[1]

Proyecto: El problema de la reconstrucción de cadenas

Alex García Castañeda - 2259517, Sebastián Gómez Agudelo - 2259474, Stiven Henao Aricapa - 2259603

Funciones implementadas

reconstruirCadenaIngenuo:

Estructura de Datos Utilizada:

La función utiliza principalmente estructuras de datos básicas de Scala, como Seq[Char] para representar secuencias de caracteres y LazyList[Seq[Char]] para generar de manera perezosa las secuencias.

Recursión:

La función generarSecuencias emplea la recursión para generar secuencias de caracteres. Se llama a sí misma decrementando n hasta alcanzar el caso base (n == 0).

Reconocimiento de Patrones:

Si bien no se emplea directamente el reconocimiento de patrones, el uso de la función flatMap se asemeja a un patrón de mapeo sobre cada elemento del alfabeto.

Mecanismos de Encapsulación:

La función generarSecuencias está encapsulada dentro de reconstruirCadenaIngenuo, lo que limita su alcance y visibilidad fuera de esta función.

Funciones de Alto Orden:

Se hace uso de funciones de alto orden, específicamente flatMap, que recibe una función como argumento y aplica esta función a cada elemento de una colección.

Iteradores, Colecciones y Expresiones For:

Se utiliza flatMap para iterar sobre los elementos del alfabeto y construir nuevas secuencias a partir de la llamada recursiva.

No se emplea expresamente un bucle for, pero el patrón de recursión en generarSecuencias realiza un recorrido similar.

Argumentaciones sobre su Implementación:

La función generarSecuencias utiliza una lógica recursiva para generar secuencias de longitud n. Cada llamada recursiva reduce n hasta llegar al caso base y genera secuencias mediante la concatenación de caracteres del alfabeto.

La función find aplicada a la secuencia de secuencias generadas busca la primera secuencia que cumpla con la condición dada por el oráculo.

La implementación perezosa mediante LazyList permite generar y procesar secuencias de manera eficiente, ya que solo se generan cuando se necesitan.

Notación matemática

reconstruirCadenaIngenuo(n,o)=generarSecuencias(n,secuenciaVacia).find(o)

donde generarSecuencias(n,s) = { {s} si n=0, generarSecuencias(n-1,s+char) si n>0 y char ϵ alfabeto}

Estos elementos se combinan para generar secuencias de caracteres y encontrar la primera que cumpla con la condición establecida por el oráculo.

reconstruirCadenaIngenuoPar:

Técnicas de paralelización:

La paralelización se implementa en la función generarSecuenciasRec mediante el método par aplicado a la colección alfabeto. Esto distribuye la generación de secuencias en múltiples hilos para procesar de manera simultánea diferentes partes del trabajo.

Además, se utiliza par en la variable secuencias, lo que paraleliza la generación de todas las secuencias posibles.

Impacto en el desempeño:

Sin embargo, debido a la naturaleza recursiva del algoritmo, la eficiencia de la paralelización puede verse limitada, ya que cada iteración depende del resultado de las anteriores, generando una posible alta latencia de comunicación entre los hilos.

Aunque la paralelización puede ser beneficiosa en escenarios con operaciones intensivas, su impacto efectivo en el rendimiento depende del tamaño de n, la complejidad de las operaciones y el hardware donde se ejecute el programa.

Evaluación comparativa

Secuencial	Paralela	Aceleración	Tamaño
0.0296	0.1077	0.2748375116063138	1
0.014	0.3247	0.043116723129042196	2
0.0414	0.2945	0.1405772495755518	3
0.1085	0.4623	0.2346960847934242	4
0.4431	9.0304	0.049067593905031819	5
0.8505	3.2971	0.257953959540202	6
1.4709	10.7164	0.13725691463551193	7
50.2609	37.5263	1.3393513349304356	8
166.1964	232.2935	0.7154586762005826	9
982.1694	1012.0242	0.9704999149229832	10
3656.2551	5039.4741	0.7255231453615367	11

Respecto a la aceleración se espera que sea mayor a 1 para lograr obtener que la versión paralela sea siempre superior en cuanto a tiempos de ejecución respecto al a versión secuencial,

[1]Artículo entregado el 12 de febrero de 2003. (Coloque la fecha en la que entrega el informe). Los autores son estudiantes de Ingeniería Mecánica en la Universidad Nacional de Colombia.

Primer Autor: código1, autor1@ ing.unal.edu.co.
Segundo Autor: código2, autor2@hotmail.com.
Tercer Autor: código3, autor3@yahoo.com.

por lo tanto, a raíz de los resultados obtenidos se puede concluir que el punto de cambio para aprovechar la ejecución paralela parece ser alrededor de tamaños de cadena 5 o 6. A partir de estos tamaños, la versión paralela ofrece tiempos de ejecución más bajos que la secuencial, lo que indica su mayor eficiencia para cadenas más grandes.

Solución Mejorada:

Estructura de Datos Utilizada:

Seq[Char]: Utilizada para representar secuencias de caracteres, en este caso, las subcadenas candidatas y la cadena resultante.

Argumentación sobre la Corrección:

- La función SolucionMejorada utiliza recursión para generar todas las posibles subcadenas de longitud n, y luego filtra aquellas que son subcadenas válidas según el oráculo.
- La recursión se detiene cuando la longitud es 0, devolviendo una secuencia vacía, o cuando se alcanza la longitud deseada n.
- La función utiliza la combinación de for-yield para generar las subcadenas y filtrarlas según el oráculo.
- El uso de headOption.getOrElse(Seq.empty[Char]) asegura que, si la secuencia es vacía, se devuelva una secuencia vacía por defecto.
- Usar recursión hace que la construcción de subcadenas se realiza de manera recursiva, generando todas las combinaciones posibles de manera estructurada.

Solución MejoradaPar:

Estructuras de Datos Utilizadas:

ParSeq[Seq[Char]]: Se utiliza una versión paralela de la secuencia para la generación de subcadenas en paralelo.

Argumentación sobre la Corrección:

- La función SolucionMejoradaPar introduce paralelización para mejorar el rendimiento en comparación con la versión secuencial.
- La paralelización se aplica en la generación de subcadenas utilizando el método .par.
- Cuando la longitud n es menor o igual al umbral especificado, la función se reduce a la versión secuencial (SolucionMejorada).
- Se ha corregido la implementación de subcadenas en la función paralelizada, utilizando .par en lugar de ParSeq.apply() para evitar posibles problemas de ordenamiento y garantizar la paralelización efectiva.

Técnicas de Paralelización:

- Paralelización de Tareas: La generación de subcadenas se realiza en paralelo utilizando la colección paralela ParSeq.
- Umbral de Paralelización: Se establece un umbral, y si la longitud n es menor o igual a este umbral, la función se reduce a la versión secuencial para evitar sobrecarga de paralelización en casos pequeños.

Impacto en el Desempeño:

- La paralelización debería tener un impacto positivo en el desempeño del programa al aprovechar la capacidad de procesamiento paralelo.
- Sin embargo, el rendimiento dependerá del tamaño del problema y de la capacidad del sistema para manejar la paralelización efectiva.
- Para problemas grandes, donde la generación de subcadenas es intensiva, la paralelización podría llevar a una mejora significativa en el tiempo de ejecución.

Evaluacion comparativa entre Solucion Mejorada y Solucion MejoradaPar:

Secuencial	Paralela	Aceleración	Tamaño
0.0313	0.0288	1.0868055555555556	1
0.027299	0.0341	0.8005571847507332	2
0.034299	0.0291	1.178659793814433	3
0.033	0.035201	0.9374733672338854	4
0.0364	1.3475	0.0270129870129870	5
0.0572	1.110901	0.051489736709211	6
0.1273	1.016	0.1252952755905511	7
0.0451	1.5976	0.0282298447671507	8
0.0717	1.241	0.0577759871071716	9
0.0607	1.341501	0.0452478231473550	10
0.049099	1.1945	0.041104227710339	11

En la comparación de rendimiento entre las funciones secuencial (SolucionMejorada) y paralela (SolucionMejoradaPar), se observa que la versión secuencial es más eficiente para tamaños pequeños de subcadenas. La estrategia de introducir un umbral en la versión paralela demuestra ser efectiva, evitando la sobrecarga en casos simples. Sin embargo, a medida que aumenta el tamaño de las subcadenas, la paralelización muestra su ventaja, superando a la versión secuencial.

reconstruirCadenaTurbo:

Estructura de Datos Utilizada:

La función reconstruirCadenaTurbo utiliza principalmente estructuras de datos básicas de Scala. Se hace uso de Set[Seq[Char]] para representar conjuntos de subcadenas y Seq[Char] para representar secuencias de caracteres. Además, la función utiliza la propiedad de concatenación de secuencias para construir nuevas subcadenas.

Recursión:

La función generarCadenaTurbo emplea la recursión

para generar subcadenas de longitud creciente. Se llama a sí misma con una longitud actual multiplicada por 2 hasta alcanzar o superar la longitud deseada n.
La recursión se detiene cuando se encuentra una subcadena válida de longitud n o cuando la longitud actual supera n.

Reconocimiento de Patrones:
Aunque no se emplea directamente el reconocimiento de patrones, el uso de flatMap se asemeja a un patrón de mapeo sobre cada elemento de las colecciones subcadenasActuales y nuevasSubcadenas, generando y combinando subcadenas de manera eficiente.

Mecanismos de Encapsulación:
La función generarCadenaTurbo está encapsulada dentro de reconstruirCadenaTurbo, limitando su alcance y visibilidad fuera de esta función. Esto favorece la modularidad y la comprensión del código.

Funciones de Alto Orden:
La función hace uso de funciones de alto orden, como flatMap y filter, que reciben funciones como argumentos y aplican estas funciones a cada elemento de las colecciones. Esto facilita la manipulación de conjuntos de subcadenas de manera concisa.

Iteradores, Colecciones y Expresiones For:
La función utiliza expresiones flatMap para iterar sobre las colecciones subcadenasActuales y nuevasSubcadenas, construyendo y combinando subcadenas a partir de llamadas recursivas. Aunque no se emplea un bucle for explícito, el patrón de recursión realiza un recorrido similar.

Argumentaciones sobre su Implementación:
La función generarSecuencias utiliza una lógica recursiva para generar secuencias de longitud n. Cada llamada recursiva reduce n hasta llegar al caso base y genera secuencias mediante la concatenación de caracteres del alfabeto.
La función find aplicada a la secuencia de secuencias generadas busca la primera secuencia que cumpla con la condición dada por el oráculo.
La implementación perezosa mediante LazyList permite generar y procesar secuencias de manera eficiente, ya que solo se generan cuando se necesitan.

Notación matemática

reconstruirCadenaTurbo(n,o)=
crearCadenaTurbo(2,subcadenaAlfabeto)
Donde:
crearCadenaTurbo(tamano, subActuales) =
find(subcadenasPrueba.filter(o).find(lenght == n).getOrElse(crearCadenaTurbo(tamano * 2, subcadenasPrueba))

La función 'crearCadenaTurbo' genera subcadenas de tamaño creciente, utilizando el doble de tamaño en cada iteración, a partir de un conjunto de subcadenas actuales.Luego, filtra estas subcadenas con el oraculo (o).Si encuentra una subcadena de tamaño n, la retorna; de lo contrario, continúa buscando con el doble de tamaño.La notación refleja la lógica de la función, donde la recursión juega un papel importante en la generación progresiva de subcadenas.

reconstruirCadenaTurboPar:

Técnicas de paralelización:
La paralelización se implementa en la función generarCadenaTurbo mediante el método par aplicado a las colecciones subcadenasActuales y nuevasSubcadenas. Esto distribuye la generación y filtrado de subcadenas en múltiples hilos para procesar de manera simultánea diferentes partes del trabajo, mejorando la eficiencia en entornos paralelos.

Impacto en el desempeño:
Sin embargo, debido a la naturaleza recursiva del algoritmo, la eficiencia de la paralelización puede verse limitada, ya que cada iteración depende del resultado de las anteriores, generando una posible alta latencia de comunicación entre los hilos.
Aunque la paralelización puede ser beneficiosa en escenarios con operaciones intensivas, su impacto efectivo en el rendimiento depende del tamaño de n, la complejidad de las operaciones y el hardware donde se ejecute el programa.
Evaluación comparativa

Secuencial	Paralela	Aceleración	Tamaño
0.2312	0.1898	1.2181243414120126	2
0.2673	0.201	1.3329850746268665 66	4
0.2005	3.1692	0.6326517733181876	8
0.9908	5.364	0.1847129008202833 8	16

En términos generales, la ejecución paralela muestra una mejora en el tiempo de ejecución en comparación con la secuencial, especialmente para tamaños pequeños de entrada (n=2 y n=4). Sin embargo, a medida que el tamaño de entrada aumenta (n=8 y n=16), la ejecución paralela puede volverse más costosa en términos de tiempo, lo que se refleja en la aceleración que disminuye en estos casos. Estos resultados sugieren que la eficacia de la paralelización puede depender del tamaño específico del problema y las características del hardware utilizado.

Turbo mejorada:

Estructuras de Datos Utilizadas:
Secuencias (Seq): Se utilizan para representar las cadenas de ADN. En la función principal TurboMejorado, se manejan las secuencias de caracteres que representan la cadena de ADN y las subcadenas generadas.
Parámetros y Variables Locales: Se utilizan variables locales para almacenar resultados intermedios y parámetros de la función.

Argumentaciones sobre la Corrección:

- Verificación de Subsecuencias (verificarSecuencias): La función verifica si una secuencia es subsecuencia de otra. Se realiza mediante recursión y comparación de subcadenas.

- Generación de Subcadenas Válidas (subcadenasTurbo): La función utiliza recursión para generar subcadenas válidas a partir de subcadenas anteriores y las verifica utilizando el oráculo.

- Manejo de Casos Especiales: Se manejan casos especiales, como cuando el tamaño de la secuencia es menor que 1, devolviendo una cadena vacía.

- Uso de headOption: Se utiliza headOption para manejar el caso en que la secuencia es vacía, evitando posibles errores.

Turbo mejoradaPar:

Estructuras de Datos Utilizadas:

- Secuencias Paralelas (ParSeq): Se utilizan para representar las subcadenas generadas de manera paralela. Esto permite realizar operaciones en paralelo, mejorando el rendimiento.

- Parámetros y Variables Locales: Se utilizan variables locales para almacenar resultados intermedios y parámetros de la función, al igual que en la solución secuencial.

Argumentaciones sobre la Corrección:

- Verificación de Subsecuencias Paralela (verificarSecuencias): La función es similar a la versión secuencial, pero ahora opera sobre ParSeq, permitiendo la paralelización de las operaciones.

- Generación de Subcadenas Válidas Paralela (subcadenasTurbo): La función es similar a la versión secuencial, pero ahora las operaciones sobre las subcadenas anteriores se realizan de manera paralela.

- Manejo de Casos Especiales: Se mantiene el manejo de casos especiales de la solución secuencial.

- Uso de Paralelización Técnica: Se utiliza la técnica de paralelización de datos (ParSeq) cuando la longitud de la cadena es mayor que el umbral definido. Esto evita la paralelización innecesaria para tareas pequeñas.

Impacto de las Técnicas de Paralelización:

- Tareas Paralelas: La paralelización se aplica en la generación y verificación de subcadenas. Al realizar operaciones en paralelo, se aprovecha el poder de procesamiento multicore, mejorando el rendimiento en tareas intensivas.

- Umbral de Paralelización: Se ha establecido un umbral para decidir cuándo aplicar la paralelización. Esto evita el sobrecosto asociado con la paralelización de tareas pequeñas, ya que el rendimiento paralelo puede ser menor en estos casos.

- Reducción de Tiempo de Ejecución: La paralelización debería tener un impacto positivo en el desempeño del programa al reducir el tiempo de ejecución, especialmente cuando se enfrenta a tareas de tamaño significativo.

Evaluación comparativa de funciones Turbo Mejorada y Turbo Mejorada Par

Secuencial	Paralela	Aceleración	Tamaño
0.0802	0.051001	1.5725181859179231	2
0.0467	0.071899	0.6495222464846521	4
0.0859	0.4435	0.1936865839909808	8
0.2448	0.927299	0.2639925202119273	16
1.017	1.713601	0.5934870486186691	32
10.4311	6.310701	1.6529225517101827	64
98.5614	45.1691	2.182053660577696	128
966.9451	378.412199	2.555269366461413	256

Los resultados obtenidos al comparar las funciones secuencial y paralela revelan un patrón claro en cuanto al rendimiento y la eficiencia de ambas implementaciones.
A medida que el tamaño de la entrada aumenta, se observa que la implementación paralela tiende a superar significativamente a la secuencial en términos de eficiencia. En tamaños pequeños, la paralelización puede introducir un cierto costo adicional debido a la sobrecarga asociada, pero a medida que la complejidad del problema crece, la ventaja de dividir las tareas entre múltiples núcleos se vuelve evidente.

Implementación trie

El código proporcionado implementa un Trie, una estructura de árbol que representa una colección de cadenas, donde cada nodo representa un carácter. En este caso, se ha definido un Trie con dos tipos de nodos: Nodo y Hoja.

Nodo: Representa un nodo interno en el Trie, con un carácter, un marcador que indica si representa el final de una cadena y una lista de hijos.
Hoja: Representa un nodo terminal en el Trie, también con un carácter y un marcador.

Funciones Principales:

1. pertenece: Esta función verifica si una subsecuencia está presente en el Trie. Utiliza una función interna (perteneceEnCabezas) para buscar recursivamente en la lista de cabezas de los nodos.
Cuando encuentra la subsecuencia vacía, retorna true indicando que está presente en el Trie.

2. adicionar: Agrega una secuencia al Trie existente. Utiliza una función interna (agregarRama) para recorrer el Trie y agregar los nodos correspondientes.
Maneja casos para nodos existentes, hojas, nodos con hijos y otros escenarios.

3. arbolDeSufijos: Crea un árbol de sufijos a partir de una secuencia de secuencias. Utiliza foldLeft para agregar cada secuencia al árbol de sufijos mediante la función adicionar.
Corrección de las Funciones:

Corrección de pertenece: Utiliza recursión para buscar la presencia de la subsecuencia en el Trie y maneja correctamente el caso base cuando la subsecuencia está vacía.

Corrección de adicionar: Agrega nodos correctamente, considerando nodos existentes, hojas, nodos con hijos, entre otros casos, preservando la estructura del Trie.

Corrección de arbolDeSufijos: Usa la función adicionar para construir un Trie con los sufijos proporcionados, asegurando que cada sufijo esté presente en el Trie resultante.

reconstruirCadenasTurboAcelerada:

Estructura de Datos Utilizada:

Seq[Seq[Char]]: Se emplea para representar secuencias de caracteres.

ArbolSufijos.Trie: Es una estructura Trie utilizada para manejar y buscar subcadenas.

Manejo de Recursión:

La función generarSecuenciasValidas hace uso de la recursión para buscar y generar las secuencias válidas.

Reconocimiento de Patrones:

Se emplea patrones en la función generarSecuenciasValidas para validar la longitud de las secuencias.

Mecanismos de Encapsulación:

La lógica para generar secuencias válidas se encapsula dentro de la función generarSecuenciasValidas.

Funciones de Alto Orden:

Se utilizan funciones de alto orden, como foldLeft y filter, para operar sobre colecciones y realizar operaciones en secuencias.

Iteradores, Colecciones y Expresiones For:

Se emplean expresiones for para generar nuevas secuencias basadas en las secuencias existentes.
Utiliza colecciones como Seq y operadores para almacenar y manipular secuencias.
Itera sobre las secuencias empleando expresiones for y generando nuevas secuencias válidas.

Argumentaciones sobre su implementación:

La función busca generar eficientemente secuencias válidas utilizando la lógica de un Trie, permitiendo búsquedas rápidas y eficientes de subcadenas.
El uso de la recursión permite explorar y construir secuencias de manera estructurada y eficiente.
La función hace un buen uso de las estructuras de datos y las operaciones sobre ellas para lograr su objetivo de reconstruir cadenas.
Esta función proporciona una manera eficiente de reconstruir cadenas a partir de un oráculo, utilizando estructuras de datos y estrategias recursivas para generar y validar secuencias válidas.

Notación matemática:

reconstruirCadenaTurboAcelerada(n,o) = {SecuenciaVacia si n<1, GenerarSecuenciasValidas(SecuenciaInicial, ArbolInicial, 1) en otro caso}

Donde: -SecuenciaVacia es una secuencia vacía cuando n<1
-SecuenciaInicial es una secuencia de caracteres filtrados por el oráculo o y luego convertidos en secuencias individuales de un solo carácter.
-ArbolInicial es un árbol de sufijos generado a partir de la SecuenciaInicial

La función GenerarSecuenciasValidas es una función interna que busca generar secuencias válidas de longitud n a partir de las secuencias iniciales y el árbol de sufijos. Esta función se define de manera recursiva, explorando y construyendo nuevas secuencias válidas hasta alcanzar una de longitud n o no encontrar más secuencias válidas.

reconstruirCadenaTurboAceleradaPar:

Paralelización de Tareas:

La generación de nuevas secuencias válidas se realiza en paralelo utilizando ParSeq.

Se aplica paralelismo al filtrar y evaluar las combinaciones de subsecuencias usando el oráculo.

La recursión en la generación de secuencias se ejecuta en paralelo con ParSeq.

Razones para Esperar un Impacto Positivo en el Desempeño:

Aprovechamiento de Recursos Multi-Core:

La paralelización distribuye tareas en múltiples núcleos de CPU, si están disponibles, para acelerar la ejecución.

Reducción del Tiempo de Ejecución:

Al ejecutar tareas en paralelo, se espera una significativa reducción del tiempo total de ejecución.

Optimización de Recursos:

Se usa paralelización solo cuando n>umbral para evitar sobrecargas innecesarias en casos donde el beneficio puede ser mínimo.

Aumento de Escalabilidad:

La implementación paralela hace que el algoritmo sea más escalable, manejando eficientemente volúmenes de datos más grandes.

Impacto en el Desempeño del Programa:

Dependencia de Parámetros:

El rendimiento depende del tamaño de entrada (n), del umbral definido y de la capacidad de paralelización del hardware.

Mejora con n Grande:

Se espera un mejor rendimiento para n grande y umbrales altos debido al aprovechamiento del paralelismo.

Posible Sobrecarga para Valores Pequeños:

Para n pequeño y umbrales bajos, la paralelización puede generar sobrecarga y no mejorar el rendimiento o incluso empeorarlo.

Evaluación comparativa:

Secuencial	Paralela	Aceleración	Tamaño
0.1525	0.0412	3.7014563106796117	2
0.04	0.0281	1.4234875444839858	4
0.1696	1.2166	0.1394048988985698	8
0.1654	1.9163	0.08631216406616918	16
0.9125	3.024	0.30175264550264547	32
7.5154	5.836	1.28776559287183	64
78.646	18.0981	4.3455390344842835	128
741.7543	140.0929	5.294731567409912	256

Se observa en la aceleración que, la mayoría son mayores que 1, lo que indica que la versión paralela es más rápida que la secuencial. Esto sugiere que para tamaños de entrada más grandes, la implementación paralela se vuelve más eficiente.

En cuanto al umbral, el análisis de rendimiento indica que para tamaños de entrada menores o iguales a 8, la versión secuencial es más rápida. Sin embargo, para tamaños de entrada mayores a 8, la versión paralela empieza a ser más eficiente.