Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico 2
Programación Dinámica

5 de marzo de 2024

Martin Tomas
100835

Adriana Macarena Iglesias Tripodi
103384

Ezequiel Lassalle
105801

Cecilia Caffaratti
72629

1. Introducción

En este trabajo practico tuvimos que resolver un problema propuesto por la cátedra mediante la utilización de una solución basada en la programación dinámica.

El problema nos propone ayudar a Scaloni a organizar el cronograma de entrenamientos y de descanso de todos los días hasta que llegue el mundial que viene. Cada día de entrenamiento da como recompensa una cantidad especifica de ganancia y el planteo nos pide lograr llegar a la mayor ganancia total acumulada siempre. El problema del trabajo es que los jugadores se van cansando a medida que pasan los días de entrenamiento y que a medida que van pasando los días cada vez tienen una capacidad menor para obtener la ganancia disponible de cada día que entrenen: si quieren recargar mas energía van a tener que descansar para así poder recargar su capacidad de obtener la energía. El problema nos pide encontrar la mejor secuencia en la que los jugadores deban descansar o entrenar para obtener la mayor cantidad de ganancia posible.

La programación dinámica es una técnica de creación de algoritmos que nos permite resolver problemas subdividiéndolos en problemas mas pequeños y reutilizándolos constantemente para ir así llegando al resultado óptimo a medida que se avanza en la ejecución del algoritmo y se reutilizan(y por ende también descartan) mas opciones. Los algoritmos de programación dinámica generalmente se basan en una ecuación de recurrencia que se usa como regla para determinar cuando tomar o cuando descartar una de las soluciones posibles y en la memorización, que consiste en ir almacenando paso a paso las soluciones en una estructura de datos con tal de reutilizarlas y volver a analizarlas después.

Una buena forma de resolver el problema propuesto es mediante la aplicación de la programación dinámica ya que, paso a paso y día a día a medida que vamos aplicando la ecuación de recurrencia para cada día y cada energía disponible del jugador nos vamos guardando las mejores opciones para días anteriores y vamos así reutilizando siempre soluciones a subproblemas anteriores ya analizados.

Resolución

2. Análisis del problema

Para resolver este problema nos inspiramos en varios de los ejercicios propuestos en las clases de programación dinámica. A medida que discutíamos que es lo que teníamos que hacer para llegar con la solución fuimos comparando el problema con varios de los problemas planteados en las clases. Sin embargo esta inspiración fue parcial debido a que este ejercicio presenta particularidades propias que lo hacen bastante diferente a los que teníamos en mente como soluciones posibles.

El problema de la mochila fue el que mas nos inspiro para pensar como debíamos enfrentarnos al trabajo practico. Esto es así ya que en el problema de la mochila tenemos una capacidad máxima y una cantidad de elementos, y generamos una matriz que paso a paso y de manera recurrente va analizando si es mejor tomar el elemento actual y sumárselo al óptimo para el elemento anterior considerando que ahora tenemos que restarle al peso total el peso del actual o si es mejor no tomar el elemento actual analizando el óptimo para el elemento anterior con el peso actual. La ecuación de recurrencia del problema de la mochila nos inspiro mucho para resolver este trabajo practico:

$$\text{OPT}(N, W) = \text{máx}(\text{OPT}(N-1, W), \text{OPT}(N-1, W-Pi) + V_i)$$

La idea de ir restando el valor que te limita a medida que vas analizando si resulta conveniente o no tomar cada elemento especifico resonó con nosotros ya que en este problema tenemos que ir analizando de manera recurrente si en el día que estamos analizando es mejor descansar o entrenar para poder obtener la mayor ganancia total.

Llegamos a la conclusión de que la mejor forma de resolver el problema es analizar si en cada día es mejor entrenar ese día habiendo descansado el día de ayer (para poder obtener la mayor ganancia posible de el día que estamos analizando) sumándole esto al valor máximo obtenido de haber entrenado hace dos días o si es mejor entrenar el día actual pero sin haber descansado el día de ayer para así poder sumar el valor de ganancia que puede obtener el jugador actualmente al máximo del día de ayer.

La mejor forma de resolver el problema teniendo en cuenta nuestra resolución y la ecuación que propusimos es armar una matriz donde cada posición de la matriz representa el máximo valor obtenido para ese día teniendo en cuenta cuando fue el ultimo día que descanso. Las filas representan el día en el que estamos iterando y las columnas hace cuanto fue la ultima vez que los jugadores descansaron. Por eso nuestra ecuación de recurrencia resulta así:

$$\text{OPT}(I, J) = \text{máx}\left(\text{OPT}(I-1, J-1) + \text{mín}(e(i-1), s(j-1)), \max_{0<j\leq I-2}(\text{OPT}(I-2, j)) + \text{mín}(e(i-1), s(0))\right)$$

Donde I son los días transcurridos desde el comienzo del plan de entrenamiento y J los días desde que se no se trabajó un día. Entonces si se descanso el día 2 el día 3 tendría una I=3 y J=1.

Por otro lado, la lógica de la ecuación es simple: si yo quiero analizar si un día tengo que descansar o no simplemente tengo dos opciones:

A) El valor de ganancia mi día actual sin haber descansado teniendo en cuenta mi energía actual mas el valor máximo que tenia para mi día de ayer con el nivel de energía disponible para el día de ayer.

B) El valor máximo acumulado que tenia disponible hace 2 días para el nivel de energía disponible que tenia en ese momento mas el valor actual con la energía máxima posible de los jugadores (la energía que tienen disponible luego de descansar).

Si B resulta mayor que A resulta conveniente descansar el día anterior: de lo contrario resulta conveniente no descansar.

Se debe tener en cuenta que partimos del caso base que se tiene solo un día y la mejor opción para este va a ser siempre el mínimo entre la energía gastada y la que se ganaría dicho día. Además, consideramos el caso los casos en que no se tenga ningún día para entrenar y por lo tanto la ganancia sería toda cero. Finalmente, se tuvo en cuenta que no se podía haber descansado hace

más días de los que se había trabajado, entonces si se está analizando el día tres todo lo que tenga la consideración de haber descansado hace más de tres días también se lo considera nulo.

2.1. Algoritmo y Complejidad

A continuación se muestra el algoritmo principal que utilizamos para solucionar nuestro problema.

```
def scaloni( d, e, s):
    G = [[0 for x in range(d + 1)] for x in range(d + 1)]
    G[1][1] = min(e[0],s[0])
    for i in range(1,d+1):
        p=1
        for j in range(1,d + 1):
            if (j>i):
                G[i][j] = 0
            else:
                G[i][j]=max(min(e[i-1],s[j-1])+ G[i-1][j-1],min(e[i-1],s[0])+ max(G
    [i-2]))

    return G
```

Como se puede ver el código realiza exactamente lo que describimos anteriormente: Llena de ceros las matrices pertenecientes a la diagonal $j > i$ ya que, por ejemplo, es imposible que si estamos en el día 9 la ultima vez que los jugadores descansaron haya sido en el día 10. Luego para la primera posición pone el máximo valor posible a ganar el primer día (la ganancia total del primer día o la energía disponible que los jugadores pueden ganar el primer día). Para el resto de los días lo que el algoritmo hace es simplemente realizar iterativamente y paso a paso la ecuación de recurrencia que planteamos anteriormente y ir llenando la matriz. Es posible ver que, para generar dicha matriz, se utilizan dos ciclos *for* sucesivos de n iteraciones, consecuentemente la creación de esta tiene un costo temporal de $n \times n = n^2$. Por lo tanto, la complejidad temporal de esta parte es de $O(n^2)$

Luego, generamos esta función que nos permite reconstruir que días tengo que entrenar y cuales no:

```

def reconstruccion(G, p, e, s):
    res=["Descanso" for x in range(len(e))]
    i=len(e)
    while(i> 1):
        if max(G[i-2])+min(e[i-1],s[p-1]) == G[i][p]:
            p=G[i-2].index(max(G[i-2]))+1
            res[i-1]="Entreno"
            i-=1
        elif(p!=0):
            res[i-1]="Entreno"
        p-=1
        i-=1
    if G[2][1]+min(e[2],s[2]) !=G[3][p+2]:
        res[0]="Entreno"
    return (res)
```

Como se puede ver, lo que la función hace es simple. La función recibe como parámetro p la columna en la que se encontraba nuestra ganancia total en nuestra matriz generada por la función anterior y pregunta constantemente si es el resultado de ser la suma del máximo dos días antes mas la energía disponible que se puede sumar en el día y posición de energía disponible actual. Si esto es cierto significa que la selección entreno en el día actual, modifica una lista res generada inicialmente para verificar que días entreno, modifica el índice p con la posición del máximo anterior para seguir el recorrido y decrezco los días a analizar en uno. Esta solución genera un vector de n posiciones siendo n la cantidad de días.

Para esta parte, como solo hay un ciclo de importancia que se produce a lo sumo n veces podemos decir que la complejidad temporal es $O(n)$. Esto es debido a que se recorren los días

establecidos para el plan como filas de la matiz y se setea su columna según la distancia a un día de descanso.

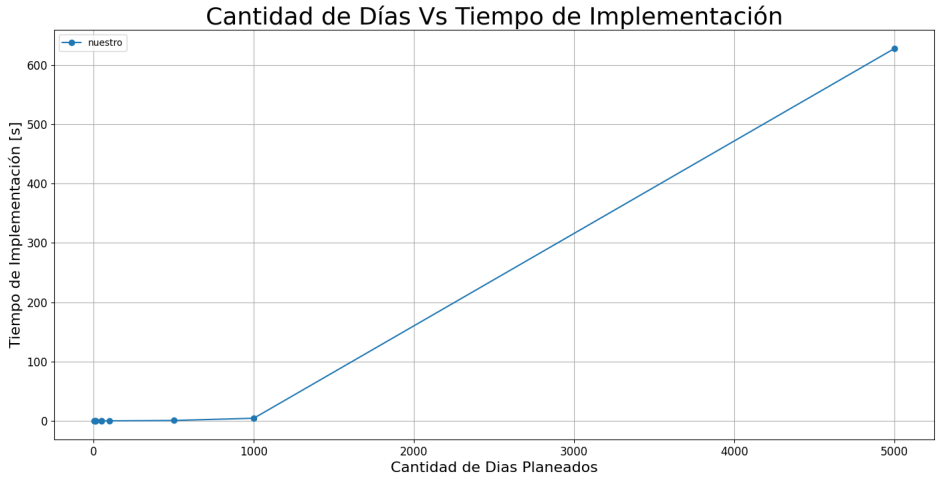
La complejidad del algoritmo planteado es entonces $O(n^2)$, ya que se puede despreciar el costo de la reconstrucción al ser mucho menor que la generación de la matriz iterativa.

La variabilidad de los esfuerzos no parece afectar mucho si nos referimos puntualmente a su valor numérico. Si parecen tener un pequeño efecto si los datos de la ganancia son cercanos entre ellos o si hay gran varianza entre los sucesivos, produciendo la necesidad de tener más descansos. Esto es consecuencia de tener que recorrer todo una fila para encontrar el máximo contra una simple operación aritmética de comparación. Además, otra cosa afecta al algoritmo es la cantidad de días que los jugadores pueden entrenar, a mayor cantidad de días, la cantidad de días con los que tenemos que armar la matriz crece y así la cantidad de días para los que tenemos que analizar si resulta conveniente descansar o entrenar.

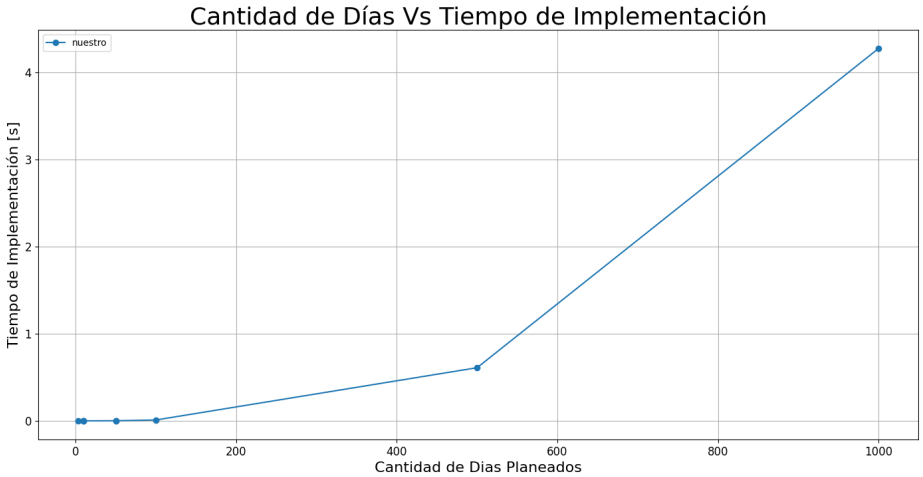
3. Ejemplos y gráficos

Para probar nuestro algoritmo realizamos pruebas sobre todos los archivos de pruebas y corroboramos que los resultados esperados sean iguales a los de nuestro algoritmo.

Los resultados de la complejidad temporal medida en el algoritmo a medida que se incrementaron los días fueron los siguientes:



Este primer gráfico muestra una clara tendencia a tener un costo temporal que aumenta fuertemente a medida que se incrementan la cantidad de días que se consideran en el plan. Esto, por un lado, soporta el análisis de la complejidad dado ya que implica una dependencia tendiendo a lo polinomial con el número de días dados. Pero, no permite observar si el grado del polinomio asociado al costo temporal es el establecido en nuestro análisis o no. Consecuentemente, se creó el gráfico 3 que, al observar un número reducido de n [días del plan de entrenamiento], permite ver más claramente la relación con un polinomio cuadrático, al mostrar principalmente el comienzo de la parte ascendiente de la curva parabólica. Esto, permite asegurar entonces que el comportamiento temporal del algoritmo sigue la línea de un polinomio de segundo orden. Dicho de otra forma que su complejidad temporal de este es $O(n^2)$



4. Conclusiones

Como conclusión podemos decir como grupo que nos resultaron muy interesantes los problemas de programación dinámica ya que su idea de reutilizar iterativamente subproblemas ya previamente calculados y intentar llegar así a la solución parece ser muy útil y eficiente a la hora de resolver problemas como este.

Una vez que entendimos bien el problema, una vez que como grupo pudimos llegar a una buena ecuación de recurrencia y una vez que logramos idear un algoritmo que ejecute paso a paso nuestra ecuación de recurrencia para la serie de datos pudimos entender lo buena que resulta esta tecnica de programacion a la hora de enfrentarse a problemas de esta naturaleza.