INFORME

Algoritmo de búsqueda y ordenamiento

ESTUDIANTE

Ana Maria Moreno Casadiego 1152073

Diego Alexander Bermudez Flores 1152067

MATERIA

Análisis de Algoritmos -Grupo A

DOCENTE

Ing. Milton Jesús Vera Cárdenas

UNIVERSIDAD FRANCISCO DE PAULA SANTANDER

INGENIERÍA DE SISTEMAS

31 DE OCTUBRE DEL 2022

INTRODUCCIÓN

En este trabajo se realizó la implementación de un algoritmo que permite ordenar de manera descendente un arreglo por la cantidad de bits que tiene cada elemento. para eso se ha realizado una aplicación que permite probar el algoritmo con datos aleatorios para cada uno con un rango de 0 a 10^4 y el tamaño máximo que puede tener el arreglo es de 700.

Además de lo anterior, se dispondrá de la explicación de un código el cual nos dice la posición que debería ocupar un elemento en un arreglo de números, tanto en los casos en los que se encuentre como los que no, tomando como números máximos el 10^4 positivos y negativos.

Algoritmo utilizado en Leetcode:

-Algoritmo de búsqueda Binaria;

Java • Auto

```java
class Solution {
    public int search(int[] nums, int target) {
        int low = 0;
        int high = nums.length - 1;

        while(low <= high){
            int mid = (low + high) / 2;
            if(nums[mid] == target) return mid;

            else if(target > nums[mid]) low = mid + 1;
            else high = mid - 1;
        }
        return -1;
    }

    }
```

-Algoritmo de Búsqueda e inserción por selección;

```java
class Solution {
    public int searchInsert(int[] nums, int target) {
        //Ordenamiento de la lista
        Arrays.sort(nums);
        //Busqueda Binaria de la lista
        Arrays.binarySearch(nums, target);

        //En caso de que el taget sea menor a 0
        if(target < 0){return (Arrays.binarySearch(nums, target))+1;}
        //En caso de que el valor dentro de la busqueda binaria sea igual al target
        else if(Arrays.binarySearch(nums, target)==target)
        {
            //regresea la posicion
            return Arrays.binarySearch(nums, target);
           //En caso de que la busqueda sea menor al target
        }else if(Arrays.binarySearch(nums, target)<0)
        {
            //retorna el valor de la busqueda por menos uno, y luego le restamos uno
            return ((Arrays.binarySearch(nums, target))*-1)-1;
        }

        //Al final regresa el valor de la busqueda
        else return Arrays.binarySearch(nums, target);
    }
}
```

Ambos algoritmos implementados fueron realizados en el lenguaje de programación JAVA.

Programa de Búsqueda Binaria:

El Programa permite probar el algoritmo utilizado en leetcode por medio de un arreglo que puede tener un tamaño máximo de 700 con datos generados aleatoriamente con un rango de 0 a 10^4.

datosAleatorios:

```
public void datosAleatorios(int n) {

    try {
        String ruta = "datosPrueba.txt";
        File file = new File( pathname: ruta);
        if (!file.exists()) {
            file.createNewFile();
        }
        FileWriter fw = new FileWriter(file);
        BufferedWriter bw = new BufferedWriter( out: fw);
        for (int i = 0; i < n; i++) {
            int valorEntero = (int) Math.floor(Math.random() * (10000 - (-10000) + 1) + (-10000));
            bw.write( str: String.valueOf( i: valorEntero));
            bw.write( str: "\n");
        }
        bw.close();

    } catch (Exception e) {
        e.printStackTrace();
    }

}
```

El procedimiento datosAleatorios que está en la clase datosAleatorios obtiene como variable un número de tipo int el cuál representa a la proporción de datos que se van a crear. El procedimiento lo cual hace es generar un documento txt denominado “PruebaF.txt” y después lo abre y le comienza a añadir los datos que se van generando aleatoriamente en cada iteración del periodo for.

```
43      public int[] leerTxt(int arr[]) {
44          try {
                File archivo = null;
                FileReader fr = null;
                BufferedReader br = null;
48              archivo = new File( pathname: "datosPrueba.txt");
49              fr = new FileReader( file: archivo);
50              br = new BufferedReader( in: fr);
51              String linea;
52              int j = 0;
53              while ((linea = br.readLine()) != null) {
54                  arr[j] = Integer.parseInt( s: linea);
55                  j++;
56              }
57              Arrays.sort( a: arr);
58
59          } catch (Exception e) {
                e.printStackTrace();
61          }
62          return arr;
63      }
64
65  }
```

Este método que también se encuentra dentro de la clase datosAleatorios permite como su nombre lo indica llenar un arreglo, tiene como variable un arreglo de tipo int el cual será llenado con los datos aleatorios el cual lo encontramos en un archivo txt que se generó anteriormente en el método datosAleatorios. Lo primero que hace el método es abrir el archivo txt donde están los datos aleatorios y más adelante se procede a recorrer línea por línea el archivo txt sacando de cada línea el número que se encuentra y agregándole al arreglo en la posición del índice.

Main (solución,java)

```java
public static void main(String args[]) {
    // TODO code application logic here

    Datos d = new Datos();
    BusquedaBinaria bb = new BusquedaBinaria();
    int array[];
    int datosPrueba = 10000;

    if (args.length > 0 && args.length <= 2 && Integer.parseInt(args[0]) > 0 && Integer.parseInt(args[0]) <= 10000 && args[1] != null)
        array = new int[Integer.parseInt(args[0])];
        d.datosAleatorios( n: Integer.parseInt(args[0]));
        System.out.println( x: bb.search( nums: d.leerTxt( arr: array),  target: Integer.parseInt(args[1])));

    } else {
        array = new int[datosPrueba];
        d.datosAleatorios( n: datosPrueba);
        System.out.println( x: bb.search( nums: d.leerTxt( arr: array),  target: 8100));
    }
    //d.datosAleatorios(datosPrueba);

}
```

en este método hacemos para guardar los datos para probar por medio de la consola o también por el ide

Programa de búsqueda e inserción por selección.

La siguiente imagen a continuación ilustra el programa que permite la generación de casos de prueba de manera aleatoria, teniendo como mínimo el número de 100 casos de prueba en caso de requerirse.

```
public void cantidadCasos(int n) throws IOException
{
    try
    {
        String ruta = "cantidadCasos.txt";
        File file = new File("cantidadCasos.txt");
        if(!file.exists())
        {
            file.createNewFile();
        }
        FileWriter fw = new FileWriter(file);
        BufferedWriter bw = new BufferedWriter(fw);
        for(int i=100;i<(n+100);i++)
        {
            int valorEntero = (int) Math.floor(Math.random()*(10000-(-10000)+1)+(-10000));
            bw.write(String.valueOf(valorEntero));
            bw.write("\n");
        }
        bw.close();
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
}
```

Como se aprecia en la imagen, el método “cantidadCasos” encontrado en la clase “Datos” obtiene como variable un número de tipo int el cuál representa a la proporción de datos que se van a crear. El procedimiento lo cual hace es generar un documento txt denominado “casosPrueba.txt” y después lo abre y le comienza a añadir los datos que se van generando de manera aleatoria en cada iteración gracias al ciclo for.

```
    public int[] leerTxt(int arr[]) throws FileNotFoundException, IOException
    {
        try
        {
            File archivo = null;
            FileReader fr = null;
            BufferedReader br = null;
            archivo = new File("CantidadCasos.txt");
            fr = new FileReader(archivo);
            br = new BufferedReader(fr);
            String linea;
            int j = 0;
            while((linea = br.readLine()) != null)
            {
                arr[j] = Integer.parseInt(linea);
                j++;
            }
            Arrays.sort(arr);
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
        return arr;
    }
}
```

El siguiente método se encuentra dentro de la clase “Datos” y su función es permitirnos a nosotros como usuarios llenar el arreglo, este método tiene una variable de arreglo de tipo “int” el cual será llenado con los datos generados dentro del archivo txt generado anteriormente en el método de “cantidadCasos”. Las primera acciones que realizaría el método sería: Abrir el archivo .txt, recorrer línea por línea los datos generados anteriormente, mientras hace esto va sacando la posición en la que se encuentran dichos datos para usarlos en un futuro método.

Main

```
public class Main
{
    public static void main(String [] args) throws IOException
    {
        Busqueda b = new Busqueda();
        Datos d = new Datos();
        int array[];
        int casosPrueba = 10000;

        if(args.length > 0 && args.length<=2 && Integer.parseInt(args[0]) > 0 && Integer.parseInt(args[0]) <= 10000 && args[1] != null)
        {
            array = new int [Integer.parseInt(args[0])];
            d.cantidadCasos(Integer.parseInt(args[0]));
            System.out.println(b.searchInsert(d.leerTxt(array), Integer.parseInt(args[1])));
        }

        else
        {
            array = new int [casosPrueba];
            d.cantidadCasos(casosPrueba);
            System.out.println(b.searchInsert(d.leerTxt(array), 1000));
        }
    }
}
```

En el método “Main” nos permite probar nuestro código tanto por el IDE como por consola en caso requerido.

Conclusiones

- Del presente trabajo se puedo apreciar los tipos de búsqueda, el de inserción y el binario, además de esto se puede corroborar que estos programas son funcionales.