Camilo Valencia Romero – 2259497

Cristian David Rivera - 2259742

Justificación de las Funciones:

Función 'generarSubcadenas' y 'ingenua'

La función 'generarSubcadenas' tiene como objetivo generar todas las posibles subcadenas de longitud 'n' a partir de un alfabeto dado. Utiliza una estructura de datos que consiste en una secuencia de secuencias de caracteres ('Seq[Seq[Char]]'). La recursión se implementa mediante la función interna 'subcadenasIter', que toma dos parámetros: 'subs', que representa las subcadenas generadas hasta el momento, y 'n', que indica la longitud actual de las subcadenas en construcción. La recursión se detiene cuando 'n' supera la longitud deseada y devuelve las subcadenas generadas hasta ese punto.

En cuanto a las estructuras de datos utilizadas, se hace uso de la función 'flatMap' para combinar las subcadenas existentes con cada caracter del alfabeto, generando así nuevas subcadenas. La función 'map' se utiliza para construir las subcadenas iniciales, cada una formada por un solo caracter del alfabeto.

La función 'ingenua' busca la primera subcadena que cumple con un oráculo dado. Se basa en la función 'generarSubcadenas' para obtener todas las posibles subcadenas de longitud 'n-1'. La estructura de datos utilizada es una secuencia de caracteres ('Seq[Char]'). La función 'find' busca la primera subcadena que satisface el oráculo, y la función 'head' devuelve dicha subcadena.

Descripción de la Función 'mejorada'

La función 'mejorada' implementa un algoritmo de búsqueda llamado "PRC (Pattern Recognition Code)" optimizado. Su objetivo es encontrar subcadenas que satisfacen un oráculo dado. A continuación, se describen las estructuras de datos utilizadas y se analiza por qué la función es correcta.

1. Implementación y Justificación:

- La función 'encontrarSubcadenasAux' maneja de manera eficiente la generación y filtrado de subcadenas, utilizando la función 'filter' para seleccionar aquellas que cumplen con el oráculo.

- La recursión se detiene cuando la lista de subcadenas (`sck`) está vacía, devolviendo una lista vacía.

- Se generan nuevas combinaciones de subcadenas y caracteres del alfabeto, y se filtran según el oráculo.

- Si alguna de las combinaciones tiene la longitud requerida (`n`), se devuelve esa combinación.

- En caso contrario, la función se llama recursivamente con las nuevas combinaciones.

- La función principal llama a la función auxiliar con la lista inicial de subcadenas y devuelve la primera subcadena encontrada que satisface el oráculo.

Descripción de la Función 'turbo'

La función 'turbo' implementa un algoritmo de búsqueda optimizado que encuentra subcadenas que cumplen con un oráculo dado. A continuación, se describen las estructuras de datos utilizadas y se analiza por qué la función es correcta.

1. Implementación y Justificación:

- La función define una función auxiliar interna llamada 'encontrarSubcadenasVelozAux', que es recursiva y eficiente.

- Se utilizan condiciones para determinar el flujo del algoritmo. La lógica se basa en generar y combinar subcadenas de manera estratégica para reducir el espacio de búsqueda.

- Si la longitud actual de las subcadenas alcanza `n`, se devuelve la lista de subcadenas. Si la longitud es menor que `n`, se realizan combinaciones y filtrados de subcadenas.

- Se utilizan diferentes estrategias de combinación según la longitud de las subcadenas, optimizando así el rendimiento del algoritmo.

- La función principal decide cómo proceder según la paridad de `n` y llama a la función auxiliar con las subcadenas correspondientes.

Entendido, puedo proporcionar una descripción más detallada sin utilizar notación matemática:

Función filtro_cadenas

- Descripción:

  - Esta función busca determinar si una cadena cumple con ciertos criterios al compararla con una lista de subcadenas de longitud `n`.

- Lógica:

  - La función tiene una función auxiliar interna llamada `incluye` que verifica si una cadena está incluida en una lista de subcadenas.

  - Si la longitud de la cadena es igual a `n`, se comprueba si la cadena está incluida en la lista de subcadenas.

  - Si la longitud es diferente, se toma un segmento de longitud `n` de la cadena y se comprueba si está incluido en la lista de subcadenas. Luego, la función se llama recursivamente con el resto de la cadena.

Función turboMejorada

- Descripción:

  - Esta función implementa un algoritmo de búsqueda optimizado para encontrar subcadenas que cumplen con un oráculo dado.

- Lógica:

  - La función utiliza una función auxiliar llamada `turboMejoradaAux` de forma recursiva para encontrar las subcadenas.

  - La lógica de la función se divide en casos:

    - Si la longitud de las subcadenas más la unidad es igual a `n`, se combinan y filtran las subcadenas, y se devuelve el resultado.

    - Si la longitud de las subcadenas es mayor o igual a `n`, se devuelven las subcadenas sin cambios.

    - Si el doble de la longitud de las subcadenas es mayor que `n`, se combinan y filtran las subcadenas utilizando la lista de subcadenas de la iteración anterior.

    - En otros casos, se combinan y filtran las subcadenas utilizando la lista de subcadenas actual.

  - La función principal decide cómo proceder según la cantidad de `n` y llama a la función auxiliar con las subcadenas correspondientes.

Función turboAcelerada

La función turboAcelerada implementa un algoritmo de búsqueda de subcadenas optimizado que aprovecha la paralelización en la reconstrucción de secuencias. A continuación, se proporciona una descripción detallada sin notación matemática.

Descripción General

La función turboAcelerada busca subcadenas que cumplen con un oráculo utilizando un enfoque optimizado que involucra la reconstrucción paralela de secuencias.

Estructuras de Datos Utilizadas

- La función utiliza la estructura de datos de un árbol de sufijos (Trie) para representar las secuencias de manera eficiente.
- Se hace uso de las funciones auxiliares turboAceleradaAux y evaluar_arbol para la manipulación y evaluación del árbol de sufijos.

Implementación y Lógica

- La función principal turboAcelerada inicia con la creación del árbol de sufijos inicial, generado a partir de las subcadenas de longitud 1 del alfabeto.
- La función hace uso de la recursión y patrones de concordancia para actualizar el árbol de sufijos de manera eficiente.
- Se emplea la función turboAceleradaAux para reconstruir secuencias de manera paralela, y la función evaluar_arbol para evaluar y actualizar el árbol según ciertos criterios.
- La función se divide en casos según la paridad de n y realiza combinaciones estratégicas para optimizar la búsqueda.

a. Función raíz:

Devuelve el carácter de la raíz de un Trie.

Utiliza patrones de concordancia para manejar tanto nodos Nodo como nodos Hoja.

b. Función cabezas:

Devuelve una secuencia de caracteres correspondientes a los hijos directos de un Trie.

Utiliza patrones de concordancia para manejar nodos Nodo y nodos Hoja.

c. Función agregar:

Agrega una cadena al árbol de sufijos de manera eficiente y correcta.

Maneja casos donde la cadena ya existe, se deben crear nuevos nodos o simplemente actualizar la marcación de un nodo.

d. Función agregar_secuencias

Agrega una secuencia de cadenas al árbol de sufijos utilizando la función agregar.

Utiliza recursión para agregar cada secuencia al árbol.

e. Función arbolDeSufijos

Construye un árbol de sufijos a partir de una lista de secuencias de cadenas.

Utiliza la función agregar_secuencias para agregar cada secuencia al árbol inicial.

f. Función pertenece

La función pertenece no se utiliza en el contexto actual del código.

Su lógica de verificación de pertenencia ha sido incorporada directamente en otras partes del código, específicamente en la función filtro_cadenas de turboMejorada, ya que se tienen todas las secuencias en una variable se aprovecha para realizar el filtro con esta variable.

Análisis paralelismo

Tamaño	Pruebas	Ingenua	IngenuaPar	Mejorada	MejoradaPar	Turbo	TurboPar	TurboMejorada	TurboMejoradaPar	TurboAcelerada	TurboAceleradaPar
2	Tiempo	76.2032	27.465	4.5341	19.2278	3.1083	0.3526	9.6503	0.9443	6.2595	2.4216
3	Tiempo	15.3089	11.5708	0.744	0.4368	10.3585	19.5267	3.8644	3.2643	4.2232	5.0849
4	Tiempo	6.1702	1.6566	0.5903	1.3604	0.6398	2.105	0.2451	5.1794	0.252	0.1485
5	Tiempo	21.5468	4.0291	0.2296	0.4895	30.367	0.5158	1.1294	0.6362	53.7244	170.8245
6	Tiempo	119.3076	9.9079	0.313	4.8739	0.3286	1.5988	3.1491	5.1739	0.4315	12.7943
7	Tiempo	190.615	22.6399	1.942	1.6717	0.1419	0.1851	0.2941	0.266	0.9043	5.9427
8	Tiempo	422.9264	164.6735	0.4431	1.042	0.2911	1.0337	0.4506	0.727	1.836	8.7326
9	Tiempo	2302.6663	250.9029	0.8151	2.5093	0.3398	0.3176	1.4686	0.2544	1.7916	7.4215
10	Tiempo	7410.0009	1660.3543	0.2096	0.4217	0.1729	0.3972	0.2708	0.5865	0.5442	9.3259
11	Tiempo	58679.2227	6825.6605	0.295	0.8255	0.2952	0.1975	0.5182	0.1991	4.856	39.3215
12	Tiempo	64135.9787	67315.6231	0.2576	53.3282	0.1517	0.519	0.3774	0.4604	0.6723	11.0065
totales	0	64150.1696	63509.2621	0.7601	0.6219	1.686	0.5125	1.1972	0.4264	2.1589	12.5134

Análisis del Paralelismo en las Funciones:

Análisis de la Función IngenuaPar

Análisis de la Función IngenuaPar:

- **Estructura de la Función:**
  - La función divide el conjunto de subcadenas (**resultado**) en dos partes, **c1** y **c2**, utilizando la función **parallel**.
  - Luego, realiza la búsqueda del oráculo en ambas mitades en paralelo utilizando la función **find**.
  - Finalmente, selecciona la primera secuencia encontrada entre ambas mitades.
- **Razones de la Mejora:**
  - La mejora observada puede atribuirse a la ejecución en paralelo de la búsqueda del oráculo en dos conjuntos de datos separados.
  - En comparación con la versión secuencial (**Ingenua**), esta versión paralela tiene el potencial de reducir el tiempo total de ejecución, especialmente cuando la operación de búsqueda del oráculo es intensiva en cómputo.

Análisis de la Función `Mejorada`:

- **Estructura de la Función:**
  - La función comienza generando una lista de subcadenas de longitud 1 (`alfabetoEnForma`), donde cada subcadena es un solo carácter del alfabeto.
  - Luego, se llama a la función auxiliar `encontrarSecuenciasParAux` con esta lista de subcadenas.
  - La función `encontrarSecuenciasParAux` opera en dos fases:
    - **Fase 1:** Llamada a la función `encontrarSecuenciasMSubs` en paralelo en dos mitades de la lista original. Cada llamada combina las subcadenas con un carácter del alfabeto y filtra aquellas que cumplen con el oráculo.
    - **Fase 2:** Combina los resultados paralelos (`c1` y `c2`) y filtra las secuencias vacías. Si hay alguna secuencia de longitud `n`, devuelve ese conjunto de secuencias. De lo contrario, realiza una llamada recursiva.
- **Razones de la Mejora:**
  - La mejora se basa en la paralelización de la búsqueda de secuencias optimizadas en dos conjuntos de datos separados.
  - La función trabaja con secuencias más pequeñas y utiliza paralelismo para evaluar diferentes combinaciones simultáneamente, lo que podría conducir a un rendimiento mejorado.
- **Datos Irregulares:**

- Los datos de rendimiento son irregulares y podrían deberse a la naturaleza de las operaciones paralelas, la carga de trabajo, o el tamaño del conjunto de datos en cada ejecución.
- Tambien puede deberse a que el proceso de combinacion sea ligero o el proceso de union es pesado o a una suma de ambos.

Análisis de la Funciónes Turbo:

Paralelismo en Turbo:

- La función **Turbo** utiliza la paralelización mediante el uso de tareas (**task**) en las fases que requieren búsqueda y combinación de subcadenas.
- La versión paralela (**TurboPar**) realiza las operaciones de búsqueda en paralelo para optimizar el tiempo de ejecución.

Analisis Comparativa:

- La versión paralela puede incluir complejidades adicionales, y si la tarea es lo suficientemente simple o no tiene suficiente trabajo para justificar la paralelización, podría resultar más eficiente en su versión secuencial.
- Tambien se puede deber a lo dicho antes pero amplificado en este caso, ya que la funcion turbo es muy rapida, la creacion de hilos no faborese su ejecucion.
- La fucion mejorada tambien sufre de lo mismo pero peor. ya que el algoritmo turbo mejorado es mucho mas rapido que su parte turbo y aun asi la creacion de hilos perjudica mucho mas sus tiempos de ejecucion.
- La funcion turboAceleradaPar al utilizar Parvector hace muchas mas divisiones de hilos haciendo ver mas este problema.

Conclusiones Generales:

La implementación y análisis de algoritmos para la búsqueda de subcadenas revela aspectos interesantes sobre el rendimiento y la eficacia de diferentes enfoques. A continuación, se resumen las observaciones clave y conclusiones derivadas de las distintas implementaciones y sus evaluaciones.

- **Eficiencia de la Versión Ingenua:**
  - La versión ingenua, que busca todas las subcadenas y selecciona la primera que satisface el oráculo, es sencilla pero puede ser costosa en términos computacionales, especialmente para tamaños de datos más grandes. La paralelización de esta versión no siempre ofrece mejoras significativas debido a posibles cargas ligeras de trabajo y al overhead asociado con la gestión de tareas.
- **Mejoras mediante la Versión Mejorada:**
  - La versión mejorada introduce una estrategia más eficiente al generar subcadenas de manera incremental, evitando la generación completa de todas las subcadenas posibles. La paralelización en este caso demuestra ser beneficiosa al dividir la tarea en partes más pequeñas y procesarlas concurrentemente.

- **Rendimiento de la Versión Turbo:**
    - La versión turbo, al forzar la longitud de la subcadena a ser impar y realizar combinaciones eficientes, muestra mejoras sustanciales en comparación con las implementaciones anteriores. Sin embargo, la versión paralela de esta implementación, a pesar de tener beneficios en algunos casos, puede no ser siempre más eficiente debido a la complejidad añadida y al manejo de datos más pequeños.
- **Importancia del Saber Cuándo Paralelizar:**
    - El rendimiento de los algoritmos paralelos no solo depende de la naturaleza de la tarea, sino también de factores como el tamaño de los datos y las características del hardware subyacente. Es crucial considerar estos elementos al diseñar e implementar soluciones paralelas.
- **Balance entre Paralelización y Complejidad:**
    - La paralelización puede no ser siempre beneficiosa, especialmente para tareas más simples o conjuntos de datos pequeños. La complejidad adicional y el overhead introducido pueden superar los beneficios potenciales de la ejecución paralela.