

Explicación del Código: Worms

En esta sección analizaremos cómo implementar, en Java, dos enfoques para resolver el problema Worms:

1. La solución “cándida” o ingenua
2. La solución óptima basada en búsqueda binaria

Ambas funciones reciben como entrada dos arreglos:

- `gusanos[]`: indica cuántos gusanos hay en cada pila.
- `jugosos[]`: representa las consultas; cada elemento es la etiqueta de un gusano que debemos ubicar dentro de alguna pila.

El objetivo es determinar en qué pila se encuentra cada gusano jugoso.

1. Método Cándido (Búsqueda Secuencial)

El primer paso consiste en calcular un arreglo auxiliar llamado `pilas[]`, que corresponde a un arreglo de sumas acumuladas. Este arreglo permite saber hasta qué etiqueta llega cada pila. Por ejemplo:

- Si la pila 1 tiene 3 gusanos,
- la pila 2 tiene 2 gusanos,
- y la pila 3 tiene 4 gusanos,

entonces el arreglo de sumas acumuladas quedaría:

Pila Hasta qué etiqueta llega

1	3
2	5
3	9

Esto significa que:

- Las etiquetas 1–3 pertenecen a la pila 1
- Las etiquetas 4–5 pertenecen a la pila 2
- Las etiquetas 6–9 pertenecen a la pila 3

Una vez construido este arreglo, recorremos cada gusano jugoso y realizamos una búsqueda lineal: verificamos, pila por pila, cuál es la primera suma acumulada que es mayor o igual al número buscado.

En cuanto la encontramos, esa es la pila correcta y pasamos a la siguiente consulta.

Este método es sencillo y correcto, pero su principal desventaja es el tiempo de ejecución:

- Para cada gusano jugoso recorremos todas las pilas, dando una complejidad aproximada de $O(N \times M)$.
- Con valores de hasta 10^5 , este enfoque puede volverse demasiado lento.

2. Método Óptimo (Búsqueda Binaria)

El método óptimo se basa en la misma idea inicial: construir el arreglo de sumas acumuladas `pilas[]`. La diferencia está en cómo ubicamos la pila correspondiente para cada etiqueta.

En lugar de buscar secuencialmente, aplicamos búsqueda binaria, aprovechando que `pilas[]` siempre está ordenado de forma creciente.

Para cada consulta:

- Buscamos la primera posición en `pilas[]` cuyo valor sea mayor o igual al gusano jugoso.
- Esa posición representa directamente el índice de la pila en la que se encuentra el gusano.

Para ello implementamos un método auxiliar llamado `binarySearch()`, que trabaja con dos punteros (izquierda y derecha), calcula un punto medio y decide en qué mitad continuar la búsqueda.

El cálculo del punto medio se hace de forma segura:

`mid = left + (right - left) / 2;`

Esto evita posibles desbordamientos al sumar dos números grandes.

Si el valor de la mitad cumple con la condición (ser \geq al gusano buscado), actualizamos el resultado temporal y seguimos buscando hacia la izquierda para encontrar el intervalo más pequeño que lo cubra.

Si no, desplazamos la búsqueda hacia la derecha.

De esta forma obtenemos una solución con complejidad:

$O(M \log N)$

Lo que la convierte en la opción ideal para grandes cantidades de datos.

Conclusiones

- Ambos métodos producen resultados correctos, pero difieren enormemente en eficiencia.
- El método ingenuo puede resultar suficiente para entradas pequeñas, pero no escala bien.
- El método óptimo, con búsqueda binaria, aprovecha la estructura ordenada del arreglo acumulado y garantiza un rendimiento adecuado incluso con $10^5 \times 10^5$ consultas.

En resumen, el uso de sumas acumuladas combinado con búsqueda binaria es la estrategia recomendada para resolver el problema Worms de forma eficiente y escalable.