INFORME PROYECTO FINAL

Kevin Steven Ramírez – 2259371

Pedro Bernal Londoño – 2259548

Jota Emilio López Ramírez – 2259394

Docente:

Carlos Andrés Delgado

Universidad del Valle sede Tuluá

Facultad de ingeniería

Ingeniería de sistemas

Seccional Tuluá

Corrección de las funciones desarrolladas:

Anotación: Para las funciones paralelas se tienen las funciones con umbral de la clase ReconstruirCadenasParUmbral, en donde se tiene una función para las cinco funciones paralelas aquí explicadas para contextualizar un poco más, la función recibe tres parámetros: umbral, n y o. Esta devuelve una secuencia de caracteres que representa la cadena reconstruida. Para ello, utiliza una condición if-else que compara el valor de n con el valor de umbral. Si n es menor que umbral, se llama a una funcíon del objeto reconstruirCadenas, que implementa el algoritmo secuencial. Si n es mayor o igual que umbral, se llama a una función del objeto reconstruirCadenasPar, que implementa el algoritmo paralelo. Ambas funciones reciben los mismos parámetros n y o, y devuelven una secuencia de caracteres.

reconstruirCadenasIngenuo:

Esta función recursiva que dados n, que es la longitud de la cadena a reconstruir y o, un oráculo asociado a esa cadena, devuelve una cadena de caracteres que representa el oráculo; para esto, hace lo siguiente:

- Define una función auxiliar llamada generarCadenas que recibe un parámetro n que representa la longitud de la cadena a generar. Si n es cero, devuelve una secuencia vacía, que representa una cadena vacía.
- Si n es mayor que cero, usa la variable alfabeto que contiene todas las letras posibles, y aplica la función flatMap para generar todas las combinaciones de cadenas de longitud n a partir de las cadenas de longitud n - 1. La función flatMap toma cada letra del alfabeto y la concatena al final de cada cadena de longitud n - 1 que se obtiene al llamar recursivamente a generarCadenas. El resultado es una secuencia de secuencias de caracteres, donde cada subsecuencia representa una cadena de longitud n. Luego, el map recibe una secuencia de caracteres y le aplica una transformación. En este caso, la transformación es agregar una letra al final de cada cadena.
- La función principal reconstruirCadenasIngenuo llama a generarCadenas con el parámetro n que recibe, y luego usa la función find para buscar la primera cadena que el oráculo reconoce como correcta. Si encuentra una, la devuelve como una secuencia de caracteres. Si no, devuelve una secuencia vacía.

reconstruirCadenasIngenuo(paralela):

Esta función es la versión paralela de reconstruirCadenasIngenuo, ambas cumplen con el mismo fin, pero tienen implementacione diferentes. A continuación, la implementación de la función:

- La función utiliza una función auxiliar llamada generarCadenas, que recibe un número entero n y devuelve una secuencia de todas las posibles secuencias de caracteres de longitud n, usando un alfabeto predefinido.
- La función generarCadenas usa la paralelización para dividir el alfabeto en dos partes y generar recursivamente las secuencias de cada parte, usando las funciones flatMap y map. Luego, concatena las dos partes con el operador ++.
- Finalmente, la función reconstruirCadenasIngenuo llama a la función generarCadenas con el argumento n y busca la primera secuencia que cumpla el oráculo o, usando la función

find. Si la encuentra, la devuelve; si no, devuelve una secuencia vacía, usando la función getOrElse.

reconstruirCadenasMejorado:

Esta función es una versión mejorada de reconstruirCadenaIngenuo que tiene el mismo fin, pero, evita generar todas las posibles cadenas de longitud n y solo hace las que el oráculo acepta. Para esto realiza los siguientes pasos:

- Se define una variable inmutable llamada SC que contiene todas las letras del alfabeto que el oráculo acepta como cadenas válidas de longitud 1. Para ello, usa la función map para convertir cada letra en una secuencia de un solo elemento, y luego la función filter para quedarse solo con las que el oráculo aprueba.
- Define una función auxiliar llamada generarCombinaciones que recibe dos parámetros: el primero es llamado cadenas, que es una secuencia de secuencias de caracteres que representan cadenas válidas, y n, que es la longitud de la cadena objetivo.
- Si n es 1, devuelve cadenas sin modificar, ya que son las cadenas válidas de longitud 1. Si no, Si n es mayor a 1, a la variable inmutable nuevas cadenas asigna cadenas.view, esta función .view es usada para crear una vista perezosa de cadenas, que evita evaluarlas hasta que sea necesario. Luego, usa la función flatMap para generar todas las combinaciones de cadenas de longitud n a partir de las cadenas de longitud n - 1 y las letras de SC. Para ello, toma cada cadena de cadenas y la concatena con cada letra de SC, usando la función map. Finalmente, usa la función toSeq para convertir la vista perezosa en una secuencia estricta.
- Llama recursivamente a generarCombinaciones con las nuevas cadenas generadas, pero solo con las que el oráculo acepta como válidas, usando la función filter. También reduce el valor de n en una unidad, para acercarse al caso base.
- La función principal reconstruirCadenasMejorado llama a generarCombinaciones con SC y n como parámetros, y luego devuelve la primera cadena de la secuencia resultante, usando la función head. Esta cadena es la que el oráculo reconoce como correcta.

reconstruirCadenasMejorado(paralela):

Esta es la versión paralela de reconstruirCadenasMejorado. Para hacer esto, se siguen los siguientes pasos:

- La función asigna a la variable inmutable llamada SC, una secuencia de todas las cadenas de longitud 1 que cumplen el oráculo. Esto se hace usando las funciones map y filter sobre el alfabeto. Luego, la función usa una función auxiliar llamada generarCombinaciones, que recibe una secuencia de cadenas y un número entero n, y devuelve una secuencia de todas las combinaciones posibles de esas cadenas de longitud n que también cumplen el oráculo.
- Se usa la paralelización para dividir la secuencia de cadenas en dos partes y generar recursivamente las combinaciones de cada parte, usando las funciones flatMap y map. Luego, filtra las combinaciones que no cumplen el oráculo, usando la función filter. El proceso se repite hasta que n sea igual a 1, en cuyo caso se devuelve la secuencia de cadenas original.

- Finalmente, la función reconstruirCadenasMejorado llama a la función generarCombinaciones con la variable SC y el argumento n, y devuelve la primera cadena de la secuencia resultante, usando la función head. Si la secuencia está vacía, significa que no hay ninguna cadena que cumpla el oráculo, y la función devolverá una excepción.

reconstruirCadenasTurbo:

Esta función es una versión mejorada de reconstruirCadenasMejorado, y son muy parecidas, su diferencia radica en que usa una técnica de división y conquista para reducir el número de combinaciones de cadenas que tiene que generar y verificar. Para estos se realizan los siguientes pasos:

- Se define una variable inmutable llamada SC que contiene todas las letras del alfabeto que el oráculo acepta como cadenas válidas de longitud 1. Para ello, usa la función map para convertir cada letra en una secuencia de un solo elemento, y luego la función filter para quedarse solo con las que el oráculo aprueba.
- Define una función auxiliar llamada generarCombinaciones que recibe dos parámetros: el primero es llamado cadenas, que es una secuencia de secuencias de caracteres que representan cadenas válidas, y n, que es la longitud de la cadena objetivo.
- Si n es menor o igual que 1, devuelve cadenas sin modificar, ya que son las cadenas válidas de longitud 1 o menor.
- Si n es mayor a 1, a la variable inmutable nuevas cadenas asigna cadenas.view, esta función .view es usada para crear una vista perezosa de cadenas, que evita evaluarlas hasta que sea necesario. Luego, usa la función flatMap para generar todas las combinaciones de cadenas de longitud 2n a partir de las cadenas de longitud n. Para ello, toma cada cadena de cadenas y la concatena consigo misma, usando la función map. El resultado es una secuencia de secuencias de caracteres, donde cada subsecuencia representa una cadena de longitud 2n. Finalmente, usa la función toSeq para convertir la vista perezosa en una secuencia estricta.
- Llama recursivamente a generarCombinaciones con las nuevas cadenas generadas, pero solo con las que el oráculo acepta como válidas, usando la función filter. También divide el valor de n entre 2, para acercarse al caso base.
- La función principal reconstruirCadenasTurbo llama a generarCombinaciones con SC y n como parámetros, y luego devuelve la primera cadena de la secuencia resultante, usando la función head. Esta cadena es la que el oráculo reconoce como correcta.

reconstruirCadenasTurbo(paralela):

Esta función usa un algoritmo más eficiente que reduce el espacio de búsqueda con respecto a la función paralela anterior. Para esto se realizan los siguientes pasos:

- La función usa una variable llamada SC, que es una secuencia de todas las cadenas de longitud 1 que cumplen el oráculo. Esto se hace usando las funciones map y filter sobre el alfabeto. Luego, la función usa una función auxiliar llamada generarCombinaciones, que recibe una secuencia de cadenas y un número entero n, y devuelve una secuencia de todas las combinaciones posibles de esas cadenas de longitud n que también cumplen el oráculo.

- Se usa el paralelismo para dividir la secuencia de cadenas en dos partes y generar recursivamente las combinaciones de cada parte, usando las funciones flatMap y map. Luego, filtra las combinaciones que no cumplen el oráculo, usando la función filter. El proceso se repite hasta que n sea menor o igual a 1, en cuyo caso se devuelve la secuencia de cadenas original.
- Finalmente, la función reconstruirCadenasTurbo llama a la función generarCombinaciones con la variable SC y el argumento n, y devuelve la primera cadena de la secuencia resultante, usando la función head. Si la secuencia está vacía, significa que no hay ninguna cadena que cumpla el oráculo, y la función devolverá una excepción.

reconstruirCadenasTurboMejorada:

Esta función es una versión más eficiente que reconstruirCadenasTurbo(paralela), para lograr esto, usa una técnica de filtrado para eliminar las cadenas que no pueden ser parte de la solución. A continuación, los pasos seguidos para realizarla:

- Se define la variable SC que almacena todas las letras del alfabeto reconocidas por el oráculo como cadenas válidas de longitud 1. Utiliza la función map para convertir cada letra en una secuencia de un solo elemento y, posteriormente, emplea la función filter para retener únicamente aquellas letras aprobadas por el oráculo.
- Se crea una función auxiliar llamada "filtrar" con dos parámetros: cadenasOriginales, una secuencia de secuencias de caracteres representando cadenas válidas de cierta longitud, y nuevasCadenas, otra secuencia de secuencias de caracteres que representan cadenas generadas a partir de las originales, pero con el doble de longitud.
- Si las nuevas cadenas tienen longitud 2, las devuelve sin cambios, ya que son las cadenas válidas de longitud 2. Si son más largas, emplea la función filter para eliminar aquellas que no cumplan una condición específica. Esta condición consiste en que cada subcadena, de longitud igual a la mitad de la nueva cadena, debe estar presente en cadenasOriginales. Para lograrlo, crea una vista perezosa de la nueva cadena usando la función view y luego utiliza la función sliding para obtener todas las subcadenas posibles de la longitud mencionada, moviéndote de uno en uno. Posteriormente, verifica con la función forall que todas esas subcadenas estén en cadenasOriginales, usando la función contains. El resultado será una secuencia donde cada subsecuencia representa una cadena que pasa el filtro.
- Se tiene otra función auxiliar, llamada generarCombinaciones, que recibe dos parámetros: "cadenas", una secuencia de secuencias de caracteres representando cadenas válidas, y "n", la longitud de la cadena objetivo.
- Ahora bien, si n es 1 o menor, retorna las cadenas sin alteraciones, ya que son válidas con longitud 1 o menor. Si n es mayor que 1, utiliza la función view para crear una vista perezosa de las cadenas, evitando evaluarlas hasta que sea necesario. Luego, emplea la función flatMap para generar todas las combinaciones de cadenas de longitud 2n a partir de las cadenas de longitud n. Esto se logra concatenando cada cadena consigo misma mediante la función map. El resultado será una secuencia donde cada subsecuencia representa una cadena de longitud 2n. Finalmente, convierte la vista perezosa en una secuencia estricta con la función toSeq.

- Se llama a la función "filtrar" con "cadenas" y las nuevas cadenas generadas como parámetros para obtener una secuencia de cadenas filtradas. Después, llama recursivamente a generarCombinaciones con las cadenas filtradas, pero únicamente con aquellas que el oráculo acepta como válidas mediante la función filter. Además, divide el valor de "n" entre 2 para acercarse al caso base.
- La función principal, reconstruirCadenasTurboMejorada, invoca generarCombinaciones con la variable que contiene las letras del alfabeto y "n" como parámetros, y luego devuelve la primera cadena de la secuencia resultante utilizando la función head. Esta cadena es reconocida como correcta por el oráculo.

reconstruirCadenasTurboMejorada(paralela):

Esta función usa la misma variable SC que antes, que es una secuencia de todas las cadenas de longitud 1 que cumplen el oráculo. A continuación, los pasos para su implementación:

- La función usa una función auxiliar llamada filtrarCadenas, que recibe una secuencia de cadenas y otra secuencia de cadenas, y devuelve una secuencia de todas las combinaciones posibles de esas cadenas que también cumplen el oráculo.
- Esta función usa las funciones flatMap para generar las combinaciones, y luego verifica si la nueva cadena tiene longitud 2 o si es una concatenación de dos cadenas que ya están en la secuencia original. Si es así, devuelve la nueva cadena; si no, devuelve None.
- la función usa otra función auxiliar llamada generarCombinaciones, que recibe una secuencia de cadenas y un número entero n, y devuelve una secuencia de todas las combinaciones posibles de esas cadenas de longitud n que también cumplen el oráculo.
- Se usa la paralelización para dividir la secuencia de cadenas en dos partes y aplicar la función filtrarCadenas a cada parte, usando la función parallel.
- Se filtran las combinaciones que no cumplen el oráculo, usando la función filter. El proceso se repite hasta que n sea menor o igual a 1, en cuyo caso se devuelve la secuencia de cadenas original.
- Finalmente, la función reconstruirCadenasTurboMejorada llama a la función generarCombinaciones con la variable SC y el argumento n, y devuelve la primera cadena de la secuencia resultante, usando la función head. Si la secuencia está vacía, significa que no hay ninguna cadena que cumpla el oráculo, y la función devolverá una excepción.

reconstruirCadenasTurboAcelerado:

Esta función es la más avanzada y eficiente que todas las anteriores, para lograr esto usa una estructura de datos llamada trie para almacenar y buscar las cadenas válidas de forma más eficiente. A continuación, los pasos de la implementación:

- Se define una función auxiliar llamada transformarCadena que recibe una secuencia de caracteres y devuelve una secuencia de secuencias de caracteres que contienen todas las subcadenas posibles de la cadena original, desde la más larga hasta la más corta. Para ello, usa la anotación @tailrec para indicar la recursión de cola, y usa el patrón de coincidencia para analizar la secuencia. Si la secuencia es vacía, devuelve el acumulador acc, que contiene las subcadenas generadas. Si la secuencia tiene al menos dos elementos, llama recursivamente a transformarCadena con el resto de la secuencia, y agrega el resto de la

secuencia al acumulador. Si la secuencia tiene un solo elemento, devuelve el acumulador sin modificar.

- Se define una variable SC que contiene todas las letras del alfabeto que el oráculo acepta como cadenas válidas de longitud 1. Para ello, usa la función map para convertir cada letra en una secuencia de un solo elemento, y luego la función filter para quedarse solo con las que el oráculo aprueba.
- Define una función auxiliar llamada filtrar que recibe dos parámetros: cadenasOriginales, que es un trie que contiene las cadenas válidas de una cierta longitud, y nuevasCadenas, que es una secuencia de secuencias de caracteres que representan cadenas generadas a partir de las originales, pero con el doble de longitud.
- Si la longitud de las nuevas cadenas es 2, devuelve nuevasCadenas sin modificar, ya que son las cadenas válidas de longitud 2. Si la longitud de las nuevas cadenas es mayor que 2, usa la función filter para eliminar las que no cumplen una condición. La condición es que cada subcadena de longitud igual a la mitad de la nueva cadena debe estar contenida en el trie cadenasOriginales. Para ello, usa la función view para crear una vista perezosa de la nueva cadena, y luego la función sliding para obtener todas las subcadenas posibles de longitud igual a la mitad de la nueva cadena, moviéndose de uno en uno. Luego, usa la función forall para verificar que todas esas subcadenas estén en el trie cadenasOriginales, usando la función buscar que se define más adelante. El resultado es una secuencia de secuencias de caracteres, donde cada subsecuencia representa una cadena que pasa el filtro.
- Tambien, define una función auxiliar llamada generarCombinaciones que recibe dos parámetros: cadenas, que es una secuencia de secuencias de caracteres que representan cadenas válidas, y n, que es la longitud de la cadena objetivo. Si n es menor o igual que 1, devuelve cadenas sin modificar, ya que son las cadenas válidas de longitud 1 o menor. Si n es mayor que 1, usa la función view para crear una vista perezosa de cadenas, que evita evaluarlas hasta que sea necesario. Luego, usa la función flatMap para generar todas las combinaciones de cadenas de longitud 2n a partir de las cadenas de longitud n. Para ello, toma cada cadena de cadenas y la concatena consigo misma, usando la función map. El resultado es una secuencia de secuencias de caracteres, donde cada subsecuencia representa una cadena de longitud 2n. Finalmente, usa la función toSeq para convertir la vista perezosa en una secuencia estricta.
- Llama a la función transformarCadena con cada cadena de cadenas, y obtiene una secuencia de secuencias de caracteres que contienen todas las subcadenas posibles de cada cadena original. Luego, usa la función flatten para aplanar la secuencia, y la función sortBy para ordenarla por longitud. El resultado es una secuencia de secuencias de caracteres ordenada por longitud, que contiene todas las cadenas válidas de longitud menor o igual que n.
- Llama a la función construirTrie con la secuencia anterior, y obtiene un trie que almacena todas las cadenas válidas de forma compacta y eficiente. La función construirTrie se define más adelante. De igual forma llama a la función filtrar con el trie y las nuevas cadenas generadas como parámetros, y obtiene una secuencia de cadenas filtradas. Luego, llama recursivamente a generarCombinaciones con las cadenas filtradas, pero solo con las que el

oráculo acepta como válidas, usando la función filter. También divide el valor de n entre 2, para acercarse al caso base.

- Por último, la función principal reconstruirCadenasTurboAcelerado llama a generarCombinaciones con SC y n como parámetros, y luego devuelve la primera cadena de la secuencia resultante, usando la función head. Esta cadena es la que el oráculo reconoce como correcta.

reconstruirCadenasTurboAcelerado(paralela):

Esta función paralela es la más avanzada, aquí se reducen aún más el espacio de búsqueda usando estructuras de datos más eficientes y código optimizado. A continuación, los pasos para su implementación:

- La función usa la variable inmutable SC, que es una secuencia de todas las cadenas de longitud 1 que cumplen el oráculo. Luego, la función usa una función auxiliar llamada transformarCadena, que recibe una secuencia de caracteres y un acumulador opcional, y devuelve una secuencia de todas las subsecuencias posibles de esa secuencia, desde la más larga hasta la más corta. Esta función usa una anotación @tailrec para indicar que es recursión de cola.
- La función usa match para analizar la secuencia de entrada y devolver el acumulador cuando está vacía, o añadir la secuencia actual al acumulador y llamar a la función con la cola de la secuencia.
- Después, la función usa otra función auxiliar llamada filtrar, que recibe una secuencia de cadenas, un Trie y otra secuencia de cadenas, y devuelve una secuencia de todas las combinaciones posibles de esas cadenas que también cumplen el oráculo. Esta función usa las funciones flatMap para generar las combinaciones, y luego verifica si la nueva cadena tiene longitud 2 o si es una concatenación de dos cadenas que ya están en el árbol de prefijos. Si es así, devuelve la nueva cadena; si no, devuelve None.
- La función usa otra función auxiliar llamada generarCombinaciones, que recibe una secuencia de cadenas y un número entero n, y devuelve una secuencia de todas las combinaciones posibles de esas cadenas de longitud n que también cumplen el oráculo. Usa la paralelización para dividir la secuencia de cadenas en dos partes y aplicar la función transformarCadena a cada parte, usando la función parallel. Luego, ordena las subsecuencias por su longitud, usando la función sortBy.
- Como siguiente paso, construye un árbol de prefijos con todas las subsecuencias, usando la función construirTrie. Finalmente, aplica la función filtrar a cada parte de la secuencia de cadenas, usando el árbol de prefijos como argumento. El proceso se repite hasta que n sea menor o igual a 1, en cuyo caso se devuelve la secuencia de cadenas original.
- Finalmente, la función reconstruirCadenasTurboAcelerado llama a la función generarCombinaciones con la variable SC y el argumento n, y devuelve la primera cadena de la secuencia resultante, usando la función head. Si la secuencia está vacía, significa que no hay ninguna cadena que cumpla el oráculo, y la función devolverá una excepción.

Anuncio:

Técnica de paralelización usada: Se usó la técnica de paralelización de datos, introduciendo la función parallel, en todas las funciones. Para esto, se divide el conjunto de datos (el alfabeto) en dos partes y se asigna cada parte a un hilo de ejecución diferente. Luego, se combinan los resultados de cada hilo para obtener la solución final. Por otra parte, en términos de desempeño del programa, se puede evidenciar que siempre que el número de caracteres de las cadenas sean más grandes, será más eficiente utilizar la paralelización de datos; en contraste, si las cadenas son muy pequeñas el costo de dividirlas hace que su eficiencia baje y sea mejor ir por la opción secuencial.

Implementación de Trie:

Esta implementación busca representar un árbol de prefijos en Scala, usando clases case y patrones de emparejamiento.

Primeramente, definimos una clase abstracta Trie, y dos subclases Nodo y Hoja, que heredan de Trie.

La clase Nodo tiene tres campos:

- car, que es un carácter que representa el prefijo actual.
- marcado, que es un valor booleano que indica si el nodo corresponde al final de una cadena.
- hijos, que es una secuencia de opciones de Trie, que pueden ser None si no hay ningún hijo con esa letra, o Some(trie) si hay un subtrie con esa letra.

La clase Hoja tiene dos campos que representan lo mismo que los de la clase nodo, pero en este caso no tienen hijos:

- Car
- Marcado

Tambien, se definieron cuatro funciones, que son:

- convertirNodosSinHijosEnHojas, que recibe un trie y devuelve otro trie con la misma estructura, pero con los nodos que no tienen hijos convertidos en hojas.
- construirTrie, que recibe una secuencia de secuencias de caracteres, y devuelve un trie que contiene todas las secuencias y sus valores asociados.
- buscar, que recibe un trie y una secuencia de caracteres, y devuelve un valor booleano que indica si la secuencia está contenida en el trie o no.
- insertarEnTrie, que recibe un trie y una secuencia de caracteres, y devuelve un nuevo trie que contiene la secuencia y su valor asociado.

Colecciones paralelas utilizadas:

Se utilizaron colecciones paralelas en las funciones reconstruirCadenasTurbo, reconstruirCadenasTurboMejorada y reconstruirCadenasTurboAcelerado. Estas funciones usan el método par para obtener versiones paralelas de las colecciones alfabeto, cadenas y SC, y luego aplican operaciones como flatMap, filter, map y reduce sobre ellas.

Impacto de las técnicas de paralelización en el desempeño del programa:

De acuerdo a los resultados presentados, se puede decir que las técnicas de paralelización implementadas generan mayor eficiencia y optimización, a la hora de analizar cadenas con un mayor número de caracteres, ya que reducen el tiempo de ejecución y aprovechan mejor los recursos del sistema; así que, siempre que se tenga un numero de caracteres mayor la paralelización tiende a ser la mejor opción a usar.

Versus de datos:

reconstruirCadenasIngenuo vs reconstruirCadenasIngenuoParalelo:

Caracteres	Secuencial	Paralelo	Aceleración
2	0.6538	0.6956	0.9399
4	2.1553	1.8014	1.1965
8	49.0589	31.2018	1.5723

reconstruirCadenasMejorado vs reconstruirCadenasMejoradoParalelo:

Caracteres	Secuencial	Paralelo	Aceleración
2	0.217	0.4301	0.5045
4	0.1779	0.4942	0.3600
8	0.3908	1.1277	0.3465
16	0.591	1.4796	0.3994
32	4.2801	6.8267	0.6270

Caracteres	Secuencial	Paralelo	Aceleración
64	54.0709	47.1303	1.1473
128	467.3766	452.5126	1.0328
256	3380.3589	3300.8187	1.0241

reconstruirCadenasTurbo vs reconstruirCadenasTurboParalelo:

Caracteres	Secuencial	Paralelo	Aceleración
2	0.1553	0.295	0.5264
4	0.2242	1.0017	0.2238
8	0.0876	0.5566	0.1574
16	0.4025	1.0709	0.3759
32	9.341	7.7583	1.2040
64	91.1123	71.8203	1.2686
128	712.1774	766.101	0.9296
256	6615.7742	6366.5013	1.0392

reconstruirCadenasTurboMejorado vs reconstruirCadenasTurboMejoradoParalelo:

Caracteres	Secuencial	Paralelo	Aceleración
2	0.2311	0.2876	0.8035
4	0.9498	0.6123	1.5512
8	2.4076	0.7076	3.4025
16	1.6879	6.9103	0.2443
32	5.5024	2.6132	2.1056
64	44.2786	28.8723	1.5336
128	415.46	236.1105	1.7596
256	2738.8749	1717.1103	1.5950

reconstruirCadenasTurboAcelerado vs reconstruirCadenasTurboAceleradoParalelo:

Caracteres	Secuencial	Paralelo	Aceleración
2	0.5145	0.555	0.9270
4	1.2287	1.0622	1.1568
8	2.008	1.1311	1.7753
16	2.7312	1.7243	1.5839

Caracteres	Secuencial	Paralelo	Aceleración
32	3.2712	5.1776	0.6318
64	30.2402	32.7784	0.9226
128	327.1197	189.9066	1.7225
256	1889.7182	1342.8718	1.4072

Conclusión sobre el grado de aceleración logrado:
De acuerdo al análisis de datos hecho, se puede concluir lo siguiente sobre el grado de aceleración:

- Si el grado de aceleración es mayor que uno, significa que la versión paralela se ejecuta más rápido que la secuencial.
- Si el grado de aceleración es menor que uno, significa que la versión paralela se ejecuta más lento que la secuencial.
- Si el grado de aceleración es igual a uno, significa que ambas versiones se ejecutan al mismo tiempo.

Tambien, se puede decir que, donde se tiene una cadena de caracteres grande, la paralelización llega a ser más eficiente, pero esto no aplica para todos los casos ya que se encuentran algunas fluctuaciones en los resultados.