El problema de la reconstrucción de cadenas

Nicolas Mauricio Rojas - 2259460

Edgar Andrés Vargas Garcia - 2259690

Juan Pablo Escobar Viveros - 2259519

18 de diciembre de 2023

1. Explicación de las funciones implementadas

1.1. Función ReconstruirCadenaIngenuo

En nuestra función ingenua primeramente recibimos dos parámetros que son el tamaño de la secuencia a encontrar y el oráculo quien nos dará las indicaciones que nos servirá para encontrar la secuencia que estamos buscando. Dentro de la función ingenua definimos una función recursiva que recibirá como parámetros una secuencia de caracteres quien ira almacenando las combinaciones de los caracteres y un segundo parámetro entero que es la longitud restante de la cadena que queremos construir. Por ultimo definimos una variable cerradura quien va contener todas las combinaciones de los caracteres y usamos una función 'find' en la variable para encontrar la primera combinación que cumpla con el predicado del oráculo.

1.2. Función ReconstruirCadenaMejorado

Al igual que en la función ingenua se reciben los dos mismos parámetros uno de tipo entero y un oráculo con la misma finalidad, pero en esta función lo primero que haremos es tomar los caracteres que son subcadenas de la cadena que queremos encontrar. Posteriormente generamos todas las combinaciones posibles para esa secuencia de caracteres y en el llamado recursivo se envía estas nuevas combinaciones pero aplicando un 'filter' para que tome solo las cadenas que si son subcadenas de la cadena que queremos encontrar y también enviamos un entero que sera la longitud restante de la cadena que queremos construir.

1.3. Función ReconstruirCadenaTurbo

Dentro de esta función tenemos definidas mas funciones las cuales realizan lo siguiente:

- Función generarCadenas: Esta función se encarga de generar todas las subcadenas de longitud k que cumplen con el predicado del oráculo. Utiliza la recursión para construir las subcadenas de longitud creciente. Cuando k es igual a 1, la función genera subcadenas de longitud 1 a partir del alfabeto y filtra aquellas que cumplen con el predicado del oráculo. Cuando k es mayor que 1, la función genera nuevas subcadenas concatenando caracteres del alfabeto a subcadenas previamente generadas.

- Función combinarSubcadenas: Esta función combina las subcadenas generadas por generarCadenas. Comienza con subcadenas de longitud 2 y las combina para formar nuevas subcadenas. Este proceso se repite hasta alcanzar la longitud n deseada. La función utiliza la recursión y flatMap para generar combinaciones de subcadenas que cumplen con el predicado del oraculo.

En el cuerpo principal de la función reconstruirCadenaTurbo, se generan subcadenas iniciales de longitud 2 mediante la llamada a generarCadenas(2, alfabeto). Luego, estas subcadenas se combinan recursivamente utilizando combinarSubcadenas. El resultado final se obtiene utilizando el método find para buscar la primera subcadena que tiene la longitud deseada n. Si no se encuentra ninguna subcadena, se devuelve una lista vacía. Cabe aclarar que por la estructura que tiene esta función turbo solo se aceptan secuencias de caracteres de tamaño 2^n.

1.4. Función ReconstruirCadenaTurboMejorada

La función reconstruirCadenaTurboMejorada toma dos argumentos: un número entero n y un objeto Oraculo. La función devuelve una secuencia de caracteres que es la concatenación de todas las secuencias de caracteres generadas por la función generarCadenas.

La función generarCadenas es una función recursiva que se llama a sí misma hasta que k sea igual a cero. En cada llamada recursiva, la función filtrar se llama con sc, k y o como argumentos. La función filtrar devuelve una secuencia de secuencias de caracteres que cumple ciertas condiciones. La función generarCadenas toma la secuencia de secuencias de caracteres devuelta por filtrar y se llama a sí misma con k - 1 y la secuencia de secuencias de caracteres devuelta por filtrar como argumentos.

La función filtrar utiliza el método flatMap para aplicar una función anónima a cada elemento de sc. La función anónima toma una secuencia de caracteres s1 y aplica el método flatMap a alfabeto. El método flatMap toma una función anónima que toma un carácter a y devuelve una secuencia de caracteres s2. La función anónima devuelve una secuencia de caracteres s2 si s2 cumple las siguientes condiciones:

- La secuencia de caracteres s2 es igual a la secuencia de caracteres s1 con el carácter a agregado al final.
- La secuencia de caracteres s2 es aceptada por el objeto Oraculo.
- La secuencia de caracteres s2 es filtrable.

La función esFiltrable toma cuatro argumentos: una secuencia de caracteres s, un número entero k, una secuencia de secuencias de caracteres sc y un objeto Oraculo. La función devuelve un valor booleano. La función esFiltrable verifica si todas las subcadenas de longitud n de s cumplen las siguientes condiciones:

- La subcadena es aceptada por el objeto Oraculo.
- La subcadena es filtrable.

1.5. Función ReconstruirCadenasTurboAcelerado

La función reconstruirCadenasTurboAcelerado toma dos argumentos: un número entero n y un objeto Oraculo. La función devuelve una secuencia de caracteres.

La función transformarCadena toma dos argumentos: una secuencia de caracteres s y una secuencia de secuencias de caracteres acc. La función devuelve una secuencia de secuencias de

caracteres. La función transformarCadena utiliza el patrón match para verificar si la secuencia de caracteres s es vacía. Si es así, la función devuelve la secuencia de secuencias de caracteres acc. De lo contrario, la función devuelve la secuencia de secuencias de caracteres acc.

La variable SC es una secuencia de secuencias de caracteres. La variable SC se crea a partir de alfabeto utilizando el método map y el método filter. El método map toma una función anónima que toma un carácter c y devuelve una secuencia de caracteres que contiene solo el carácter c. El método filter toma una función anónima que toma una secuencia de caracteres s y devuelve un valor booleano. La función anónima devuelve true si el objeto Oraculo acepta la secuencia de caracteres s; de lo contrario, devuelve false.

La función generarCombinaciones toma dos argumentos: una secuencia de secuencias de caracteres cadenas y un número entero n. La función devuelve una secuencia de secuencias de caracteres. La función generarCombinaciones es una función recursiva que se llama a sí misma hasta que n sea menor o igual a 1. En cada llamada recursiva, la función filtrado se llama con cadenas como argumento. La función filtrado devuelve una secuencia de secuencias de caracteres que cumple ciertas condiciones. La función generarCombinaciones toma la secuencia de secuencias de caracteres devuelta por filtrado y se llama a sí misma con la secuencia de secuencias de caracteres devuelta por filtrado y n / 2 como argumentos.

La función filtrado utiliza el método flatMap para aplicar una función anónima a cada elemento de cadenas. La función anónima toma dos secuencias de caracteres s1 y s2 y crea una nueva secuencia de caracteres nuevaCadena que es la concatenación de s1 y s2. La función anónima devuelve nuevaCadena si nuevaCadena cumple las siguientes condiciones:

- La longitud de nuevaCadena es igual a 2.
- Todas las subcadenas de longitud nuevaCadena.length / 2 de nuevaCadena se pueden encontrar en un árbol construido a partir de cadenas.

La función generarCombinaciones devuelve la primera secuencia de caracteres de la secuencia de secuencias de caracteres devuelta por filtrado.

1.6. Función ReconstruirCadenaIngenuoParalela

La función reconstruirCadenaIngenuoParalela es una versión paralela de la función reconstruirCadenaIngenuo. La estructura de ambas funciones es similar. La diferencia principal es que la función reconstruirCadenaIngenuoParalela tiene una nueva variable entera llamada umbral. Esta variable determina si la función se debe ejecutar en paralelo o secuencialmente. Si umbral es mayor que n, la función se ejecuta secuencialmente. Si umbral es menor o igual que n, la función se ejecuta paralelamente. Para la parte de la paralelización, la función utiliza la palabra clave task para paralelizar el llamado recursivo. El resto de la lógica funciona de manera similar a la función reconstruirCadenaIngenuo.

1.7. Función reconstruirCadenaMejoradoParalelo

La función reconstruirCadenaMejoradoParalelo es una versión paralela de la función reconstruirCadenasMejorado. La función reconstruirCadenaMejoradoParalelo se ejecuta en paralelo si umbral es mayor que n. La paralelización se realiza en la función generarCombinacionesParalelo. Si umbral es mayor que n, la función utiliza el método flatMap para aplicar una función anónima a cada elemento de cadenas. La función anónima devuelve una tarea que llama a generarCombinacionesParalelo con s1 ++ s2 y n - 1 como argumentos. La función generarCombinacionesParalelo devuelve una secuencia de secuencias de caracteres que es la concatenación de todas las secuencias de caracteres devueltas por las tareas.

1.8. Función reconstruirCadenaTurboParalelo

La función reconstruirCadenaTurboParalelo incorpora notables modificaciones respecto a reconstruirCadenaTurbo. Principalmente, se destaca la introducción de la ejecución paralela en las fases de generación y combinación de subcadenas cuando el umbral supera la longitud deseada, es decir, cuando umbral es mayor que n.

En la función generarCadenasParalelo, se adopta un enfoque paralelo mediante el uso de tareas (task) para la generación concurrente de subcadenas, aprovechando potencialmente múltiples núcleos de CPU. Este enfoque se activa únicamente cuando la condición umbral ¿n es satisfecha, lo que determina la elección entre ejecución secuencial y paralela.

De manera análoga, la función combinarSubcadenasParalelo implementa una estrategia paralela para la combinación de subcadenas, también condicionada por umbral ¿n. Aquí, nuevamente se emplean tareas para realizar operaciones de manera concurrente, buscando mejorar el rendimiento en sistemas con capacidad de paralelismo.

El manejo de tareas y paralelismo en reconstruirCadenaTurboParalelo constituye un punto clave. La decisión de activar la ejecución paralela se toma de forma inteligente según la relación entre el umbral y la longitud deseada, permitiendo una adaptabilidad dinámica al entorno de ejecución.

Ambas funciones siguen un enfoque específico para longitudes de cadena que son potencias de 2, pero la versión paralela introduce un componente adaptativo que puede resultar en un rendimiento mejorado en sistemas multiprocesador.

2. Desempeño de las funciones

2.1. Evaluación comparativa

Cuadro 1: Función Ingenua

Tamaño Cadena	Tiempo Secuencial	Tiempo Paralela	Aceleración
4	0.4823	1.2786	0.3772
6	0.5829	1.5624	0.3731
8	25.998	9.9528	2.6121
9	80.1796	68.515	1.1702

Cuadro 2: Función Mejorada

Tamaño Cadena	Tiempo Secuencial	Tiempo Paralela	Aceleración
4	0.1054	0.4005	0.2632
6	0.2554	0.6492	0.3934
8	0.0727	0.234	0.3107
9	0.1388	0.419	0.3313

Cuadro 3: Función Turbo

Tamaño Cadena	Tiempo Secuencial	Tiempo Paralela	Aceleración
8	1.1074	0.3698	2.9946
16	1.3505	0.2977	4.5364
32	6.9409	1.3269	5.2309
64	12.4801	1.1513	10.8400

Cuadro 4: Función Turbo Mejorada

Tamaño Cadena	Tiempo Secuencial
8	0.3252
16	1.8317
32	4.5905
64	12.4944

- ¿Que impacto tuvieron las paralelizaciones en el programa?

El impacto de las paralelizaciones en el programa se evidencia en la mejora significativa de los tiempos de ejecución cuando el tamaño de las cadenas es más grande. En los casos donde las cadenas son pequeñas, la diferencia en los tiempos de ejecución entre las versiones paralelas y secuenciales no es tan notable. Sin embargo, a medida que el tamaño de las cadenas aumenta, la versión paralela demuestra una aceleración considerable en comparación con la versión secuencial. Esta mejora en el rendimiento se debe al aprovechamiento de la capacidad de procesamiento concurrente proporcionada por la paralelización. Específicamente, cuando el tamaño de las cadenas es más grande, la versión paralela puede distribuir la carga de trabajo entre múltiples

núcleos de procesador, permitiendo una ejecución más eficiente y una reducción significativa en los tiempos totales.

- ¿De que depende que la aceleración sea mejor?

La aceleración en las versiones paralelas va depender de la naturaleza del problema, el tamaño de los datos, la arquitectura del hardware y la calidad de la implementación la función haciendo uso del paralelismo.

- ¿Cuáles son los algoritmos que presentaron desafíos significativos al trabajar con secuencias de caracteres extensas?

Las funciones mejoradas e ingenuas, diseñadas para reconstruir secuencias de caracteres, presentan limitaciones significativas cuando operan con secuencias considerables. Este obstáculo surge del elevado coste computacional asociado a la generación de todas las combinaciones posibles de caracteres en secuencias largas, lo que restringe su eficacia para secuencias de más de 10 caracteres. Aunque el hardware del computador puede ser potente, la complejidad computacional inherente a la generación exhaustiva de combinaciones limita la ejecución eficiente de estos algoritmos en el programa para secuencias más largas, lo que pone de manifiesto la necesidad de enfoques más eficientes en situaciones de mayor dimensión.

2.2. Desempeño de las funciones secuenciales y de las funciones paralelas

A continuación las siguientes comparaciones serán evaluadas con secuencias de caracteres pequeñas debido a las limitaciones computacionales que generan las funciones ingenua y mejorada.

Cuadro 5: Funciones Secuenciales

Tamaño	Ingenua	Mejorada	Turbo	Turbo Mejorada	Acelerada
2	0.3302	0.0515	0.0839	0.0701	0.0703
4	0.5527	0.1059	0.5967	0.1281	0.2986
8	31.7434	0.1789	0.5349	0.4382	1.6514

Cuadro 6: Funciones Paralelas

Tamaño	Ingenua	Mejorada	Turbo
2	0.1374	0.0974	0.0748
4	1.2249	0.3093	0.6903
8	25.8614	0.3258	0.4453

3. Análisis Comparativo

- ¿Las paralelizaciones fueron de ayuda?

Las paralelizaciones tuvieron un buen desempeño por lo cual logramos observar que tienen un mejor tiempo de entrega sobre todo cuando las combinaciones de las secuencias son demasiado grandes.

- ¿Cual es el algoritmo mas eficiente?

A pesar de que no logramos hacer su implementación sabemos que la función turbo acelerada paralela muy posiblemente sea la mas eficiente debido a que tiene en consideración mas factores los cuales optimizan mucho mas la combinación de caracteres para llegar a encontrar la secuencia que estamos buscando además de que las funciones paralelas implementan un umbral que cuando este es superado se entiende que la paralelización no va generar mucha diferencia entonces empieza a ejecutarse secuencialmente dando así una mayor efectividad en la entrega del resultado y en el ahorro de recursos computacionales.

- ¿Que se puede concluir al respecto?

En conclusión, analizar problemas considerando las características inherentes y tendencias de los datos puede llevar a soluciones más eficientes en términos computacionales, resolviendo problemas de rendimiento y permitiendo ejecutar las tareas de forma efectiva. También es importante señalar que distribuir funciones de forma paralela puede ser una estrategia útil para acelerar los resultados. Sin embargo, al implementar esta distribución es crucial hacerlo de manera cuidadosa ya que cuán bien se organice el trabajo concurrente afecta mucho a la eficacia de esta estrategia.

Ciertas funciones, especialmente aquellas que generan un gran número de combinaciones, pueden suponer un coste computacional demasiado alto. En última instancia, la combinación de un diseño algorítmico inteligente y la exploración de estrategias de paralelización bien implementadas pueden ofrecer soluciones más rápidas y eficientes, mejorando significativamente la capacidad de respuesta de los programas y reduciendo los tiempos de ejecución en problemas computacionales complejos.