INFORME PFC || PROYECTO FINAL

CARLOS STIVEN RUIZ ROJAS 2259629

KAREN GRIJALBA ORTIZ 2259623

JHONY FERNANDO DUQUE 2259398

Solución Ingenua secuencial (reconstruirCadenaIngenuo):

1. Generación de Secuencias:

La función **generarCadenas(n)** genera todas las secuencias de longitud **n** sobre el alfabeto Σ. En notación matemática, esto se expresa como: $\boldsymbol{generarCadenas}(\boldsymbol{n}) = \{\boldsymbol{s} \in \boldsymbol{Sec}(\boldsymbol{\Sigma}) \mid \boldsymbol{longitud}(\boldsymbol{s}) = \boldsymbol{n}\}$

Aquí, $Sec(\Sigma)$ representa el conjunto de todas las secuencias sobre el alfabeto Σ, y estamos filtrando las secuencias que tienen una longitud específica **n**.

2. Filtrado con el Oráculo:

Luego, la función **reconstruirCadenaIngenuo** toma las secuencias generadas y las filtra usando el oráculo "(o)". La expresión matemática para esto sería: $\boldsymbol{reconstruirCadenaIngenuo}(\boldsymbol{n}, \boldsymbol{o}) = \{\boldsymbol{s} \in \boldsymbol{generarCadenas}(\boldsymbol{n}) \mid \boldsymbol{o}(\boldsymbol{s})\}$. Aquí, estamos seleccionando todas las secuencias "(s)" en el conjunto generado que cumplen con la condición especificada por el oráculo "(o)".

En resumen, la función "reconstruirCadenaIngenuo" crea un conjunto de todas las secuencias de longitud (n) y luego aplica el oráculo (o) para filtrar aquellas que cumplen con la condición.

Solución Ingenua paralelo (reconstruirCadenaParalelo):

1. Generación de Secuencias:

La función **"generarCadenas(n)"** genera todas las secuencias de longitud (n) sobre el alfabeto Σ. En notación matemática, esto se expresa como: $\boldsymbol{generarCadenas}(\boldsymbol{n}) = \{\boldsymbol{s} \in \boldsymbol{Sec}(\boldsymbol{\Sigma}) \mid \boldsymbol{longitud}(\boldsymbol{s}) = \boldsymbol{n}\}$

Aquí, $Sec(\Sigma)$ representa el conjunto de todas las secuencias sobre el alfabeto Σ, y estamos filtrando las secuencias que tienen una longitud específica (n).

2. División y Paralelización:

La función luego divide el conjunto de secuencias en dos partes, y aplica la función **"parallel"** para procesar ambas partes en paralelo. La expresión matemática para esto sería:

$$\boldsymbol{reconstruirCadenaParalelo}(\boldsymbol{n}, \boldsymbol{o}) = \boldsymbol{parallel}(\boldsymbol{cadenas.take}(\boldsymbol{n/2}), \boldsymbol{cadenas.drop}(\boldsymbol{n/2}))$$

Esto representa la división y paralelización de las cadenas según la paridad de (n).

3. Filtrado con el Oráculo:

Después de la paralelización, se aplican las funciones de filtrado para cada parte resultante. Para la primera parte (cuando **(n)** es par), tenemos:

$$\boldsymbol{resultado1} = \boldsymbol{cadena1.flatmap}(\boldsymbol{s} \mapsto \{\ \boldsymbol{s} \quad \boldsymbol{si}\ \boldsymbol{o}(\boldsymbol{s}) \wedge \emptyset\ \ \boldsymbol{si\ no})$$

Y para la segunda parte (cuando \(n\) es impar), es análogo:

$$\boldsymbol{resultado2} = \boldsymbol{cadena2.flatmap}(\boldsymbol{s} \mapsto \{\ \boldsymbol{s} \quad \boldsymbol{si}\ \boldsymbol{o}(\boldsymbol{s}) \wedge \emptyset\ \ \boldsymbol{si\ no})$$

4. Concatenación de Resultados:

Finalmente, se concatenan los resultados de ambas partes:

$$\boldsymbol{reconstruirCadenaParalelo(n, o) = resultado1 \cup resultado2}$$

En resumen, la función **"reconstruirCadenaParalelo"** divide y procesa en paralelo las secuencias generadas y luego filtra y concatena los resultados según la paridad de (**n**).

Solución cadena mejorado secuencial (reconstruirCadenaMejorado):

1. Generación de Cadenas Candidatas:

La función **"generarCadenasCandidatas(k, candidatas) "** genera todas las cadenas candidatas de longitud k que cumplen con la condición del oráculo.

$$\text{generarCadenasCandidatas}(k, \text{candidatas}) = \begin{cases} \text{candidatas}[0] & \text{si } k = \text{n} \\ \text{generarCadenasCandidatas}(\text{k+1}, \text{candidatas'}) & \text{si } k \neq \text{n} \end{cases}$$

Aquí, candidatas' se obtiene expandiendo cada cadena en candidatas con todos los elementos de A que, al ser añadidos al final de la cadena, cumplen con la condición del oráculo.

2. Filtrado con el Oráculo:

La función **"reconstruirCadenaMejorado"** toma las cadenas candidatas generadas y las filtra usando el oráculo o.

$$reconstruirCadenaMejorado(n, o) = generarCadenasCandidatas(k, A)$$

donde A es el alfabeto de caracteres y generarCadenasCandidatas es la función que genera secuencialmente las cadenas candidatas aplicando el oráculo en cada paso y filtrando las cadenas que no cumplen con la condición. La generación se detiene cuando se alcanza la longitud n de la cadena objetivo.

En resumen, la función **"reconstruirCadenaMejorado"** sigue una estructura similar a la función **"reconstruirCadenaIngenuo"**, generando secuencialmente las cadenas candidatas y aplicando el oráculo para filtrar aquellas que cumplen con la condición.

Solución Mejorada Paralela (reconstruirCadenaMejoradoParalela):

1. Generación de Cadenas Candidatas:

La función auxiliar **"generarCadenasCandidatas(k, candidatas)**" genera todas las cadenas candidatas de longitud k que cumplen con la condición del oráculo. Se implementa como en la función secuencial con la diferencia de que en esta realizamos una división en paralelo

2. División en Paralelo:

La función principal reconstruirCadenaMejoradoParalela(n, o) realiza una división en paralelo de las cadenas candidatas generadas. Si n es par, divide las cadenas en dos conjuntos de longitud n/2; si n es impar, divide las cadenas de manera que el primer conjunto tenga longitud (n + 1) /2 y el segundo conjunto tenga longitud (n − 1) /2.

3. Filtrado con el Oráculo y Concatenación:

Después de la división, aplica el oráculo o a cada conjunto de cadenas de manera paralela. Luego, concatena los resultados obtenidos de cada conjunto.

$$reconstruirCadenaMejoradoParalela(n, o) = \{concatenar(filtrar(generarCadenasCandidatas(k, candidatas1)), filtrar(generarCadenasCandidatas(k, candidatas2)))$$

donde:

- **candidatas1** y **candidatas2** son las divisiones de las cadenas candidatas.
- **filtrar** representa la aplicación del oráculo y la eliminación de las cadenas que no cumplen con la condición.
- **concatenar** es la operación de concatenación de secuencias.

En resumen, la función realiza una generación secuencial mejorada de cadenas candidatas y luego divide estas cadenas en dos conjuntos (paralelos) dependiendo de si la longitud n es par o impar. Luego, aplica el oráculo a cada conjunto por separado, filtra las cadenas que cumplen con la condición y finalmente concatena los resultados.

Solución turbo secuencial (reconstruirCadenaTurbo):

1. Generación de Subcadenas:

La función principal **"reconstruirCadenaTurbo"** inicia generando subcadenas mediante la función auxiliar **"construirSubcadena(*k*, subcadena)".** La generación se realiza recursivamente y se detiene cuando k alcanza la longitud n deseada.

$$\text{generarCadenasCandidatas}(k, \text{candidatas}) = \begin{cases} \text{subcadena}[0] & \text{si } k = \text{n} \\ \text{filtrarSubcadenas}(\text{construirSubcadena}(\text{k} \times 2, \text{nuevoSubcadena}), o) & \text{si } k \neq \text{n} \end{cases}$$

donde nuevoSubcadena se obtiene expandiendo cada secuencia en subcadena concatenándola con todas las demás secuencias posibles y filtrarSubcadenas representa la aplicación del oráculo para filtrar las subcadenas que cumplen con la condición.

2. Generación Inicial de Cadenas:

Antes de la generación de subcadenas, se inicia con la generación inicial de cadenas de longitud 1 sobre el alfabeto, luego se llama a la función **"construirSubcadena(*k*, subcadena)".**

En resumen, la función reconstruirCadenaTurbo genera subcadenas de manera exponencial al concatenar secuencias posibles, trabajando únicamente con las subcadenas que cumplen con la condición del oráculo. La generación comienza con un conjunto inicial de secuencias de longitud 1 y continúa duplicando la longitud en cada iteración hasta alcanzar la longitud deseada n. El resultado final es la última subcadena generada que satisface la condición del oráculo.

Solución turbo paralela (reconstruirCadenaTurboParalela):

1. Generación de Subcadenas:

La función principal reconstruirCadenaTurboParalela inicia generando subcadenas mediante la función auxiliar construirSubcadena(k, subcadena). La generación se realiza de manera paralela, dividiendo las subcadenas en dos conjuntos, dependiendo de si k es par o impar. La generación se detiene cuando k alcanza la longitud n deseada.

$$\text{construirSubcadena}(\text{k}, \text{candidatas}) = \begin{cases} \text{subcadena}[0] & \text{si } k = \text{n} \\ \text{parallel}(\text{nuevasCandidatas1}, \text{nuevasCandidatas2}) & \text{si } k \neq \text{n} \end{cases}$$

2. División en Paralelo:

La función divide las subcadenas en dos conjuntos y dependiendo de si k es par o impar. Se realiza la ejecución paralela de las operaciones, mejorando la eficiencia computacional.

3. Filtrado con el Oráculo y Concatenación:

Luego se aplica el oráculo y se concatenan los resultados.

$$resultado = concatenar(filtrarSubcadenas(nuevasCandidatas1, o), filtrarSubcadenas(nuevasCandidatas2, o))$$

donde filtrarSubcadenas representa la aplicación del oráculo para filtrar las subcadenas que cumplen con la condición y concatenar es la operación de concatenación de secuencias.

En resumen, la función **"reconstruirCadenaTurboParalela"** genera subcadenas de manera exponencial, aplicando la ejecución paralela en la generación de nuevas candidatas. La generación comienza con un conjunto inicial de secuencias de longitud 1 y continúa duplicando la longitud en cada iteración hasta alcanzar la longitud deseada n. La ejecución paralela en la función **"construirSubcadena"** permite realizar la operación de construcción de subcadenas en dos conjuntos simultáneamente, mejorando la eficiencia computacional en entornos paralelos.

Solución turbo mejorada secuencial (reconstruirCadenaTurboMejorada):

1. Función Filtrar:

La función **"filtrar"** genera todas las subcadenas posibles a partir de las subcadenas dadas y luego filtra estas subcadenas $filtrar(subCadena, k) = \{s \in \Sigma * \mid esConcatenacionDe(subCadena, s) \wedge o(s)\}$ donde $\Sigma *$ representa todas las secuencias posibles sobre el alfabeto Σ, y esConcatenacionDe(subCadena,s) verifica si s es una concatenación de las subcadenas en subCadena.

2. Construcción de Subcadenas:

La función **"construirSubcadena"** construye las subcadenas resultantes, aplicando la función **"filtrar"** en cada iteración y duplicando k hasta alcanzar la longitud deseada n.

$$\text{construirSubcadena}(\text{k}, \text{candidatas}) = \begin{cases} \text{subcadena}[0] & \text{si } k = \text{n} \\ \text{filtrar}(\text{subCadena}, \text{k}/2) & \text{si } k \neq \text{n} \end{cases}$$

3. Generación Inicial de Cadenas:

Antes de la construcción de subcadenas, se inicia con la generación inicial de cadenas de longitud 1 sobre el alfabeto Σ

En resumen, la función **"reconstruirCadenaTurboMejorada"** genera subcadenas de manera exponencial, trabajando únicamente con las subcadenas que cumplen con la condición del oráculo. La generación comienza con un conjunto inicial de secuencias de longitud 1 y continúa duplicando la longitud en cada iteración hasta alcanzar la longitud deseada n. El resultado final es la última subcadena generada que satisface la condición del oráculo.

Solución turbo mejorada paralela (reconstruirCadenaTurboMejoradaParalela):

1. Función Filtrar:

La función filtrar genera todas las subcadenas posibles a partir de las subcadenas dadas y luego filtra estas subcadenas $filtrar(subCadena, k) = parallel(subCadenas1, subCadenas2)$ donde $subCadenas1$ y $subCadenas2$ son las subcadenas generadas a partir de la subdivisión de $subCadena$ en dos conjuntos, y $parallel$ indica la ejecución paralela de las operaciones dentro del parentesis.

2. División de Subcadenas:

La función divide las subcadenas generadas en dos conjuntos, realizando la operación de filtrado de manera paralela.

3. Construcción de Subcadenas (construirSubcadena):

La función construirSubcadena construye las subcadenas resultantes, aplicando la función **"filtrar"** en cada iteración y duplicando k hasta alcanzar la longitud deseada n.

4. Generación Inicial de Cadenas:

Antes de la construcción de subcadenas, se inicia con la generación inicial de cadenas de longitud 1 sobre el alfabeto Σ.

En resumen, la función **"reconstruirCadenaTurboMejoradaParalela"** genera subcadenas de manera exponencial, trabajando únicamente con las subcadenas que cumplen con la condición del oráculo. La generación comienza con un conjunto inicial de secuencias de longitud 1 y continúa duplicando la longitud en cada iteración hasta alcanzar la longitud deseada n. La ejecución paralela en la función filtrar permite realizar la operación de filtrado en dos conjuntos de subcadenas simultáneamente, mejorando la eficiencia computacional en entornos paralelos.

A continuación, describiré cómo se utilizaron diferentes elementos de programación en las funciones proporcionadas:

1. Recursión:

- “reconstruirCadenaParalelo” y “reconstruirCadenaIngenuo”: Utilizan la recursión en la función “generarCadenas” para generar todas las combinaciones posibles de cadenas de longitud “n”. Además, “reconstruirCadenaIngenuo” utiliza recursión en la función interna “resultado”.

- “reconstruirCadenaMejorado” y “reconstruirCadenaMejoradoParalela”: Ambas funciones utilizan la recursión en la función “generarCadenasCandidatas” para generar candidatas de cadenas recursivamente.

-” reconstruirCadenaTurbo” y “reconstruirCadenaTurboParalela”: Usan recursión en la función “construirSubcadena” para construir subcadenas recursivamente.

- “reconstruirCadenaTurboMejorada” y “reconstruirCadenaTurboMejoradaParalela”: Hacen uso de la recursión en la función “construirSubcadena” para construir subcadenas recursivamente y en la función “filtrar” para generar subcadenas posibles de manera recursiva.

2. Reconocimiento de patrones:

- ”reconstruirCadenaParalelo” y “reconstruirCadenaTurboParalela”: Utilizan patrones en la verificación de paridad para decidir cómo dividir las cadenas y realizar la recursión.

3. Mecanismos de encapsulación:

-” reconstruirCadenaMejorado” y “reconstruirCadenaMejoradoParalela”: Encapsulan la lógica de generación de cadenas en la función interna “generarCadenasCandidatas”.

-” reconstruirCadenaTurbo” y “reconstruirCadenaTurboParalela”: Encapsulan la lógica de construcción de subcadenas en la función interna “construirSubcadena”.

-” reconstruirCadenaTurboMejorada” y “reconstruirCadenaTurboMejoradaParalela”: Encapsulan la lógica de filtrado de subcadenas en la función interna “filtrar”.

4. Funciones de alto orden:

-” reconstruirCadenaTurboMejoradaParalela”: Utiliza funciones de alto orden en “parallel” y “flatMap” para generar y filtrar subcadenas de manera paralela.

5. Iteradores:

-” reconstruirCadenaMejorado” y “reconstruirCadenaMejoradoParalela”: Utilizan iteración a través de las candidatas generadas mediante “flatMap”.

6. Colecciones:

- Todas las funciones: Utilizan colecciones como “Seq” para representar y manipular secuencias de caracteres y candidatas de cadenas.

7. Expresiones for:

-” reconstruirCadenaMejorado” y “reconstruirCadenaMejoradoParalela”: Utilizan expresiones “flatMap” para generar y manipular candidatas de cadenas.

- ”reconstruirCadenaTurboMejorada” y “reconstruirCadenaTurboMejoradaParalela”: Emplean expresiones “flatMap” para generar subcadenas de manera más eficiente.

Medidas de tiempo y procesamiento de resultados:

La función llamada **compareTime**, compara los tiempos de ejecución entre las dos funciones, una secuencial (**Fsec**) y otra paralela (**Fpar**). La función permite especificar el número de ejecuciones para obtener resultados más representativos.
A partir de ella podemos calcular la diferencia relativa entre los tiempos secuencial y paralelo, así como también retornar una tupla que contiene el tiempo promedio de la versión secuencial, el tiempo promedio de la versión paralela y la relación entre ambos tiempos.
Durante la prueba de tiempo y la ejecución de las funciones tanto su versión secuencial como su versión paralela se pudieron observar diferentes sucesos, pero veamos detalladamente los resultados para diferentes longitudes de la cadena objetivo:
Veamos los resultados con respecto al tiempo:

Solución ingenua vs Solución Paralela

N	Fsec(t)	Fpar(t)	Fsec(t)/Fpar(t)
4	2.2574	1.8562	1.191862
8	58.9069	70.298	0.772539
12	34227.8591	29614.5034	1.155180
16	42471.3579	89983.1754	0.4719922

Solución Mejorada vs Mejorada Paralela

N	Fsec(t)	Fpar(t)	Fsec(t)/Fpar(t)
4	0.5132	0.6088	0.842969700
8	0.2545	0.441	0.57709750
12	03456	0.5420	0.639868
16	1.0918	2.5489	0.372559

Solución Turbo vs Turbo Paralela

N	Fsec(t)	Fpar(t)	Fsec(t)/Fpar(t)
4	0.396	2.5528	0.15512378
8	0.0917	0.3243	0.282762
12	0.2055	2.041	0.100685
16	0.5623	1.3866	0.405524

Turbo MejoradaSec vs Turbo Mejorada Paralela

N	Fsec(t)	Fpar(t)	Fsec(t)/Fpar(t)
4	0.3562	1.8432	0.19625086
8	0.1558	0.3374	0.46176644
12	0.4104	0.4155	0.99638889
16	0.7199	1.1639	0.61852392

Durante la comparación, podemos evidenciar que:

los resultados de la tabla muestran que la función reconstruirCadenaMejorado es significativamente más eficiente que la función reconstruirCadenaIngenuo. El tiempo de ejecución promedio de la función reconstruirCadenaMejorado es de aproximadamente un 50% menor que el tiempo de ejecución promedio de la función reconstruirCadenaIngenuo.
Esta diferencia en el rendimiento se debe a la diferencia en el enfoque de las dos funciones. La función reconstruirCadenaIngenuo genera todas las cadenas posibles de longitud n a la vez. Esto puede ser muy ineficiente para valores grandes de n, ya que el número de cadenas posibles crece exponencialmente con n.La función reconstruirCadenaMejorado, por otro lado, genera las cadenas candidatas de forma incremental. Esto significa que solo genera las cadenas candidatas que son necesarias para encontrar una solución. Esto puede ser mucho más eficiente para valores grandes de n.
No obstante, es necesario recordar que puede haber momentos en los que la versión secuencial obtenga menores tiempos que la versión paralela para valores pequeños de n, aplicado también a todas las funciones. Esto se debe a que la sobrecarga de la paralelización puede superar la mejora en el rendimiento que se obtiene al ejecutar las operaciones en paralelo, mayoritariamente cuando n es un valor pequeño.
En el caso específico de las funciones, la sobrecarga de la paralelización se debe a los siguientes factores:

- El costo de crear y administrar los hilos y procesos paralelos.
- El costo de la comunicación entre los hilos y procesos paralelos.

En el caso del algoritmo de reconstrucción de cadena, la paralelización puede ser muy efectiva para valores grandes de n. Esto se debe a que el número de operaciones que se deben realizar crece exponencialmente con n. Para valores grandes de n, la sobrecarga de la paralelización es menor que la mejora en el rendimiento que se obtiene al ejecutar las operaciones en paralelo.

Conclusiones

En conclusión, la paralelización de procesos es una técnica que puede ser muy efectiva para mejorar el rendimiento del algoritmo de reconstrucción de cadena. Sin embargo, es importante tener en cuenta la sobrecarga de la paralelización para valores pequeños de n.

En el caso del proyecto, la paralelización de procesos puede ser una buena opción para mejorar el rendimiento del algoritmo de reconstrucción de cadena si el proyecto requiere resolver problemas con valores grandes de n.

Recomendaciones

Para aprovechar al máximo la paralelización de procesos, se recomienda tener en cuenta los siguientes factores:

El número de procesadores disponibles.

El tamaño de los datos que se deben procesar.

El costo de la paralelización.

Si el proyecto cuenta con suficientes procesadores y los datos que se deben procesar son lo suficientemente grandes, la paralelización de procesos puede ser una buena opción para mejorar el rendimiento del algoritmo de reconstrucción de cadena.