PROYECTO FINAL

EL PROBLEMA DE RECONSTRUCCIÓN DE CADENAS

Juan Manuel Arango Rodas - 2259571 [Grupo 50]
Juan Camilo Valencia Rivas - 2259459 [Grupo 51]

1. Informe de Corrección de Funciones

A continuación se expone de manera formal el proceso de las funciones implementadas durante la realización del proyecto.

1.1. Función *PRC-ingenuo*

La función busca subconjuntos de cadenas de longitud n en un lenguaje L que cumplen la propiedad de ser subcadenas o en este caso la propia cadena y devuelve una representación en forma de cadena de esos subconjuntos. La eficiencia de esta búsqueda puede ser limitada, ya que considera todas las combinaciones posibles, lo que puede resultar en un enfoque 'ingenuo' o poco eficiente para conjuntos de datos grandes.

La función PRC-ingenuoParalelo toma una lenguaje L, un entero positivo n, y un oráculo ora como argumentos. Inicialmente, calcula la cerradura completa de las cadenas de longitud n. Luego, divide esta cerradura en cuatro partes aproximadamente iguales y aplica de manera paralela el oráculo a cada subconjunto. Finalmente, concatena los resultados filtrados de cada parte para generar una cadena de salida, representando así un enfoque paralelo en el procesamiento de cadenas según las especificaciones del oráculo proporcionado.

1.2. Función *PRC-mejorado*

La función PRC-mejorado busca subcadenas en un conjunto de cadenas representado por la lista L. Utiliza un enfoque mejorado a través de una función auxiliar llamada PRC-mejorado-aux. En cada paso, se genera una cerradura de las cadenas en L (una especie de conjunto expandido) y se filtran las subcadenas resultantes utilizando un oráculo. Este proceso se repite de manera recursiva, aumentando gradualmente el tamaño de las cerraduras, hasta que se encuentra una subcadena de la longitud deseada n o hasta que se alcanza un punto

donde no se pueden encontrar subcadenas más largas. El resultado final es una lista de subcadenas que cumplen con los criterios establecidos por el oráculo y la longitud deseada. La recursividad y la acumulación de resultados en la lista res ayudan a optimizar el proceso al evitar el procesamiento innecesario cuando ya se ha encontrado una subcadena de la longitud requerida.

1.3. Función *PRC-mejoradoParalelo*

La función PRC-mejoradoParalelo mejora la eficiencia al introducir paralelismo en la búsqueda de subcadenas. Divide la cerradura de L en cuatro partes y utiliza operaciones paralelas para filtrar simultáneamente cada subdivisión. El uso de la función parallel agiliza el procesamiento paralelo. La recursividad y la acumulación de resultados en filtrado se mantienen para optimizar el rendimiento y evitar redundancias en el procesamiento de la cerradura.

1.4. Función *PRC-TurboSolucion*

PRC-TurboSolucion representa una optimización en la búsqueda de subcadenas al filtrar la lista inicial antes de iniciar el proceso recursivo y al incrementar el tamaño de la cerradura en pasos de 2 donde busca cadenas de tamaño 2^n y n es el numero de veces que se va a realizar este proceso. Esta estrategia optimizada busca mejorar el rendimiento al reducir el tamaño de la lista de entrada inicial y al ajustar la generación de cerraduras de manera más eficiente. Esta diferencia clave en la estrategia de filtrado inicial y en el incremento de tamaño acelera el proceso en comparación con las versiones anteriores, priorizando la eficiencia en la búsqueda de subcadenas.

1.5. Función *PRC-TurboSolucionParalelo*

PRC-TurboSolucionParalelo representa una mejora en la eficiencia de búsqueda de subcadenas al incorporar paralelismo. Inicia filtrando la lista inicial L con el oráculo antes de entrar en el proceso recursivo. Durante cada iteración recursiva, la cerradura se calcula con un tamaño de búsqueda de 2^n, y se realiza un filtrado paralelo al dividir la cerradura en cuatro partes, ejecutando operaciones concurrentes mediante la función parallel. La optimización se mantiene al incrementar el tamaño de la cerradura en pasos de 2. La función termina cuando encuentra una subcadena de longitud n o cuando la lista filtrada ya contiene una subcadena de esa longitud y es la cadena reconstruida que esta en el oraculo. Esta combinación de estrategias optimizadas y paralelismo destaca la eficiencia de PRC-TurboSolucionParalelo en la búsqueda de subcadenas.

1.6. Función *PRC-Turbomejorada*

PRC-Turbomejorado presenta una mejora adicional en la búsqueda de subcadenas al introducir un filtro más sofisticado, representado por la función filtrar. Esta función compara cada elemento en la cerradura con la mitad de su longitud, asegurando que todas las subcadenas parciales estén presentes en la lista anterior. La función principal, PRC-mejorado-aux, filtra la lista de entrada L utilizando el oráculo antes de iniciar el proceso recursivo. La cerradura se calcula con un tamaño de búsqueda de 2^n, y luego se aplica el filtro más sofisticado. El proceso recursivo continua con el filtrado y verificación de condiciones de terminación.

1.7. Función *PRC-TurbomejoradaParalelo*

La función PRC-TurbomejoradoParalelo representa una versión mejorada y paralelizada de un algoritmo para el reconocimiento de subcadenas en un conjunto de cadenas. La optimización se logra mediante el uso de filtros más sofisticados y la ejecución de operaciones en paralelo.

En particular, la función utiliza la función filtrar para comparar cada elemento de la cerradura con la mitad de su longitud, mejorando el filtrado de subcadenas. Además, se implementa paralelismo en la generación y filtrado de la cerradura, dividiendo la tarea en cuatro partes y utilizando operaciones concurrentes mediante el uso de la función parallel.

La estrategia general consiste en ajustar el tamaño de la cerradura en pasos iniciales de 2 y, posteriormente, en pasos de 4, aplicando filtros y paralelismo en cada iteración. La recursividad y la acumulación de resultados en filtrado aseguran la optimización del proceso, evitando el procesamiento innecesario y mejorando la eficiencia del algoritmo.

1.8. Función *Cerradura*

La función cerradura en Scala calcula la cerradura de una lista de cadenas hasta un nivel específico n. Utiliza recursión y comprensiones de listas para generar todas las combinaciones posibles de subcadenas hasta la longitud n. La función toma una lista de cadenas l y el nivel n como parámetros. Internamente, la función concatenarRec realiza la concatenación de las cadenas originales con las subcadenas acumuladas de manera recursiva. El resultado es una lista que contiene todas las combinaciones posibles de subcadenas hasta el nivel especificado. La anotación @tailrec indica que la recursión se realiza en cola para optimizar el rendimiento. Este cálculo de cerradura es útil en la teoría de autómatas y gramáticas formales, podemos evidenciar el uso de elementos puramente funcionales, y el manejo de expresiones for además de recursión de cola.

1.9. Función *CerraduraCompleta*

La función cerraduraCompleta en Scala calcula la cerradura completa de una lista de cadenas hasta un nivel específico n. Utiliza recursión y la función flatMap para generar todas las combinaciones posibles de subcadenas hasta la longitud n. La función toma una lista de cadenas l y el nivel n como parámetros. Internamente, la función generarCerraduraRec realiza

la generación de cerradura concatenando cada cadena con todas las combinaciones previas de manera recursiva. El resultado es una lista que contiene todas las combinaciones posibles de subcadenas hasta el nivel especificado.

2. Informe de Desempeño

Tras ejecutar repetidamente las pruebas de rendimiento, fue posible obtener algunas conclusiones a partir de las implementaciones. En las siguientes tabla puede verse reflejado el promedio de tiempo que tomó la ejecución de cada implementación según el tamaño de la cadena que se desea reconstruir:

2.1. Cadena de tamaño 2

Cuadro 1: Promedio de tiempos de ejecución por función para PRC de tamaño 2

	Implementación secuencial (ms)	Implementación paralela (ms)
PRC-ingenua	0.082	0.093
PRC-mejorado	0.072	0.034
PRC-TurboSolucion	0.017	0.081
PRC-Turbomejorado	0.082	0.145

2.2. Cadena de tamaño 4

Cuadro 2: Tiempos de ejecución por función para PRC de tamaño 4

	Implementación secuencial (ms)	Implementación paralela (ms)
PRC-ingenua	0.477	0.295
PRC-mejorado	0.122	0.234
PRC-TurboSolucion	0.017	0.070
PRC-Turbomejorado	0.220	0.116

2.3. Cadena de tamaño 8

Cuadro 3: Tiempos de ejecución por función para PRC de tamaño 8

	Implementación secuencial (ms)	Implementación paralela (ms)
PRC-ingenua	9.623	6.732
PRC-mejorado	2.392	2.881
PRC-TurboSolucion	0.008	0.049
PRC-Turbomejorado	0.242	0.455

2.4. Cadena de tamaño 16

Cuadro 4: Tiempos de ejecución por función para PRC de tamaño 16

	Implementación secuencial (ms)	Implementación paralela (ms)
PRC-TurboSolucion	0.065	0.289
PRC-Turbomejorado	0.568	0.981

2.5. Cadena de tamaño 32

Cuadro 5: Tiempos de ejecución por función para PRC de tamaño 32

	Implementación secuencial (ms)	Implementación paralela (ms)
PRC-TurboSolucion	0.229	0.417
PRC-Turbomejorado	2.912	2.066

2.6. Cadena de tamaño 64

Cuadro 6: Tiempos de ejecución por función para PRC de tamaño 64

	Implementación secuencial (ms)	Implementación paralela (ms)
PRC-TurboSolucion	0.605	0.924
PRC-Turbomejorado	19.730	7.759

2.7. Cadena de tamaño 128

Cuadro 7: Tiempos de ejecución por función para PRC de tamaño 128

	Implementación secuencial (ms)	Implementación paralela (ms)
PRC-TurboSolucion	2.420	1.798
PRC-Turbomejorado	239.344	89.145

2.8. Cadena de tamaño 256

Cuadro 8: Tiempos de ejecución por función para PRC de tamaño 256

	Implementación secuencial (ms)	Implementación paralela (ms)
PRC-TurboSolucion	20.216	11.0414
PRC-Turbomejorado	3297.157	1197.531

2.9. Cadena de tamaño 512

Cuadro 9: Tiempos de ejecución por función para PRC de tamaño 512

	Implementación secuencial (ms)	Implementación paralela (ms)
PRC-TurboSolucion	109.615	88.546
PRC-Turbomejorado	44149.406	20943.211

2.10. Cadena de tamaño 1024

Cuadro 10: Tiempos de ejecución por función para PRC de tamaño 1024

	Implementación secuencial (ms)	Implementación paralela (ms)
PRC-TurboSolucion	2047.357	1436.928

El análisis de estos resultados nos otorga bastante información respecto a la ejecución de nuestro código. En primera instancia, podemos observar que obtenemos mejores resultados a medida que el tamaño de cadena crece, mientras más pequeña la cadena la paralelización se vuelve más ineficiente, esto nos permite responder algunas preguntas:

- **¿Cuál de las implementaciones es más rápida?** La implementación que nos dio mejores resultados fue la turbo solución, la cual es muy eficiente y baja bastante los tiempos de ejecución.
- **¿Porque razón las técnicas de paralelización utilizadas tienen un impacto positivo en el desempeño del programa?** El enfoque que se utilizo para paralelizar todas las funciones fue muy similar, lo que se realizo para tener un impacto positivo en el desempeño, fue paralelizar las partes que eran posibles de dividir en tareas más pequeñas sin afectar el resultado final, esto se uso sobre todo en la parte de filtrado de las cerradura, ya que se puede dividir la lista y filtrar más eficientemente en secciones más pequeñas y luego unir los resultados de cada una de estas tareas.

3. Análisis Comparativo

Para efectos de brevedad, se utilizarán únicamente los resultados de las cadenas de tamaño 8 y 512 durante el desarrollo de esta sección.

3.1. Tiempos de ejecución en implementaciones secuenciales en cadenas de tamaño 8

Implementación	**Tiempo (ms)**
PRC-ingenua	9.623
PRC-mejorado	2.392
PRC-TurboSolucion	0.008
PRC-Turbomejorado	0.242

A partir de estos resultados es posible inferir que, en caso de utilizar una implementación secuencial, la turbo solución puede llegar a ser la mejor alternativa ya que disminuye en gran medida los tiempos de ejecución, por el contrario la solución ingenua es la más lenta de todas por ende la menos recomendable.

3.2. Tiempos de ejecución en implementaciones secuenciales en cadenas de tamaño 512

Implementación	**Tiempo (ms)**
PRC-TurboSolucion	109.615
PRC-Turbomejorado	44149.406

Si tenemos en cuenta la conclusión anterior se puede evidenciar y ratificar que la turbo solución de manera secuencial es la más eficiente, igualmente solo la turbo solución y la Turbo mejorada son capaces de reconstruir cadenas de mayor tamaño.

3.3. Tiempos de ejecución en implementaciones paralelas con cadenas de tamaño 8

Implementación	**Tiempo (ms)**
PRC-ingenua	6.732
PRC-mejorado	2.881
PRC-TurboSolucion	0.049
PRC-Turbomejorado	0.455

Como se puede evidenciar solamente la solución ingenua tuvo una mejora con respecto a sus versiones secuenciales, pero aún así, sigue por debajo de las otras en rendimiento, en cadenas pequeñas se nota que las otras soluciones son más ineficientes en comparación con su versión secuencial y esto se debe a que en cadenas pequeñas dividir y concatenar las listas es más ineficiente que filtrar en una sola lista con todos los datos.

3.4. Tiempos de ejecución en implementaciones paralelas con cadenas de tamaño 512

Implementación	**Tiempo (ms)**
PRC-TurboSolucion	88.546
PRC-Turbomejorado	20943.211

A diferencia de la conclusión anterior en este caso se puede evidenciar mejoras en los tiempos de ejecución, los cual nos indica que mientras más grande la cadena la mejora de rendimiento aumenta, la turbo solución es mucho más rápida y eficiente que la solución turbo mejorada.

3.5. Aceleración en cadenas de tamaño 8

Implementación	Secuencial (ms)	Paralelo (ms)	Aceleración
PRC-ingenua	9.623	6.732	1.429
PRC-mejorado	2.392	2.881	0.830
PRC-TurboSolucion	0.008	0.049	0.163
PRC-Turbomejorado	0.242	0.455	0.531

Es posible evidenciar una aceleración prominente en la versión ingenua, en el resto se evidencia una ralentización del proceso, por ende en cadenas pequeñas en caso de usar una versión paralela la más recomendable es usar la solución ingenua, aún así la solución más rapida es la turbo solución secuencial por mucha diferencia.

3.6. Aceleración en cadenas de tamaño 512

Implementación	Secuencial (ms)	Paralelo (ms)	Aceleración
PRC-TurboSolucion	109.615	88.546	1.237
PRC-Turbomejorado	44149.406	20943.211	2.106

A diferencia de la conclusión anterior podemos evidenciar una gran aceleración en ambas implementaciones donde la que obtuvo saco mayor provecho de la paralelización fue la turbo mejorada, la turbo solución también tuvo una mejora significativa y sigue siendo la más eficiente y de ejecución más rápida.