Teoría de Algoritmos I

Curso de verano 2024

Trabajo Práctico 2: Programación Dinámica

Alumno	Padrón	E-mail
Juan Ignacio Pérez Di Chiazza	109.887	jperezd@fi.uba.ar
Joaquín Parodi	100.752	jparodi@fi.uba.ar
Máximo Utrera	109.651	mutrera@fi.uba.ar

Índice

Introducción

Luego de haber analizado a todos los rivales gracias a tu ayuda, Scaloni definió un cronograma de entrenamiento. Tiene definido qué hacer para cada día de acá al mundial que viene, e incluso más. Para hacerlo más simple, para los próximos n días. El entrenamiento del día i demanda una cantidad de esfuerzo e_i, y podemos decir que la ganancia que nos da dicho entrenamiento es igual al esfuerzo. El entrenamiento que corresponde al día i (así como su esfuerzo y ganancia) son inamovibles: el Chiqui Tapia alquiló las herramientas específicas para cada día, y la AFA está muy ocupada organizando el torneo de 2^{30} equipos del año que viene para andar moviendo cosas. Si la cantidad de energía que se tiene para un día i es $j < e_i$, entonces la ganancia que se obtiene en ese caso es justamente j. (si se tiene más energía que e_i, no es que se pueda usar más para tener más ganancia).

A su vez, los jugadores no son máquinas. La cantidad de energía que tienen disponible para cada día va disminuyendo a medida que pasan los entrenamientos. Suponiendo que los entrenamientos empiezan con los jugadores descansados, el primer día luego de dicho descanso los jugadores tienen energía s_1. El segundo día luego del descanso tienen energía s_2, etc... Para cada día hay una cantidad de energía, y podemos decir que $s_1 >= s_2 >= ... >= s_n$. Scaloni puede decidir dejarlos descansar un día, haciendo que la energía "se renueve" (es decir, el próximo entrenamiento lo harían con energía s_1 nuevamente, siguiendo con s_2, etc...). Obviamente, si descansan, el entrenamiento de ese día no se hace (y no se consigue ninguna ganancia).

Scaloni no sabe bien cómo hacer para definir los días que deba entrenarse y los días que convenga descansar de tal forma de tener la mayor ganancia posible (y tener mayores probabilidades de ganar el mundial que viene), pero Menotti, exponente del juego bonito en Argentina, le recomendó usar Programación Dinámica para resolver este problema. Nos está pidiendo ayuda para poder resolver este problema.

Consigna

1. Hacer un análisis del problema, plantear la ecuación de recurrencia correspondiente y proponer un algoritmo por programación dinámica que obtenga la solución óptima al problema planteado: 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, determinar la máxima cantidad de ganancia que se obtener de los entrenamientos, considerando posibles descansos.
2. Escribir el algoritmo planteado. Describir y justificar la complejidad de dicho algoritmo. Analizar si (y cómo) afecta la variabilidad de los valores de los esfuerzos (disponibles y necesarios) a los tiempos y optimalidad del algoritmo planteado.
3. Realizar ejemplos de ejecución para encontrar soluciones y corroborar lo encontrado. Adicionalmente, el curso proveerá con algunos casos particulares que deben cumplirse su optimalidad también.
4. De las pruebas anteriores, hacer también mediciones de tiempos para corroborar la complejidad teórica indicada. Realizar gráficos correspondientes.
5. Agregar cualquier conclusión que parezca relevante.

1. Análisis del problema

El problema involucra la toma de decisiones secuenciales por parte de Scaloni para maximizar la ganancia acumulada a través de entrenamientos, considerando la variabilidad en la cantidad de energía disponible para cada día y la posibilidad de descansar para renovar la energía de los jugadores.

- El entrenamiento del día i demanda una cantidad de esfuerzo e_i.
- La cantidad de energia en un momento dado es s_j con j entre 0 e i (j no puede superar i porque es imposible que por ejemplo el primer día se tenga una energía diferente a la primera).
- Si la cantidad de energía que se tiene en un día i es $s_j < e_i$, entonces la ganancia que se obtiene en ese caso es s_j, es decir que esta se limita a la energía disponible. Pero si $s_j > e_i$ la ganacia será e_i ya que por mas que haya energía demas no se puede sobrepasar el esfuerzo designado al día i.
- Para cada día hay una cantidad de energía, y podemos decir que $s_0 >= s_1 >= ... >= s_n$, es decir, la energía disponible para cada día es igual o disminuye a medida que se realizan entrenamientos consecutivos.
- La posibilidad de descansar un día permite renovar la energía, pero esto implica no entrenar ese día y por consiguiente no se consigue ganancia.
- El primer día luego de un descanso los jugadores vuelven a tener energía s_0.

Se observa una situación donde los esfuerzos para los días $1-4$ son $1, 1, 1, 3$ respectivamente y las energías $3, 1, 1, 1$:

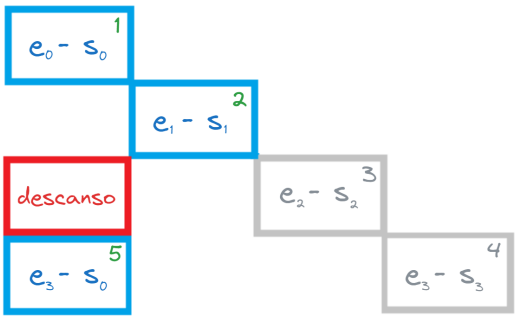

Figura 1: En azul se ven los días entrenados, en rojo el día de descanso, y en gris los escenarios ignorados. En la esquina superior derecha de cada caja esta la ganancia total luego de ese entrenamiento.

En esta situación se puede notar que de hacer todos los entrenamientos consecutivamente se llegaría a una ganancia de 4, pero si en lugar de eso se descansa el día 3 la ganancia aumenta a 5.

Hay dos detalles importantes a analizar:

El primero es que la ganancia total luego de un entrenamiento e_i con energía s_j es igual a la ganancia del mismo sumado a la ganancia hasta el entrenamiento e_{i-1} con energia s_{j-1}, siempre y cuando no se haya descansado el día anterior. Esto se puede ver en el segundo día del ejemplo, ya que la ganancia total es $e_1 + e_0 = 1 + 1 = 2$

El segundo es que para obtener la ganancia hasta el cuarto día después de descansar el día anterior, **se depende del valor calculado en el segundo día** (dos días atrás) al cual se le debe sumar el número mas chico entre e_3 y s_0 (ganancia del día 4).

- Ganancia hasta el día $2 = 2$
- $e_3 = 3$
- $s_0 = 3$

Entonces la ganancia hasta el día 4 es $2 + 3 = 5$.

1.1. Ecuación de recurrencia

A partir del análisis previo se puede crear una solución al problema teniendo en cuenta los siguientes 3 puntos:

- Dado un día i se calcula su ganancia parcial como $min(s_j, e_i)$ siendo s_j la energía disponible.
- Si se descansa el día $i - 1$ entonces la ganancia hasta el día i será la mayor obtenida hasta el día anterior al descanso $(i - 2)$ sumado a la ganancia parcial actual.
- Si no se descansa el día anterior entonces la ganancia hasta el día i se calcula como la del día $i - 1$ con una energía de s_{j-1} siendo que la energía actual es s_j, sumado a la ganancia parcial actual.

Entonces para un dado día la ganancia maxima se podrá calcular a partir de la siguiente relación de recurrencia:

$$OPT(i) = \max\left(OPT(i-2) + \min(s_0, e_i), (Ganancias[i-1][j-1] + \min(s_j, e_i), \forall j \in \{1, 2, \ldots, i\})\right)$$

2. Planteo del algoritmo

2.0. Algoritmo no óptimo considerado

En un principio una ecuación de recurrencia considerada fue la siguiente:

$OPT(i,j) = max(OPT(i-1,j-1) + min(s_j, e_i), OPT(i-2,j-2) + min(s_1, e_i))$

Esta tiene en cuenta que conviene mas para un día i entre seguir entrenando sin descanso, o haber descansado el día anterior y volver a entrenar el día actual.

Esta solución no siempre es óptima ya que en un determinado día solamente elige descansar el día anterior si la suma de la ganancia hasta hace 2 días y la ganancia parcial del nuevo día es mayor a la de seguir entrenando, es decir que no toma en cuenta que el haber descansado un día anterior mas lejano podria desembocar en mayor ganancia en un futuro día.

Para demostrar la no optimalidad de este algoritmo se usó el siguiente **contraejemplo**:

- Dados 4 días ($i = 0, 1, 2, 3$)
- Dados los esfuerzos $8, 2, 8, 7$
- Con energías $5, 4, 4, 2$

Figura 2: El camino azul representa el camino óptimo y el gris representa el camino al que se llega siguiendo dicha ecuación de recurrencia.

Como se puede apreciar en el gráfico si se sigue la lógica del algoritmo considerado, en este caso se llega a una ganancia máxima de 13 cuando en realidad la óptima es 14. Esto tiene sentido para el algoritmo ya que se ve como siempre va en aumento la ganancia, pero como se adelantó previamente, no tiene en cuenta descansos posibles (que a la larga podrian ser beneficiosos) si la ganancia sigue en aumento y ese descanso representa una pérdida de ganancia en la inmediatez.

2.1. Algoritmo

Se propone un algoritmo que utiliza programación dinámica para calcular la solución óptima. Este compara las ganancias acumuladas considerando tanto la opción de descansar en el día anterior como la opción de seguir entrenando sin descansar.

En este caso se utilizó el enfoque 'Bottom-Up' (iterativo), el cual consiste en resolver los problemas del caso base en adelante, contrario al enfoque 'Top-Down' (recursivo), en el cuál se parte del caso general y se avanza hacia el caso base progresivamente.

Se utilizó también Memoización para almacenar los resultados obtenidos previamente.

- **Inicialización de la Matriz de Memoización:** Se crea una matriz de tamaño $n*n$, donde n es la longitud de las listas de esfuerzos y energías. Esta matriz, llamada memo, se utilizará para almacenar las ganancias máximas acumuladas en cada día de entrenamiento y para cada posible nivel de energía disponible.
- **Caso base:** Se establece el caso base de la matriz memo para el primer y segundo día de entrenamiento. La ganancia máxima acumulada para el primer día es el mínimo entre el esfuerzo del primer día y la energía disponible inicial. Similarmente, se establece la ganancia máxima acumulada para el segundo día (con energia inicial, es decir tomando como descansado el primer dia).
- **Cálculo de Ganancias Máximas:** Se itera sobre los días de entrenamiento, empezando desde el segundo día, ya que los casos base ya han sido inicializados. Para cada día i:

  Si i no es igual a 1 (es decir, no es el segundo día), se calcula la ganancia máxima acumulada considerando la opción de descansar el día anterior y entrenar en el día actual. Esto se hace tomando el máximo valor de ganancia acumulada hasta el día i-2 y sumando el mínimo entre el esfuerzo del día actual y la energía disponible en el primer día.

  Luego, para cada nivel de energía disponible j desde 1 hasta i, se calcula la ganancia máxima acumulada considerando la opción de entrenar en el día actual después de descansar en algún día anterior. Esto se hace tomando la ganancia acumulada en el día anterior y sumando el mínimo entre el esfuerzo del día actual y la energía disponible s_j.
- **Obtención de la Ganancia Máxima Total:** Después de calcular todas las ganancias máximas acumuladas, se encuentra el máximo valor en la última fila de la matriz memo, que representa la ganancia máxima total acumulada a lo largo de todos los días de entrenamiento y se toma el valor mas alto de la misma.

2.2. Por qué es programación dinamica

El algoritmo planteado pertenece a la técnica de diseño programación dinámica, ya que:

- **Se divide el problema en subproblemas mas pequeños:** se dividió el problema en días, calculando para cada día una posible forma de llegar *utilizando subproblemas resueltos con anterioridad.*
- **Se utilizan las soluciones para construir una solucion al problema original:** luego de la obtención del máximo final, se reconstruye la solución del problema original siguiendo un patrón de escalera, asignando como 'Entreno' a aquellos que pertenecen a la 'escalera' actual, luego de llegar a la base se asigna como 'descanso' en caso que corresponda, y se reinicia el procedimiento desde el óptimo del día anterior al descanso.

74	0	0	0	0	0	0	0	0	0
74	143	0	0	0	0	0	0	0	0
128	128	197	0	0	0	0	0	0	0
179	164	164	233	0	0	0	0	0	0
271	248	229	229	297	0	0	0	0	0
269	307	284	265	265	333	0	0	0	0
341	313	351	328	309	302	356	0	0	0
407	410	378	416	392	346	325	378	0	0
404	455	458	425	464	429	369	347	400	0
481	469	520	523	490	501	452	391	369	407

Figura 3: Patrón de 'escalera' para 10 elementos. Las filas representan cada día i, mientras que las columas representan la 'racha' de días seguidos (o los indices j de las energías).

2.3. Por qué es óptimo

El algoritmo propuesto es óptimo porque utiliza programación dinámica que garantiza la obtención de la solución óptima para el problema. Éste tiene una propiedad de subestructura óptima, lo que

significa que una solución óptima global puede ser construida a partir de soluciones óptimas a subproblemas más pequeños. Esto se refleja en la ecuación de recurrencia, que descompone el problema en subproblemas más simples y encuentra la solución óptima para cada uno de ellos, utilizando memoización para evitar recalcular soluciones para subproblemas ya resueltos. Esto mejora significativamente la eficiencia del algoritmo al evitar la repetición de cálculos, pero sin comprometer la obtención de la solución óptima.

Dado que el algoritmo considera todas las posibles combinaciones de entrenamiento y descanso para cada día y selecciona la que maximiza la ganancia acumulada, se garantiza que la solución obtenida es óptima en términos de maximizar la ganancia total acumulada a lo largo de todos los días.

2.4. Complejidad

La complejidad temporal del algoritmo es $O(n^2)$ ya que se itera desde 1 hasta n, y por cada una de estas iteraciones se itera desde 1 hasta el valor del contador del bucle exterior, cuyo máximo valor es n.

La complejidad espacial del algoritmo también es $O(n^2)$, dado que se almacena la matriz de memoización cuya dimensión es n x n. Se podría hacer una optimización de espacio creando sólamente la mitad de dicha matriz, ya que es lo único que se utiliza, como se menciona en el análisis previo esto es debido a que no es posible hacer el entrenamiento del primer día con una energía $s_i < s_1$ o para el segundo día una energía $s_i < s_2$ y lo mismo para cada día. Sin embargo la complejidad no cambiaría.

2.5. Código en Python

```python
# ALGORITMO:
def obtener_ganancia_maxima(esfuerzos, energias) -> int:
    memo = [[0] * len(esfuerzos) for _ in range(len(esfuerzos))]  # matriz nxn

    memo[0][0] = min(esfuerzos[0], energias[0])  # caso base
    memo[1][0] = min(esfuerzos[1], energias[0])

    for i in range(1, len(esfuerzos)):

        if i != 1:
            memo[i][0] = max(memo[i-2]) + min(esfuerzos[i], energias[0])

        for j in range(1, i+1):
            memo[i][j] = memo[i-1][j-1] + min(esfuerzos[i], energias[j])

    max_memo = max(memo[-1])
    return max_memo, reconstruir_camino(max_memo, memo)


# RECONSTRUCCION DE CAMINO DE ENTRENAMIENTOS Y DESCANSOS:
def reconstruir_camino(max_memo, memo) -> list[str]:
    """
    Utiliza la escalera producida en la busqueda del maximo, la cual al llegar a la
    'base': idx = 0 se agrega un dia de descanso, y en otro caso uno de entrenamiento.
    """

    camino = []
    idx = memo[-1].index(max_memo)
    i = len(memo)-1
    while i >= 0:
        while idx >= 0:
            camino.insert(0, 'Entreno')
            idx -= 1
            i-=1
            if i < 0:
                return camino
        camino.insert(0, 'Descanso')
        i -= 1
        idx = memo[i].index(max(memo[i]))
    return camino
```

3. Ejemplos de ejecución

3.1. 3 elementos

```
esfuerzos = [1,5,4]
energias = [610,2,2]

El resutado optimo es 7 y el orden de entrenamientos y descansos es:
['Descanso', 'Entreno', 'Entreno']

El resultado obtenido es el mismo que el optimo junto con el mismo orden de
entrenamientos y descansos.
```

3.2. 10 elementos

```
esfuerzos = [36,2,78,19,59,76,65,64,33,41]
energias = [63,61,49,41,40,38,23,17,13,10]

El resutado optimo es 380 y el orden de entrenamientos y descansos es:
['Entreno', 'Descanso', 'Entreno', 'Descanso', 'Entreno', 'Entreno', 'Entreno', 'Entreno'
    , 'Entreno', 'Entreno']

El resultado obtenido es el mismo que el optimo junto con el mismo orden de
entrenamientos y descansos.
```

3.3. 10 elementos bis

```
esfuerzos = [75,77,54,36,78,36,44,77,48,65]
energias = [74,69,65,65,64,37,23,22,22,7]

El resutado optimo es 523 y el orden de entrenamientos y descansos es:
['Entreno', 'Entreno', 'Entreno', 'Entreno', 'Entreno', 'Descanso', 'Entreno', 'Entreno',
     'Entreno', 'Entreno']

El resultado obtenido es el mismo que el optimo junto con el mismo orden de
entrenamientos y descansos.
```

3.4. Contraejemplo del primer algoritmo considerado

En este ejemplo se ve el resultado de la ejecución del algoritmo propuesto con la entrada que se uso de **contraejemplo** para otro algoritmo considerado en la seccion 2.0. **Algoritmo no optimo considerado**.

```
esfuerzos = [8,2,8,7]
energias = [5,4,4,2]

La ejecucion de este algoritmo con esta entrada da como resultado una ganancia de 14, y
un orden de entrenamientos y descansos: ['Entreno', 'Descanso', 'Entreno', 'Entreno']

Y como se puede ver en la figura 2 de la seccion mencionada, estos resultados son los
optimos esperados.
```

4. Mediciones de tiempo

4.1. Cómo afecta la cantidad de entrenamientos n al tiempo de ejecución

En el siguiente benchmark se analizó el tiempo del algoritmo en función del tamaño n de la entrada. La complejidad teórica del algoritmo está explícita en la sección 2.4. **Complejidad**.

Para el benchmark se tomaron diferentes largos desde 0 hasta 10.000 con un salto de 500, es decir se mide el tiempo con el largo 0, luego el largo 500, el 1000 y así sucesivamente.

Los valores de esfuerzos y energías fueron randomizados entre 1 y 50.000 (las energías están en orden decreciente).

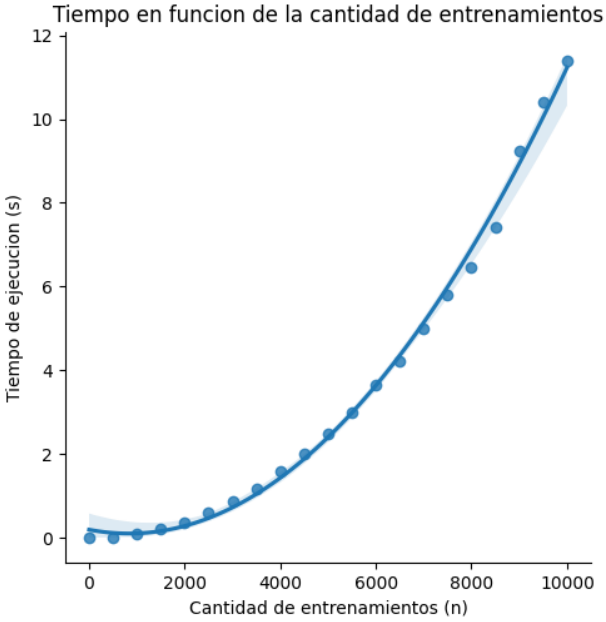

Figura 4: Se puede observar que a medida que aumenta n, el gráfico se parece a una parábola y esto confirma la complejidad temporal teórica del algoritmo $O(n^2)$.

4.2. Cómo afecta la variabilidad de e_i al tiempo de ejecución

A continuación se ve un segundo benchmark, esta vez analizando los cambios en el tiempo del algoritmo al variar los valores de esfuerzo y energía.

Para esto, los mismos son randomizados con un límite máximo que va de 0 a 10.000 con un salto de 100, por lo que se toma el tiempo que tarda el algoritmo con los valores siendo randomizados entre 0 y 100, luego 0 y 200 y así sucesivamente, luego a partir de estas mediciones se hace el gráfico de la figura 5.

La cantidad de entrenamientos se fija en 1000 para todas las mediciones.

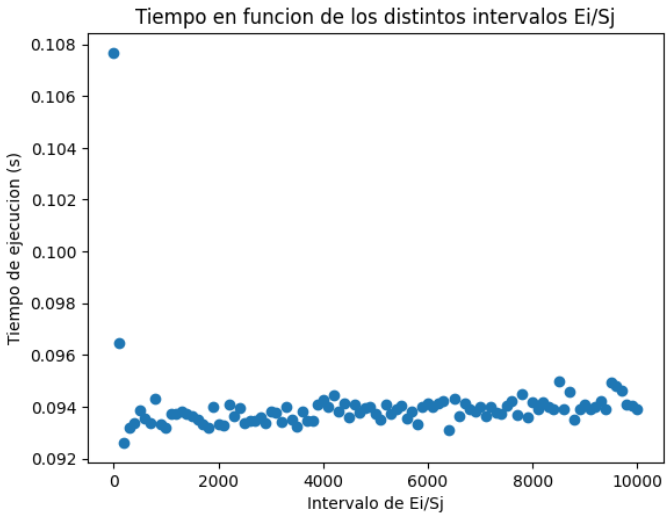

Figura 5: En este gráfico se puede observar que el tiempo que tarda el algoritmo no se ve afectado por la variabilidad de los números e_i y s_j.

5. Conclusiones

Al analizar la situación con lo solicitado, se halló cierta similaridad con el problema de "Juan el vago", sin embargo en este problema se agregaba una dimensión (la lista de energía con sus implicancias) y por lo tanto no era factible usar la solución. Cierta lógica de la solución del problema mencionado está embebida en la solución planteada en el informe, al poder dejar días de descanso para obtener mayores ganancias.

El algoritmo propuesto utiliza una matriz de memoización para almacenar las ganancias máximas acumuladas en cada día de entrenamiento y para cada posible nivel de energía disponible. Utilizando la técnica de programación dinámica, este calcula eficientemente la ganancia máxima total acumulada considerando todas las posibles combinaciones de entrenamiento y descanso para cada día.

Además, se realizaron ejemplos de ejecución para ilustrar el funcionamiento del algoritmo, y se proporcionaron mediciones de tiempo para corroborar su complejidad teórica. Los resultados obtenidos fueron consistentes con las expectativas, lo que respalda la óptimalidad y eficiencia del algoritmo propuesto.

En conclusión, la técnica de diseño de algoritmos *programación dinámica* es una herramienta poderosa y efectiva para abordar problemas de optimización secuencial como el planteado. Su capacidad para garantizar la obtención de la solución óptima lo hace adecuado para una variedad de aplicaciones prácticas en el campo del deporte y más allá.