Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico 2
Programación Dinámica

9 de octubre de 2023

Tomas Caporaletti
108598

Helen Chen
110195

Lucas Garcia Campos
110099

1. Introducción

Luego de haber ayudado a Scaloni y el equipo técnico a ordenar los análisis de los siguientes rivales de la selección **CAMPEONA DEL MUNDO**, Scaloni planteó un cronograma de entrenamiento para los jugadores, pero se cruzó con un problema: no sabe qué días conviene entrenar y cuales descansar para que los jugadores tengan la mayor ganancia posible y puedan dar el mejor desempeño en el próximo mundial. Pero que no cunda el pánico, Menotti le recomendó usar una técnica que será la que nos ayudará a resolver este problema: *"Programación Dinámica"*.

Ahora bien, ¿En qué consiste la *Programación Dinámica*? Para entender esto, es necesario entender ciertos puntos:

- Usa una técnica de optimización llamada *Memoization* (memorización en inglés). Esta técnica se basa en guardar los resultados ya calculados de un problema para poder ser reutilizados más adelante sin ser necesario que se calculen nuevamente.
- El problema que se desea solucionar debe poder descomponerse en subproblemas que a su vez estos permitan construir las soluciones a problemas más grandes.
- La cantidad de subproblemas debe ser polinomial.
- Esta técnica nos permite reducir la complejidad temporal de un algoritmo, evitando explorar un espacio exponencial de soluciones.

Una vez planteado esto, lo que se refiere a la *Programación Dinámica* es hacer uso de la *Memoization* y guardar las soluciones de los subproblemas más pequeños para poder ir construyendo las soluciones cada vez más grandes hasta llegar a dar con la solución deseada. Pero para poder aplicar esta técnica, es necesario antes saber la forma que tienen los subproblemas y como estos se combinan para dar solución a un problema más grande. Viendo por donde viene el asunto, se nos puede venir a la cabeza la *recurrencia*, y esto es correcto ya que esta técnica se sustenta gracias al uso de este método, y la solución se hallara gracias a haber hallado la ***Ecuación de Recurrencia*** del problema. Una vez obtenida, se puede plantear el problema iterativamente (en vez de hacerlo recursivo) pero siguiendo los principios de la técnica y hacer uso de la memorización. Hacerla de forma iterativa nos será de gran ayuda ya que es más fácil de entender qué es lo que está pasando y cuándo, y porque es mucho más fácil de calcular la complejidad.

En nuestro problema que se nos planteó, necesitamos saber que días conviene entrenar y cuáles no dependiendo de la ganancia obtenida del entrenamiento y la energía de los jugadores. Tener en cuenta lo siguiente:

- Los entrenamientos son inamovibles. No podemos cambiar el entrenamiento e_i con el entrenamiento e_j $(i \neq j)$.
- Para el dia del entrenamiento e_i, la energía disponible va a ser menor o igual a la del dia e_{i-1}, por lo tanto se cumple
$$s_1 \geq s_2 \geq ... \geq s_n$$
siendo estas las energías disponibles para cada e_i después de haber entrenado consecutivamente. La energía s_1 corresponde al día e_1.
- Si se descansa un dia, el primer dia de volver a entrenar vuelve a tener la energía del primer dia.
- Si el valor del entrenamiento del día e_i tiene valor j, y la energía disponible para ese dia es s_i, la ganancia será del mínimo de estos dos. Es decir,
$$Ganancia(i) = min(j, s_i)$$

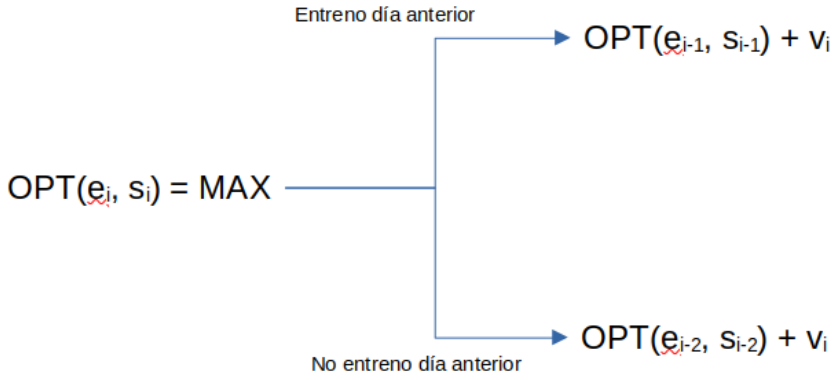
Entonces teniendo en mente lo anterior mencionado, nuestros subproblemas serían proporcionales a la cantidad de días que hayan programado. A su vez, la forma de estos subproblemas es calcular

el óptimo del día `i` dependiendo de si conviene haber entrenado o no el día anterior y tener en cuenta la ganancia de este día. La forma en la que los subproblemas se combinan para dar lugar a la solución de los problemas más grandes es a través de, una vez calculado los *óptimos* de los días hasta *i-1* y teniendo en cuenta descansos y sus ganancias, para el dia i la solucion seria hacer uso de la memorización de lo ya calculado y elegir la mejor opcion segun la ***Ecuación de Recurrencia*** de nuestro problema.

Para definir nuestra ecuación de recurrencia, planteamos lo siguiente:

- Para el primer día, la ganancia va a ser el mínimo entre la energía de ese día y el valor del entrenamiento.
- Para el segundo dia va a depender de si conviene haber entrenado el primer día y obtener la ganancia del entrenamiento como si fuera el primer dia de haber entrenado o continuar entrenando después del primer dia pero **no** con la energía que tendrían disponible los jugadores si hubieran descansado.
- Para el entrenamiento e_i, el optimo seria el que maximiza la ganancia entre haber entrenado el dia e_{i-1} y tener menos energía disponible para obtener la ganancia de ese dia, o no haber entrenado el día anterior y tener la posibilidad de obtener mas ganancia teniendo más energía disponible.

Entonces teniendo todo esto en mente, nuestra ecuación de recurrencia tendría la siguiente forma:



Consideraciones a tener en cuenta:

- v_i hace referencia al valor de la ganancia de ese día, que será el mínimo entre s_i y e_i, los valores de energía y entrenamiento de ese día respectivamente.
- La opción de no entrenar el dia anterior solo es válida cuando estamos en el óptimo de s_1, ya que este corresponde a tener toda la energía disponible por haber descansado el día anterior.
- La opción de entrenar el dia anterior es válida cuando me encuentro en un s_i donde i es mayor a 0, ya que estará entrenando en el i día seguido.
- Tendremos dos casos bordes:
  1. En el dia 1 no habrá días anteriores, por lo que la única posible ganancia de ese día estará dada por el valor de v_1.

2. En el dia 2, ya que no tendra dos días atrás para ver la ganancia de dicho dia. Entonces sus únicas dos posibilidades del óptimo es no haber entrenado el día 1 y tener la ganancia del mínimo entre e_2 y s_1, o haber entrenado el día anterior y tener el optimo como indica la ecuación.

2. Algoritmo para encontrar la mayor ganancia en los entrenamientos

A continuación se detallan el código y los pasos que se siguieron para llevar a cabo el algoritmo planteado utilizando *Programación Dinámica*.

2.1. Obtener el cronograma

Una vez que se obtuvieron los datos para los entrenamientos y la energía disponible dependiendo la cantidad de días seguidos de haber entrenado, se arma una matriz de tamaño $n \times n$ siendo n la cantidad de días en el cronograma:

```
1
2  def getWorkOut(earn, energy):
3      matrix = []
4
5      for idx, e in enumerate(earn):
6          arr = [0] * len(energy)
7          for i in range(idx+1):
8              gain = min(e, energy[i])
9              if i == 0:
10                 prevTwoDays = matrix[idx-2][idx-2] + gain if idx > 1 else 0
11                 arr[i] = max(prevTwoDays, gain)
12                 continue
13             contWorkOut = matrix[idx-1][i-1] + gain if idx > 0 else 0
14             lastWorkOut = arr[i-1]
15             arr[i] = max(contWorkOut, lastWorkOut)
16         matrix.append(arr)
17
18     print("Highest possible profit: ", matrix[len(earn)-1][len(energy)-1])
19     return matrix
```

El algoritmo se basa en crear una matriz de $(n \times n)$ siendo n la cantidad de días. ¿Porque una matriz y no un simple arreglo? En este problema, tenemos que considerar no solo una variable a la hora de buscar los óptimos sino dos. Una será la ganancia del entrenamiento, y la otra la cantidad de energía que tendremos ese día dependiendo si es el primer dia que entrenamos desde el último descanso, o el segundo, etc. Entonces la forma que planteamos es usar la energía para cada dia como filas, y la ganancia de cada entrenamiento como la columna.

A la hora de calcular los óptimos para cada dia, se seguirá el siguiente algoritmo:

- Si el dia e_i es el primero que se entrena desde un descanso (s_1), hay que obtener el óptimo del dia e_{i-2} ya que el anterior se supone que se descanso.[1]
- Si el dia e_i es el segundo que se entrena desde un descanso (fila dos), habrá que sumar a la ganancia del dia e_{i-1} la del dia actual teniendo en cuenta la energía que se redujo respecto el primer dia, pues recordemos
$$s_1 \geq s_2 \geq ... \geq s_n$$
- Para el dia e_i la cantidad máxima de posibles entrenamientos es i, ya que siempre el óptimo es respecto a los días descansados como la ganancia de ese entrenamiento, y estos son proporcionales a la cantidad de días, lo que nos resulta en una matriz cuadrada.

A su vez, esta forma de resolver el problema nos resultara en una matriz triangular, ya que la cantidad máxima de filas es la cantidad de días y lo mismo con los entrenamientos, pero el entrenamiento del dia e_i completará hasta maximo i filas, dejando el resto de las filas hasta n en 0.

Respecto a la complejidad del algoritmo, todas las operaciones que se hacen son constantes, pues acceder a una posición específica de un arreglo, hacer comparaciones, y asignar valores son

[1] Esto solo es valido para los entrenamientos pasados el segundo dia.

$\mathcal{O}(1)$. Pero al mismo tiempo, cada una de estas operaciones se realizan i veces para el entrenamiento e_i. Y para cada entrenamiento, se realiza i veces para la energía de cada posible situación de los días seguidos entrenando, pues por cada entrenamiento e_i tengo que considerar la ganancia para la energia $s_1, s_2, ..., s_i$, formando así la matriz triangular. Considerando todo esto, la complejidad final del algoritmo es $\mathcal{O}(n \times n) = \mathcal{O}(n^2)$.

2.2. Reconstruir cronograma

Para reconstruir el cronograma se implementó el siguiente algoritmo:

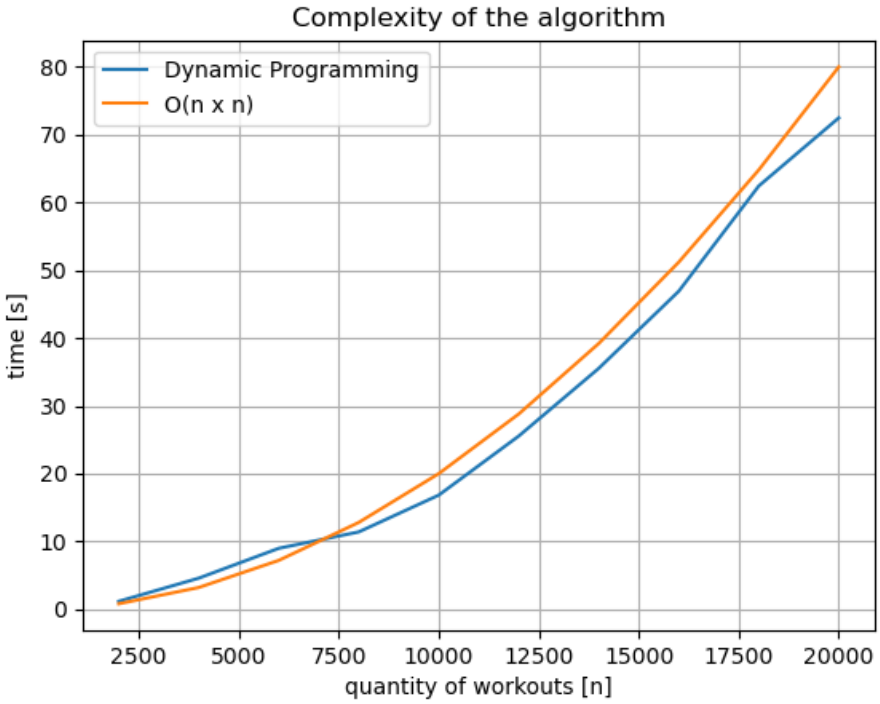
```
1
2  def getSchedule(matrix):
3      res = [""] * len(matrix)
4      i = len(matrix) - 1
5      rest = False
6      for idx in range(len(matrix)-1, -1, -1):
7          if rest:
8              res[idx] = "Rest"
9              rest = False
10             i = idx - 1
11             continue
12         while i != 0 and matrix[idx][i] == matrix[idx][i-1]:
13             i -= 1
14         if i == 0: rest = True
15         res[idx] = "Train"
16         i -= 1
17     return res
```

Para reconstruir la solución se siguió el siguiente razonamiento:

1. Se iterará de adelante para atrás, de manera de saber si se entreno el dia e_{i-1} o no dependiendo del óptimo del dia e_i
2. Se buscará en que dia consecutivo se ejercitó el último entrenamiento (e_i). Recordemos que para esto, habrá que iterar la última columna de la matriz de arriba hacia abajo, hasta que el elemento en la posición s_i difiera del elemento s_{i-1} (dentro de la misma columna).
3. Empezaremos a bajar de forma .^escalonada", es decir, bajando una posición y yendo para la izquierda en otra.
4. Si llegamos a la fila 0, significa que ese día es el primero luego de haber descansado, así que el anterior día no se entrenó y el último entrenamiento fue el anterior a este último. Volvemos al paso 2.
5. En caso de haber llegado a la primer columna, terminará la reconstrucción.

3. Mediciones

Se realizaron mediciones en base a crear pruebas de distintos tamaños y tomar su tiempo de ejecución individualmente, y en base a los datos recolectados hacer el gráfico. Los datos fueron desde 2000 hasta 20.000 elementos, los cuales las ganancias de cada entrenamiento fue generada aleatoriamente entre valores de 1 y 100, y para las energías fueron decayendo en porcentajes. Es decir, el 1 % del total fue 100, otro 99, otro 98... hasta llegar a 1. De esta manera, nos aseguramos de que cada día seguido entrenando cumple con la condición de que la energía de ese día es **menor o igual** al anterior.



Como se puede apreciar, el algoritmo planteado con *Programación Dinámica* efectivamente cumple con la complejidad indicada anteriormente, ya que su diferencia con la complejidad de $\mathcal{O}(n \times n)$ es casi nula[2].

4. Conclusiones

El algoritmo planteado y la forma en la que se diseñó la solución pudo ser llevada a cabo de forma efectiva y encontrar la forma óptima de resolver el problema. Ahora bien, el hecho de que en vez de haber usado un simple arreglo como estructura de datos para resolver el problema nunca hubiera sido acorde a la solución. Como se explicó, no solo hay que tener en cuenta la ganancia del entrenamiento e_i, sino también la energía que corresponda a ese entrenamiento, y encontrar de esta manera la óptima planificación para obtener la mayor ganancia posible.

Por otro lado, se pudo también cumplir con las ideas planteadas sobre la *Programación Dinámica*, el uso de la memorización para recordar subproblemas ya resueltos y aprovechar estos para resolver problemas mayores hasta llegar a la solución óptima. De igual manera, se pudo crear el algoritmo para reconstruir el cronograma a partir de la matriz construida y poder si se lo desea obtener los días a entrenar y los días a descansar.

Finalmente, una vez más, valoramos la oportunidad que se nos brindó de colaborar en esta tarea crucial para que los jugadores tengan el mejor desempeño y estamos determinados a contribuir el éxito continuo de la selección **CAMPEONA DEL MUNDO**.

[2] Aclaración: para la complejidad indicada por la traza perteneciente a $\mathcal{O}(n \times n)$, se ha dado cierto valor al parámetro a en la ecuación $y = ax^2$ ya que sino para 20.000 elementos el valor y sería igual a 400.000.000, haciendo parecer que nuestro algoritmo es constante.