0 Identificación:

Juan Camilo Ruiz

Sergio Guzmán

1 Algoritmo de solución

Para este problema se eligió la exploración de grafos por tanto es una opción muy viable, ya que el problema propuesto es un problema NP-Completo. En esta solución, se hace la exploración de grafos con una función dominó sin necesidad de marcar estados.

Por el lado de la función dominó, si una solución sobrepasa cierta longitud de texto, no tiene como disminuir dicha longitud y por tanto se pueden descartar todos los sucesores de dicho estado. Por otro lado, en cuestión de marcar estados, no es necesario debido a que nunca se va a existir una repetición, ya que es imposible que dos estados diferentes tengan el mismo texto con los mismos subtextos. Así, en un comienzo se buscan las soluciones para las entradas recibidas, de modo que se agrega el estado inicial, el cual posee todos los subtextos recibidos junto con un texto vacío y el nivel N=0, a la agenda, y se empieza a sacar los estados de la agenda, se ve si son viables, si son soluciones, y por último se calculan sus sucesores para agregarlos a la agenda. Dicho cálculo surge añadiendo al texto del estado recibido uno de sus subtextos, verificar cuales de los otros subtextos se pueden haber formado para así calcular los subtextos del sucesor y aumentar el nivel N en 1. A su vez, para la función dominó, se efectúa un algoritmo *greedy* que elige aleatoriamente cual subtexto agregar al texto, calculando también los subtextos que se pueden haber formado y retornando el texto formado; así, en el algoritmo principal, se llama 5 veces a este algoritmo y se escoge la menor longitud de texto retornada, el cual da una referencia de longitud para dicho corte de rama de forma que si alguno de los estados poseen una longitud de texto mayor que la de referencia, se efectúa la selección. Por otro lado, un estado será una solución sí su arreglo de subtextos esta vació y su texto tiene una longitud menor que la referencia obtenida del *greedy*, en caso de que sea solución se actualiza la referencia de longitud de manera que ahora esta sea la longitud del estado que ya se sabe que es una solución, para disminuir aún más la cantidad de posibles soluciones. Así, se obtienen todas las posibles soluciones y posteriormente se elige la solución óptima, siendo esta la que posea la menor longitud de texto.

* Anotaciones: El único método que posee anotaciones de pre y pos condición, es el método de encontrar las soluciones, debido a que es el único que es predecible; es decir, es el único que se sabe con certeza lo que debe ocurrir antes y después, ya que en los otros dependerá del caso y el estado en el que se encuentre el algoritmo.

**private** List<CharacterChainState> findFeasibleSolutions(String sb [], **int** k) {

List<CharacterChainState> answer = **new** ArrayList<>();

//Estado inicial

CharacterChainState state = **new** CharacterChainState("", sb, 0);

//Agenda

agenda.add(state);

// P1: (estadoInicial.text = empty) ^ (estadoInicial.subtexts = sb) ^ (estadoInicial.N = 0) ^ (estadoInicial BELONGS agenda) ^ answer.isEmpty()

//En otras palabras, el estado inicial debe poseer el texto vacio, todos los subtextos recibidos en la entrada, y su nivel debe

//ser 0, además, este tiene que estar en la agenda

**while**(agenda.size()>0) {

//Elige el proximo estado de la agenda

state = agenda.poll();

//Verifica si es una solucion

**if**(isSolution(state)) answer.add(state);

//Agrega los sucesores a la agenda

List<CharacterChainState> successors = getSuccessors (state, k);

agenda.addAll(successors);

//R1: agenda.isEmpty ^ (ForAll state | isSolution(state) : state BELONGS answer)

// Al final de este while, la agenda debe estar vacía y todos los estados que sean solucion deben estar en el arrayList answer.

}

**return** answer;

}

2 Análisis de complejidades espacial y temporal

La complejidad espacial, en el peor de los casos, es de O(N!) siendo N la cantidad de subtextos, debido a que no va a haber ningún corte de ramas, por tanto ningún subtexto estará contenido al concatenar otros subtextos con el texto original, de este modo, se van a tener que calcular todos los sucesores, los cuales son N! sucesores, haciendo de esta la complejidad espacial. En sí todos los estados estarían en la agenda y en la respuesta, que son las únicas estructuras usadas.

La complejidad temporal es sumamente difícil de calcular, ya que siempre depende del caso en el que se esté, además de que aun considerando únicamente el peor caso, la cantidad que efectúa de ciclos depende de diferentes formas de N siendo imposible calcular una única función de N para todos sus posibles valores. Sin embargo, procurando ser lo más precisos posible, se sabe que al comienzo, se posee un ciclo donde se calcula el *greedy* que se efectúa 5 veces, obteniendo una complejidad de O(5); posteriormente, se llama la función *findFeasibleSolutions*() la cual efectúa el *while* aproximadamente, en el peor de los casos, , dando una complejidad de O( ), sin embargo, dentro del *while* se calculan los sucesores, los cuales dan una complejidad cercana a N! Ya que cada vez se tiene 1 subtexto menos, de este modo, la complejidad total del método findFeasibleSolutions() es de O(). Finalmente, se realiza un *for each* para seleccionar la mejor solución, siendo en el peor caso todos los sucesores soluciones, este ciclo tendría complejidad de O(N!). De este modo, la complejidad aproximada de todo el algoritmo, es O(5)+ O() + O(N!), lo cual termina siendo únicamente O() siendo N la cantidad de subtextos.

3. Comentarios finales

La solución posee complejidades tanto espacial como temporal muy altas, pero esto se debe a que se considera el peor de los casos. Sin embargo, al poder cortar ramas se disminuye enormemente ambas complejidades, debido a que se descartan muchos sucesores; por tanto, en el peor de los casos tiene sentido que sea una complejidad tan alta considerando que debe, literalmente, calcular todos los posibles sucesores y además todos los estados son posibles soluciones.