UNIVERSIDAD FRANCISCO DE PAULA SANTANDER

FACULTAD DE INGENIERÍA

PROGRAMA DE INGENIERÍA DE SISTEMAS

DESARROLLADO POR:

- 1152139 CRISTIAN JULIAN LAMUS LAMUS

ASIGNATURA: Análisis de Algoritmos

Semestres I 2023

El presente documento tiene como objetivo exponer el informe de todo el trabajo realizado en conformidad con la actividad que dejó el docente de la materia Análisis de algoritmos en https://leetcode.com.

CONTENIDO

SOLUCIÓN DEL PROBLEMA EN LEETCODE

El problema propuesto por el profesor para resolver fue: *922. Sort Array By Parity II*, para el cual su solución requiere de un método que:

*Dado un arreglo de números enteros, la mitad de los números enteros en números son impares y la otra mitad son pares. Ordene la matriz de modo que siempre que **nums[i]** sea impar, **i** sea impar, y siempre que **nums[i]** sea par, **i** sea par.*
Devuelve cualquier arreglo de respuesta que satisfaga esta condición.

Propuse el código de solución en java, **Main.java**, para resolver el problema.

```java
/**
     * Ordena un Arreglo, los números pares en posiciones pares, y los números impares
     * en posiciones impares.
     *
     * @param nums un Arreglo de números enteros, con número igual de pares e impares.
     * @return un Arreglo con los números ordenados.
     */
    public static int[] sortArrayByParityII(int[] nums) {
        int even = 0;
        int odd = 1;
        while (true) {
            while (even < nums.length && nums[even] % 2 == 0) {
                even += 2;
            }
            while (odd < nums.length && nums[odd] % 2 == 1) {
                odd += 2;
            }
            if (odd >= nums.length || even >= nums.length) {
                return nums;
            }

            int temp = nums[odd];
            nums[odd] = nums[even];
            nums[even] = temp;
        }
    }
```

El método principal tiene como nombre ***sortArrayByParityII()*** que recibe un arreglo de enteros como ya antes se había mencionado, hace uso de dos variables enteras como apuntadores ***even*** como apuntador de posiciones pares y ***odd*** como apuntador de posiciones impares, inicializadas en 0 y 1 respectivamente.

Se crea un ciclo infinito que no parará hasta que el arreglo se encuentre ordenado, dentro del ciclo se crea otro ciclo para recorrer los números pares de forma que el index **(even)** comienza en 0 y va aumentando de 2 en 2 para solo pasar por las posiciones pares, y de igual modo se crea otro ciclo, esta vez para recorrer los números impares index **(odd)** en 1 y va aumentando de 2 en 2 para solo pasar por las posiciones impares, si se cumple la condición siguen recorriendo.

Luego, se prueba una condición donde ***even*** u ***odd*** son mayores al tamaño del arreglo retorna el arreglo ordenado.

En caso de que haya un número impar en una posición par o un número par en un posición par hará un intercambio con una variable ***temp*.**

GENERACIÓN DE CASOS DE PRUEBA

Según lo especificado en el ejercicio debemos tener en cuenta en qué casos puede o no puede fallar el algoritmo creado, es así que basándonos en la información base para este problema de leetcode, hallé los siguientes casos de prueba que debe recibir el algoritmo, teniendo en cuenta las restricciones de entrada dadas por el propio enunciado de LeetCode. Excluí los arreglos de tamaño superiores a 1000 para facilitar el manejo de datos, cabe resaltar que el algoritmo aún funciona para cantidades superiores de datos. Cree un código en java capaz de generar los casos de prueba de manera controlada llamado "GeneradorDeCasos.java".

Restricciones:

```
- 2 <= nums.length <= 2 * 10^4.
- nums.length es par.
- La mitad de los números enteros en nums son pares.
- 0 <= nums [i] <= 1000.
```

Caso 1. Tal que, el arreglo se encuentra desordenado. Elegí hacer 20 de estos casos porque son entradas que el algoritmo debe resolver correctamente.
Input: `[5, 4, 3, 2] , [27, 12, 10, 43] , [34, 34, 27, 17, 8, 15]`.

Caso 2. Tal que, el contenido del arreglo sean números pares hasta la mitad, seguido de números impares. Elegí hacer 20 de estos casos porque son entradas que el algoritmo debe resolver correctamente.
Input: `[2, 4, 3, 5] , [10, 12, 27, 43] , [34, 34, 8, 17, 27, 15]`.

Caso 3. Tal que, el contenido del arreglo sean números impares hasta la mitad, seguido de números pares. Elegí hacer 20 de estos casos porque son entradas que el algoritmo debe resolver correctamente.
Input: `[3, 5, 2, 4] , [27, 43, 10, 12] , [17, 27, 15, 34, 34, 8]`.

Caso 4. Tal que, el contenido del arreglo sean con números pares en posiciones impares y números impares en posiciones pares. Elegí hacer 20 de estos casos porque son entradas que el algoritmo debe resolver correctamente.
Input: `[3, 2, 5, 4] , [27, 10, 43, 12] , [17, 34, 15, 34, 27, 8]`.

Caso 5. Tal que, el contenido del arreglo sean con números pares en posiciones pares y números impares en posiciones impares. Elegí hacer 20 de estos casos porque son entradas que el algoritmo debe resolver correctamente.
Input: `[2, 5, 4, 5] , [10, 43, 12, 27] , [34, 27, 8, 17, 34, 15]`.

DESARROLLO DEL PROGRAMA FUERA DE LEETCODE

Para el desarrollo del algoritmo fuera de la plataforma de leetcode, cree un proyecto en el IDE Netbeans con la finalidad de ejecutarlo correctamente, cree un método main para las entradas del programa por el usuario y llamar al método ***sortArrayByParityII()*** para cada una de ellas.

El programa se ejecuta 100 veces, una vez por caso de prueba, leyendo una línea de entrada con números enteros separados por espacios en cada línea, los convierte en un arreglo de enteros y los usa como parámetros para el método ***sortArrayByParityII().***

Cree un Script en windows tipo .bat, **GeneradorDeCasos.bat**,para la ejecución de los dos archivos por fuera de los IDE, este ejecuta el archivo **GeneradorDeCasos.java** y redirige la salida a un archivo llamado *CasosGenerados.txt*. Luego ejecuta el archivo **Main.java**, y redirige la entrada desde el archivo *CasosGenerados.txt* y este da la salida en un archivo llamado *Salidas.txt*.
Los casos de entrada se generan con el programa **GeneradorDeCasos.java**, y se procesan con **Main.java**, y los resultados se almacenan en los archivos *CasosGenerados.txt* y *Salidas.txt*, respectivamente.

LINK AL VIDEO EXPLICATIVO Y AL REPOSITORIO

Link del problema:
https://leetcode.com/problems/sort-array-by-parity-ii/

Link del repositorio:
https://github.com/JulianL76/SortArrayByParityII

Link del video explicativo:
https://youtu.be/LJTTZ2t-SW0