

Miguel Angel Ceballos Yate 2259619

Cristhian leonardo Albarracin zapata 1968253

Nicolás Gutiérrez Ramírez 2259515

Justificación de las Funciones:

Función 'generarCombinaciones' y 'Ingenuo'

La función 'generarCombinaciones' 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 'generarCombinaciones' 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

$\text{generarCombinaciones}: C^* \times N \rightarrow C^*$

$\text{Ingenuo}() = \text{find}(\text{generarCombinaciones}(\text{alfabeto}, n-1), \text{oraculo})$

Descripción de la Función 'Mejorado'

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

encontrarSubcadenasAux(*sck*,oraculo)`=`

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

- 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=

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

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

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

- 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()=

$$\begin{cases} \text{head}(\text{combinar_dos_listas}(\text{sub_cadenas}, \text{sub_cadenas}).\text{filter}(\text{oraculo}), & \text{si } n \text{ es par} \\ \text{head}(\text{combinar_dos_listas}(c1, c2).\text{filter}(\text{oraculo})), & \text{si } n \text{ es impar} \end{cases}$$

Donde:

- $\text{arbolDeSufijos}:\{\text{alfabeto}\} \rightarrow \text{Trie}$ es una función que construye el árbol de sufijos inicial a partir de las secuencias de un solo carácter del alfabeto.
- $\text{cadenas_del_arbol}:\text{Trie} \rightarrow \{\text{subcadenas}\}$ es una función que extrae todas las secuencias almacenadas en el árbol de sufijos.
- $\text{turboAceleradaAux}:\text{Trie} \times \{\text{subcadenas}\} \times \{\text{subcadenas}\} \rightarrow \text{Trie}$ es una función auxiliar que realiza la reconstrucción de secuencias en paralelo y actualiza el árbol de sufijos.
- $\text{filtro_cadenas}:\{\text{subcadenas}\} \times \{\text{subcadenas}\} \times \text{Int} \rightarrow \text{Boolean}$ 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`:

Tam año	Prue bas	Ingenu o	IngenuoP ar	Mej orad o	Turb o_m ejor ada	Turb o	Turb oPar	Turb o_m ejor ada	Turb o_m ejor ada	turb oAc eler ada	turboAcel eradaPar
2	Tie mpo	4,3037	29,345	2,79 34	4,66 07	1,65 13	26,2 556	1,12 88		5,24 79	29,1408
3	Tie mpo	1,5645	3,3989	0,59 81	2,13 19	1,84 72	8,91 98	1,76 69	7,68 36	1,79 8	7,9196
4	Tie mpo	3,1266	3,8587	0,61 57	2,43 03	0,34 03	5,54 78	0,34 46	5,21 07	0,38 31	5,0867
5	Tie mpo	5,3722	5,222	0,31 03	2,26 58	1,24 26	7,82 61	2,23 18	8,56 35	6,13 4	12,6504
6	Tie mpo	10,644	13,648	0,35 9	1,94 94	0,36 6	4,62 16	0,69 82	5,57 82	1,03 23	6,4694
7	Tie mpo	25,682	24,0351	0,38 34	3,59 94	0,47 09	6,60 62	0,82 66	8,80 11	1,31 91	9,8354
8	Tie mpo	56,833	30,2801	0,38 9	1,92 96	3,59 45	5,96 72	0,56 92	6,40 62	0,92 36	9,2276
9	Tie mpo	183,00	169,8932	0,33 67	2,63 41	0,76 08	9,23 85	0,82 68	12,1 371	1,12 92	88,7479
10	Tie mpo	915,99	471,1452	0,30 76	2,09 44	0,42 05	5,72 2	5,92 63	5,75 75	3,73 53	9,7868
11	Tie mpo	3186,9	2933,643	0,29 99	2,48 2	1,06 51	8,22 16	1,83 44	10,6 357	9,50 33	15,8222

	Tie	28324,	253919,8	0,24	2,00	0,22	3,60	0,41		0,72	
12	mpo	2253	925	97	23	88	71	65	4,15	14	9,1487
	Tie	45373,	217367,6	0,36	2,37	1,10	11,5	5,98	25,7	15,5	
13	mpo	2704	604	29	53	83	604	79	726	091	68,8388
	Tie	42764,	264438,4	0,42	2,38	0,34	6,09	0,87	5,35	1,02	
14	mpo	8366	34	96	52	89	2	05	27	18	6,5578
							110,		112,		
		12085	739410,4	7,43	32,9	13,4	185	23,4	738	48,4	
total		5,8344	569	53	404	452	9	285	9	581	279,2321

Tabla 2 paralelismo por division:

Ta ma ño	Pru eba s	Ingen uo	Ingenuo Par	Mej ora do	Tur bo_ mej ora da	Tur bo	Turb oPar	Turb o_me jorad a	Turbo_ mejora daPar	turboA celerad a	turboAce leradaPar
	Tie mp	32,65		3,0	3,8	2,3	11,0	1,189			
2	o	87	11,5614	893	26	398	567	5	1,5781	6,09	1,4482
	Tie mp	1,873		0,7	0,4	2,1	1,82	1,607			
3	o	4	2,1492	878	253	572	61	3	1,6019	1,9045	1,7958
	Tie mp			0,6	0,5	0,5	2,59	0,322			
4	o	5,029	3,4639	307	577	828	57	7	2,238	0,3162	2,3986
	Tie mp	11,11		0,5	1,0	2,8	1,48	1,839			
5	o	84	4,1612	016	845	122	79	9	2,2595	8,1661	3,3577
	Tie mp	9,392		0,4	0,6	0,4	0,46	0,379			
6	o	1	7,1808	763	269	48	81	3	0,633	0,7157	0,9357
	Tie mp	33,83		0,3	4,2	0,4	0,54	0,722			
7	o	36	25,1595	475	102	154	87	9	1,6954	2,5331	1,3096
	Tie mp	99,19		0,4	0,7	0,5	0,45	0,326			
8	o	77	56,7337	573	491	588	56	7	0,5961	0,8619	1,1022
	Tie mp	193,0	193,913	0,3	0,8	3,7	0,51	0,768			
9	o	761	2	559	378	373	33	3	0,8369	4,7722	1,7726
	Tie mp	645,8	816,581	0,2	0,9	0,3	1,41	0,627			
10	o	096	6	495	816	167	3	4	0,892	0,6239	0,7458
	Tie mp	2369,	2177,50	0,3	1,0	1,0	1,13	2,488			
11	o	0252	48	257	439	741	8	8	2,4129	7,4	6,6472

	Tie											
	mp	30235	28468,0	0,3	1,1	0,2	0,40	0,443				
12	o	,6947	434	505	21	279	78	5	0,7449	0,9106	1,226	
	Tie											
	mp	37965	44346,2	0,3	1,2	0,5	0,71	1,201				
13	o	,0797	972	809	037	453	08	6	1,0067	2,1505	1,3755	
	Tie											
	mp	39138	30517,5	0,3	1,0	0,2	0,60	0,424				
14	o	,0797	319	557	082	74	09	1	0,6061	0,7606	1,0887	
	Tie											
	mp	30785	30238,8	0,3	1,2	0,5	0,64	0,954				
15	o	,7457	478	436	709	192	77	1	0,7061	1,6233	1,3278	
	Tie											
	mp	41523	39824,8	0,4	1,1	0,3	0,83	0,674				
16	o	,019	344	509	275	686	77	1	0,9662	0,9803	1,254	
	Tie											
	mp	29859	42957,0	0,5	1,6	0,5	0,79	1,181				
17	o	,389	515	848	304	976	92	4	0,8953	2,7044	2,8825	
	Tie											
	mp	32798	36949,4	0,4	1,3	0,3	0,91	0,687				
18	o	,151	076	845	252	973	92	4	1,0285	0,7937	0,9909	
		24570		10,	23,	17,						
tota		6,172	256600,	172	029	372	26,4	15,83	20,697			
l		6	4231	5	9	2	264	9	6	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 Mejorada 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 utilizar ParVector, muestra un rendimiento similar, sugiriendo que una implementación más profunda del código podría mejorar considerablemente la eficiencia de TurboAceleradaPar si se elimina el primer caso de la turboacelerada.

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.