PROYECTO

EL PROBLEMA DE LA RECONSTRUCCIÓN DE CADENAS

LAURA SOFIA PEÑALOZA LÓPEZ - 2259485 - 3743

ESMERALDA RIVAS GUZMÁN - 2259580 - 3743

LAURA TATIANA COICUE POQUIGUEGUE – 2276652-3743

DIRECTOR

CARLOS ANDRES DELGADO

UNIVERSIDAD DEL VALLE SEDE TULUÁ

FACULTAD DE INGENIERÍA

PROGRAMA ACADÉMICO DE INGENIERÍA DE SISTEMAS

CURSO FUNDAMENTOS DE PROGRAMACIÓN FUNCIONAL Y CONCURRENTE (750013C)

TULUÁ – VALLE DEL CAUCA

2023

1. Funciones secuenciales

1.1. Implementando la solución ingenua

reconstruirCadenaIngenuo recibe dos argumentos n de tipo entero y o de tipo oráculo que proporciona información sobre qué subcadenas son válidas o aceptables, devolviendo una secuencia de caracteres, dentro de esta función tenemos la función auxiliar generarCadena la cual recibe como parámetro n de tipo entero cadena de tipo secuencias de caracteres iniciada en una secuencia vacía por defecto esta función devuelve una secuencia de secuencias de caracteres esta función es recursiva y genera todas las posibles cadenas de longitud n. el flatMap genera las nuevas cadenas nuevas agregando cada carácter del alfabeto a la cadena nueva. Al final se llama la función generaCadena la cual tiene como valor n luego busca una cadena que cumpla la condición de o de tipo oráculo, si encuentra una cadena que cumpla con la condición, la devuelve, si no devuelve una secuencia vacía.

1.2.Implementando la solución mejorada

Estas funciones tienen un mejor enfoque y es más eficiente que el método ingenuo, ya que evita generar todas las posibles cadenas de longitud n.

reconstruirCadenaMejorado recibe dos argumentos, n de tipo entero y o de tipo oráculo que proporciona información sobre que subcadenas son válidas o aceptables, devolviendo una secuencia de caracteres, dentro de esta función tenemos la función auxiliar generarSubC que recibe como argumento k de tipo entero, subcadena un conjunto de secuencias de caracteres, devolviendo un conjunto de secuencias de caracteres. Si k es mayor a n devuelve un conjunto subCadena de lo contrario crea un nuevo conjunto mediante operaciones de mapeo y filtración en subCadena. Se crea una variable nSubC donde el flatMap crea nuevas secuencias concatenando s1 con cada elemento de s2 del alfabeto, luego filtra estas secuencias utilizando el objeto o obteniendo como resultado un conjunto nSubC. Se crea una variable subCadena la cual llama la función generarSubC que recibe como valor K = 1 y un conjunto de secuencias vacías generando subconjuntos, después de esto se busca una cadena en subCadena que tenga una longitud igual a n.

1.3. Implementando la solución turbo

reconstruirCadenaTurbo recibe dos argumentos, n de tipo entero, o de tipo oráculo que proporciona información sobre qué subcadenas son válidas o aceptables, devuelve una secuencia de caracteres, dentro de ella tenemos una función auxiliar generarSubC la cual recibe dos argumentos k de tipo entero, subcadena un conjunto de secuencias de caracteres, devuelve un conjunto de secuencias de caracteres. Si k es mayor que n devuelve un conjunto de subCadena, de los contrario crea una variable nSubC la cual crea nuevas secuencias, con el método flatMap crea todas las combinaciones posibles de concatenaciones de s1 con cada elemento de s2 en subCadena, se filtra utilizando el objeto o, el resultado final es un conjunto de nSubC. La variable ISubC crea un conjunto de ISubC a partir del alfabeto, cada elemento de ISubC es una secuencia de un solo carácter. En la variable subCadena se llama la función generarSubC la cual recibe como valor k = 2 y el conjunto de ISubC, esto genera subconjuntos de cadenas, la siguiente función subCadena.find busca una cadena en subCadena que tenga una longitud igual a n si encuentra una cadena con esa condición la devuelve si no devuelve una secuencia vacía.

1.4.I*mplementando la solución turbo mejorada*

reconstruirCadenaTurboMejorado toma dos argumentos n de tipo entero, o de tipo oráculo que proporciona información sobre qué subcadenas es válida o aceptables, dentro de esta función tenemos una función auxiliar filtrar recibe dos argumentos subCadena un conjunto de secuencias de caracteres, k un entero la función devuelve un conjunto de secuencias de caracteres, el método flatMap crea todas las combinaciones posibles de concatenaciones de s1 con cada elemento de s2 en subCadena el método filter filtra las secuencias de caracteres que cumplan con la condición, para cada índice i en el rango 0 a s.length-k, existe una subcadena en subCadena que coincida con s.drop(i).take(k) , el oráculo o devuelve true para la secuencia s.

1.5.I*mplementando la solución turbo acelerada*

reconstruirCadenaTurboAcelerada toma dos parámetros, n de tipo entero, o de tipo oráculo, esta función devuelve una secuencia de caracteres. Dentro de ella se

implementó dos funciones la primera generarSubC que toma como argumento k un valor entero, subCadena un conjunto de secuencias de caracteres si k es mayor o igual que n, devuelve una subCadena, de lo contrario crea una nueva secuencia nSubc concatenando todas las combinaciones posibles de elementos de subCadena, filtra nSubC utilizando la función filtrar, llama recursivamente a generarSubC con un valor duplicado de k y los resultados filtrados. La función filtrar la cual recibe tres parámetros Cactual un conjunto de secuencias de caracteres, Canterior un conjunto de secuencias de caracteres y k de tipo entero. Si la longitud del primer elemento de la cadenaActual es mayor a 2, crea una variable t donde donde llama a la funcion arbolDeSufijos con el argumento Canterior donde se crea un arbol de sufijos a partir de Cantero, filtra Cactual utilizando una funcion que verifica si todos los segmentos de longitud k de cada subcadena pertenecen al arbol de sufijos, devuelve la secuencia resultante si no se cumple la condicion devuelve Cactual.

1.5.1. Implementación del Trie

En general, esta implementación de Trie está diseñada para construir y manipular árboles de prefijos, que son útiles para almacenar y buscar cadenas de manera eficiente.

1.5.2. Funciones porque son correctas estas funciones: la función pertenece verifica si una secuencia de caracteres está presente en el trie y devuelve true si está marcada (completa) o falsé si no lo está.

- pertenece contiene dos parámetros s secuencia de caracteres t es el trie en el que estamos buscando la secuencia s, devuelve un booleano. Si s está vacía entonces se hacen dos casos donde si t es un nodo interno, devuelve un valor de marcada, si t es un hoja también devuelve un valor de marcada y si s no es vacía entonces si t si es un nodo interno, verifica si hay algún hijo cuya raíz coincida con el primer carácter s, si es así llama a pertenecer con el resto de la secuencia s.tail y ese hijo o si t es una hoja devuelve false porque no pueden haber más caracteres después de una hoja.
- Adicionar recibe dos argumentos s secuencia de caracteres que deseamos agregar al trie t es el trie en el que queremos insertar la secuencia s,

devuelve un objeto de tipo trie, dentro de esta función encontramos dos funciones internas crearRama que toma como parámetro s una secuencia de caracteres, devolviendo un trie Utiliza el patrón match para descomponer s en dos partes, cabeza y cola . Si la cola no está vacía, crea un nodo, con cabeza como valor, no marcado, y una lista que contiene una rama recursiva creada a partir de cola. Si la cola está vacía, crea una hoja con cabeza como valor y marcada. Si s está vacía, crea un nodo con un espacio en blanco como valor y no marcado. En resumen, crearRama construye una rama del árbol trie a partir de una secuencia de caracteres. agregarRama toma tres argumentos arbolActual un objeto de tipo Trie, prefix: una secuencia de caracteres, remaining: otra secuencia de caracteres. Utiliza el patrón match para manejar diferentes casos: Si el árbol actual es un nodo y el primer carácter de remaining está en la lista de cabezas de los hijos del nodo, se actualizan los hijos según el camino deseado. Si el árbol actual es una hoja y aún quedan caracteres en remaining, se crea un nuevo nodo con la rama generada a partir del remaining. Si el árbol actual es un nodo y el camino se detiene en un nodo, se agrega un nuevo nodo a la lista de hijos. Si el árbol actual es un nodo no marcado y no quedan caracteres en remaining, se marca el nodo. - En cualquier otro caso, se devuelve el árbol actual sin cambios. La función adicionar devuelve el resultado de la función agregarRama aplicada al árbol actual, el prefijo y la secuencia restante.

- arbolDeSufijos recibe como parámetro ss que representa un conjunto de secuencias de caracteres a partir del cual se quiere construir el árbol de sufijos, devuelve un objeto de tipo trie, el método foldLeft se utilizada para iterar sobre los elementos del conjunto ss y acumular el resultado, el valor inicial del acumulados es una hoja del árbol de sufijos con un carácter cualquiera y un indicador de final de sufijo igual a falso devolviendo un objeto trie, la función lambda de pliegue (t, s) => tomas dos argumentos donde t es el acumulador actual y s es la secuencia de caracteres, llama a la función adicionar con la secuencia de caracteres s y el acumuludor t esta

funcion se encarga de agregar la secuencia s al árbol de sufijos representado por t.

2. Funciones paralelas en estas funciones se usó paralelismo de datos.

Cabe resaltar que en estas funciones se aplica la paralelización en la generación y filtrado de subcadenas utilizando la función parallel. Esto permitió mejorar el rendimiento al procesar conjuntos grandes de datos de manera simultánea.

2.1. Implementando la solución ingenua paralela

reconstruirCadenaIngenuoPar toma tres argumentos, umbral de tipo entero, n de tipo entero, o de tipo oráculo, devuelve una secuencia de caracteres. Si umbral es menor o igual que n se llama a la función reconstruirCadenaIngenuo con argumentos n y o para construir la cadena con un enfoque secuencial. En caso contrario se define una función llamada generarCadena donde toma dos argumentos n una secuencia de caracteres inicialmente vacía, si n es igual a cero se devuelve una secuencia que contiene solo la cadena, de lo contrario se calcula el tamaño del alfabeto, se divide en dos partes sub1 y sub2 donde p1 contiene las cadenas resultantes al agregar cada carácter de sub1 a la cadena, p2 contiene las cadenas resultantes al agregar cada carácter de sub2 a la cadena. La función parallel paraleliza las llamadas a la función generarCadena con los subconjuntos de sub1 y sub2, donde se ejecutan las llamadas en paralelo y devuelve un resultado, se concatenan p1 y p2. Finalmente se busca una cadena en la secuencia generada por la función generarCadena(n) que cumpla con la condición de o, si no se encuentra una se devuelve una secuencia vacía.

2.2. *Implementando la solución mejorada paralela*

reconstruirCadenaMejoradoPar toma tres argumentos umbral de tipo entero n de tipo entero, o de tipo oráculo, devuelve una secuencia de caracteres. Si umbral es menor o igual que n se llama a la función reconstruirCadenaMejorado con argumentos n y o para construir la cadena con un enfoque secuencial. En caso contrario se define una función llamada generarSubC que toma dos argumentos k de tipo entero, subCadena un conjunto de secuencias de caracteres, donde si k es mayor a n se devuelve una subCadena si no, calcula el tamaño m del conjunto subCadena, divide

subcadena en dos partes, sub1 y sub2 tomando la mitad de elementos en cada parte, la funcion parallel ejejuta las operaciones en paralelo el flatMap realiza para cada elemento de s1 en el conjunto sub1 se aplica una funcion que genera nuevas secuencias concatenando cada elemento s1 con cada elemento del alfabeto, el filter(o) filtra los resultados que cumplan la condicion proporcionada por 'o', esto se realiza de igual modo para el sub2. El resultado de la ejecución paralela se asigna a las variables (p1, p2) los resultados de las operaciones paralelas aplicadas a sub1 y sub2. Finalmente llama a generarSubC con k + 1 y la concatenación de p1 con p2. La función principal devuelve la secuencia de caracteres encontrada en subcadena que tiene una longitud igual a n, o una secuencia vacía si no se encuentra ninguna.

2.3. Implementando la solución turbo paralela

reconstruirCadenaTurboPar toma tres argumentos umbral de tipo entero, n de tipo entero, o de tipo oráculo, devuelve una secuencia de caracteres. Si umbral es menor o igual que n se llama a la función reconstruirCadenaTurbo con argumentos n y o, para construir la cadena con un enfoque secuencial. De los contrario se define una función interna llamada generarSubC que recibe dos argumentos k de tipo entero y subCadena un conjunto de secuencias de caracteres, si k es mayor que n devuelve una subCadena, si no se crea una variable (p1, p2) que contiene el resultado de la división de subCadena sub1 que toma la mitad inicial de la colección de subCadena y sub2 toma la mitad restante de subCadena, con el método flatMap para cada subcadena s1 en una mitad, genera todas las combinaciones posibles con todas las subcadenas de s2 en subCadena, luego filtra esa combinaciones según la condición dada por el oráculo o, la operación parallel ejecuta estas operaciones en paralelo en ambas mitades de subcadena, dividiendo el trabajo. los resultados filtrados se combinan en nuevos subconjuntos p1 y p2, se llama la función generarSubC con k*2 y la concatenación de p1 y p2. Se crea una variable ISubC que contiene secuencias de un solo carácter del objeto alfabeto. Al finalizar la función devuelve la secuencia de caracteres encontrada en subCadena que tiene una longitud igual a n, o una secuencia vacía si no se encuentra.

2.4. Implementando la solución turbo mejorada paralela

reconstruirCadenaTurboMejoradaPar toma tres parámetros, umbral de tipo entero, n de tipo entero, o un objeto de tipo oráculo, devuelve una secuencia de caracteres. Si umbral es menor o igual que n se llama a la función reconstruirCadenaTurboMejorada con argumentos n y o, para construir la cadena con un enfoque secuencial. De lo contrario se definen dos funciones la primera generarSubC que recibe como parámetro k de tipo entero subCadena un conjunto de secuencias de caracteres, si k es mayor que n devuelve una subCadena si no crea una variable nSubC donde se llama la función filtrar que tiene como argumento una subCadena y k. finalmente se llama la función generarSubC con k*2 y la colección de nSubC filtrada nuevamente con la condición dada por el oráculo. La función filtrar toma los mismos parámetros de la función generarSubC, subCadena y k divide subCadena en dos mitades para cada par de secuencias s1 y s2 las concatena en una nueva secuencia s, luego crea ventanas deslizantes de tamaño k/2 a partir de s, si k es 2 devuelve una secuencia de caracteres de lo contrario verifica si todas las ventanas deslizantes están contenidas en la subCadena original. Si es así, devuelve una secuencia de caracteres y si no devuelve una secuencia vacía. Finalmente combina los resultados de las dos mitades p1 y p2 y devuelve su concatenación.

2.5. Implementación de la solución turbo acelerada paralela

reconstruirCadenaTurboAceleradaPar toma tres argumentos umbral de tipo entero, n de tipo entero y o de tipo oraculo, esta función devuelve una secuencia de caracteres. Si umbral es menor o igual que n se llama a la función reconstruirCadenaTurboAceleradaPar con argumentos n y o, para construir la cadena con un enfoque secuencial. En caso contrario se define una función llamada generarSubC que toma dos argumentos k de tipo entero, subCadena un conjunto de secuencias de caracteres, donde si k es mayor o igual a n se devuelve una subCadena si no, calcula el tamaño m del conjunto subCadena, divide subcadena en dos partes, sub1 y sub2 tomando la mitad de elementos en cada parte, la funcion parallel ejecuta las operaciones en paralelo el flatMap realiza para cada elemento de s1 en el conjunto

sub1 se aplica una función que genera nuevas secuencias concatenando cada elemento s1 con cada elemento del alfabeto, luego se filtran los resultados combinados en (p1 ++ p2). finalmente se llama se llama recursivamente a generarSubC con un valor duplicado de k y los resultados filtrados .La función filtrar toma tres argumentos, Cactual un conjunto de secuencias de caracteres, Canterior un conjunto de secuencias de caracteres k un valor entero. Si la longitud de la cabeza de Cactual es mayor que 2, se crea un árbol de sufijos (t) a partir de Canterior. Se divide Cactual en dos partes y se filtran ambas partes utilizando una función que verifica si todos los segmentos de longitud k de cada subcadena pertenecen al árbol de sufijos. Los resultados filtrados se combinan en p1 ++ p2. si no se cumple la condicion anterior devuelve Cactual. Finalmente se crea una secuencia inicial ISubC a partir del alfabeto y se filtra utilizando el objeto o. Luego, se llama a generarSubC con un valor inicial de k = 1 y se toma la cabeza de la cadena resultante como el resultado final.

3.Comparacion de resultados secuenciales y paralelizados

3.1 *Algoritmo Ingenuo*

Tamaño Cadena	Paralelizada	Normal	Aceleración
2	0.0465	0.0225	2.066666666666667
4	0.3683	0.6683	0.5510998054765824
8	16.1354	16.2155	0.9950602818291142

3.2. *Algoritmo Mejorada*

Tamaño Cadena	Paralelizada	Normal	Aceleración
2	0.2971	0.096	3.094791666666666
4	0.4178	0.1007	4.148957298907646
8	0.4072	0.3434	1.1857891671520093
16	1.1154	0.6218	1.7938243808298486
32	1.4261	2.3477	0.607445585040678
64	11.4449	10.5608	1.0837152488447845

128	102.6149	104.4884	0.9820697799947172
256	970.989	895.9096	1.0838024282807106
512	10400.8798	12158.6811	0.8554282914780946

3.3. *Algoritmo Turbo*

Tamaño Cadena	Paralelizada	Normal	Aceleración
2	0.099	0.0801	1.2359550561797752
4	0.1509	0.0726	2.078512396694215
8	0.1425	0.1366	1.0431918008784773
16	0.2998	0.1416	2.117231638418079
32	1.3585	2.2941	0.5921712218299116
64	22.45	15.0343	1.4932520968718197
128	157.5304	156.2408	1.0082539259911623
256	3669.5875	1669.5499	2.1979501780689517
512	22455.2293	15403.0897	1.4578392866205276

3.4. *Algoritmo Turbo Mejorada*

Tamaño Cadena	Paralelizada	Normal	Aceleración
2	0.2682	0.0966	2.7763975155
4	0.863	0.2071	4.1670690487
8	0.5274	0.4849	1.0876469375
16	0.737	0.6632	1.1112786489
32	2.0433	1.9161	1.0663848442
64	16.8534	19.3048	0.8730160374
128	160.3452	162.5076	0.9866935454
256	1363.6007	1366.1789	0.9981128383

3.5. *Algoritmo Turbo Acelerada*

Tamaño Cadena	Paralelizada	Normal	Aceleración
2	0.1383	0.0262	5.278625954198473
4	0.2158	0.0714	3.0224089635854336
8	0.2791	0.2132	1.3090994371482176
16	0.4004	0.375	1.0677333333333332
32	2.0684	1.9472	1.0622432210353328
64	8.0444	5.0713	1.5862599333504228
128	59.0513	51.366	1.149618424638866
256	440.2564	448.015	0.9826822762630716
512	9048.3066	7651.0003	1.1826305378657482

4. Análisis comparativo de las diferentes soluciones, versión secuencial versión paralela

En el algoritmo de ingenuo podemos observar que la eficiencia parece variar según el tamaño de la cadena. Para tamaños de cadena pequeños como 2 el enfoque más eficiente es el secuencial. Sin embargo para tamaños con cadenas más grandes como la 4 y 8 la eficiencia en el de paralelizada es más eficiente, la aceleración varía dependiendo del tamaño de las cadenas.

En el algoritmo mejorado se observa que la eficiencia en los tiempos varían lo que quiere decir que la paralelización no es tan eficiente en las cadenas pequeñas, es más eficiente desde la cadena 64, esto quiere decir que el enfoque es más eficiente en las cadenas más grandes. La aceleración podemos observar que en la cadena 2 la aceleración es de 3 lo que indica que el enfoque paralelo es más eficiente, la aceleración en este caso es variada.

En el algoritmo de turbo podemos observar que para algunas cadenas la paralelización no es la más eficiente, es eficiente en la cadena 32 el enfoque paralelo. La aceleración no es constante.

El algoritmo turbo mejorada, la eficiencia es mejor en el enfoque paralelo desde la cadena 64 en las demás cadenas es más eficiente en enfoque secuencial. La aceleración es variada.

El algoritmo de turbo acelerada, podemos observar que en enfoque paralelo es eficiente solo en la cadena 256, en las demás cadenas es más eficiente el enfoque secuencial.

Con este análisis podemos decir que la aceleración no es constante, depende del tamaño de la cadena y la configuración del algoritmo, los tamaños de cadenas más grandes se benefician en la paralelización.

5. Conclusiones

- Se logró evidenciar que el uso de la programación funcional y concurrente es una herramienta fundamental para resolver problemas complejos y al mismo tiempo mejorar la eficiencia de los programas.
- La implementación de conceptos de programación funcional, como lo son las funciones de orden superior y la recursión, demuestran la habilidad de abordar y resolver problemas complejos de una manera eficiente.
- La construcción de pruebas y tests dentro del programa amplió significativamente el aprendizaje más allá de los conceptos bases del curso, brindando a los estudiantes conocimientos que serán fundamentales en proyectos y trabajos futuros.
- El trabajo en equipo permitió desarrollar habilidades de participación activa y comunicación efectiva dentro del grupo fortaleciendo las habilidades blandas en entornos de desarrollo colaborativo.
- Se lograron concretar conceptos de eficiencia y optimización comparando diferentes algoritmos buscando la forma de que el resultado sea el más rápido posible y además, compile de forma correcta.
- La solución de este gran problema de forma exitosa demuestra los conocimientos adquiridos a lo largo del curso, resaltando una actitud proactiva hacia el aprendizaje continuo formando una mentalidad orientada al crecimiento profesional y la adaptación a las evoluciones tecnológicas.