Departamento de Computación y Sistemas Inteligentes
Computación y Estructuras Discretas II
Tarea Integradora 1

Objetivos Terminales

OT1. Aplicar el paradigma funcional en el análisis, diseño, evaluación, selección e implementación de algoritmos para dar solución a problemas cuya estructura es naturalmente autocontenida.
OT2. Aplicar inducción matemática para definir estructuras discretas, demostrar sus propiedades y verificar algoritmos formalmente.
OT7. Expresar o comunicar con el vocabulario y lenguaje adecuado/especializado las ideas principales sobre estructuras discretas o la programación funcional.

Objetivos Específicos

- Calcular la complejidad temporal de algoritmos recursivos.
- Resolver ecuaciones de recurrencia que sean resultado del análisis de complejidad temporal de algoritmos recursivos.
- Diseñar e implementar programas funcionales puros con estructuras de datos inmutables utilizando recursión, reconocimiento de patrones para resolver problemas de programación.
- Aplicar conceptos fundamentales de la programación funcional, utilizando un lenguaje de programación adecuado como SCALA, para analizar un problema, modelar, diseñar y desarrollar su solución
- Razona sobre la estructura de programas funcionales utilizando la inducción como mecanismo de argumentación para demostrar propiedades de los programas que construye

Problema: Algoritmos de ordenamiento en el paradigma funcional

Los algoritmos de ordenamiento describen un proceso computacional que produce una salida a partir de la entrada especificada así:

Entrada: Una secuencia de n objetos $\langle a_1, a_2, a_3, \ldots, a_n \rangle$
Salida: Una permutación (reorganización) $\langle a'_1, a'_2, a'_3, \ldots, a'_n \rangle$ de la secuencia de entrada, tal que, $a'_1 \leq a'_2 \leq a'_3 \leq \ldots \leq a'_n$ para una relación de orden $\leq$

El problema de ordenar una secuencia de objetos es estudiado frecuentemente en los primeros cursos de programación porque permite introducir diferentes técnicas de diseño y análisis de algoritmos [1]. En esta tarea integradora cada equipo de trabajo debe implementar los siguientes algoritmos de ordenamiento:

1. HeapSort, de acuerdo a la descripción del capítulo 6 del texto de Cormen[1]
2. CountingSort, de acuerdo a la descripción del capítulo 8 del texto de Cormen[1]
3. RadixSort, de acuerdo a la descripción del capítulo 8 del texto de Cormen[1]

Ordenamiento genérico y de alto orden

Para cada uno de los algoritmos de ordenamiento implemente una versión de alto orden que reciba como parámetro la función de ordenamiento

Condiciones de la implementación:

- Los algoritmos deberán implementarse con el paradigma funcional usando el lenguaje Scala. De acuerdo a los objetivos de aprendizaje los programas deben ser funcionales puros con estructuras de datos inmutables y utilizando recursión.
- La entrada y salida debe estar representada usando listas de enteros. Además, los algoritmos deben procesar las listas usando *pattern matching*.

- Defina al menos cuatro funciones recursivas que lancen procesos iterativos.
- Las funciones recursivas que lanzan procesos iterativos deben estar anotadas con la etiqueta @tailrec
- Los identificadores y comentarios del código deben estar en inglés.
- El código debe estar documentado usando las etiquetas de javadoc.

Condiciones de la implementación:

- Los algoritmos deberán implementarse con el paradigma funcional usando el lenguaje Scala. De acuerdo a los objetivos de aprendizaje los programas deben ser funcionales puros con estructuras de datos inmutables y utilizando recursión.
- La entrada y salida debe estar representada usando listas de enteros. Además, los algoritmos deben procesar las listas usando pattern matching.
- Su tarea integradora debe tener al menos cuatro funciones recursivas que lancen procesos iterativos.
- Las funciones recursivas que lanzan procesos iterativos deben estar anotadas con la etiqueta @tailrec
- Los identificadores y comentarios del código deben estar en inglés.
- El código debe estar documentado usando las etiquetas de javadoc.

Equipos de trabajo

Los equipos pueden formarse con estudiantes de los dos grupos del curso y **deben registrarse a tiempo en una hoja de equipos y en GitHub classroom**. Equipos formados por fuera de los tiempos establecidos podrán o no ser tenidos en cuenta o disueltos a potestad de los profesores. Teniendo en cuenta la cantidad total de estudiantes, los equipos serán **estrictamente** de **tres** o **cuatro** integrantes.

Para registrar su equipo usted deberá:

1. Llenar la siguiente hoja de cálculo con los nombres y los usuarios de GitHub de cada integrante. El identificador del equipo (Ex) en la hoja de registro se usará en el siguiente paso [link a la hoja de registro].
2. Cada estudiante debe ingresar a GitHub classroom usando la invitación y el identificador de su equipo. Tenga en cuenta que el primer integrante en registrarse debe crear el equipo usando el identificador de la hoja de registro [Invitación].

Entregables

1. Para cada problema, usted debe formular al menos una definición recursiva de un algoritmo (función auxiliar, o principal) y probar la correctitud de la función (sea un bloque de código o una función recursiva). Para este paso usted debe utilizar inducción estructural: probando el caso base, formulando la hipótesis inductiva y probando la correctitud de su algoritmo suponiendo válida la hipótesis inductiva del paso previo acorde con los lineamientos de la inducción estructural.
2. A cada problema se le debe calcular su complejidad, en el caso de los algoritmos con la estrategia divide y vencerás esta complejidad debe asociarse a una relación de recurrencia que permita estimar la complejidad del mismo. Usted debe resolver dicha ecuación para mostrar la complejidad utilizando la notación O, Θ u Ω (solo una).

3. Su proyecto debe estar en GitHub classroom y debe contener lo siguiente
    - Archivos de la implementación del proyecto en Scala con las carpetas main para la solución a la tarea integradora y test para las pruebas unitarias.
    - Tener al menos dos commits. El **primer commit** debe hacerse a más tardar el **lunes 2 de septiembre de 2024 (con diseño de pruebas e implementación de las pruebas)**. La **entrega final** a más tardar el **lunes de la semana 9** a las 12:00 pm.
    - Un archivo README.md con la presentación del proyecto y los integrantes. Este archivo debe escribirse usando Markdown.
    - Una carpeta doc con los documentos que contienen el diseño de las pruebas, las demostraciones y el análisis de la complejidad de los algoritmos. Estos documentos también deben escribirse usando Markdown.

Nota: El proyecto debe cumplir con las condiciones de implementación, el código debe compilar/ejecutar y los entregables deben estar en el formato solicitado para que pueda ser revisado.
Sustentación: Después de la entrega, todos los estudiantes serán citados para la sustentación.

Rúbricas

- **Rubrica de la tarea**
- **Rubrica de la sustentación**

Referencias

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, Third Edition. The MIT Press, 3rd edition, 2009.