República Bolivariana de Venezuela
Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Técnicas Avanzadas de Programación
Semestre 1-2025

Problema DP: Conciliación de programaciones radiales

Estudiante:
César Carios
C.I 30.136.117

28 de julio del 2025

Introducción al Problema

El problema consiste en conciliar dos cronogramas de reproducción de una canción, generados por dos empresas distintas, A y B. El objetivo es determinar el número máximo de emparejamientos (matches) entre las dos listas, donde un match es válido si la diferencia de tiempo entre dos registros no supera una tolerancia S en segundos. En caso de existir múltiples soluciones con el mismo número máximo de matches, se debe seleccionar aquella con la menor diferencia de tiempo promedio. El problema especifica el uso de programación dinámica para su resolución.

Planteamiento de la Solución

La estrategia implementada se basa en la programación dinámica, siguiendo la sugerencia del enunciado. Esta técnica es ideal para problemas de optimización que pueden ser descompuestos en subproblemas superpuestos.

Procesamiento Inicial de Datos

Como primer paso, todos los tiempos en formato `hh:mm:ss` se convierten a una unidad única de segundos desde la medianoche. Esto simplifica enormemente los cálculos de diferencias. Seguidamente, ambas listas de tiempos (A y B) se ordenan de manera ascendente. Este ordenamiento es crucial para que la subestructura óptima del problema sea válida y el algoritmo de programación dinámica pueda construir la solución de forma correcta, evitando emparejamientos cruzados.

Programación Dinámica

Se define una tabla (matriz) de programación dinámica, a la que llamaremos DP, de dimensiones $(N_a + 1) \times (N_b + 1)$, donde N_a y N_b son el número de entradas en las programaciones A y B, respectivamente.

Cada celda $DP[i][j]$ de la tabla almacena el resultado óptimo al considerar los primeros i elementos de la lista A y los primeros j elementos de la lista B. El resultado óptimo se define como un par de valores:

1. El número máximo de matches.
2. La suma total de las diferencias de tiempo para esos matches.

La tabla se llena iterativamente basándose en la siguiente relación de recurrencia para cada celda $DP[i][j]$:

1. **No emparejar el tiempo** $A[i]$: El resultado sería el que ya se calculó para $DP[i-1][j]$.
2. **No emparejar el tiempo** $B[j]$: El resultado sería el que ya se calculó para $DP[i][j-1]$.

3. **Intentar emparejar** $A[i]$ **con** $B[j]$: Se calcula la diferencia $d = |A[i] - B[j]|$. Si $d \leq S$, este es un match potencial. El resultado de esta opción se obtiene a partir de la solución para $DP[i-1][j-1]$, sumando 1 al número de matches y d a la suma de diferencias.

La celda $DP[i][j]$ se actualiza con la mejor opción entre las disponibles. La mejor opción es la que tiene más matches. Si hay un empate en el número de matches, se prefiere la que tiene la menor suma de diferencias acumuladas.

El resultado final del problema se encuentra en la última celda de la tabla, $DP[N_a][N_b]$. Si el número de matches es cero, la salida es No matches. De lo contrario, se imprime el número de matches y el tiempo promedio de diferencia, que se calcula dividiendo la suma total de diferencias entre el número de matches.

Análisis de Complejidad

El rendimiento del algoritmo se puede analizar en términos de tiempo y espacio. Las restricciones del problema son $1 \leq N_a, N_b \leq 200$.

Complejidad en Tiempo

La complejidad total del algoritmo está determinada por la suma de sus partes principales:

- **Conversión y lectura de datos**: Se leen $N_a + N_b$ tiempos. Esta operación tiene una complejidad de $O(N_a + N_b)$.
- **Ordenamiento**: Se ordenan dos listas de tamaños N_a y N_b. Usando un algoritmo eficiente como Quicksort o Mergesort, la complejidad es $O(N_a \log N_a + N_b \log N_b)$.
- **Llenado de la tabla DP**: Se recorre una tabla de $(N_a + 1) \times (N_b + 1)$ celdas. Para cada celda, se realizan operaciones de tiempo constante (comparaciones y sumas). Por lo tanto, esta fase tiene una complejidad de $O(N_a \cdot N_b)$.

La complejidad en tiempo total es la suma de estas partes: $O(N_a + N_b + N_a \log N_a + N_b \log N_b + N_a \cdot N_b)$. El término dominante es $O(N_a \cdot N_b)$, por lo que la complejidad en tiempo del algoritmo es:

$$T(N_a, N_b) = O(N_a \cdot N_b)$$

Complejidad en Espacio

La complejidad en espacio se refiere a la memoria adicional utilizada por el algoritmo:

- **Almacenamiento de tiempos**: Se usan dos vectores para guardar los tiempos convertidos a segundos, lo que requiere un espacio de $O(N_a + N_b)$.

- **Tabla DP**: La tabla de programación dinámica es la mayor consumidora de memoria. Su tamaño es $(N_a + 1) \times (N_b + 1)$, y cada celda almacena dos enteros. Esto resulta en una complejidad de espacio de $O(N_a \cdot N_b)$.

El término dominante es el de la tabla DP. Por lo tanto, la complejidad en espacio del algoritmo es:

$$S(N_a, N_b) = O(N_a \cdot N_b)$$

Conclusión

La solución propuesta utilizando programación dinámica es eficiente y se ajusta perfectamente a la naturaleza del problema. La complejidad cuadrática, $O(N_a \cdot N_b)$, es totalmente aceptable para las restricciones dadas ($N_a, N_b \leq 200$), garantizando que el programa se ejecute en un tiempo muy bajo. El enfoque permite encontrar de manera sistemática la solución óptima global basándose en las soluciones de subproblemas más pequeños, cumpliendo con todos los requisitos del enunciado.