Instituto Tecnológico de Costa Rica
Escuela de Ingeniería en Computación
IC-4700 Lenguajes de programación
Asignación #2 [Proyecto]: *Aprender características imperativas y secuenciales del lenguaje Go*

Historial de revisiones:
- 2021.09.22,28,30: Versión base incompleta (v0).
- 2021.10.01: Versión base (v1).

Lea con cuidado este documento. Si encuentra errores en el planteamiento[1], por favor comuníqueselos inmediatamente al profesor.

Objetivo
Al concluir esta asignación, Ud. estará familiarizado con las principales características de programación *secuencial* del lenguaje **Go** (Golang). El aprendizaje será auto-dirigido.

Bases
- Libros y documentos sobre el lenguaje de programación **Go** (Golang).
- Artículos y libros referentes a algoritmos eficientes para el manejo de árboles binarios de búsqueda y generación de número pseudo-aleatorios.

Desarrollo

Funciones por desarrollar

Cada grupo de estudiantes trabajará los aspectos indicados a continuación, conforme la distribución indicada más abajo:

1. Diseñar y construir una función que **genere** un *arreglo* de tamaño ***n*** con números enteros pseudo-aleatorios obtenidos mediante el método de *congruencia lineal multiplicativa*, a partir de una *semilla* dada. La semilla deberá ser un número *primo* entre 11 y 101. Los valores generados deben ser convertidos al intervalo 0 .. 53. El período debe ser $\geqslant 2048$. ***n*** puede ser cualquier número en el intervalo 200 .. 1000.
2. Diseñar y construir una función que genere un gráfico de barras a partir de un arreglo de *n* números enteros, cada uno de los cuales pertenece al intervalo 0 .. 53. Las barras deben ser verticales y de tamaño directamente proporcional al número que representan.
3. Diseñar y construir una función que haga la **inserción** de un valor entero (la *llave*) en un arreglo de números enteros. Si la *llave* ***no*** se encuentra en el arreglo, inserta la llave al final del arreglo. Si la *llave* ***ya*** se encuentra en el arreglo, no la inserta. La función retorna un número entero, que será la cantidad de comparaciones realizadas, incluida la que llevó a la inserción.
4. Diseñar y construir una función que ordene ascendentemente un arreglo de enteros mediante el método de ordenamiento de *selección*. Este algoritmo es cuadrático en promedio.
5. Diseñar y construir una función que ordene ascendentemente un arreglo de enteros mediante el método de ordenamiento conocido como *Quicksort*. Este algoritmo es $O(n \log_2 (n))$ en promedio.
6. Diseñar y construir una función que haga la **búsqueda** de un valor entero (la *llave*) en un arreglo que ***no esté ordenado***, mediante el algoritmo de **búsqueda secuencial**. La función debe retornar un par ordenado: un valor booleano que indique si encontró la llave buscada (**true** = la encontró, **false** = no la encontró), junto con un entero que indique la cantidad de comparaciones realizadas hasta determinar si la llave está o no presente en el arreglo.
7. Diseñar y construir una función que haga la **búsqueda** de un valor entero (la *llave*) en un arreglo que ***esté ordenado***, mediante el algoritmo de **búsqueda binaria**. La función debe retornar un par ordenado: un valor booleano que indique si encontró la llave buscada (**true** = la encontró, **false** = no la encontró), junto con un entero que indique la cantidad de comparaciones realizadas hasta determinar si la llave está o no presente en el arreglo.

[1] El profesor es un ser humano, falible como cualquiera.

8. Diseñar una **representación** en Go para los árboles binarios de búsqueda *ordinarios*, donde cada nodo almacene un número entero.
9. Diseñar y construir una función que haga la **inserción** de un valor entero (la *llave*) en un árbol binario de búsqueda *ordinario*. Si la *llave* ***no*** se encuentra en el árbol, crea un nuevo nodo con la llave. Si la *llave* ***ya*** se encuentra en el árbol, no la inserta. La función retorna un número entero, que será la cantidad de comparaciones realizadas, incluida la que llevó a la inserción.
10. Diseñar y construir una función que haga la **búsqueda** de un valor entero (la *llave*) en un árbol binario de búsqueda *ordinario*, sin hacer inserciones. La función debe retornar un par ordenado: un valor booleano que indique si encontró la llave buscada (**true** = la encontró, **false** = no la encontró), junto con un entero que indique la cantidad de comparaciones realizadas hasta determinar si la llave está o no presente en el árbol.
11. Diseñar y construir una función que **transforme** un árbol binario de búsqueda *ordinario* en un árbol binario de búsqueda *quasi-completo*, conforme lo logra el algoritmo *DSW*. Un árbol de altura (o profundidad) *a* es quasi-completo cuando los nodos más profundos se encuentran todos en los niveles *a* o *a – 1*. Base su función en el algoritmo DSW (Day-Stout-Warren); ver referencias al final y en el tecDigital.

Distribución

Considere los ítemes de la lista anterior.

- Si el grupo es de 1 o 2 miembros: desarrollar en Go los ítems 1 al 10.
- Si el grupo es de 3 miembros: desarrollar en Go los ítems 1 al 11.

Experimentos

- Con su función para generar número pseudo-aleatorios (ítem 1), realizar un experimento *para cada uno* de valores de $\boldsymbol{n} \in \{200, 400, 600, 800, 1000\}$.
  - **a)** Crear un arreglo *A* de tamaño ***n***, cuyos elementos son números pseudo-aleatorios generados por la función que resuelve el ítem 1 (rango de valores 0 .. 53). Los números pseudo-aleatorios deben ser guardados en *A* en el orden que los fue generando su función. Puede usar las *tajadas* (*slices*) de Go para esto.
  - **b)** Crear un arreglo *Distr* de enteros, con índices en el rango 0 .. 53. Inicialice todos los elementos del arreglo en 0. Luego, recorra el arreglo *A*, y cada vez que aparezca un número $k \in \{ i : \mathbb{N} \mid 0 \leqslant i \leqslant 53\}$, aumente en 1 el contador del índice *k* en el arreglo *Distr*.
  - **c)** Crear una *tajada* (*slice*) de ***n*** elementos enteros; llamémosla *TS*. *Insertar* los elementos de *A*, uno a uno, mediante su algoritmo de inserción en un arreglo (ítem 3). Recoger estadísticas.
  - **d)** Crear una *tajada* (*slice*) con una copia de los ***n*** elementos del arreglo *A*; llamémosla *TOS*. Ordenar *TOS* usando su implementación del algoritmo de ordenamiento por *selección* (ítem 4).
  - **e)** Crear una *tajada* (*slice*) con una copia de los ***n*** elementos del arreglo *A*; llamémosla *TOQ*. Ordenar *TOQ* usando su implementación del algoritmo de ordenamiento *Quicksort* (ítem 5).
  - **f)** Crear *un* árbol binario de búsqueda vacío, *Abb*, congruente con la representación diseñada por su grupo (ítem 8). Sobre ese árbol, *insertar todos* los ***n*** elementos del arreglo *A*, conforme a las indicaciones dadas para el ítem 9. Recoger estadísticas.
  - **g)** En relación con el método DSW (ítem 11): crear *un* árbol binario de búsqueda vacío, *AbbDSW*. Seguidamente, *insertar todos* los ***n*** elementos del arreglo *A*, conforme a las indicaciones dadas para el ítem 9. Después de poblar el árbol *AbbDSW* con esos datos, someterlo a la transformación DSW (ítem 11).
  - **h)** Grafique el arreglo *Distr* (ítem 2).
  - **i)** Mediante la función generadora de número pseudo-aleatorios creada como parte de su solución al resolver el ítem 1: generar una secuencia de 10000 números enteros pseudo-aleatorios (en el rango 0 .. 53). Para *cada* valor ***v*** generado, *buscarlo* en las estructuras creadas en los pasos **c)** a **g)**, de esta manera:
    - Buscar ***v*** en *TS* (método del ítem 6). Recoger estadísticas.
    - Buscar ***v*** en *TOS* (método del ítem 7). Recoger estadísticas.
    - Buscar ***v*** en *TOQ* (método del ítem 7). Recoger estadísticas.
    - Buscar ***v*** en *Abb* (método del ítem 10). Recoger estadísticas.
    - Buscar ***v*** en *AbbDSW* (método del ítem 11). Recoger estadísticas.
  - **j)** Al finalizar, obtener estas estadísticas para el punto **i)**, conforme a los alcances correspondientes a su grupo:
    - Altura (máxima) del árbol, para cada uno de los árboles *Abb* y *AbbDSW*.

- Densidad del árbol[2], para cada uno de los árboles *Abb* y *AbbDSW*..
- Cantidad total de comparaciones realizadas en las *inserciones* sobre *TS*, *Abb*.
- Cantidad total de comparaciones realizadas en las *búsquedas* sobre *TS*, *TOS*, *TOQ*, *Abb*, *AbbDSW*.

Informe técnico
Deberán preparar un informe técnico que incluya:

- Portada que identifique a los autores del informe, con sus carnets.
- Introducción al informe.
- Secciones:
  - Generación de número pseudo-aleatorios (ítem 1).
    - Estrategia desarrollada.
    - Código.
    - Resultados obtenidos.
    - Referencias consultadas.
  - Generación del gráfico de barras para un arreglo (ítem 2).
    - Estrategia desarrollada.
    - Código.
    - Resultados obtenidos.
    - Referencias consultadas.
  - Representación diseñada para los árboles binarios de búsqueda ordinarios (ítem 8).
    - Código.

  {Ítems 3 al 7, 9 al 11}
  - *Ítem[k]*, $k \in \{ 3, 4, 5, 6, 7, 9, 10, 11 \}$.
    - Descripción sucinta del método.
    - Código.
    - Resultados obtenidos en los experimentos.
    - Referencias consultadas.

- Análisis general de los resultados obtenidos: una tabla que resuma las estadísticas obtenidas y una interpretación de esos resultados.
- Reflexión sobre la experiencia de programar en el lenguaje Go y los aprendizajes logrados en este trabajo.
- Referencias: los libros, revistas y sitios Web que utilizaron durante la investigación y el desarrollo de su proyecto. Citar toda fuente consultada. Usar uno de estos formatos para citar y preparar la bibliografía: ACM, APA o IEEE.
- Apéndices: con ejemplos extendidos o ampliaciones detalladas.

Entrega: jueves **2021.10.14, antes de las 23:55.**
Deben enviar por correo-e el ***enlace***[3] a un archivo comprimido almacenado en alguna nube, con todos los elementos de su solución a estas direcciones: `itrejos@itcr.ac.cr`, `susana.cob.3@gmail.com` (Susana Cob García, asistente), `a.tapia1908@gmail.com` (Alejandro Tapia Álvarez, asistente). El archivo comprimido debe llamarse **`Asignación 2 carnet carnet carnet carnet`**

El asunto (*subject*) de su mensaje debe ser: **`IC-4700 Asignación 2 carnet carnet carnet carnet`**

Si su mensaje no tiene el asunto en la forma correcta, su trabajo será castigado con **-20** puntos; puede darse el caso de que su proyecto no sea revisado del todo (y sea calificado con **0**) sin responsabilidad alguna del profesor o de los

---

[2] "*The density of the binary tree is obtained by dividing the size of the tree by the height of the tree. The size of the binary tree is the total number of nodes present in the given binary tree. The height of the binary tree is the max depth of any leaf node from the root node.*" Fuente: https://www.tutorialspoint.com/density-of-binary-tree-in-one-traversal-in-cplusplus-program

[3] Los sistemas de correo han estado rechazando el envío o la recepción de carpetas comprimidas con componentes ejecutables. Suban su carpeta comprimida (en formato **`.zip`**) a algún 'lugar' en la nube y envíen el hipervínculo al profesor y a nuestros asistentes mediante un mensaje de correo con el formato indicado. Deben mantener la carpeta viva hasta 31 de diciembre del 2021.

asistentes (caso de que su mensaje fuera obviado por no tener el asunto apropiado). Si su mensaje no es legible (por cualquier motivo), o contiene un virus, o es entregado en formato **`.rar`**, la nota será **0**.

La redacción y la ortografía deben ser correctas. La citación de sus fuentes de información debe ser acorde con los lineamientos de la Biblioteca del TEC. En la carpeta 'Referencias', bajo 'LENGUAJES DE PROGRAMACION GR 2's Public Files', ver 'Citas y referencias'. La Biblioteca del TEC usa mucho las normas de la APA. Yo prefiero las de ACM; está bien si usan las normas bibliográficas de la ACM, la APA o el IEEE para su trabajo académico en el TEC (conmigo como profesor).

El profesor tiene *altas expectativas* respecto de la calidad de los trabajos escritos y de la programación producidos por estudiantes universitarios de la carrera de Ingeniería en Computación del Tecnológico de Costa Rica. Los profesores esperamos que los estudiantes tomen en serio la comunicación profesional.

Referencias

- Day, A. Colin (1976). "Balancing a Binary Tree". *Comput. J.* **19** (4): 360–361. *doi*:*10.1093/comjnl/19.4.360*.
- Dromey, G. (1982). *How to solve it by computer*. Reino Unido: Prentice Hall International.
- Drozdek, Adam (2013). *Data Structures and Algorithms in C++, 4th ed.* PWS Publishing Co. pp. 173–175. *ISBN* *0-534-94974-6*.
- Kommadi, Bhagvan (2019). *Learn Data Structures and Algorithms with Golang - Level up your Go programming skills to develop faster and more efficient code*. Birmingham, Reino Unido: Packt Publishing.
- Rolfe, Timothy J. (December 2002). "One-Time Binary Search Tree Balancing: The Day/Stout/Warren (DSW) Algorithm". *SIGCSE Bulletin. ACM SIGCSE. **34** (4): 85–88.* doi:10.1145/820127.820173. Archived from the original on 2012-12-13.
- Stout, Quentin F.; Warren, Bette L. (September 1986). "Tree rebalancing in optimal space and time" (PDF). *Communications of the ACM*. **29** (9): 902–908. *doi*:*10.1145/6592.6599*. *hdl*:*2027.42/7801*.
- Wikipedia (2021). *Day–Stout–Warren algorithm*. *https://en.wikipedia.org/wiki/Day-Stout-Warren_algorithm*
- Wikipedia (2021). *Linear congruential generator*. https://en.wikipedia.org/wiki/Linear_congruential_generator

Ponderación

Este proyecto tiene un valor del 20% de la calificación del curso.