de entrenamiento. El cuadrado surge de la necesidad de recorrer la matriz mediante un doble ciclo anidado. Además, las operaciones dentro de ambos ciclos tienen complejidad constante.

1.8. Análisis de Tiempos de Ejecución

Para corroborar que la complejidad analizada es correcta, se realizaron pruebas de volumen y se midieron los tiempos de ejecución mientras se incrementaba progresivamente la cantidad de días a ser analizados por el algoritmo.



Se puede observar en el gráfico que los tiempos de ejecución crecen de acuerdo con la línea de guía trazada por una función cuadrática, corroborando así la complejidad algorítmica $O(n^2)$ que se había propuesto.

2. Optimalidad de la Solución

Además de analizar la complejidad temporal de una solución propuesta, es importante corroborar el comportamiento de esta ante diversos casos de uso, asegurando de esta manera la robustez de la lógica propuesta ante volúmenes de datos variables.

2.1. Casos de Prueba

En el GitHub de la entrega se pueden observar los diversos casos de prueba a los que fue sometido el algoritmo propuesto. Con el objetivo de mantener la simplicidad del presente documento, optamos por describir un resumen de lo que se realizó:

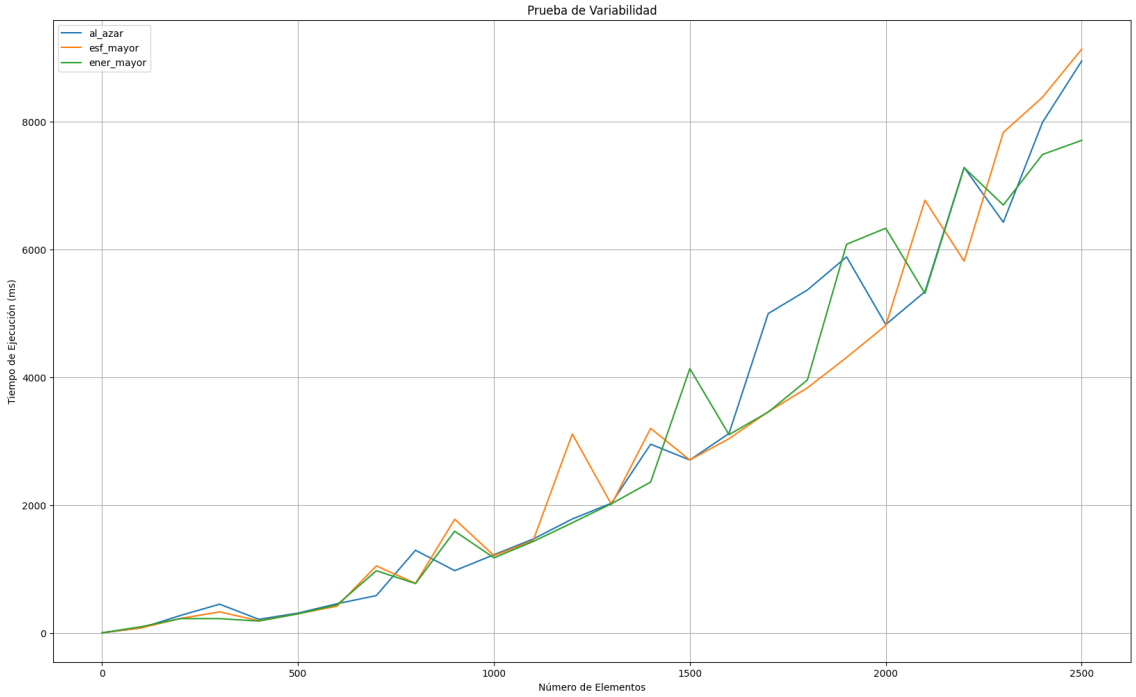
Utilizando los casos de prueba proporcionados por la cátedra, el algoritmo propuesto como solución cumple con todos los casos de optimalidad.

2.2. Variabilidad de Datos

Para comprobar que el algoritmo propuesto conservara su comportamiento cuadrático con datos de distinto orden de magnitud, se sometió a las siguientes pruebas:

- Igualdad: Las energías y esfuerzos proporcionados contaban con la misma variabilidad en sus generaciones.
- Esfuerzos Mayores: Los esfuerzos proporcionados se generaban siempre con un piso mayor que la cota superior de las energías, de esta manera eran siempre mayores.
- Energías Mayores: Las energías proporcionadas se generaban siempre con un piso mayor que la cota superior de los esfuerzos, de esta manera eran siempre mayores.

El análisis temporal de ejecución es el siguiente:



Es observable que en los 3 casos (obviando algunos spikes de CPU no relacionados), la evolución del tiempo de ejecución es la misma para todos, lo cual nos indica que no afecta significativamente el rendimiento la variabilidad de los datos proporcionados al algoritmo.

3. Conclusiones

Tras los análisis realizados en las secciones previas podemos concluir que el algoritmo realizado mediante programación dinámica que cumple con la solución de manera óptima es el siguiente:

```
def calcular_ganancia_maxima(esfuerzos, energias, dias):
    ganancia = [[0] * (dias) for _ in range(dias)]

    for i in range(1,dias):
        for j in range(1,dias):
          ganancia_entrenando = ganancia[i-1][j-1] + min(energias[j], esfuerzos[i])
          ganancia_descansando = 0 if (i < 2 or j < 2) else ganancia[i-2][j-2] +
    min(energias[1], esfuerzos[i])
          ganancia[i][j] = max(ganancia_entrenando, ganancia_descansando)
          ganancia[i][1] = max(ganancia[i][1], ganancia_descansando)

    return ganancia, max(ganancia[dias-1])
```

Primero fue sometido a los casos de optimalidad propuestos por la materia, llegando en cada uno de estos a la solución correcta y óptima.

Acto seguido se corroboró su complejidad algorítmica mediante pruebas de volumen y se pudo concluir que era $O(n^2)$. A su vez, la complejidad espacial se encuentra en la manera de almacenar las soluciones previas utilizadas para construir nuevas soluciones, justamente en este caso termina siendo: $O(\text{dias}^2)$ tomando en cuenta que la matriz es cuadrada y de tamaño igual a los días a analizar.

Las pruebas de variabilidad realizadas consistieron en alterar la cantidad de los datos, tanto energías como esfuerzos en relación a la otra, con el objetivo de observar cómo estas variaciones afectan al desempeño del algoritmo. Lo observado indica que es indistinto para este.