Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico 2
Programación Dinámica

9 de octubre de 2023

Agostina Picchetti - 108028

Nehuén Cabibbo - 108025

Roberta Sprenger - 104239

Índice

1. Introducción

Programación Dinámica es una técnica de diseño de algoritmos que resuelve los problemas explorando implícitamente todo el espacio de soluciones posibles mediante la descomposición en subproblemas más pequeños que permiten luego ir construyendo soluciones más grandes hasta conseguir la solución final.

En este trabajo práctico buscamos implementar una solución usando Programación Dinámica que permita maximizar la ganancia obtenida luego de finalizar el cronograma de entrenamiento propuesto por Scaloni.

2. Análisis del problema y diseño del algoritmo

Tras analizar y evaluar el problema, observamos que:

- Las datos involucrados son: la cantidad de días contemplados en el plan de entrenamiento, los esfuerzos demandados por cada día que se entrena, la energía disponible de los jugadores para cada día.
- El esfuerzo demandado puede ser mayor a la energía disponible para un día determinado o viceversa, la energía disponible puede ser mayor al esfuerzo demandado. Entonces, la ganancia del entrenamiento para un día i va a estar definida por $min(e_i, s)$ correspondiente.
- La energía disponible de los jugadores disminuye con los entrenamientos sucesivos, por lo tanto un s_i de un día determinado no puede ser mayor al del anterior. Los jugadores pueden descansar entre entrenamientos sin sumar ganancia para ese día, pero con la opción de reponer y restaurar la energía a su máximo valor, la del primer día. Si este es el caso, se produce un desplazamiento de los valores de s respecto de los de e ya que la secuencia de energía comienza desde su primer valor.
- Cada día del plan de entrenamiento existen dos opciones: descansar o entrenar.
- El último día del plan de entrenamiento la mejor opción siempre será entrenar. No importa con cuanta energía lleguen los jugadores al día n, si descansan no sumarían nada de ganancia, pero por más mínima que sea convendrá sumarla. Esta ganancia dependerá de cuántos días consecutivos se entrenaron porque el descanso previo condiciona cuánto será la energía disponible y por lo tanto la ganancia.
- No conviene descansar dos días seguidos porque implicaría perder un día de ganancia sin ningún beneficio sobre la energía disponible
- En principio, no podemos asumir nada respecto a la cantidad de descansos que tendrá un plan de n días, pero sí inferimos del punto anterior que, al no haber días consecutivos de descanso, no podrán superar los días de entrenamiento.
- Considerando las observaciones anteriores, entendemos que hay otra variable que tendremos que considerar para la optimización del plan. Además de la cantidad de días del plan con los respectivos esfuerzos necesarios y energía disponible, debemos considerar cuándo fue el último descanso y vincular cada día del cronograma con la cantidad de días consecutivos de entrenamiento.

A partir de las observaciones anteriores entendemos que existen diferentes opciones para planificar el entrenamiento de acuerdo a la cantidad de días que tenga el cronograma y cuándo se tomen los descansos:

- Para un plan de un día:
  - entreno

- Para un plan de dos días:
  - descanso-entreno
  - entreno-entreno
- Para un plan de tres días:
  - descanso-entreno-entreno
  - entreno-descanso-entreno
  - entreno-entreno-entreno
- Para un plan de cuatro días:
  - descanso-entreno-entreno-entreno
  - entreno-descanso-entreno-entreno
  - entreno-entreno-descanso-entreno
  - descanso-entreno-descanso-entreno
  - entreno-entreno-entreno-entreno

Para encontrar el plan que optimice la ganancia y evitar evaluar todas las opciones posibles según la cantidad de días del cronograma, aplicamos Programación Dinámica evitando así explorar todas las opciones posibles basándonos en la idea de *Memoization* para guardarnos los óptimos locales en una matriz que relaciona los días del cronograma con los días de entrenamiento consecutivos.

2.1. Ecuación de recurrencia

Para plantear la ecuación de recurrencia tenemos que tener en cuenta las variables del problema:

- i : numero de día
- d : días desde descanso / días de entrenamiento consecutivo
- e_i : cantidad de esfuerzo para el día i
- s_d : energía disponible acorde a cuando fue el ultimo día de descanso

Luego pensamos en nuestro subproblema y la decisión de descansar o entrenar.

Para esta parte trataremos de buscar el máximo entre la ganancia que nos dé cada decisión.

$$\text{óptimo: } max\,(descansar, entrenar)$$

Siendo que empezamos a calcular desde el ultimo día y que no lo descansamos, podemos saber cual será nuestra ganancia según cuantos días de entrenamiento de corrido tuvimos.

Con estos datos, podemos iterar por los días anteriores y calcular cada posibilidad:

- descansar: $optimo(i+1, d=1)$

  La ganancia será 0 para este día por lo que la ganancia parcial nos queda igual a la del día posterior con $d = 1$ (s_1)
- entrenar: $min(e_i, s_d) + optimo(i+1, d+1)$

  Calculamos la ganancia de ese día (mínimo entre el esfuerzo y la energía) con d entre $[1, i]$

  Y además sumamos la ganancia que habrá el día siguiente considerando un día más de entrenamiento consecutivo.

Así nos quedará definida nuestra ecuación de recurrencia como:

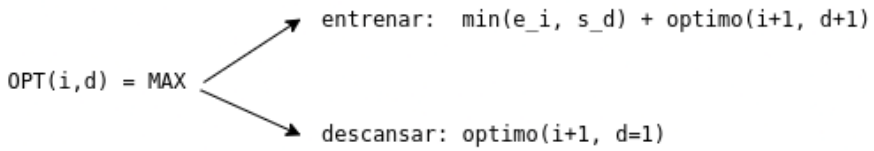

Figura 1: Ecuación de Recurrencia

2.2. Algoritmo propuesto

Proponemos un algoritmo iterativo que aplica *Memoization* usando una matriz de nxn (los días consecutivos de entrenamiento no pueden ser mayor a los días del cronograma) para guardar los óptimos que se van calculando.

Esta matriz tiene como filas a los días de entrenamiento i y como columnas a los días de entrenamiento consecutivo d.

i_3	4	2	2
i_2	7	2	
i_1	7		
d	1	2	3

Cuadro 1: Ejemplo matriz completa con óptimos locales para 3 días

Esta será la implementación de nuestro algoritmo:

```
def planificar_entrenamiento(n, esfuerzo, energia):
  memoria = [[0 for _ in range(n)] for _ in range(n)]
  memoria[0] = [min(esfuerzo[n-1], energia[d]) for d in range(n)]

  for i in range(1,n):
    for d in range(n-i):
      memoria[i][d] = max(memoria[i-1][d+1] + min(esfuerzo[n-1-i], energia[d]),
    memoria[i-1][0])

  return memoria[n-1][0], memoria
```

Como ejemplo de seguimiento mostramos como funcionaria nuestro algoritmo con el plan de 3 días:

días	1	2	3
e_i	1	5	4
s_i	10	2	2

Cuadro 2: Arreglos de esfuerzo y de energía

El primer paso será completar la fila del último día considerando que se entrena.

i_3	$\min(e_3, s_1) = \mathbf{4}$	$\min(e_3, s_2) = \mathbf{2}$	$\min(e_3, s_3) = \mathbf{2}$
i_2			
i_1			
d	1	2	3

Cuadro 3: Matriz de Memoization - 1era fila

Luego iteraremos por completo la matriz desde la segunda fila, calculando los óptimos locales con nuestra ecuación de recurrencia:

Celda M[i_2][1] : `max(M[`i_3`][1], min(`e_2`, `s_1`) + M[`i_3`][`s_2`]) -> max(4,5+2)`

Celda M[i_2][2] : `max(M[`i_3`][1], min(`e_2`, `s_2`) + M[`i_3`][`s_3`]) -> max(2,2+2)`

i_3	4	2	2
i_2	**7**	**4**	
i_1			
d	1	2	3

Cuadro 4: Matriz de Memoization - 2da fila

En la última iteración estaremos calculando la ganancia total:

Celda M[i_1][1] : `max(M[`i_2`][1], min(`e_1, s_1`) + M[`i_2`][s2]) -> max(7,1+4)`

i_3	4	2	2
i_2	7	4	
i_1	**7**		
d	1	2	3

Cuadro 5: Matriz de Memoization - Ultima fila/celda

Como podemos observar:

1. La primera fila de la matriz, que corresponde al ultimo día de entrenamiento, se completa calculando el mínimo entre el esfuerzo designado para ese día y la energía restante según la columna de días desde el ultimo descanso.
2. Al finalizar la iteración de la matriz memoria, nos quedará el resultado de la ganancia total máxima, en la primera columna de la última fila (primer día siendo que empezamos desde el ultimo)

3. Complejidad

Para hallar la ganancia máxima, se recorre una vez la matriz, donde cada fila corresponde al día del entrenamiento y cada columna al tiempo transcurrido desde el último descanso, haciendo operaciones $\mathcal{O}(1)$, por lo que la complejidad de hallar la ganancia total queda $\mathcal{O}(n^2)$, siendo n la cantidad de días de entrenamiento que es la misma que la cantidad máxima de días transcurridos desde el último descanso

4. Mediciones

Para programar nuestra solución elegimos el lenguaje `Python`.

Generamos set de datos con valores pseudo aleatorios usando el módulo `random` que exportamos en archivos de texto con el módulo `csv`.

Para las pruebas realizadas, medimos el tiempo de ejecución cinco veces para cada set de datos usando el módulo `time`. Estos valores se promediaron y el resultado lo usamos para realizar los gráficos usando la biblioteca `Seaborn`.

4.1. Tiempo de ejecución vs cantidad de días del plan de entrenamiento

Para el estudio del tiempo de ejecución en función de días del plan de entrenamiento (n), generamos set de datos con valores enteros pseudoaleatorios para s_i (procurando que estén de mayor a menor) y e_i de hasta diez mil elementos con intervalos de cien elementos.

A su vez, se ajustó por una función cuadrática para poder corroborar la complejidad antes justificada.

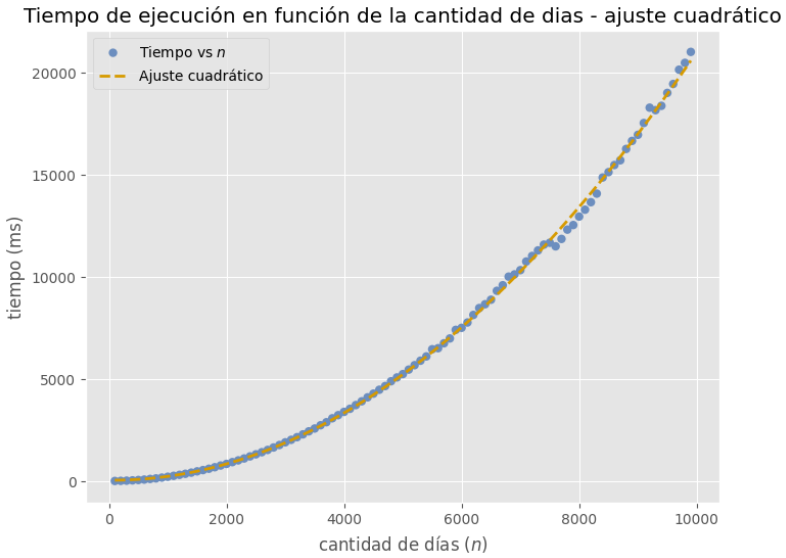

Figura 2: Tiempo de ejecución según la cantidad de días del plan de entrenamiento

Se puede apreciar que el algoritmo tiene una tendencia n^2 en función del tamaño de la entrada, lo que coincide con la complejidad teórica calculada anteriormente.

4.2. Variabilidad

En esta sección analizaremos brevemente el efecto de la variabilidad de s_i y los tiempos de ejecución del algoritmos. Generamos set de datos con valores pseudo aleatorios usando el módulo `random` acotando a un rango fijo los valores de e_i y variando en tres rangos s_i: 1-100, 1-100000, 1-10000000.

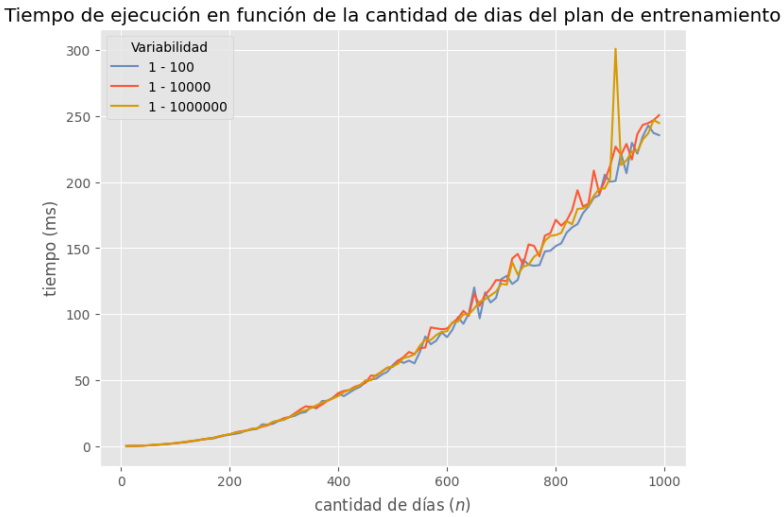

Figura 3: Tiempo de ejecución para diferentes rangos de variabilidad de s_i

Teniendo en cuenta el gráfico de la sección 4.1, podemos observar que, al variar los valores de s_i entre rangos de 1 a 10000000, no hay cambios significativos en los tiempos de ejecución.

Análogamente, generamos los sets de datos para e_i y visualizamos su variabilidad y los tiempos de ejecución y observamos que tampoco influyen en los tiempos de ejecución:

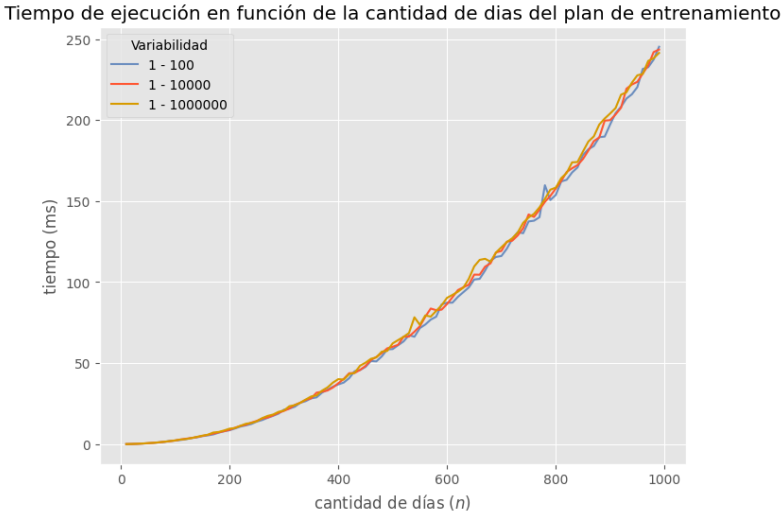

Figura 4: Tiempo de ejecución para diferentes rangos de variabilidad de e_i

5. Conclusiones

En este trabajo práctico, pudimos aplicar la técnica de diseño de algoritmos de Programación Dinámica para resolver un problema que pudimos relacionar con el problema de la mochila, con el objetivo de maximizar la ganancia obtenida en los entrenamientos por Scaloni.

Pudimos plantear nuestra ecuación de recurrencia luego de identificar la forma de los subproblemas, entendiendo la relación entre ellos, y cómo se componen para obtener la solución. Por último, al aplicar Programación Dinámica evitamos explorar un espacio exponencial de posibles soluciones.

A. Anexo: correcciones

Se incluyen en el presente anexo las correcciones sobre las secciones:

- 2.2 Algoritmo propuesto

Donde también incluimos la explicación de la reconstrucción de la solución.

2.2 Algoritmo propuesto

Proponemos un algoritmo iterativo que aplica *Memoization* usando una matriz de $n \times n$ (los días consecutivos de entrenamiento no pueden ser mayor a los días del cronograma) para guardar los óptimos que se van calculando.

Esta matriz tiene como filas a los días del plan de entrenamiento i y como columnas a los días de entrenamiento consecutivo d.

i_3	4	2	2
i_2	7	2	
i_1	7		
d	1	2	3

Cuadro 6: Ejemplo matriz completa con óptimos locales para tres días

Esta será la implementación de nuestro algoritmo:

```
def planificar_entrenamiento(n, esfuerzo, energia):
  memoria = [[0 for _ in range(n)] for _ in range(n)]
  memoria[0] = [min(esfuerzo[n-1], energia[d]) for d in range(n)]

  for i in range(1,n):
    for d in range(n-i):
      memoria[i][d] = max(memoria[i-1][d+1] + min(esfuerzo[n-1-i], energia[d]),
    memoria[i-1][0])

  return memoria[n-1][0], memoria
```

Los parámetros que recibe la función son:

- `n`: la cantidad de días que tiene el plan de entrenamiento
- `esfuerzo`: un arreglo con los esfuerzos necesarios (e_i) para cada día
- `energía`: un arreglo con los esfuerzos disponibles (s_i) para cada día

Como ejemplo de seguimiento mostramos como funcionaría nuestro algoritmo con el plan de tres días:

días	1	2	3
e_i	1	5	4
s_i	10	2	2

Cuadro 7: Arreglos de esfuerzo y de energía

El primer paso será completar la fila del último día considerando que se entrena.

i_3	$min(e_3, s_1) = \mathbf{4}$	$min(e_3, s_2) = \mathbf{2}$	$min(e_3, s_3) = \mathbf{2}$
i_2			
i_1			
d	1	2	3

Cuadro 8: Matriz de Memoization - 1era fila

Luego iteraremos por completo la matriz desde la segunda fila, calculando los óptimos locales con nuestra ecuación de recurrencia:

Celda $M[i_2][1]$: $max(M[i_3][1],\ min(e_2,\ s_1) + M[i_3][s_2]) \rightarrow\ max(4,\ 5+2)$
Celda $M[i_2][2]$: $max(M[i_3][1],\ min(e_2,\ s_2) + M[i_3][s_3]) \rightarrow\ max(2,\ 2+2)$

i_3	4	2	2
i_2	**7**	**4**	
i_1			
d	1	2	3

Cuadro 9: Matriz de Memoization - 2da fila

En la última iteración estaremos calculando la ganancia total:

Celda $M[i_1][1]$: $max(M[i_2][1],\ min(e_1, s_1) + M[i_2][s2]) \rightarrow\ max(7,\ 1+4)$

i_3	4	2	2
i_2	7	4	
i_1	**7**		
d	1	2	3

Cuadro 10: Matriz de Memoization - Ultima fila/celda

Como podemos observar:

1. La primera fila de la matriz, que corresponde al ultimo día de entrenamiento, se completa calculando el mínimo entre el esfuerzo designado para ese día y la energía restante según la columna de días desde el ultimo descanso.
2. Al finalizar la iteración de la matriz memoria, nos quedará el resultado de la ganancia total máxima, en la primera columna de la última fila (primer día siendo que empezamos desde el ultimo)

2.2.1 Reconstrucción de la solución

A partir de la matriz resultante, podemos reconstruir la solución con el plan de entrenamiento óptimo.

Para eso, implementamos el siguiente algoritmo:

```
def reconstruir_plan_entrenamiento(plan_entrenamiento):
    reconstruccion = []
    dias = len(plan_entrenamiento)
    d = 0
    for i in range(dias-1,-1,-1):
        if plan_entrenamiento[i][d] != plan_entrenamiento[i-1][0]:
            reconstruccion.append("Entreno")
            d+=1
            continue
        reconstruccion.append("Descanso")
        d=0

    return reconstruccion
```

La función recibe como parámetro la matriz resultante de la ejecución de `planificar_entrenamiento` y devuelve una lista de cadenas con la secuencia de entrenamientos y descansos correspondiente al plan óptimo.

Explicación del funcionamiento del algoritmo:

La reconstrucción comienza en la posición de la celda $M[i = 1][d = 1]$ que contiene el valor de la ganancia máxima y corresponde al primer día del plan que inicia con los jugadores descansados (como es el primer día no pueden haber entrenado el día anterior).

Como mencionamos anteriormente, cada día existe la posibilidad de entrenar o descansar y para reconstruir qué decisión fue tomada en el plan óptimo, siempre compararemos con lo que sucede al día siguiente (la fila siguiente) en $d = 1$. La celda $M[i + 1][d = 1]$ almacena la ganancia parcial considerando que $i + 1$ es el primer día de entrenamiento tras un descanso.

Como cada celda guarda las ganancias óptimas para cada caso, si los valores son distintos indica que el día i que estamos analizando se acumula ganancia y por lo tanto los jugadores entrenaron. Entonces, debemos avanzar en diagonal por la matriz, sumando 1 al valor de i y de d, entendiendo que al día siguiente llevaremos un día más consecutivo de entrenamiento.

De lo contrario, si ambos valores son iguales indica que el día i que estamos analizando no se acumula ganancia y por lo tanto se descansa. En este caso, avanzamos en la dirección de las filas $(i + 1)$, pero permanecemos en la primera columna $(d = 1)$ que indica que es el primer día de entrenamiento tras un descanso.

Iterativamente vamos comparando el valor de la posición actual con la posición $M[i+1][d = 1]$ y dependiendo del resultado de esa comparación se define:

- Si son distintos $\rightarrow$ implica día de `Entreno` y se avanza hacia la posición $M[i + 1][d + 1]$
- Si son iguales $\rightarrow$ implica `Descanso` y se avanza hacia la posición $M[i + 1][d = 1]$

El último día, como ya describimos en el análisis del problema, siempre será de `Entreno`.

Como ejemplo de seguimiento mostramos como funcionaría el algoritmo de reconstrucción con la matriz obtenida para el plan de tres días en la sección anterior.

Iniciamos comparando las celdas $M[1][1]$ y $M[2][1]$

i_3	4	2	2
i_2	**7**	2	
i_1	**7**		
d	1	2	3

Cuadro 11: Matriz solución para plan de tres días - Comparación de $M[1][1]$ con $M[2][1]$

Como ambos valores son iguales, el primer día es de descanso.
En la siguiente iteración, se comparan las celdas $M[2][1]$ y $M[3][1]$

i_3	**4**	2	2
i_2	**7**	2	
i_1	7		
d	1	2	3

Cuadro 12: Matriz solución para plan de tres días - Comparación de $M[2][1]$ con $M[3][1]$

Como los valores son distintos, el segundo día es de entrenamiento.
La siguiente iteración corresponde al último día del plan que siempre será de `Entreno`.
Finalmente, la reconstrucción del plan de entrenamiento óptimo resultará en la secuencia:
`Descanso, Entreno, Entreno`