Instituto Tecnológico de Costa Rica
Escuela de Ingeniería en Computación
IC-4700 Lenguajes de programación
Profesor: Ignacio Trejos Zelaya

Asignación #2 [Proyecto]: Aprender características imperativas y secuenciales del lenguaje Go

Integrantes
Juan Sebastián Gamboa Botero - 2020030303
David Suárez Acosta - 2020038304

Fecha de entrega:
18 de Octubre, 2021

Índice

Introducción

Existen distintos tipos de programación que son clave de entender para poseer un vasto conocimiento para la resolución de problemas. Uno de estos tipos es la “programación secuencial”, la cual como el nombre dice, ejecuta las instrucciones secuencialmente. El lenguaje de Go (Golang) consiste en este tipo de programación, por lo que los estudiantes van a estar aprendiendo las características imperativas y secuenciales acerca de este por medio de esta asignación.

Con esta asignación, y por medio de los ítems especificados (del 1 al 10) y los experimentos realizados con estos, se desea familiarizarse con todo lo mencionado anteriormente y demostrarlo con los resultados obtenidos al realizar los experimentos.

Github: https://github.com/Orantoon/Lenguajes-Asignacion2.git

Generación de número pseudo-aleatorios

Estrategia desarrollada

El método de Congruencia Lineal Multiplicativa funciona para crear una secuencia de números aleatorios con el uso de una fórmula. Esta fórmula es la siguiente: $X_i = (a X_0 + c) \, mod \, m$, en donde X_i es el valor que se quiere buscar, a es el multiplicador y que se puede conseguir con las fórmulas $3 + 8k$ o $5 + 8k$ con "k" como una constante $0 \ldots \infty$, X_0 como la semilla o valor anterior de la secuencia, c como el incremento (en este caso es 0 para que se considere Lineal Multiplicativa) y m como el módulo o periodo.

Según las indicaciones de este ejercicio, se sabe que n será un parámetro de la función (además se sabe que el valor será un número entre 200 y 1000 pero como en los experimentos se utiliza además el valor 10000, esto se omite), la semilla debe ser un número primo entre 11 y 101 y el periodo o "m" debe ser mayor o igual a 2048. Con esta información, se asumió que estas variables serán parámetros y que deben ser validadas; junto con estas, también se agrega la constante "k" como parámetro ya que no se da información acerca del multiplicador "a" en las indicaciones, su única validación es que debe ser un número natural.

Una vez se tienen estas variables, se puede comenzar con el método principal en donde se introducen los números que se van generando con la fórmula en el arreglo "arr" con un loop de "for". Dentro del "for", se usa el algoritmo para generar un número en "num", este se convierte al rango 0 .. 53 haciéndole la operación de módulo de 54, se le agrega el resultado al siguiente espacio disponible del arreglo y por último, se sobreescribe la semilla con el último valor que se ingresó el arreglo, con esto, el próximo valor a convertir utilizará el valor previo como semilla. Además de esto, está comentada una forma de recortar el algoritmo cuando se comienza a repetir el valor, como no se solicita no se utilizó.

Código

```
func RandArray(n int, seed int, k int, m int) []int {

    // Validating "n"

    /*
    if n < 200 || n > 1000 {
        fmt.Println("El valor n es incorrecto")
        return nil
    }
    */
```

```
    // Validating Seed
    if seed < 11 || seed > 101 {
        fmt.Println("El valor de la semilla es incorrecto")
        return nil
    }

    for i := 2; i < seed; i++ { // Prime Number
        if seed % i == 0 {
            fmt.Println("El valor de la semilla es incorrecto")
            return nil
        }
    }

    // Validating "k"
    if k < 0 {
        fmt.Println("El valor k es incorrecto")
        return nil
    }

    // Validating "m"
    if m < 2048 {
        fmt.Println("El valor m es incorrecto")
        return nil
    }

    arr := make([]int, n)
    a := 8*k + 3    // 8k + 5 can also be used
    //first := seed

    for i := 0; i < n; i++ {    // Generating the Array
        num := (a * seed) % m   // Main Algorithm, X = (a * [seed or
previous number]) % m
        num = num % 54  // Changed to 0..53

        /*
        if num == first && i != 0{  // We can stop the loop to avoid
repeating the pattern
            arr = arr[:i]
            break
        }
```

```
        */

        arr[i] = num
        seed = num  // Seed is now the previous number
    }

    //fmt.Println("Resultado: ", arr)

    return arr
}
```

Resultados obtenidos

En el experimento a) se generan los siguientes arreglos:

n = 200

```
A:  [49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47
31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7
23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19
47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13
35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43
41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1
11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49
53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53]
```

n = 400

```
A:  [49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47
31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7
23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19
47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13
35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43
41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1
11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49
53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25
5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29
49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17
25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37
```

```
29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31
17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23
37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47
31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7
23 37 29 49 53 43 41]
```

n = 600

```
A:  [49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47
31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7
23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19
47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13
35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43
41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1
11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49
53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25
5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29
49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17
25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37
29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31
17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23
37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47
31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7
23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19
47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13
35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43
41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1
11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49
53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25
5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29
49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47]
```

n = 800

```
A:  [49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47
31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7
23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19
47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13
35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43
41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1
```

```
11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49
53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25
5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29
49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17
25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37
29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31
17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23
37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47
31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7
23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19
47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13
35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43
41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1
11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49
53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25
5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29
49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17
25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37
29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31
17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23
37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47
31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7
23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19
47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13
35 7 23 37 29 49 53 43 41 19 47 31 17]
```

n = 1000

```
A:  [49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47
31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7
23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19
47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13
35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43
41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1
11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49
53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25
5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29
49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17
25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37
29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31
```

```
17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23
37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47
31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7
23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19
47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13
35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43
41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1
11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49
53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25
5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29
49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17
25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37
29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31
17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23
37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47
31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7
23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19
47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13
35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43
41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1
11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49
53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25
5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29
49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17
25 5 1 11 13 35 7 23 37 29 49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37
29 49 53 43 41 19 47 31 17 25 5]
```

Tomar en cuenta que como siempre se utilizaron los mismos valores de variables (semilla = 29, k = 1 y m = 2048), los arreglos siempre son el mismo con la diferencia del tamaño. Además, esta combinación resulta en un patrón de aproximadamente 30 valores donde se repiten hasta alcanzar el tamaño “n”.

Referencias consultadas

("Generador lineal congruencial - Wikipedia, la enciclopedia libre", n.d.)
(García, 2017)

Generación del gráfico de barras para un arreglo

Estrategia desarrollada

Para generar un gráfico de barras en Go, se utilizó la librería “go-chart”. En esta, se construye un gráfico con diferentes datos del mismo, por ejemplo el tipo de gráfico, título y valores. En este caso, se utiliza el tipo de gráfico “BarChart” el cual representa los datos como líneas verticales, como es solicitado. Además, se le agregó información adicional: el título, el fondo de la imagen, la altura de la imagen y el ancho de las barras.

Con respecto a los valores de las barras, se utilizó una función llamada “createValues” que convierte un arreglo de “int” a un arreglo de “Values”, tipo de estructura usado por la librería. Por último, el resultado es guardado en una imagen llamada “resultado.png” la cual se crea dentro del folder del proyecto cuando se corre.

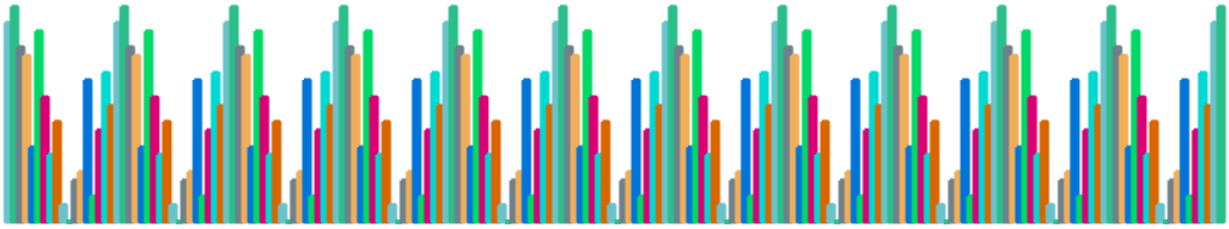
Código

```
func ArrChart(arr []int) {

    arrVal := createValues(arr)

    graph := chart.BarChart{
        Title : "Gráfico de Barras",
        Background: chart.Style{
            Padding: chart.Box{
                Top: 30,
            },
        },
        Height : 256,
        BarWidth: 50,

        Bars: arrVal,
    }
    f , _ := os.Create("resultado.png")
    defer f.Close()
    graph.Render(chart.PNG,f)
}

func createValues(arr []int) []chart.Value{ // Creates a Value array from
a normal array
    res := []chart.Value{}
```

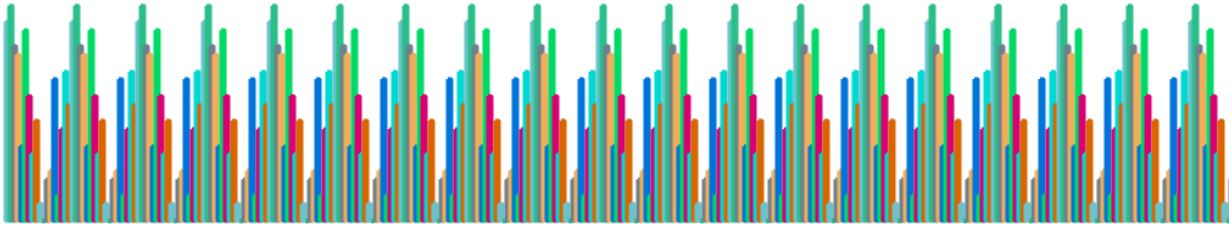
```
    val := chart.Value{}

    for i := 0; i < len(arr); i++{
        val.Value = float64(arr[i])
        res = append(res, val)
    }
    return res
}
```

Resultados obtenidos

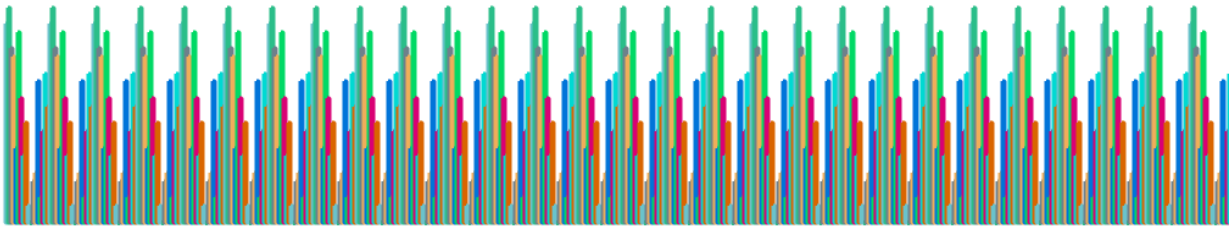
n = 200



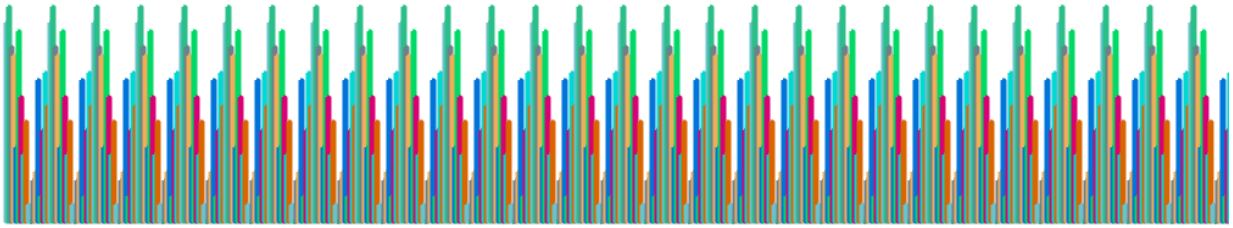
n = 400



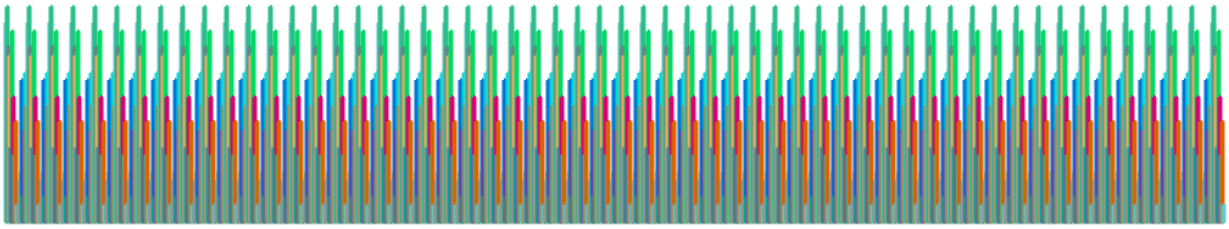
n = 600



n = 800



n = 1000



Referencias consultadas

(wcharczuk, 2016)

Representación diseñada para los árboles binarios de búsqueda ordinarios

Descripción sucinta

Para representar un árbol binario se crearon 2 structs:

- Node: cada nodo individual del árbol
- BSTree (Binary Search Tree): el árbol que contiene un puntero al nodo raíz y un int que es utilizado para guardar el número de iteraciones realizadas dependiendo del método utilizado (cambia por cada uso de los métodos).

Código

```
type Node struct {
    key int
    Left *Node
    Right *Node
}

type BSTree struct {
    Root *Node
    count int
}

func (n *Node) print() {
    if n != nil {
        fmt.Println(n.key)
    } else {
        fmt.Println("Nil Node")
    }
}
```

Adicionalmente también se incluye el método de print para *Node que fue utilizado para verificar las pruebas y experimentos realizados.

Ítem 3

Descripción sucinta

insertVal recibe como parámetros un puntero a un arreglo/slice de números enteros y la llave a insertar. Esta consiste en un for que recorre el arreglo por cada elemento, consiguiendo su valor e índice. Si encuentra la llave retorna el índice + 1 de donde fue encontrado (cantidad de comparaciones hechas). En el caso contrario inserta la llave y retorna la longitud del arreglo antes de esta inserción (debido a que lo recorrió por completo).

Código

```
func InsertVal(arr *[]int, key int) int {
    for index, value := range *arr {
        if value == key {

            return index + 1 // Found, returns number of comparisons done
[index + 1 because it starts at 0]
        }
    }; len := len(*arr)

    *arr = append(*arr, key)

    return len // Not found, checked all the values in the original array/slice
and because it wasn't, it is the length of it.
}
```

Resultados obtenidos en experimentos

n = 200

Experimento c: Se realizó el slice TS y guardar las estadísticas para el experimento j

```
TS:  [49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29]
```

Experimento j: Cantidad total de comparaciones realizadas en las inserciones sobre en TS: 1866

n = 400

Experimento c: Se realizó el slice TS y guardar las estadísticas para el experimento j

```
TS:  [49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29]
```

Experimento j: Cantidad total de comparaciones realizadas en las inserciones sobre en TS: 3754

n = 600

Experimento c: Se realizó el slice TS y guardar las estadísticas para el experimento j

```
TS:  [49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29]
```

Experimento j: Cantidad total de comparaciones realizadas en las inserciones sobre en TS: 5646

n = 800

Experimento c: Se realizó el slice TS y guardar las estadísticas para el experimento j

```
TS:  [49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29]
```

Experimento j: Cantidad total de comparaciones realizadas en las inserciones sobre en TS: 7542

n = 1000

Experimento c: Se realizó el slice TS y guardar las estadísticas para el experimento j

```
TS:  [49 53 43 41 19 47 31 17 25 5 1 11 13 35 7 23 37 29]
```

Experimento j: Cantidad total de comparaciones realizadas en las inserciones sobre en TS: 9442

Referencias consultadas

("What is the for-range loop in Golang?", n.d.)
(Wright, 2018)

Ítem 4

Descripción sucinta

La función selectionSort prácticamente consiste en que por cada índice del arreglo/slice va buscando el menor valor posible de los elementos siguientes y este se cambia de lugar con el valor original, esto se repite con cada posición hasta que quede ordenado.

Código

```
func SelectionSort(arr *[]int) {
    arr2 := *arr
    len := len(arr2)

     for currentIndex := 0; currentIndex < len-1; currentIndex++ { // Done to
all the indexes in the array
        indexMin := currentIndex

          for i := currentIndex + 1; i < len; i++ { // Get the index of the
smallest value from the numbers to the right
            if arr2[i] < arr2[indexMin] {
                indexMin = i
            }
        }

        //Swap numbers
        arr2[currentIndex], arr2[indexMin] = arr2[indexMin], arr2[currentIndex]
    }

    *arr = arr2 // Assign changes to original array
}
```

Resultados obtenidos en experimentos

n = 200

Experimento d: Se ordenan los elementos correctamente por medio del Selection Sort.

```
TOS:  [1 1 1 1 1 1 1 1 1 1 1 5 5 5 5 5 5 5 5 5 5 5 7 7 7 7 7 7 7 7 7 7 7
11 11 11 11 11 11 11 11 11 11 11 13 13 13 13 13 13 13 13 13 13 13 17 17 17
17 17 17 17 17 17 17 17 19 19 19 19 19 19 19 19 19 19 19 23 23 23 23 23 23
23 23 23 23 23 25 25 25 25 25 25 25 25 25 25 25 29 29 29 29 29 29 29 29 29
```

```
29 29 31 31 31 31 31 31 31 31 31 31 31 35 35 35 35 35 35 35 35 35 35 35 37
37 37 37 37 37 37 37 37 37 37 41 41 41 41 41 41 41 41 41 41 43 43 43 43
43 43 43 43 43 43 43 47 47 47 47 47 47 47 47 47 47 47 49 49 49 49 49 49 49
49 49 49 49 49 53 53 53 53 53 53 53 53 53 53 53 53]
```

n = 400

Experimento d: Se ordenan los elementos correctamente por medio del Selection Sort.

```
TOS:  [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 11 11 11
11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 13 13 13 13 13 13
13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 17 17 17 17 17 17 17 17 17
17 17 17 17 17 17 17 17 17 17 17 17 19 19 19 19 19 19 19 19 19 19 19 19
19 19 19 19 19 19 19 19 19 19 23 23 23 23 23 23 23 23 23 23 23 23 23 23
23 23 23 23 23 23 23 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29
29 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 35 35
35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 37 37 37 37 37
37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41 41 41 41 41 41 41 43 43 43 43 43 43 43 43 43 43
43 43 43 43 43 43 43 43 43 43 43 43 47 47 47 47 47 47 47 47 47 47 47 47
47 47 47 47 47 47 47 47 47 47 49 49 49 49 49 49 49 49 49 49 49 49 49 49
49 49 49 49 49 49 49 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53
53 53 53 53 53 53]
```

n = 600

Experimento d: Se ordenan los elementos correctamente por medio del Selection Sort.

```
TOS:  [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 11 11 11 11 11 11
11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11
11 11 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13
13 13 13 13 13 13 13 13 13 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17
17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 19 19 19 19 19 19 19
19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19
19 19 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23
```

```
23 23 23 23 23 23 23 23 23 23 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 29 29 29 29 29 29 29
29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29
29 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
31 31 31 31 31 31 31 31 31 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35
35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 37 37 37 37 37 37 37 37 37
37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41 41 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43
43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 47 47 47 47 47 47 47 47 47
47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47
47 47 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49
49 49 49 49 49 49 49 49 49 49 49 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53
53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53]
```

n = 800

Experimento d: Se ordenan los elementos correctamente por medio del Selection Sort.

```
TOS:  [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 11 11 11 11 11 11 11 11 11
11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11
11 11 11 11 11 11 11 11 11 11 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13
13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13
13 13 13 13 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17
17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 19
19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19
19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 23 23 23 23 23 23 23 23
23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23
23 23 23 23 23 23 23 23 23 23 23 23 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25 25 25 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29
29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29
29 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 35 35 35 35 35 35
35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35
35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 37 37 37 37 37 37 37 37 37 37 37
```

```
37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37
37 37 37 37 37 37 37 37 37 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
41 41 41 41 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43
43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 47
47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47
47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 49 49 49 49 49 49 49 49
49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49
49 49 49 49 49 49 49 49 49 49 49 49 49 53 53 53 53 53 53 53 53 53 53 53 53
53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53
53 53 53 53 53 53 53 53 53]
```

n = 1000

Experimento d: Se ordenan los elementos correctamente por medio del Selection Sort.

```
TOS:  [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 11 11 11 11 11 11 11 11 11 11 11
11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11
11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 13 13 13 13 13 13
13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13
13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 17
17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17
17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17
17 17 17 17 17 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19
19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19
19 19 19 19 19 19 19 19 19 19 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23
23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23
23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 29 29 29
29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29
29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29
29 29 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
31 31 31 31 31 31 31 31 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35
```

```
35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35
35 35 35 35 35 35 35 35 35 35 35 35 35 37 37 37 37 37 37 37 37 37 37 37 37
37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37
37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 43
43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43
43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43
43 43 43 43 43 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47
47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47
47 47 47 47 47 47 47 47 47 47 47 49 49 49 49 49 49 49 49 49 49 49 49 49 49
49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49
49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 53 53 53 53 53 53 53 53
53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53
53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53]
```

Referencias consultadas

("Selection Sort (With Code)", n.d.)

(Wright, 2018)

Ítem 5

Descripción sucinta

3 funciones: partition que consigue el pivote a utilizar para ir dividiendo y haciendo cambios en el arreglo/slice (valores menores al pivote a la izquierda y mayores a la derecha), quicksort que es recursiva y utiliza partitions para el pivote, y quisortCall que funciona como auxiliar de quicksort asignando los valores necesarios y para que el usuario solo ingrese como parámetro el arreglo.

Código

```
func Partition(arr *[]int, low int, high int) int { //
    arrC := *arr
    pivot := arrC[high]

    i := low - 1

    for j := low; j < high; j++ {
        if arrC[j] <= pivot {
            i++
             arrC[i], arrC[j] = arrC[j], arrC[i] //Gets the lesser values to the
left of the pivot
        }
    }

    //Swap pivot with the next element to i
    arrC[i+1], arrC[high] = arrC[high], arrC[i+1]

    *arr = arrC // Assign changes to original array

    return i + 1 //new pivot
}

func Quicksort(arr *[]int, low int, high int) {
     if low < high { //Divides into subarrays by selecting a pivot and organizes
them
        pivot := Partition(arr, low, high)
        Quicksort(arr, low, pivot-1)
        Quicksort(arr, pivot+1, high)
    }
}
```

```
func QuicksortCall(arr *[]int) {
    Quicksort(arr, 0, len(*arr)-1)
}
```

Resultados obtenidos en experimentos

n = 200

Experimento e: Se ordenan los elementos correctamente por medio del Quicksort.

```
TOQ:  [1 1 1 1 1 1 1 1 1 1 1 5 5 5 5 5 5 5 5 5 5 5 7 7 7 7 7 7 7 7 7 7 7
11 11 11 11 11 11 11 11 11 11 11 13 13 13 13 13 13 13 13 13 13 13 17 17 17
17 17 17 17 17 17 17 17 19 19 19 19 19 19 19 19 19 19 19 23 23 23 23 23 23
23 23 23 23 23 23 25 25 25 25 25 25 25 25 25 25 25 29 29 29 29 29 29 29 29 29
29 29 31 31 31 31 31 31 31 31 31 31 31 35 35 35 35 35 35 35 35 35 35 35 37
37 37 37 37 37 37 37 37 37 37 41 41 41 41 41 41 41 41 41 41 41 43 43 43 43
43 43 43 43 43 43 43 47 47 47 47 47 47 47 47 47 47 47 49 49 49 49 49 49 49
49 49 49 49 49 53 53 53 53 53 53 53 53 53 53 53 53]
```

n = 400

Experimento e: Se ordenan los elementos correctamente por medio del Quicksort.

```
TOQ:  [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 11 11 11
11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 13 13 13 13 13 13
13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 17 17 17 17 17 17 17 17 17
17 17 17 17 17 17 17 17 17 17 17 19 19 19 19 19 19 19 19 19 19 19 19
19 19 19 19 19 19 19 19 19 23 23 23 23 23 23 23 23 23 23 23 23 23 23
23 23 23 23 23 23 23 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29
29 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 35 35
35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 37 37 37 37 37
37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41 41 41 41 41 43 43 43 43 43 43 43 43 43 43
43 43 43 43 43 43 43 43 43 43 43 43 47 47 47 47 47 47 47 47 47 47 47
47 47 47 47 47 47 47 47 47 47 49 49 49 49 49 49 49 49 49 49 49 49 49
49 49 49 49 49 49 49 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53
53 53 53 53 53 53]
```

n = 600

Experimento e: Se ordenan los elementos correctamente por medio del Quicksort.

```
TOQ:  [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 11 11 11 11 11 11
11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11
11 11 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13
13 13 13 13 13 13 13 13 13 13 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17
17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 19 19 19 19 19 19 19
19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19
19 19 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23
23 23 23 23 23 23 23 23 23 23 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 29 29 29 29 29 29 29
29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29
29 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
31 31 31 31 31 31 31 31 31 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35
35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 37 37 37 37 37 37 37 37
37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41 41 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43
43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 47 47 47 47 47 47 47
47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47
47 47 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49
49 49 49 49 49 49 49 49 49 49 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53
53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53]
```

n = 800

Experimento e: Se ordenan los elementos correctamente por medio del Quicksort.

```
TOQ:  [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 11 11 11 11 11 11 11 11
11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11
11 11 11 11 11 11 11 11 11 11 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13
13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13
13 13 13 13 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17
```

```
17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 19
19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19
19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 23 23 23 23 23 23 23
23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23
23 23 23 23 23 23 23 23 23 23 23 23 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25 25 25 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29
29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29
29 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 35 35 35 35
35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35
35 35 35 35 35 35 35 35 35 35 35 35 35 35 37 37 37 37 37 37 37 37 37 37
37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37
37 37 37 37 37 37 37 37 37 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
41 41 41 41 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43
43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 47
47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47
47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 49 49 49 49 49 49
49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49
49 49 49 49 49 49 49 49 49 49 49 49 49 53 53 53 53 53 53 53 53 53 53 53
53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53
53 53 53 53 53 53 53 53 53]
```

n = 1000

Experimento e: Se ordenan los elementos correctamente por medio del Quicksort.

```
TOQ:  [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 11 11 11 11 11 11 11 11 11 11 11
11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11
11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 13 13 13 13 13 13
13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13
13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 17
17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17
17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17
```

```
17 17 17 17 17 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19
19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19
19 19 19 19 19 19 19 19 19 19 19 23 23 23 23 23 23 23 23 23 23 23 23 23 23
23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23
23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 25 25 25 25 25 25 25 25 25
25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 29 29 29
29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29
29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29
29 29 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
31 31 31 31 31 31 31 31 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35
35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35
35 35 35 35 35 35 35 35 35 35 35 35 35 37 37 37 37 37 37 37 37 37 37 37 37
37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37
37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 43
43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43
43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43
43 43 43 43 43 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47
47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47
47 47 47 47 47 47 47 47 47 47 47 49 49 49 49 49 49 49 49 49 49 49 49 49 49
49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49
49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 53 53 53 53 53 53 53 53
53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53
53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53]
```

Referencias consultadas

("QuickSort (With Code)", n.d.)

Ítem 6

Descripción sucinta

LinearSearch usa un for para ir recorriendo linealmente un arreglo/slice y buscar la llave, si la encuentra retorna true y el índice + 1 (cantidad de comparaciones hechas). En el caso contrario retorna false y la longitud del arreglo (debido a que lo recorrió por completo).

Código

```
func LinearSearch(arr []int, key int) (bool, int) {
    for index, value := range arr {
        if value == key {
              return true, index + 1 // Found, index of where it was + 1
[because it starts at 0]
        }
    }
    return false, len(arr) // Not Found, the whole length of the array
}
```

Resultados obtenidos en experimentos

n ∈ {200, 400, 600, 800, 1000}

Experimento i: Se buscó cada valor de v generado, se muestran las comparaciones en el experimento j.

Experimento j: Cantidad total de comparaciones realizadas en las búsquedas sobre TS: 94960

*En todos los experimentos con distintos n, el valor del experimento j siempre se mantiene igual.

Referencias consultadas

No se consultaron referencias online.

Ítem 7

Descripción sucinta

BinarySearchCall es un auxiliar de BinarySearch al asignar los valores necesarios. BinarySearch es una función recursiva que consiste en conseguir el índice intermedio entre los índices de left y right ingresados (hasta que left sea mayor que right, condición de parada) y revisar si el valor con este índice calculado es el buscado. Si el valor conseguido es mayor a la llave, en la siguiente iteración se tiene que disminuir el índice de right al índice intermedio - 1. En el caso contrario, viceversa. Retorna un bool indicando si lo encontró, y con cada iteración se incrementa el valor de n (comparaciones realizadas).

Código

```
func BinarySearch(arr []int, key int, left int, right int, n int) (bool, int) {
    if left <= right {
        mid := (left + right) / 2 // Mid value between left & right

           //fmt.Println("Mid: ", arr[mid], " Left: ", arr[left], " Right: ",
arr[right])

        if arr[mid] == key { // Found
            return true, n
        } else if arr[mid] > key {
            return BinarySearch(arr, key, left, mid-1, n+1) // Key is lesser
        } else {
            return BinarySearch(arr, key, mid+1, right, n+1) // Key is greater
        }
    }
    return false, n // Not found
}

func BinarySearchCall(arr []int, key int) (bool, int) {
    return BinarySearch(arr, key, 0, len(arr)-1, 1)
}
```

Resultados obtenidos en experimentos

n ∈ {200, 400, 600, 800, 1000}

Experimento i: Se buscó cada valor de v generado, se muestran las comparaciones en el experimento j con los slices TOS y TOQ.

Experimento j:

- Cantidad total de comparaciones realizadas en las búsquedas sobre TOS: 35558
- Cantidad total de comparaciones realizadas en las búsquedas sobre TOQ: 35558

*En todos los experimentos con distintos n, el valor del experimento j siempre se mantiene igual.

Referencias consultadas

("Binary Search - GeeksforGeeks", 2021)

Ítem 9

Descripción sucinta

El método de Insert es utilizado como auxiliar de InsertNode debido a que solo recibe la llave a insertar y prepara los valores utilizados para insertNode. insertNode es recursiva y con cada iteración realizada se incrementa en 1 b.count (atributo de BSTree mencionado anteriormente). Se mueve por los nodos del árbol y si el nodo no existe o encuentra el nodo (condiciones de parada) respectivamente crea un nuevo nodo que es asignado o simplemente solo lo retorna.

Código

```
func (b *BSTree) InsertNode(value int, node *Node) *Node{
    b.count++ //New iteration

    if node == nil {
        node = &Node{key: value}
        return node
    } else if node.key == value {
        return node
    } else if value < node.key {
        node.Left = b.InsertNode(value, node.Left)
    } else {
        node.Right = b.InsertNode(value, node.Right)
    }; return node
}

func (b *BSTree) Insert(key int) int{
    b.count = 0
    b.Root = b.InsertNode(key, b.Root)
    return b.count
}
```

Método extra utilizado para calcular altura maxima en experimentos:

```
func (b *BSTree) MaxHeight(n *Node) int{
    if n == nil{
        return -1
    } else {
        return 1 + Max(b.MaxHeight(n.Left), b.MaxHeight(n.Right))
    }
```

```
}

func Max(a int, b int) int{
    if a > b {
        return a
    }; return b
}
```

Resultados obtenidos en experimentos

n = 200

Experimento f: Se creó el árbol Abb y se insertaron los valores correctamente

Experimento j:

- Altura (máxima) del árbol Abb: 17
- Densidad del árbol: Misma que la altura debido a que los nodos del árbol Abb solo tienen hijos derechos.
- Cantidad total de comparaciones realizadas en las inserciones sobre Abb: 1916

n = 400

Experimento f: Se creó el árbol Abb y se insertaron los valores correctamente

Experimento j:

- Altura (máxima) del árbol Abb: 17
- Densidad del árbol: Misma que la altura debido a que los nodos del árbol Abb solo tienen hijos derechos.
- Cantidad total de comparaciones realizadas en las inserciones sobre Abb: 3826

n = 600

Experimento f: Se creó el árbol Abb y se insertaron los valores correctamente

Experimento j:

- Altura (máxima) del árbol Abb: 17
- Densidad del árbol: Misma que la altura debido a que los nodos del árbol Abb solo tienen hijos derechos.
- Cantidad total de comparaciones realizadas en las inserciones sobre Abb: 5730

n = 800

Experimento f: Se creó el árbol Abb y se insertaron los valores correctamente

Experimento j:

- Altura (máxima) del árbol Abb: 17
- Densidad del árbol: Misma que la altura debido a que los nodos del árbol Abb solo tienen hijos derechos.
- Cantidad total de comparaciones realizadas en las inserciones sobre Abb: 7628

n = 1000

Experimento f: Se creó el árbol Abb y se insertaron los valores correctamente

Experimento j:

- Altura (máxima) del árbol Abb: 17
- Densidad del árbol: Misma que la altura debido a que los nodos del árbol Abb solo tienen hijos derechos.
- Cantidad total de comparaciones realizadas en las inserciones sobre Abb: 9520

Referencias consultadas

("Binary Search Tree | Set 1 (Search and Insertion) - GeeksforGeeks", 2021)

Ítem 10

Descripción sucinta

El método de search es utilizado como auxiliar de searchNode debido a que solo recibe la llave a buscar y preparar los valores utilizados para searchNode. searchNode es recursiva y con cada iteración realizada se incrementa en 1 b.count. Esta recorre el árbol de igual manera que insertNode, y retorna true si encuentra el valor y false en el caso contrario.

Código

```
func (b *BSTree) SearchNode(value int, node *Node) bool {
    b.count++

    if node == nil {
        return false
    } else if node.key == value {
        return true
    } else if value < node.key {
        b.SearchNode(value, node.Left)
    } else {
        b.SearchNode(value, node.Right)
    }; return false
}

func (b *BSTree) Search(key int) (bool, int){
    b.count = 0
    return b.SearchNode(key, b.Root), b.count
}
```

Resultados obtenidos en experimentos

n ∈ {200, 400, 600, 800, 1000}

Experimento i: Se buscó cada valor de v generado, se muestran las comparaciones en el experimento j.

Experimento j: Cantidad total de comparaciones realizadas en las búsquedas sobre Abb: 95020

*En todos los experimentos con distintos n, el valor del experimento j siempre se mantiene igual.

Referencias consultadas

("Binary Search Tree | Set 1 (Search and Insertion) - GeeksforGeeks", 2021)

Análisis de resultados

Se puede observar en su totalidad los experimentos realizados en el archivo experimentos.go, con esto en mente se tiene el siguiente análisis de las estadísticas conseguidas en el experimento j:

Inserción	200	400	600	800	1000
TS	1866	3754	5646	7542	9442
Abb	1916	3826	5730	7628	9520

Búsqueda	200	400	600	800	1000
TS	94960	94960	94960	94960	94960
TOS	35558	35558	35558	35558	35558
TOQ	35558	35558	35558	35558	35558
Abb	95020	95020	95020	95020	95020

Según los datos de la tabla de inserciones, se puede ver que las comparaciones incrementan junto con el incremento del tamaño del arreglo utilizado. Esto se debe a que, al haber más datos, habrá que hacer más inserciones y comparaciones de todos los datos.

En el caso de la búsqueda, se puede observar que las cantidad de comparaciones se mantiene igual a pesar del tamaño n de los experimentos, esto es debido a que al utilizar los distintos métodos de búsqueda con arreglos ordenados, como es el caso de binary search, con los distintos arreglos sin importar que tan grande sean los arreglos. Además, podemos concluir que los métodos más efectivos son los utilizados en TOS y TOQ por igual, ya que dan la menor cantidad de búsquedas.

Reflexión y Conclusiones

Por medio de los distintos ejercicios realizados, los estudiantes aprendieron de manera autodirigida como programar en el lenguaje de Go y sus características imperativas y secuenciales que este posee. Ambos estudiantes pueden afirmar que esta fue la primera vez que trabajaban con este lenguaje y que lo utilizaron de manera eficiente así como la librería utilizada para la generación de gráficos.

Hablando de la experiencia de haber programado en este nuevo lenguaje, el principal problema que los estudiantes tuvieron fue la importación de librerías de github. Este fue el mayor contratiempo que se presentó debido a que inicialmente se había intentado importar por medio del comando de go get pero no funciono, pero se soluciono descargando la librería en sí y aplicando los comandos necesarios. Un dato que les pareció interesante a los estudiantes fue el hecho de que para utilizar funciones en múltiples archivos, estas tienen que comenzar con mayúscula.

Adicionalmente se repasaron conceptos y estructuras claves como los distintos tipos de ordenamiento, inserciones y búsqueda de valores en arreglos/slices, la representación de un árbol binario de búsqueda y sus métodos relacionados, entre otros.

Referencias

Binary Search - GeeksforGeeks. GeeksforGeeks. (2021). Retrieved 14 October 2021, from https://www.geeksforgeeks.org/binary-search/.

Binary Search Tree | Set 1 (Search and Insertion) - GeeksforGeeks. GeeksforGeeks. (2021). Retrieved 16 October 2021, from https://www.geeksforgeeks.org/binary-search-tree-set-1-search-and-insertion/.

García, G. (2017). *Algoritmo Congruencial Multiplicativo para la generación de números pseudo aleatorios: Implementación en Java -.* Retrieved 16 October 2021, from https://naps.com.mx/blog/algoritmo-congruencial-multiplicativo-para-la-generacion-de-numeros-pseudo-aleatorios-implementacion-en-java/

Generador lineal congruencial - Wikipedia, la enciclopedia libre. Es.wikipedia.org. (2021). Retrieved 16 October 2021, from https://es.wikipedia.org/wiki/Generador_lineal_congruencial.

QuickSort (With Code). Programiz.com. (n. d.). Retrieved 13 October 2021, from https://www.programiz.com/dsa/quick-sort.

Selection Sort (With Code). Programiz.com. (n.d.). Retrieved 13 October 2021, from https://www.programiz.com/dsa/selection-sort.

wcharczuk. (2016). *GitHub - wcharczuk/go-chart: go chart is a basic charting library in native golang.*. GitHub. Retrieved 15 October 2021, from https://github.com/wcharczuk/go-chart.

What is the for-range loop in Golang?. Educative: Interactive Courses for Software Developers. (n.d.). Retrieved 13 October 2021, from https://www.educative.io/edpresso/what-is-the-for-range-loop-in-golang.

Wright, J. (2018). *Learn Go in 12 Minutes* [Video]. Retrieved 12 October 2021, from https://www.youtube.com/watch?v=C8LgvuEBraI&t=14s.