Instituto Tecnológico de Costa Rica
Escuela de Ingeniería en Computación
IC-4700 Lenguajes de programación
Asignación #3 [Proyecto], Características imperativas y secuenciales del lenguaje Go

Historial de revisiones:
- 2022.10.08: Versión base (v0).

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** una secuencia de tamaño ***n*** con números pseudo-aleatorios obtenidos mediante el método de *congruencia lineal multiplicativa*, a partir de una *semilla* dada. Los valores generados deben ser convertidos al intervalo 0 .. 255. La semilla deberá ser un número primo entre 11 y 257. El período debe ser $\geqslant 2048$. ***n*** puede ser cualquier número en el intervalo 500 .. 5000.
2. Diseñar una **representación** en Go para los árboles binarios de búsqueda *ordinarios*, donde cada nodo almacene dos campos: una llave que sea un número entero y un campo adicional para almacenar otro número entero. El campo adicional servirá como un contador.
3. 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.
4. 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 el par (llave, 1). Si la llave *ya* se encuentra en el árbol, modifica el nodo para sumar 1 al número entero que está junto a la llave. La función retorna un número entero, que será la cantidad de comparaciones realizadas, incluida la que llevó a la inserción.
5. 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 de este documento.
6. Diseñar una **representación** en Go para los árboles binarios de búsqueda *AVL*, donde los nodos almacenen una llave que sea un número entero, un campo adicional para almacenar otro número entero y cualesquiera otros campos que se requieran mantener la propiedad *AVL* del árbol.
7. Diseñar y construir una función que haga la **búsqueda** de un valor entero (la llave) en un árbol binario de búsqueda *AVL*, 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.

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

[Si el árbol ya tenía la propiedad *AVL*, no es necesario rebalancear, porque solo estamos buscando (no insertando).]

8. Diseñar y construir una función que haga la **inserción** de un valor entero (la llave) en un árbol binario de búsqueda *AVL*. Si la llave *no* se encuentra en el árbol, crea un nuevo nodo con el par (llave, 1) y rebalancea el árbol *AVL* si fuera necesario. Si la llave *ya* se encuentra en el árbol, modifica el nodo para sumar 1 al número entero que está junto a la llave. La función retorna un número entero, que será la cantidad de comparaciones realizadas, incluida la que llevó a la inserción.
9. Investigar sobre las heurísticas o estrategias de rotación dinámica de árboles de búsqueda que se *auto-ajustan*, asociadas a los trabajos de [Allen & Munro], [Bitner] y [Sleator & Tarjan (*Splay Trees*)].

Distribución del trabajo

Considere los íteme de la lista anterior.

- Si el grupo es de 1 o 2 miembros: desarrollar en Go los algoritmos correspondientes a los ítems 1 al 5.
- Si el grupo es de 3 miembros: desarrollar en Go los algoritmos correspondientes a los ítems 1 al 8.

10. Si el grupo es de 4 miembros: desarrollar en Go los algoritmos correspondientes a los ítems 1 al 8. Además, escoger *una* de las tres propuestas indicadas en el ítem 9, e implementar una representación apropiada para esos árboles binarios de búsqueda, programar y probar los algoritmos de búsqueda e inserción correspondientes. Asegurar que el algoritmo de *inserción* está programado para manejar los contadores de manera equivalente a la indicada para los ítems 4 y 8. Los algoritmos de *búsqueda* podrían cambiar la estructura del árbol, pues la auto-ajustan (según sea la elección hecha por su grupo, de entre las tres posibles).

Experimentos

- Con su función para generar número pseudo-aleatorios (ítem 1), realizar un experimento *para cada uno* de valores de $\boldsymbol{n} \in \{500, 1000, 2000, 3500, 5000\}$.
  - **a)** Crear un *arreglo* **A** de tamaño ***n***, cuyos elementos son números pseudo-aleatorios generados por la función que resuelve el problema 1 (rango de valores 0 .. 255).
  - **b)** Crear *un* árbol vacío *para cada uno* de los métodos para el manejo de árboles binarios de búsqueda que su grupo implementó. Sobre cada uno de esos árboles, *insertar* todos los elementos del arreglo A (de tamaño ***n***), conforme a las indicaciones dadas para cada método (o lo que corresponda al método escogido entre los tres indicados en el ítem 9). Para el método DSW (ítem 5), crear primero un árbol binario ordinario (inicialmente vacío) y someterlo a las inserciones ordinarias (hacer los conteos indicados) y *después* someterlo a la transformación DSW, antes de realizar las búsquedas del paso **c)**.
  - **c)** Generar una secuencia de 100,000 (cien mil) números pseudo-aleatorios. Para *cada uno* de los valores generados, *buscarlo* en *cada uno* de los árboles pre-llenados en el paso **b)**. Generar los números pseudo-aleatorios mediante la función que resuelve el problema 1 (rango de valores 0 .. 255), a partir de una semilla distinta a la usada en el paso **a)**.
  - **d)** Al finalizar, obtener estadísticas *para cada método* desarrollado por su grupo:
    - Altura (máxima) del árbol.
    - Profundidad promedio del árbol.
    - Densidad del árbol[2].
    - Cantidad total de comparaciones realizadas.
    - Cantidad promedio de comparaciones para las inserciones realizadas, *ponderadas* por la altura donde se encuentra cada nodo. Recuerden que al hacer una inserción en los nodos nuevos el contador se inicializa en 1 y en los nodos donde la llave ya existía, se aumenta en 1 el contador.

Informe técnico

Deberán preparar un informe técnico que incluya:

- Portada que identifique a los autores del informe, con sus carnets.
- Introducción al informe.

---

[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

- Secciones:
  - Generación de número pseudo-aleatorios.
    - Estrategia desarrollada.
    - Código.
    - Resultados obtenidos.
    - Referencias consultadas.

  {Una sección para cada método* M *para el manejo de árboles binarios de búsqueda}
  - Método *x*.
    - Descripción sucinta de cada algoritmo correspondiente al método **M**.
    - Código de la construcción (programación) de cada algoritmo correspondiente al método **M**.
    - Resultados obtenidos en los experimentos.
    - Referencias consultadas.
- Análisis general de los resultados obtenidos: una tabla que resuma las estadísticas obtenidas, seguida por un una interpretación y un análisis 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.
- Apéndices:
  - El código fuente de los programas desarrollados por su grupo.
  - Ejemplos de las corridas de sus programas, con todas las semillas utilizadas para generar números pseudo-aleatorios.
  - Las estadísticas obtenidas en sus experimentos: una sección por cada experimento.

Archivos por entregar

- Guardar su trabajo en *una* carpeta comprimida (formato **`.zip`**)[3]. Esto debe incluir:
  - Informe técnico, en un solo documento, según se indicó arriba. El documento debe estar en formato `.pdf`.
  - El código fuente, con todas las funciones, propias y ajenas, en una sub-carpeta aparte.
- Subir su carpeta comprimida a alguna nube, obtener un enlace de solo-lectura, para pasarlo al profesor y al asistente del curso. Deben mantener la carpeta accesible hasta 28 de febrero del 2023.

Entrega

Fecha límite: **miércoles 2022.11.02,** antes de las **23:45**.

Los grupos pueden ser de *hasta* **4** personas. El punto 10 es ***obligatorio*** para los grupos de 4 miembros.

Deben enviar por correo-e el ***enlace***[4] a un archivo comprimido almacenado en alguna nube, con todos los elementos de su solución a estas direcciones: `itrejos@itcr.ac.cr`, `svengra01@gmail.com` (Steven Granados Acuña, Asistente). El archivo comprimido debe llamarse **`Asignación 3`** *carnet carnet carnet*

El asunto (*subject*) de su mensaje debe ser: **`IC-4700 Asignación 3`** *carnet carnet carnet*

Sustituir *carnet* por los números de carnet de cada persona miembro del grupo.

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 del asistente (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**.

---

[3] No use formato **`.rar`**, porque es rechazado por el sistema de correo-e del TEC.

[4] 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 nuestro asistente mediante un mensaje de correo con el formato indicado. Deben mantener la carpeta accesible hasta 28 de febrero del 2023.

Habilitaremos también la entrega de los trabajos vía el tecDigital.

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 (bibliografía), en OneDrive, ver 'Citas y referencias'. La Biblioteca del TEC usa mucho las normas de la APA. El profesor prefiere 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 (con este profesor).

El profesor tiene *altas expectativas* respecto de la calidad de los trabajos escritos y de la programación que produzcan 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

Referencias suministradas por el profesor. En OneDrive: carpeta 'Referencias', sub-carpeta 'Algoritmos y estructuras de datos'.
https://tecnube1-my.sharepoint.com/:f:/g/personal/itrejos_itcr_ac_cr/EsNQxZIXRmVNkpSiSi5x8psBE2yFm5Zx7LKWJ6vfseJD8w?e=dzxyW5

- Allen, Brian, and Munro, Ian, "Self-Organizing Binary Search Trees," *Journal of the ACM* 25 (1978), 526–535.
- Bitner, James R., "Heuristics That Dynamically Organize Data Structures," *SIAM Journal on Computing* 8 (1979), 82–110.
- 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*.
- 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.
- Sedgewick, Robert. Algorithms in C, 3rd ed. Addison-Wesley, 1998. [caps. 12 y 13]
- Sleator, Daniel D., and Tarjan, Robert E., "Self-Adjusting Binary Search Trees," *Journal of the ACM* 32 (1985), 652–686.
- 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
- Tarjan, Robert E.(1987). Algorithm design. *Commun. ACM* 30, 3 (March 1987), 204–212. DOI: https://doi.org/10.1145/214748.214752
- 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 30% de la calificación del curso.