Miguel Angel Ceballos Yate 2259619

Cristhian leonardo Albarracin zapata 1968253

Nicolás Gutiérrez Ramírez 2259515

Justificación de las Funciones:

Función 'gemerarCombinaciones' y 'Ingenuo'

La función 'gemerarCombinaciones' se encarga de producir todas las subcadenas posibles de longitud 'n' a partir de un alfabeto específico. Esta función emplea una estructura de datos que es una secuencia de secuencias de caracteres ('Seq[Seq[Char]]'). La recursión se lleva a cabo a través de la función interna 'subcadenasIter', que acepta dos argumentos: 'subs', que representa las subcadenas generadas hasta ahora, y 'n', que indica la longitud actual de las subcadenas en proceso. Cuando 'n' excede la longitud deseada, la recursión se detiene y se devuelven las subcadenas generadas hasta ese punto.

En términos de las estructuras de datos utilizadas, se utiliza la función 'flatMap' para combinar las subcadenas existentes con cada carácter del alfabeto, generando así nuevas subcadenas. La función 'map' se utiliza para construir las subcadenas iniciales, cada una compuesta por un solo carácter del alfabeto.

La función 'Ingenuo' busca la primera subcadena que cumple con un oráculo dado. Se basa en la función 'gemerarCombinaciones' 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.

La función 'Mejorado' implementa un algoritmo de búsqueda más 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:

Notación

gemerarCombinaciones:$C*\times\mathbb{N}\rightarrow C*$

Ingenuo()=find(gemerarCombinaciones(alfabeto,$n-1$),oraculo)

Descripción de la Función 'Mejorado'

Mejorado()=encontrarSubcadenasAux(subcadenas,oraculo)[0]

encontrarSubcadenasAux(*sck*,oraculo)=

$$\begin{cases} \{\}, \text{ si } sck=\{\} \\ c, \text{ si } \exists c \in \text{combinaciones} \mid \text{longitud}(c)=n \\ \text{encontrarSubcadenasAux(combinaciones,oraculo), en otro caso} \end{cases}$$

- 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:

Turbo=

$$\begin{cases} \text{alfabetoEnForma.filter(oraculo(\_)).head}, & \text{si } n=1 \\ \text{uniones.head}, & \text{si } n \text{ es par} \\ \text{uniones.head}, & \text{si } n \text{ es impar} \end{cases}$$

encontrarSubcadenasVelozAux(*sck*,*n*,combinacion_anterior)=

$$\begin{cases} \text{uniones.filter(oraculo(\_))}, & \text{si } sck\text{.head.size} + 1 = n \\ sck, & \text{si } sck\text{.head.size} \geq n \\ \text{encontrarSubcadenasVelozAux(uniones, n, sck\_anterior)}, & sck\text{.head.size} * 2 > n \\ \text{encontrarSubcadenasVelozAux(uniones, n, sck\_ anterior)}, & \text{en otros casos} \end{cases}$$

- 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.

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 Turbo_mejorada

- Implementación:

-Igual que la función Turbo, solo que con un filtro de verificación de subcadenas, para eliminar las subcadenas que no pertenezcan a la lista anterior, para 'ahorrar' intentos del oráculo

- Descripción:

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

- Lógica:

- 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.

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 **turboAcelerada** se implementa de la siguiente manera:

turboAcelerada()=

$$\begin{cases} \text{head(combinar\_dos\_listas(sub\_cadenas,sub\_cadenas).filter(oraculo)}, \text{ si } n \text{ es par} \\ \text{head(combinar\_dos\_listas(c1,c2).filter(oraculo))}, \text{ si } n \text{ es impar} \end{cases}$$

Donde:

- arbolDeSufijos:{alfabeto}→Trie es una función que construye el árbol de sufijos inicial a partir de las secuencias de un solo carácter del alfabeto.
- cadenas_del_arbol:Trie→{subcadenas} es una función que extrae todas las secuencias almacenadas en el árbol de sufijos.
- turboAceleradaAux:Trie×{subcadenas}×{subcadenas}→Trie es una función auxiliar que realiza la reconstrucción de secuencias en paralelo y actualiza el árbol de sufijos.
- filtro_cadenas:{subcadenas}×{subcadenas}×Int→Boolean es una función que filtra las cadenas según ciertos criterios.

- 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 Turbo_mejorada, ya que se tienen todas las secuencias en una variable se aprovecha para realizar el filtro con esta variable.

Análisis paralelismo

Tabla uno paralelismo con Parvector:

Tamaño	Pruebas	Ingenuo	IngenuoPar	Mejorado	Turbo_mejorada	Turbo	TurboPar	Turbo_mejorada	Turbo_mejoradaPar	turboAceleradaPar	turboAceleradaPar
2	Tiempo	4,3037	29,345	2,7934	4,6607	1,6513	26,2556	1,1288	6,69	5,2479	29,1408
3	Tiempo	1,5645	3,3989	0,5981	2,1319	1,8472	8,9198	1,7669	7,6836	1,798	7,9196
4	Tiempo	3,1266	3,8587	0,6157	2,4303	0,3403	5,5478	0,3446	5,2107	0,3831	5,0867
5	Tiempo	5,3722	5,222	0,3103	2,2658	1,2426	7,8261	2,2318	8,5635	6,134	12,6504
6	Tiempo	10,6444	13,648	0,359	1,9494	0,366	4,6216	0,6982	5,5782	1,0323	6,4694
7	Tiempo	25,6822	24,0351	0,3834	3,5994	0,4709	6,6062	0,8266	8,8011	1,3191	9,8354
8	Tiempo	56,8335	30,2801	0,389	1,9296	3,5945	5,9672	0,5692	6,4062	0,9236	9,2276
9	Tiempo	183,0083	169,8932	0,3367	2,6341	0,7608	9,2385	0,8268	12,1371	1,1292	88,7479
10	Tiempo	915,9913	471,1452	0,3076	2,0944	0,4205	5,722	5,9263	5,7575	3,7353	9,7868
11	Tiempo	3186,9754	2933,6438	0,2999	2,482	1,0651	8,2216	1,8344	10,6357	9,5033	15,8222

12	Tiempo	28324,2253	253919,8925	0,2497	2,0023	0,2288	3,6071	0,4165	4,15	0,7214	9,1487
13	Tiempo	45373,2704	217367,6604	0,3629	2,3753	1,1083	11,5604	5,9879	25,7726	15,5091	68,8388
14	Tiempo	42764,8366	264438,434	0,4296	2,3852	0,3489	6,092	0,8705	5,3527	1,0218	6,5578
total		120855,8344	739410,4569	7,4353	32,9404	13,4452	110,1859	23,4285	112,7389	48,4581	279,2321

Tabla 2 paralelismo por division:

Tamaño	Pruebas	Ingenuo	IngenuoPar	Mejorado	Turbo_mejorada	Turbo	TurboPar	Turbo_mejorada	Turbo_mejoradaPar	turboAceleradaPar	turboAceleradaPar
2	Tiempo	32,6587	11,5614	3,0893	3,8826	2,3398	11,0567	1,1895	1,5781	6,09	1,4482
3	Tiempo	1,8734	2,1492	0,7878	0,4253	2,1572	1,8261	1,6073	1,6019	1,9045	1,7958
4	Tiempo	5,029	3,4639	0,6307	0,5577	0,5828	2,5957	0,3227	2,238	0,3162	2,3986
5	Tiempo	11,1184	4,1612	0,5016	1,0845	2,8122	1,4879	1,8399	2,2595	8,1661	3,3577
6	Tiempo	9,3921	7,1808	0,4763	0,6269	0,448	0,4681	0,3793	0,633	0,7157	0,9357
7	Tiempo	33,8336	25,1595	0,3475	4,2102	0,4154	0,5487	0,7229	1,6954	2,5331	1,3096
8	Tiempo	99,1977	56,7337	0,4573	0,7491	0,5588	0,4556	0,3267	0,5961	0,8619	1,1022
9	Tiempo	193,0761	193,9132	0,3559	0,8378	3,7373	0,5133	0,7683	0,8369	4,7722	1,7726
10	Tiempo	645,8096	816,5816	0,2495	0,9816	0,3167	1,413	0,6274	0,892	0,6239	0,7458
11	Tiempo	2369,0252	2177,5048	0,3257	1,0439	1,0741	1,138	2,4888	2,4129	7,4	6,6472

12	Tiempo	30235,6947	28468,0434	0,3505	1,121	0,2279	0,4078	0,4435	0,7449	0,9106	1,226
13	Tiempo	37965,0797	44346,2972	0,3809	1,2037	0,5453	0,7108	1,2016	1,0067	2,1505	1,3755
14	Tiempo	39138,0797	30517,5319	0,3557	1,0082	0,274	0,6009	0,4241	0,6061	0,7606	1,0887
15	Tiempo	30785,7457	30238,8478	0,3436	1,2709	0,5192	0,6477	0,9541	0,7061	1,6233	1,3278
16	Tiempo	41523,019	39824,8344	0,4509	1,1275	0,3686	0,8377	0,6741	0,9662	0,9803	1,254
17	Tiempo	29859,389	42957,0515	0,5848	1,6304	0,5976	0,7992	1,1814	0,8953	2,7044	2,8825
18	Tiempo	32798,151	36949,4076	0,4845	1,3252	0,3973	0,9192	0,6874	1,0285	0,7937	0,9909
total		245706,1726	256600,4231	10,1725	23,0299	17,3722	26,4264	15,8396	20,6976	43,307	31,6588

Análisis del Paralelismo en las Funciones:

Evaluación de la Función IngenuaPar:

La estructura inicial de la función implica la sustitución de Seq por ParVector en todas las funciones, manteniendo la implementación sin cambios adicionales, ya que la adición de más hilos resulta redundante debido a la capacidad de ParVector. La función divide el conjunto de subcadenas resultantes en dos partes, c1 y c2, utilizando la función parallel. A continuación, realiza la búsqueda del oráculo en ambas mitades simultáneamente mediante la función find. Finalmente, selecciona la primera secuencia encontrada entre ambas mitades.

Observaciones:

En un principio, el uso de ParVector parece innecesario, ya que las operaciones son relativamente simples. La ineficiencia de unir múltiples hilos para operaciones tan

básicas se destaca, especialmente dada la naturaleza secuencial de los algoritmos que dependen de datos anteriores en cada ciclo.

Además, los resultados de las pruebas con ParVector pueden atribuirse a la secuencialidad inherente de estos algoritmos, ya que requieren datos de ciclos anteriores, realizan operaciones en orden y se repiten.

La otra función paralela muestra mejoras significativas respecto a la original, pero con el tiempo de ejecución acercándose gradualmente al de la versión secuencial. Esto sugiere que la división debe realizarse de manera más inteligente, considerando el tipo de operaciones y la posible simplicidad de estas.

Análisis de la Función Mejorada:

Estructura de la Función Mejorada:

En la Fase 1, se realiza una llamada paralela a la función encontrarSecuenciasMSubs 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. En la Fase 2, se combinan los resultados paralelos (c1 y c2) y se filtran las secuencias vacías. Si hay alguna secuencia de longitud n, se devuelve ese conjunto de secuencias; de lo contrario, se realiza una llamada recursiva.

Mejoras y Razones:

La mejora se basa en la paralelización de la búsqueda de secuencias optimizadas en dos conjuntos de datos separados, aunque con el tiempo de ejecución acercándose a la parte secuencial. Esto puede deberse a que las operaciones siguen siendo simples y la paralelización no es eficaz en tamaños de datos más pequeños.

La función opera con secuencias más pequeñas y utiliza el paralelismo para evaluar diferentes combinaciones simultáneamente, lo cual debería mejorar el rendimiento, aunque esto no se refleje debido al tamaño limitado de n.

Paralelismo en Turbo:

Las funciones Turbo y TurboMejorada emplean la paralelización mediante tareas (tasks) en las fases que requieren búsqueda y combinación de subcadenas. La parte de ParVector no se considera, ya que su rendimiento ineficiente fue explicado previamente en otras secciones, aunque se sigue observando en esta función.

Comparativa de Rendimiento:

La versión paralela puede introducir complejidades adicionales, y en casos de tareas simples o con poco trabajo, la versión secuencial puede ser más eficiente. Aunque las diferencias no son tan notables en Turbo, la función paralela tiende a ser más constante y rápida en la mayoría de los casos, especialmente en números impares. Sin embargo, se destaca que el umbral para la TurboMejorada debería ser mayor, ya que no mejora el rendimiento en este rango de datos, indicando que la versión secuencial sigue siendo eficiente.

La función Mejorado muestra mejoras con las cadenas pares, pero es más lenta en cadenas impares en comparación con la versión secuencial. La función Turbo intenta abordar este problema sin mucho éxito.

La función TurboAceleradaPar, al usar la misma paralelisacion sufre de algu muy parecido que las demas, en las cadenas impares, parece funcionar mejor en algunos casos impares.

Conclusiones Generales:

El análisis e implementación de algoritmos para la búsqueda de subcadenas revela aspectos intrigantes sobre el rendimiento y la eficacia de distintos enfoques. Las observaciones clave y conclusiones derivadas de las implementaciones y evaluaciones son resumidas a continuación.

Eficiencia de la Versión Ingenua:

La versión Ingenua, que busca todas las subcadenas y selecciona la primera que satisface el oráculo, puede ser computacionalmente costosa, especialmente para tamaños de datos mayores. La paralelización no siempre mejora significativamente debido a cargas de trabajo ligeras 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 incrementalmente, evitando la generación completa de todas las subcadenas posibles. La paralelización resulta 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. Aunque la versión paralela puede usarse en casos impares, la mejor función sería aquella que combine lo mejor de ambas.

Importancia de Saber Cuándo Paralelizar:

Es crucial paralelizar en casos de grandes conjuntos de datos y cuando el algoritmo se presta naturalmente a la paralelización. La función TurboParalela destaca la importancia de calcular cuándo se obtiene rendimiento adicional y cuándo la paralelización es aplicable de manera natural al algoritmo.