Explicación del Código: Diverse Team

En esta sección analizaremos la implementación en Java del problema número ocho, “Diverse Team”, mediante dos enfoques:

- Método Cándido (enfoque ingenuo)
- Método Óptimo (usando una tabla hash)

Ambos métodos reciben los mismos parámetros:

1. N → cantidad total de estudiantes.
2. K → número de estudiantes que debe tener el equipo.
3. ratings[] → arreglo con los ratings de cada estudiante.

1. Método Cándido (enfoque básico)

El método cándido busca estudiantes con ratings diferentes revisando manualmente los valores que ya han sido seleccionados. Su funcionamiento se puede describir así:

Estructura general

1. Se crea un arreglo indices[] de tamaño K, donde almacenaremos los índices finales.
2. Se inicializa un contador que indicará cuántos estudiantes válidos llevamos seleccionados.
3. Se recorre el arreglo ratings desde la posición 0 hasta N − 1.
4. Para cada rating, verificamos si ya ha sido seleccionado anteriormente. Esto se hace comparándolo con los ratings ya guardados en el arreglo indices.

Detección de repetidos

- Se declara un booleano existe que inicialmente será false.
- Se recorre únicamente la parte del arreglo indices que ya ha sido llenada.
- Cada comparación se hace así:

ratings[i] == ratings[indices[j]−1]

Esto es necesario porque los índices almacenados se guardan como posiciones 1-based (empezando en 1), pero el arreglo en Java comienza en 0.

Si encontramos que el rating actual ya está registrado, marcamos existe = true y salimos del ciclo.

Agregar un nuevo estudiante

Si el rating no existe y todavía no hemos alcanzado los K estudiantes, entonces:

- Guardamos el índice i + 1 en indices[].
- Incrementamos el contador.

Resultado final

- Si al final del proceso el contador es menor que K, significa que fue imposible formar el equipo → se retorna null.
- En caso contrario, se retorna el arreglo indices[].

2. Método Óptimo (usando Hash Table)

El segundo método resuelve el mismo problema, pero de forma mucho más eficiente.

En vez de comparar un rating con todos los seleccionados, usaremos una tabla hash donde cada rating se registra una sola vez.

Estructura del método

1. Se declara una tabla hash de tamaño aproximado a los valores del dominio (en el ejemplo, 10007).
2. Se crea un arreglo resultado[] de tamaño K para almacenar los índices válidos.
3. Se inicializa un contador que rastrea cuántos ratings distintos llevamos.

Recorrido del arreglo

- El ciclo principal avanza desde i = 0 hasta N − 1,
- y únicamente continúa mientras todavía no hayamos seleccionado K estudiantes.

Inserción en la tabla hash

Para cada rating:

- Se verifica si ya está registrado utilizando un método de búsqueda de la tabla.
- Si no aparece en la tabla:
    - Se inserta el rating.
    - Se almacena su índice i + 1 en el arreglo resultado.
    - Se incrementa el contador.
- Si el rating ya existe, simplemente se ignora.

Resultado final

- Si al final no se lograron obtener K ratings únicos → retorna null.
- Si sí se encuentran → retorna el arreglo de índices.

3. Comparación de tiempos y eficiencia

Para demostrar la diferencia entre los métodos, se ejecutaron tres escenarios:

Escenario 1: entrada pequeña

- Ambos métodos entregan la misma respuesta.
- El método cándido es ligeramente más rápido debido a que la entrada es mínima y la estructura hash tiene una sobrecarga inicial.

Escenario 2: entrada mediana

- El método cándido comienza a presentar retardos notables.
- El método óptimo ejecuta considerablemente más rápido.

Valores mostrados:

- Método Cándido: ~6000 microsegundos
- Método Óptimo: ~1000 microsegundos

Escenario 3: entrada grande

- La diferencia se vuelve muy evidente:
- Método Cándido: ~179,000 microsegundos
- Método Óptimo: ~15,000 microsegundos

Esto confirma que el método cándido tiene una complejidad aproximada de:

$O(N \times K)$

Mientras que el método óptimo opera en:

$O(N)$

Conclusión

- Aunque el método cándido funciona, solo es adecuado para entradas pequeñas debido a su elevado costo computacional.
- El método óptimo, basado en una tabla hash, es superior para valores grandes de N y K, logrando una reducción significativa del tiempo de ejecución.
- En todos los casos, ambos métodos encuentran la misma solución, pero su desempeño es drásticamente distinto en entradas de mayor tamaño.