INFORME PROYECTO FINAL

PROGRAMACIÓN FUNCIONAL Y CONCURRENTE

Alejandro Marin Hoyos 2259353-3743

Yessica Fernanda Villa Nuñez 2266301-3743

Manuel Antonio Vidales Duran 2155481-3743

UNIVERSIDAD DEL VALLE SEDE TULUÁ

PROGRAMA ACADÉMICO DE INGENIERÍA DE SISTEMAS

2023

1. Alternativas solución

Para implementar los algoritmos presentados de manera funcional, se definen el alfabeto y el tipo de dato Oraculo:

1.1.GenerarOraculo:

def generarOraculo(longitud: Int): Seq[Char]: Define una función llamada **generarOraculo** que toma un parámetro longitud de tipo entero (Int) y devuelve una secuencia (Seq) de caracteres (Char).

Seq.fill(longitud)(alfabeto(Random.nextInt(alfabeto.length))): Crea una secuencia de caracteres de longitud especificada (longitud).

La función **fill** toma dos argumentos: El primero es la longitud de la secuencia que se va a crear, y el segundo es el valor que se utilizará para llenar la secuencia.

alfabeto(Random.nextInt(alfabeto.length)): Esto genera un carácter aleatorio seleccionado del "alfabeto". Aquí, se asume que alfabeto es una variable o constante que contiene una secuencia de caracteres de la cual se elige uno al azar. Random.nextInt(alfabeto.length) genera un índice aleatorio dentro del rango de la longitud del alfabeto, y luego se utiliza para seleccionar un carácter específico de ese índice en el alfabeto.

La función generarOraculo crea y devuelve una secuencia de caracteres aleatorios de la longitud especificada, seleccionando cada carácter de forma aleatoria del alfabeto proporcionado. La aleatoriedad está basada en la generación de números aleatorios utilizando Random.nextInt.

1.2.ReconstruirCadenaIngenuo:

reconstruirCadenaIngenuo: Es la función principal que toma dos parámetros: n (un entero) y o (un objeto de tipo Oraculo).

auxiliar: Es una función interna dentro de reconstruirCadenaIngenuo. Toma dos parámetros: secuencia (una secuencia de caracteres) y n (un entero).

Si n es 0, la función devuelve la secuencia actual.

Si n no es 0, la función usa map para generar nuevas secuencias añadiendo cada letra del alfabeto a la secuencia actual y llamando recursivamente a auxiliar con la nueva secuencia y n - 1.

alfabeto: Es una variable o constante que contiene una secuencia de caracteres (un alfabeto).

find(o): Después de generar todas las posibles secuencias con n letras del alfabeto, la función busca la primera secuencia que satisface la condición definida por el oráculo o.

getOrElse(Seq()): Si no se encuentra ninguna secuencia que cumpla con la condición del oráculo, la función devuelve una secuencia vacía.

Finalmente, la función principal reconstruirCadenaIngenuo llama a auxiliar con una secuencia vacía y n, lo que inicia el proceso de construcción de secuencias y la búsqueda de aquellas que cumplen con la condición del oráculo.

1.3.ReconstruirCadenaMejorado:

ReconstruirCadenaMejorado: Es la función principal que toma dos parámetros: n (un entero) y o (un objeto de tipo Oraculo).

auxiliar: Es una función interna dentro de ReconstruirCadenaMejorado. Toma dos parámetros: secuencia (una secuencia de caracteres) y k (un entero).

Condición Principal: Si k es igual a n (la longitud deseada de la secuencia) y la secuencia es aceptada por el oráculo o secuencia, entonces devuelve la secuencia actual.

Caso Recursivo: Si k es menor que n, la función genera nuevas secuencias añadiendo cada letra del alfabeto a la secuencia actual y llamando recursivamente a auxiliar con la nueva secuencia y k + 1.

La función utiliza map para generar varias secuencias y luego usa find(o) para encontrar la primera que satisface la condición del oráculo.

Caso Base: Si k es mayor o igual a n, la función devuelve una secuencia vacía (Seq()).

Llamada Inicial:La función principal ReconstruirCadenaMejorado llama a auxiliar con una secuencia vacía (Seq()) y k inicializado en 0.

Este código genera secuencias de caracteres utilizando una estrategia mejorada que divide la construcción de la secuencia en ramas más pequeñas, lo que podría mejorar la eficiencia en comparación con un enfoque ingenuo. La secuencia resultante debe cumplir con la longitud deseada n y la condición del oráculo o.

1.4.ReconstruirCadenaTurbo:

Tiene dos parámetros: n de tipo entero (Int) y o de tipo Oraculo. La función devuelve una secuencia (Seq) de caracteres (Char).

La función utiliza dos funciones auxiliares: buscarCadena y Auxiliar.

buscarCadena: Esta función toma una secuencia de secuencias de caracteres (Seq[Seq[Char]]) como argumento y devuelve la primera secuencia que cumple con la condición especificada por el oráculo o. Utiliza el método **collectFirst** para encontrar la primera secuencia que satisface la condición del oráculo, y en caso de no encontrar ninguna, devuelve una secuencia vacía (Seq.empty).

Auxiliar: Esta función toma dos argumentos, un entero k y una secuencia de secuencias de caracteres secuencia. Si k es mayor o igual a n, la función devuelve la secuencia tal como está. Si k es menor que n, entonces la función llama recursivamente a sí misma con k + 1 y una nueva secuencia obtenida aplicando la función flatMap sobre la secuencia actual. La función flatMap concatena cada elemento de la secuencia con cada letra del alfabeto, generando así nuevas secuencias.

Finalmente, la función principal ReconstruirCadenaTurbo llama a buscarCadena con el resultado de llamar a Auxiliar(1, alfabeto.map(Seq(_))). Aquí, alfabeto parece ser una variable no definida en el código proporcionado, pero se asume que es un iterable (por ejemplo, una secuencia o lista) de caracteres.

2. Acelerando los datos con paralelismo de tareas y de datos.

2.1.ReconstruirCadenaIngenuoParallel:

reconstruirCadenaIngenuoParallel: Es la función principal que toma dos parámetros curried: umbral (un entero) y luego una tupla (n: Int, o: Oraculo).

auxiliar: Es una función interna que realiza la reconstrucción de la cadena. Toma dos parámetros: secuencia (una secuencia de caracteres) y n (un entero).

Condición Base:Si n es igual a 0, la función devuelve la secuencia actual.

Caso Recursivo:Si n es menor que el umbral y la secuencia actual es aceptada por el oráculo (o(secuencia)), realiza las llamadas de manera secuencial utilizando la función **reconstruirCadenaIngenuo**.

Si no, utiliza el método par.map para realizar llamadas en paralelo con el tipo de colección paralela.

La función task se utiliza para envolver cada llamada a auxiliar en una tarea, que se ejecuta en paralelo.

Utiliza join() para esperar a que todas las tareas paralelas se completen.

Después de obtener los resultados paralelos, utiliza find(o) sobre la secuencia resultante de manera no paralela para encontrar la primera que satisface la condición del oráculo.

Llamada Inicial:La función principal inicia la llamada a auxiliar con una secuencia vacía (Seq()) y el valor n proporcionado.

Esta función realiza la reconstrucción de una cadena de caracteres de manera paralela para tamaños mayores o iguales al umbral especificado y de manera secuencial para tamaños menores al umbral. El objetivo es aprovechar la paralelización para mejorar el rendimiento en tamaños grandes, mientras que para tamaños pequeños se prefiere la ejecución secuencial, posiblemente para evitar la sobrecarga asociada con la ejecución paralela en

tales casos. La función utiliza la función task para crear tareas paralelas y par.map para realizar llamadas en paralelo.

2.2.ReconstruirCadenaMejoradoPar:

ReconstruirCadenaMejoradoPar: Es la función principal que toma dos parámetros curried: umbral (un entero) y luego una tupla (n: Int, o: Oraculo).

auxiliarParalelo: Es una función interna que realiza la reconstrucción de la cadena. Toma dos parámetros: secuencia (una secuencia de caracteres) y k (un entero).

Condición Principal:Si k es igual a n y la secuencia actual es aceptada por el oráculo o secuencia, la función devuelve la secuencia actual.

Caso Recursivo:Si k es menor que n, y si n es menor que el umbral, realiza llamadas de manera secuencial utilizando la función **ReconstruirCadenaMejorado** para tamaños pequeños.

Si n es mayor o igual al umbral, utiliza el método par.map para realizar llamadas en paralelo con el tipo de colección paralela.

La función task se utiliza para envolver cada llamada a auxiliarParalelo en una tarea, que se ejecuta en paralelo.

Utiliza join() para esperar a que todas las tareas paralelas se completen.

Después de obtener los resultados paralelos, utiliza find(o) sobre la secuencia resultante de manera no paralela para encontrar la primera que satisface la condición del oráculo.

Caso Base:Si k es mayor o igual a n, la función devuelve una secuencia vacía (Seq()).

Llamada Inicial:La función principal inicia la llamada a **auxiliarParalelo** con una secuencia vacía (Seq()) y el valor k inicializado en 0.

Esta función realiza la reconstrucción de una cadena de caracteres de manera paralela para tamaños mayores o iguales al umbral especificado y de manera secuencial para tamaños menores al umbral. El objetivo es aprovechar la paralelización para mejorar el rendimiento en tamaños grandes, mientras que para tamaños pequeños se prefiere la ejecución secuencial, posiblemente para evitar la sobrecarga asociada con la ejecución paralela en tales casos. La función utiliza la función task para crear tareas paralelas y par.map para realizar llamadas en paralelo.

2.3.ReconstruirCadenaTurboPar: Esta función toma un parámetro adicional llamado umbral, que es de tipo entero (Int) y se utiliza como argumento para la función **parallel**. La función **ReconstruirCadenaTurboPar** devuelve una secuencia (Seq) de caracteres (Char).

La función utiliza las mismas funciones auxiliares que el código anterior: buscarCadena y Auxiliar.

buscarCadena: Esta función es idéntica a la versión anterior y toma una secuencia de secuencias de caracteres (Seq[Seq[Char]]) como argumento. Devuelve la primera secuencia que cumple con la condición especificada por el oráculo o, utilizando el método collectFirst. Si no encuentra ninguna secuencia que cumpla con la condición, devuelve una secuencia vacía (Seq.empty).

Auxiliar: Esta función es similar a la versión anterior, pero con una diferencia crucial. En lugar de llamar recursivamente a la función de manera secuencial, se utiliza la función parallel para realizar las llamadas recursivas en paralelo. La función parallel toma dos tareas (expresadas como bloques de código) y las ejecuta en paralelo, devolviendo un par de resultados. En este caso, las dos tareas corresponden a las llamadas a flatMap sobre la secuencia actual, cada una con el alfabeto. Luego, se concatenan los resultados de ambas tareas antes de realizar la llamada recursiva.

La función ReconstruirCadenaTurboPar es similar a ReconstruirCadenaTurbo, pero utiliza paralelismo para acelerar el proceso de generación de secuencias. El paralelismo se logra mediante la función parallel que ejecuta tareas en paralelo.

3. Tabla comparativa.

3.1. Tabla de comparación del método ingenuo versión secuencial y paralela:

Tamaño	Ingenuo (ms)	IngenuoPar(ms)	Aceleración (ms)
2	0.0360	1.6329	0.020
4	0.1753	1.8780	0.0933
8	14.0568	47.1289	0.2983
10	306.2730	611.5581	0.5008
12	4488.5878	10050.7203	0.4466

3.2. Tabla de comparación del método mejorado versión secuencial y paralela:

Tamaño	Mejorado (ms)	MejoradoPar(ms)	Aceleración (ms)
2	0.0655	0.3287	0.1993
4	0.1801	1.9174	0.0939
8	9.1524	20.8146	0.4397
10	128.909	688.8033	0.1861

3.3. Tabla de comparación del método turbo versión secuencial y paralela:

Tamaño	Turbo (ms)	TurboPar(ms)	Aceleración (ms)
2	0.0830	0.3281	0.2530
4	0.4115	1.2142	0.3389

3.4. CompararAlgoritmos: Compara el rendimiento de dos algoritmos: uno secuencial (funcionSecuencial) y otro paralelo (funcionParalela). La función toma tres parámetros:

funcionSecuencial: Una función que toma un entero n y un oráculo o como argumentos y devuelve una secuencia de caracteres.

funcionParalela: Una función similar a funcionSecuencial, pero diseñada para ejecutarse en paralelo.

(n: Int, o: Oraculo): La entrada para ambos algoritmos, consistente en un entero n y un oráculo o.

La función devuelve una tupla (Double, Double, Double), que contiene tres valores:

tiempoSecuencial: El tiempo de ejecución de funcionSecuencial medido con el objeto Warmer de Scala.

tiempoParalelo: El tiempo de ejecución de **funcionParalela** medido con el mismo objeto Warmer.

aceleracion: El factor de aceleración, calculado como el cociente entre el tiempo secuencial y el tiempo paralelo (**aceleracion** = **tiempoSecuencial** / **tiempoParalelo**).

La función utiliza la función **withWarmer** de ScalaTest para medir el tiempo de ejecución de las funciones proporcionadas en **funcionSecuencial** y **funcionParalela**. La idea es comparar el rendimiento secuencial y paralelo de los algoritmos dados el tamaño de la entrada n y el oráculo o.

Esta función **compararAlgoritmos** proporciona una herramienta para evaluar el rendimiento relativo de dos algoritmos dados bajo diferentes condiciones de entrada. El resultado incluye información sobre los tiempos de ejecución secuencial y paralelo, así como la aceleración lograda por el enfoque paralelo.

3.5. PruebasCompararAlgoritmos:

Realiza pruebas y comparaciones entre dos algoritmos de reconstrucción de cadenas: uno ingenuo (**reconstruirCadenaIngenuo**) y otro paralelo (**reconstruirCadenaIngenuoParallel**(4)). La función utiliza un conjunto de diferentes tamaños de cadena (**tamanios**) para realizar las pruebas y luego imprime los resultados en forma de una tabla.

La función realiza las siguientes acciones:

Definición de Tamaños de Cadena: Se define una secuencia tamanios que contiene diferentes tamaños de cadena (longitudes) para probar los algoritmos. Estos tamaños son 2, 4, 8, 10 y 12.

Impresión del Encabezado de la Tabla: Se imprime en la consola el encabezado de la tabla, que incluye las columnas "Tamaño", "Turbo (ms)", "Turbo Parallel (ms)", "Aceleración (ms)" y "Oráculo". Esto proporciona una presentación clara de los resultados.

Iteración a través de los Tamaños de Cadena: Se utiliza un bucle for para iterar a través de cada tamaño de cadena en la secuencia **tamanios**.

Generación de Oráculo y Comparación de Algoritmos: Para cada tamaño de cadena, se genera un oráculo usando la función **generarOraculo**. Luego, se llama a la función **compararAlgoritmos** con los algoritmos reconstruirCadenaIngenuo y **reconstruirCadenaIngenuoParallel**(4), el tamaño de cadena actual y un oráculo que verifica si la cadena generada es igual al oráculo.

Impresión de Resultados en la Tabla: Se imprime en la consola una fila de la tabla con información sobre el tamaño de la cadena, los tiempos de ejecución secuencial y paralelo, la aceleración y el oráculo asociado.

La función **pruebasCompararAlgoritmos** automatiza la comparación de dos algoritmos de reconstrucción de cadenas bajo diferentes condiciones de entrada. Proporciona una visión rápida del rendimiento relativo de los algoritmos para tamaños de cadena específicos, lo que puede ser útil para evaluar la eficiencia y la escalabilidad de los algoritmos en cuestión.

4. Conclusiones:

Método ingenuo versión secuencial y paralela

La tabla de comparación entre las versiones secuencial y paralela revela que la implementación paralela mejora significativamente el rendimiento, especialmente para tamaños de cadena más grandes. Aunque la versión ingenua tiene un desempeño inferior en comparación con el método mejorado, la ejecución paralela logra una aceleración considerable, destacando su eficacia para tareas más intensivas. La elección entre las versiones secuencial y paralela dependerá del tamaño típico de las cadenas a procesar, con la ejecución paralela mostrando su mayor impacto en situaciones que involucran tamaños considerables de cadena.

Método mejorado versión secuencial y paralela

La tabla de comparación entre las versiones secuencial y paralela revela un rendimiento mejorado en términos de tiempo de ejecución para tamaños de cadena más grandes. La ejecución paralela muestra una aceleración más significativa cuando se procesan cadenas más extensas (8 y 10), indicando que la paralelización es más beneficiosa en estos casos. Sin embargo, para tamaños de cadena más pequeños (2 y 4), la aceleración es menor, sugiriendo que la ejecución paralela puede no ser tan ventajosa en estas situaciones. La elección entre las versiones secuencial y paralela dependerá de las características específicas del problema y del tamaño típico de las cadenas a procesar.

Método turbo versión secuencial y paralela

El tiempo de ejecución, tanto para la versión secuencial como para la paralela, tiende a aumentar a medida que crece el tamaño de la cadena. Este comportamiento es esperado, ya que reconstruir cadenas más largas generalmente requiere más tiempo de procesamiento.

La ejecución paralela, en algunos casos, demuestra ser más eficiente que la ejecución secuencial, como se evidencia en la aceleración menor a 1 para ciertos tamaños de cadena. Sin embargo, no siempre se observa una mejora constante en la eficiencia al utilizar la ejecución paralela, debido a que la ejecución puede depender del tamaño específico de la cadena. Por ejemplo, para tamaños de cadena pequeños, la ejecución paralela puede ser menos eficiente que la secuencial, pero para tamaños de cadena más grandes, la ejecución paralela puede proporcionar beneficios en términos de rendimiento.

La tabla de comparación entre las versiones secuencial y paralela del método Turbo revela que la eficiencia de la ejecución paralela varía con el tamaño de la cadena. Si bien la ejecución paralela puede ofrecer mejoras en el rendimiento para ciertos tamaños de

cadena, esta mejora no es constante. La elección entre la versión secuencial y paralela debe considerar cuidadosamente el tamaño típico de las cadenas y las características específicas del hardware, ya que la eficiencia de la ejecución paralela depende de estos factores.