Documentación Proyecto 1
Juan Pablo Reyes Romero – 202310852
Juan Sebastián Maldonado - 202311840

1. Re-Contextualización del problema
   - El problema como se sabe, consiste en que se tienen k celdas, de enteros no negativos, cuya suma debe ser exactamente n. Cada celda aporta cierta creatividad según los dígitos establecidos (3, 6 y 9) y según la creatividad por posición decimal. La idea entonces es repartir la suma n entre las k celdas para así poder maximizar la suma total de creatividad.
2. Explicación el algoritmo empleado
   - Habiendo recapitulado el problema a resolver, cabe destacar que se empleó programación dinámica para poder llegar a las soluciones óptimas. Básicamente, lo que proponemos es la descomposición del problema por posiciones decimales, evaluando desde las posiciones más significativas a las menos significativas (P = 5 -> 0). Ahora bien, para cada posición p, se lleva a cabo el proceso pensando "cuál sería la creatividad máxima que puedo abstraer de esta posición". Eso se hace considerando los casos anteriores, en donde ya se hizo en cada uno dicha consideración. Sabiendo eso, cada "caso particular" se resuelve por separado, para posteriormente combinarse entre sí.
     En términos del código, se trata cada una de las posiciones decimales por separado, y manteniendo una lista **dp** que almacena para cada posible resumen de la suma que está hasta esa posición, la máxima creatividad alcanzable en las posiciones ya procesadas. Luego, al procesar una posición p se construye otro vector **resultado_actual** a partir de **dp**, que contiene la solución para las posiciones ya combinadas. Posteriormente, **dp** se actualiza a **resultado_actual**. Finalmente, se retorna **dp [0]**, el cual representa la mayor creatividad total compatible con la suma n.

   CASOS BASE Y VALIDACIONES INICIALES

   - Caso base: siguiente = [0] -> antes de procesar la suma en cualquier posición, a suma evidentemente es 0.
   - Gracias a **INF NEG,** es posible representar casos imposibles y asi no habrán contribuciones que puedan alterar la conformación de la solución verdadera.
   - Validaciones: En consultas del máximo por intervalo, se rectifica **inicio>fin**, para luego retornar **INF NEG**. Además de eso, al calcular **limite_inferior** y **limite_superior,** se tienen en cuenta los límites que restringen la suma de dígitos, para así no tener en cuenta variables que estén fuera de rango.

   ESTRUCTURAS USADAS

   - **dp**: es el arreglo DP que contiene la solución parcial para los estados ya solucionados.
   - En su recorrido, para cada posición se construye:
     - **Resultado_actual**: esta lista tiene entradas que corresponden a posibles valores **c_actual**, para dicha posición.
     - **Valores_transformados**: este es un vector intermedio que combina las soluciones de **dp** con ganancias locales lineales, así se reutilizan casos ya calculados, para reducir la complejidad temporal.

- **Lista_residuos:** es una construcción de vectores con los valores transformados indexados por bloques **3*q + residuo**. Esto se hace ya que la restricción del problema es la creatividad dada por los dígitos 3, 6 y 9.

ACTUALIZACIÓN DE DP

Para cada posible **c_actual**, el cual es un índice de la lista **resultado_actual**:

1. Se determinan los limites mencionados antes para la variable q. A través de este paso se reúnen restricciones de suma exacta, y lo que se permite con las **K** celdas.
2. Se iteran las tres clases de residuo posibles (0, 1 y 2). Para cada uno de estos, se convierte el intervalo de posibles q en índices de inicio y fin, en **lista_residuos[residuo]**
3. Si existe un intervalo válido, se pide lo máximo posible sobre ese intervalo con una **tabla_logaritmos**, y a ello se le suma lo que aporta la posición actual (tomo lo demás, y considero la opción que tengo actualmente).
4. De los tres residuos posibles mencionados anteriormente, se toma el valor maximizado, y se guarda en **resultado_actual[c_actual]**
5. Ya finalmente, resultado se asigna a siguiente, y se continua con la posición posterior (**p--**).

3. Complejidades de la Solución

FUNCIÓN CONSTRUIR(ARREGLO)

- **Que hace**: Arma un Sparse Table para RQM-máximo y un arreglo de logaritmos.
- **Bucles**: Niveles: nivel = 1 ..[log2 n]: ~log n niveles. En cada nivel, un recorrido de tamaño n para combinar pares.
- **Tiempo:** Suma por niveles: n + (n-1) + (n+3) + ... = n * log n => O(n log n)
- **Espacio:** La tabla guarda n + n/2 + n/4 ... < 2n elementos. El arreglo logaritmos es de tamaño n+1 => O(n)

FUNCIÓN MAX(TABLA_MAXIMOS, LOGARITMOS, INICIO, FIN)

- **Que hace**: responde RQM maximo en un intervalo con Sparse Table.
- **Tiempo**: busca k, toma dos bloques y compara. Todo indice constante. => O(1)
- **Espacio**: Constante => O(1)

FUNCIÓN RESOLVER_CASO(K, N, DATOSCREATIVIDAD)

Idea de estructura: se itera por 6 digitos (de p=5 a). En cada paso: Paso 1, se calcula dp para un rango más grande (max_entrada = digito + 10*max_siguiente). Paso 2, se arma un vector valores_transformados de tamaño max_siguiente+1. Paso 3, para cada residuo r contendio en {0,1,2} se crea un arreglo de tamaño max_siguiente/3 y se construye una

Spare Table (via construir). Paso 4, se rellena resultado_actual para c_actual en 0...max_entrada, haciendo a lo sumo 3 consultas max(...) de tiempo O(1) cada una.

Crecimiento de los tamaños: Max_siguiente empieza en 0 y se multiplica por 10 en cada digito: p=5 <= 9 ; p=4 <= 99 ; p=3 <= 999 ; p=2 <= 9999 ; p=1 <= 99999 ; p=0 <= 999999

Denotemos por m el tamaño dominante en la última iteración m = 10^D con D=6

Desglose por iteración:

- Construir valores_transformados: O(S)
- 3 arreglos por residuos, cada uno de tamaño S/3; por cada uno se llama construir:
  - Tiempo por residuo: O((S/3) log (S/3))
  - 3 residuos => O (S log S)
- Relleno de resultado_actual para max_entrada = digito + 10*S: 10*S iteraciones, cada una hace solo operaciones O(1) + 3 consultas RQM O(1).
  => O(S)

Tiempo por iteración: O(S log S) + O(S) = O(S log S)

Tiempo total en las 6 iteraciones: Dominado por la ultima: O(m log m) con m=10^6

Espacio: O(m) con m = 10^6

- dp/resultado_final: tamaño = O(S).
- 3 Sparse Tables sobre arreglos que en total suman O(S) elementos y cada una almacena O(tamaño) por nivel => O(S) total
- No se acumulan entre iteraciones.

En resumen…

- Complejidad Temporal Asintótica: O(m log m)
- Complejidad Espacial Asintótica: O(m)

4. Gráfico de apoyo

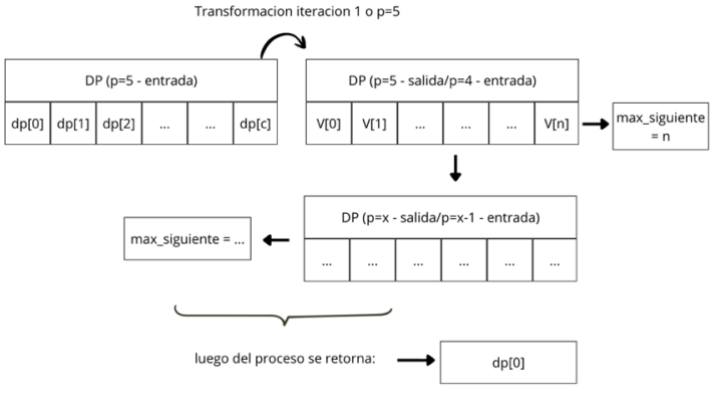

Imagen 1 / Gráfico proceso actualización DP - Elaborado por Juan Pablo y Juan Sebastián – (Canva, 2025)