.UBAfiuba
FACULTAD DE INGENIERÍA

Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico 2
Informe

Octubre 2023

Brzoza Valeria	Cichero Tomás	Covela Maximiliano
107523	107973	102547

Índice

1. Consigna

Para este trabajo práctico, fuimos nuevamente reclutados por Scaloni, ya que como buen DT de la selección, está armando un cronograma de entrenamientos diarios, que varían en esfuerzo que los jugadores tendrían que poner. Sin embargo, como nuestros jugadores son seres humanos, sabemos que cada vez van a estar más cansados y no van a rendir tanto, por lo que necesitan descansar. Scaloni quiere nuestra ayuda para saber qué días conviene entrenar y qué días conviene descansar para maximizar la ganancia de nuestros jugadores. Para esto nos va a dar los n días que planificó, los esfuerzos e_i requeridos de cada día y la energía que nuestros jugadores van a ir teniendo a medida que sigan entrenando de forma consecutiva.

1.1. Datos

Para realizar esto, contamos con cierta información y restricciones:

1. Contamos con la cantidad de días (n), los esfuerzos requeridos de cada día (e_i) y la energía que nuestros jugadores tendrán cada día consecutivo de entrenamiento (s_i) donde $s_1 >= s_2 >= ... >= s_n$.
2. Por cada día de entrenamiento los jugadores obtendrán una ganancia, la cual será del esfuerzo e_i en caso de que $e_i <= s_i$ o de la energía s_i en caso contrario.
3. No obtendrán mayor ganancia por tener más energía que el esfuerzo requerido para un día de entrenamiento.
4. El entrenamiento del día i, así como su esfuerzo y ganancia, son inamovibles.
5. Si los jugadores descansan en el día i, los jugadores contarán con s_1 al día $i + 1$.

2. Análisis inicial

Con estos primeros datos pudimos deducir algunos aspectos de nuestro conflicto. Por un lado, podemos ver que nunca va a ser útil descansar dos días seguidos. Esto se debe a que con descansar un día, la energía de los jugadores vuelve a maximizarse con s_1. Supongamos que esta maximización se logra con que en el día i tengamos s_1. Para lograr esto, necesitaríamos que el día *i-1* descansen. Sin embargo, no importa cuanta fuera la energía del día *i-2*, si entrenan ese día, ganaríamos un poco más que si lo descansaramos. Siguiendo esta misma lógica, fuimos capaces de deducir que sin importar cuanta energía tengan el último día, siempre será conveniente que ese día se entrene.

3. Algoritmos

3.1. Primer algoritmo pensado

El primer algoritmo que pensamos no funcionó realmente. Este algoritmo resultó que no se podía calificar como programación dinámica sino más bien como greedy. El mismo se basaba en ir desde el primer día hasta el último y, dependiendo si era mejor haber descansado el día anterior o no, establecer si se descansa o no.

El código de este algoritmo se puede encontrar en el repositorio del proyecto en el directorio lib/algoritmo_no_optimo.

Este algoritmo fallaba en no contemplar, para cada día, todas las opciones de haber entrenado i días antes. En el siguiente diagrama se puede observar el seguimiento del algoritmo:

	N° Día que voy entrenando desde el último descanso									
Fecha N°	1	2	3	4	5	6	7	8	9	10
1	36									
2	2	38								
3	99		87							
4	57	118								
5	158		167							
6	181			208						
7	230				248					
8	271					286				
9	281						309			
10	327							326		

Figura 1: Seguimiento del primer algoritmo pensado

Siendo el número 327 la ganancia máxima, y las filas sin ninguna celda con fondo azul, los días que se descansa.

La representación del problema en el formato de una tabla se concibió después de haber escrito este algoritmo y luego de varios intentos hasta encontrar la manera correcta de representar el problema.

3.2. Algoritmo definitivo

Habiendo reconocido las falencias del algoritmo anterior, descubrimos un nuevo requisito. El algoritmo debe tener en cuentá todos los niveles de energía s_i posbiles para cada día. Es decir debe considerar todas las posibilidades de haber entrenado i días previamente

Para encarar este nuevo requerimiento, pensamos en utilizar una tabla. La tabla que terminamos utilizando tiene en el eje vertical los días de entrenamiento y en el eje horizontal la cantidad de días entrenados desde el último descanso (o el índice de energía).

	N° Día que voy entrenando desde el último descanso									
Fecha N°	1	2	3	4	5	6	7	8	9	10
1	36									
2	2	38								
3	99	63	87							
4	57	118	82	106						
5	158	116	167	123	146					
6	181	219	165	208	163	184				
7	230	242	268	206	248	201	207			
8	282	291	291	309	246	286	224	224		
9	301	315	324	324	342	279	309	241	237	
10	350	342	356	365	364	380	302	326	254	247

Figura 2: Seguimiento del algoritmo definitivo (basado en archivo de prueba 10.txt)

En la **figura 2** podemos ver cómo se rellena nuestra tabla para un caso específico. Adicionalmente se marca en azul el seguimiento de la solución. Llamaremos i al índice en el eje horizontal y n al índice en el eje vertical.

En la primera columna de la tabla es el caso de que el día anterior no se entrenó. Para rellenar esta columna, se le suma la ganancia de ese día *(mín(e[n], s[0]))* y el máximo posible del último día de entrenamiento antes del descanso *(máx(fila n - 2))*.

Para el resto de las columnas, en cada celda se suma la ganancia del día (mín(e[n], s[i])) y la ganancia hasta el día previo (fila n - 1, columna i - 1).

El máximo valor de la última fila es la máxima ganancia del cronograma completo.

A continuación se muestra la implementación en código.

```
(List<List<int>> matriz, List<int> indicesMaximosPorDia) rellenarMatriz(Cronograma
    cronograma) {
  final cantidadDias = cronograma.length;

  // Generamos la matriz con forma de triangulo
  final matriz = List.generate(cantidadDias, (index) => List.filled(index + 1, 0));

  // Indices de energia con la maxima ganancia posible de cada dia (valores optimos
    )
  final indicesMaximosPorDia = List.filled(cantidadDias, 0);

  for (int indiceDia = 0; indiceDia < cantidadDias; indiceDia++) {
    for (int indiceEnergia = 0; indiceEnergia < matriz[indiceDia].length;
    indiceEnergia++) {
      // Ganancia de la posicion actual
      matriz[indiceDia][indiceEnergia] = min(
        cronograma[indiceDia].esfuerzo,
        cronograma[indiceEnergia].energia,
      );

      // Si tiene toda la energia posible sumamos la ganancia maxima posible del
    dia previo al descanso
      if (indiceEnergia == 0 && indiceDia > 1) {
        matriz[indiceDia][indiceEnergia] += matriz[indiceDia - 2][
    indicesMaximosPorDia[indiceDia - 2]];
      }
      // Si hay cansancio, sumamos la ganancia del dia anterior
      else if (indiceEnergia > 0) {
        matriz[indiceDia][indiceEnergia] += matriz[indiceDia - 1][indiceEnergia -
    1];
      }
```

```
26
27        // Guardamos la ganancia maxima posible para ese dia
28        if (matriz[indiceDia][indiceEnergia] > matriz[indiceDia][indicesMaximosPorDia
      [indiceDia]]) {
29          indicesMaximosPorDia[indiceDia] = indiceEnergia;
30        }
31      }
32    }
33
34    return (matriz, indicesMaximosPorDia);
35  }
```

Podemos observar que *indicesMaximosPorDia* representa el índice en la matriz del valor óptimo de cada día. Con esta información se puede reconstruir que días se entrena y cuáles se descansa. El código para reconstruir la planificación del entrenamiento se puede encontrar en el repositorio del proyecto en *lib/calcular_planificacion.dart*

3.2.1. Ecuación de recurrencia

Para la ecuación de recurrencia, **tomando n como el índice del último día de entrenamiento siendo 0 el índice del primer día**, primero pensamos esta forma en donde no era necesaria una ecuación auxiliar:

$$OPT(n) = \max_{i=0}^{n}(OPT(n-i-2) + \sum_{j=0}^{i}(\text{mín}(e_{n-j}, s_{i-j})))$$

En ella vemos que la ganancia óptima para $n+1$ días será el máximo entre una cantidad $n+1$ de opciones, en donde en cada opción lo que hacemos es obtener, recursivamente, la ganancia óptima de un día anterior al último descanso y sumarle a esta la sumatoria de las ganancias de cada día partiendo desde un día posterior al último descanso con la máxima energía, hasta el último día analizado, cada día consecutivo teniendo un nivel de energía menos.

La ecuación anterior repite varias veces las mismas operaciones en la sumatoria. Además es difícil de interpretar y no representa correctamente la manera en la que está escrito el algoritmo en código. Por este motivo decidimos escribir otra función de recurrencia que representa adecuadamente cómo se comporta el algoritmo propuesto. Este utiliza una matriz para guardar las ganancias por nivel de energía a cada día del cronograma.

$$M(n,i) = \begin{cases} OPT(n-2) + \text{mín}(e_n, s_i) & \text{si i} = 0 \\ M(n-1, i-1) + \text{mín}(e_n, s_i) & \text{si i} > 0 \end{cases}$$

$$OPT(n) = \max_{i=0}^{n}(M(n,i))$$

Finalmente, de esta forma tendremos, para el último día de entrenamiento, todas las ganancias posibles para todos los niveles de energía y nos quedamos con la máxima.

4. Mediciones y complejidad

4.1. Complejidad teórica

El algoritmo que se encarga de rellenar la tabla funciona iterativamente n veces y tiene que llenar una cantidad de celdas equivalente al número de iteración. Es decir, tiene que llenar

$$\sum_{i=1}^{n} i$$

lo que se aproxima a $\frac{n^2}{2}$ y en cuanto a complejidad equivale a $\mathbf{O}(n^2)$.

5. Gráficos

5.1. Mediciones de tiempo según cantidad de elementos

En el siguiente gráfico podemos observar el tiempo que demora el programa en calcular la solución según la cantidad de días que se tengan que utilizar. El análisis se hizo haciendo 20 pruebas por cada punto y tomando un promedio de cada una. Además se proyectó sobre el mismo la curva de la regresión cuadrática sobre los datos de la tabla:

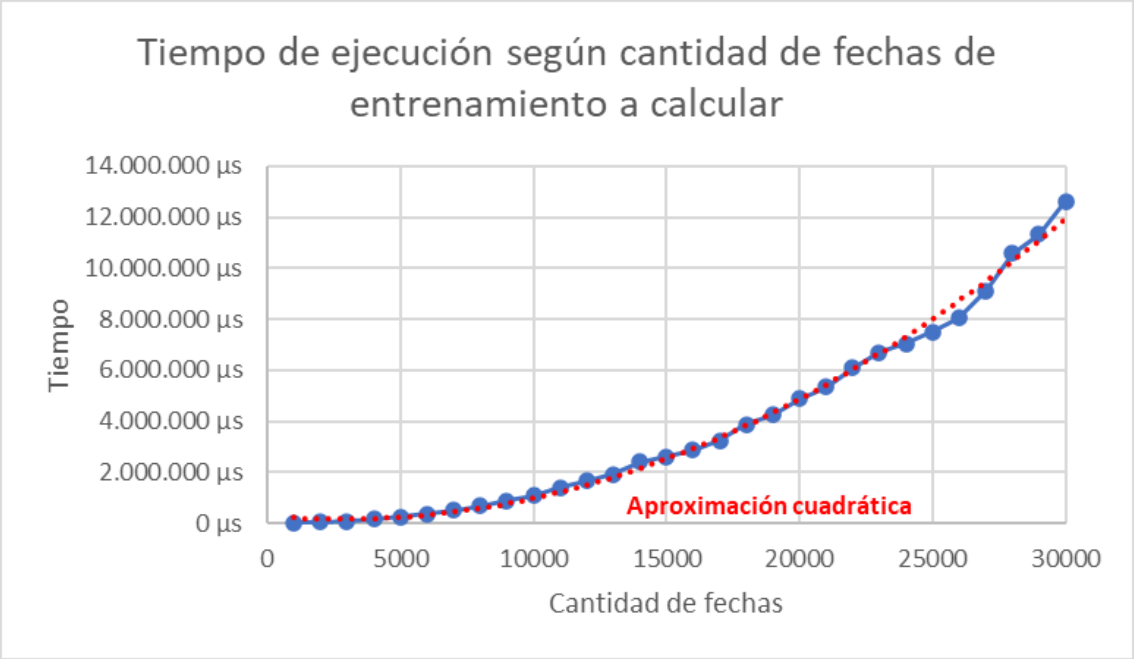

Figura 3: Tiempo de ejecución vs cantidad de fechas

En esta **figura 3** se puede observar que efectivamente el comportamiento es cuadrático tal cual esperábamos.

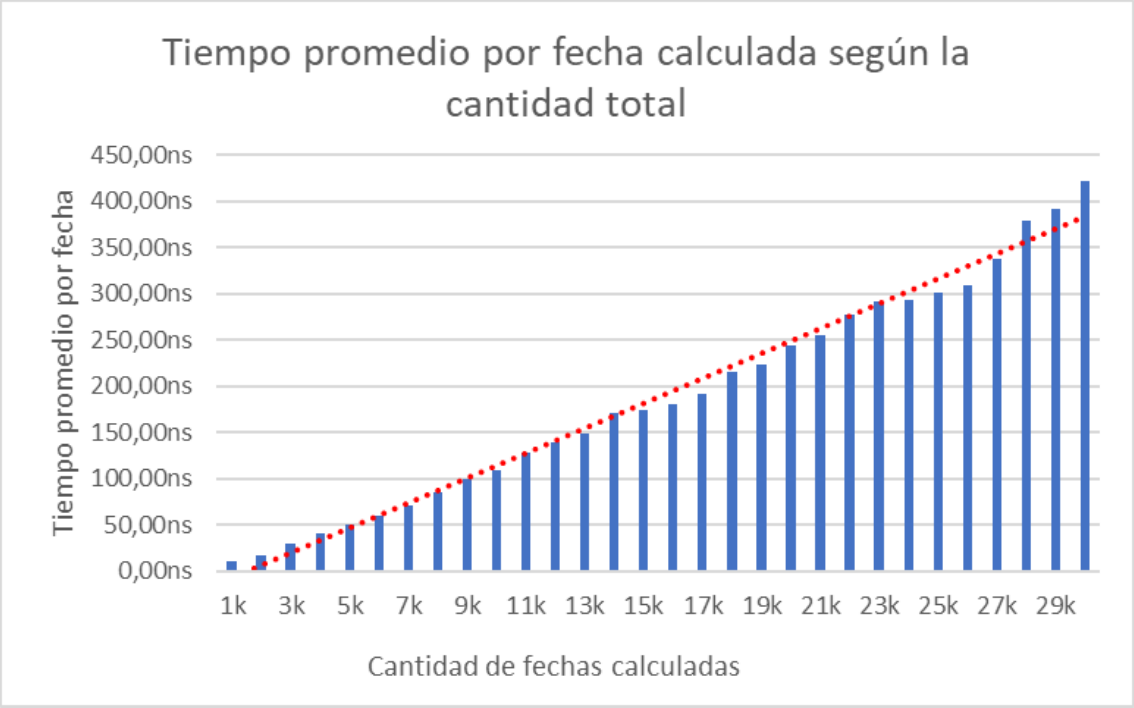

Figura 4: Tiempo promedio por fecha vs cantidad total

En esta **figura 4** podemos observar el mismo patrón pero visto por el tiempo que tarda en promedio por cada elemento individual. Viendo que efectivamente por ser cuadrático, mientras más elementos, más tarda en promedio en calcular cada uno.

6. Referencias

- `https://github.com/MaximilianoCovela/TDA-TP2-2C2023/blob/main/lib/calcular_planificacion.dart`
- `https://github.com/MaximilianoCovela/TDA-TP2-2C2023/blob/main/lib/algoritmo_no_optimo/calcular_planificacion_greedy.dart`