Problem 1: Josués y Karles balanceados.

Rainel Fernández Abreu C412

Lázaro Alejandro Castro Arango C411

El problema:

Alejandra está comenzando a quedarse dormida en mitad de un turno de `**censura**`. La verdad es que ya conoce todo el contenido de la conferencia, pero no quiere cerrar los ojos y herir los sentimientos del profesor. Así que, para dejar que su mente se entretenga en algo, comienza a imaginar clones de Josué y Karel, colocados uno al lado del otro formando una lista. La imaginación de Alejandra es tan poderosa que empieza a realizar el siguiente proceso: Primero imagina dos listas cualesquiera de Josués y Karels. Luego trata de imaginar una tercera lista tal que las dos primeras son sublistas no necesariamente continuas de esta y que además esté balanceada.

Alejandra define una lista de Josués y Karels balanceada como una lista que cumple lo siguiente:

- Una lista formada por sólo un josué y un karel, en ese orden, está balanceada.
- Si a una lista balanceada, se le agrega un Josué al principio y un Karel al final, el resultado sigue estando balanceado.
- La concatenación de dos listas de Josués y Karel balanceadas es también una lista de Josués y Karel balanceada.

Ayude a Alejandra (a perder el tiempo) con su proceso. Dadas dos listas de Josués y Karel cualesquiera, encuentre la menor lista tal que las dos primeras son sublistas no necesariamente continuas de esta y además está balanceada.

Resumen

Dadas dos cadenas $str1, str2$ paréntesis, se quiere formar otra tal que cumpla:

- Debe estar balanceada.
- Debe ser de tamaño mínimo.
- Debe contener a $str1, str2$ como subsecuencias.

Solución fuerza bruta :

Una primera idea para atacar el problema pasa por generar todas las cadenas S de paréntesis balanceados de tamano mayor igual que la mayor de las cadenas de entradas dadas y comprobar que se cumplan los requisitos de salida. Esto es extremadamente costoso puesto que para cada índice i de la cadena generada existen dos posibilidades colocar "(" o ")"; lo que derivaría en un algoritmo exponecial de complejidad temporal $O(2^n)$ solo para generar las cadenas.

Como la cadena a formar debe ser balanceada una mejora al algoritmo anterior consta de formar solo aquellas cadenas balanceadas e ir comprobando si alguna contiene a las dadas como subsecuencia no continua. Para una cadena de tamano $2 * n$ deben existir n parentesis abiertos y la misma cantidad de cerrados pues es condición necesaria para que una cadena este balanceada. Luego basado en el algoritmo backtraking anterior agregamos la siguiente poda:

Coloco "(" cada vez que pueda hacerlo, esto sucede cuando el número de abiertos es menor que n y coloco) solamente cuando el número de cerrados colocados es menor que el número de abiertos colocados.

Con esas dos ideas fundamentales se contruyó el algoritmo tipo backtraking para saber dado un n (numero de parentesis abiertos y cerrados) cuantas cadenas de tamano $2 * n$ balanceadas existen. Es conocido que este número es coincidente con el numero de Catalán en n (a partir de ahora $C(n)$). Por tanto existen $C(n)$ cadenas válidas por lo que la complegidad de este código sería $O(C(n))$ y $O((2n)!/((n+1)! * n!)$ si se desarrolla $C(n)$.

Usando el generador anterior se va comprobando si cada cadena encontranda posee como subsecuencia a las dos dadas. Esto se realiza mediante un recorrido por la cadena resultante y si el caracter actual coincide con el primero en la cadena dada este se elimina y se continua el chequeo. Si al final las cadenas de entradas estan vacias es pq estan contenidas y se ha encontrado una solución. Sumando este paso la complejidad pasa a ser $O(2 * n * C(n))$, representando $2 * n$ el tamano de la cadena resultante. De no encontrarse solución para un tamano n se continua con $n + 1$ hasta encontar una cadena válida. La complejidad total de esta solución fuerza bruta es entonces $O(m * n * C(n))$ siendo m la cantidad de veces que se aumenta el tamano de la cadena.

A continuación se ilustra la idea anterior con imagenes.

Solución solución recursiva :

En esta idea se buscarán todas las cadenas que contengan a las cadenas de entrada como subsecuencias, sean estas A, B respectivamente. La cadena mínima que contiene a A, B como subcecuencias no es necesariamente la cadena mínima balanceada que las contiene,

ya no se cumple $A < B => Bal(A) < Bal(B)$. Para ello solo basta un contra ejemplo: Sea $A = ((((, B = ((())$ entonces tenemos $|Bal(A) = ((((()))))| > |Bal(B) = ((()))|$

Solución dinámica :

A continuación se recogen una serie de definiciones que serán útiles para la comprensión de la solución.

Definiciones:

- *Factor de balace:* El factor de balance de una cadena es el número de paréntesis abiertos menos el de paréntesis cerrados.
- *Cadena válida:* Una cadena se dice válida si cumple que, al recorrerla de incio a fin su factor de balance nunca se vuelve negativo.
- *Contiene:* Se dice que una cadena b está contenida dentro de una cadena a si b es sublista de a.

Se construye una lista de tres dimensiones llamada dp, donde la posición $dp[i][j][k]$ guarda el tamaño de la menor cadena válida que contiene los i primeros caracteres del $str1$, j primeros caracteres del $str2$ y tiene factor de balance k.

El tamaño de dp es $[len(str1) + 1] \times [len(str2) + 1] \times 301$. Para el caso del índice i, este va desde 0 hasta $len(str1) + 1$ porque son todos los prefijos del $str1$ incluyendo el prefijo vacío. Analogamente sucede para $str2$ y j. El índice k va desde 0 hasta 300 porque a lo sumo la cantidad de paréntesis abiertos en ambas cadenas es 300, nunca se hace negativo pues en el dp solo se analizan cadenas válidas.

Como el dp no contiene como está formada la cadena sino el tamaño de la menor cadena válida, se necesita de alguna manera construir la cadena, pues es la salida que se pide en el problema. Para ello se tiene una lista prv de 3 dimensiones, donde en cada posición de esta se almacena una tupla de tamaño 3. Luego $prv[i][j][k]$ nos dice la casilla desde la que se llego a esta. Esta matriz se actualiza junto con el dp, que se explica más adelante. Para construir la solución se recorre prv con un `bfs` partiendo de la casilla $prev[len(str1)], [len(str2)], [c]$. Cuando estamos en una casilla comprobamos como es el valor k de la casilla que venimos respecto a la posición actual; si $prvK < k$,con $prvK$ el valor de k de la anterior casilla entonces agregamos a nuestra solución un paréntesis "(", sino pues se agrega un paréntesis ")" . Al llegar a la casilla $(0, 0, 0)$ hemos contruido la solución, solo que está en orden inverso, basta hallar el reverso de esta y agregar los paréntesis cerrados para balancearla, se asegura que solo se necesitan cerrados para balancear pues siempre se forman cadenas válidas en el dp. Hacer el

recorrido hacia atrás con un `bfs` es efectivo pues en cada paso de iteración la solución crece en 1.

El caso base será $dp[0][0][0] = 0$ que representa la mejor solución de este subproblema del temaño de la menor cadena válida que contiene el prefijo vacío de $str1$ y $str2$; y tiene factor de balance 0, dicha solución es la cadena vacía. Los demás valores de la matriz se inicializan con valor 1000.

Para llenar la matriz del dp se recorre la misma por los índices i, j, k en ese orden. Suponga que en algún momento del ciclo para llenar la matriz dp se está en una casilla $dp[i][j][k] = l$, esto quiere decir que la cadena óptima de este subproblema para llegar a esta casilla tiene tamaño l, esta cadena es un posible prefijo de alguna solución general entonces es necesario hacerla crecer hasta alcanzar la óptima del problema. La única forma de crecerla es agregando un paréntesis al final, para no romper la consistencia del dp. Por lo anterior dicho se asume que no se lleva la cadena de forma explícita, si de forma implícita dado que se tiene el tamaño de la cadena. Para analizar el $dp[i][j][k]$ se consideran dos casos:

- Colocar un paréntesis abierto
- Colocar un paréntesis cerrado

Para el primer caso, si alguna cadena $str1$ o $str2$ para los índices i, j respectivamente tiene un paréntesis abierto entonces su índice mueve a la posicion siguiente, de esta menera se actualiza el prefijo que está contenido de cada cadena en la solución óptima. Como se agregó un paréntesis abierto el factor de balance aumentó en 1.

Para el caso de añadir un paréntesis cerrado, es análogo al primero solo que disminuyendo el factor de balance en 1. El factor de balance nunca se hace negativo porque el ciclo que recorre los k va desde 1 hasta 301, sin incluir este último.