Tecnológico de Monterrey

Actividad Integradora 1

Jeshua Nava Avila A01639282

Luis David López Magaña A00344656

Análisis y Diseño de Algoritmos Avanzados (gpo 601)

Profesor Gildardo Sánchez Ante

13 de septiembre de 2022

Con el propósito de practicar distintos algoritmos relacionados a la manipulación y búsqueda en strings, en esta actividad se implementó el algoritmo de Knuth–Morris–Pratt para la búsqueda de un string dentro de otro, así como la búsqueda de la subsecuencia más larga que comparten dos strings por medio de programación dinámica y el algoritmo de Manacher para poder encontrar el palíndromo más largo en cada transmisión.

En el caso del algoritmo KMP es necesario crear un array auxiliar denominado LPS (*Longest proper Prefix that is also a Suffix*). Tal como lo indica su nombre, en este array se guardará la longitud del prefijo más largo que también es un sufijo para cada substring que se extiende desde el índice 0 hasta cada posición. Al iterar sobre el patrón con dos apuntadores -uno que indica el *LPS* actual y otro que continúa sobre la cadena- se va almacenando el *LPS* en cada substring, aumentando si los caracteres en cada posición son equivalentes. En caso contrario, el apuntador auxiliar regresará a la posición que indique el *LPS* anterior con la finalidad de que pueda seguir buscando un posible candidato para substrings más grandes -si el *LPS* era 0 se indicará en el array que el siguiente también será 0-. Dado que se itera una sola vez por el string y se crea un array del mismo tamaño, las complejidades temporal y espacial son de O(n).

Esto facilita el funcionamiento de KMP, en el cuál también hay dos apuntadores -uno para el string principal y otro para el patrón-. Si los caracteres concuerdan ambas posiciones avanzan; en caso contrario, el apuntador del patrón regresará a la posición que indique el *LPS* anterior. De esta manera, si este apuntador logra recorrer todo el patrón significa que existe dentro del string principal. Dado que se itera a través del string principal dos veces en el peor caso donde los caracteres no concuerden y se deba regresar, la complejidad temporal es de O(n). Si ignoramos el array generado previamente en el preprocesamiento del *LPS*, no hay espacio utilizado para esta función, así que la complejidad espacial es de O(1).

Para determinar el substring más largo que comparten dos secuencias se utilizó una tabla que almacena la cantidad de caracteres equivalentes para cada substring. Se compara cada carácter del primer string con cada carácter del segundo; en caso de un par sea igual, se checa la cantidad de caracteres concordantes para la posición anterior de cada string y se le suma una ocurrencia. De esta manera es posible llevar un registro que facilita computar las ocurrencias sin recalcular las anteriores, mejorando el tiempo de ejecución. Se lleva la cuenta la mayor cuenta de caracteres concordantes y se regresa la posición del substring en cada secuencia con límites inclusivos. Al recorrer cada string por cada carácter de su contraparte, la complejidad temporal es de O(n*m), representando los tamaños de ambos. La tabla se genera de manera similar para guardar cada combinación, por lo que la complejidad espacial es de O(n*m).

Para el algoritmo Manacher, tuvimos que usar una función auxiliar que ayuda a ingresar carácteres especiales al string que se vaya a analizar. Se agregan estos caracteres especiales para evitar *bound checking* y para poder tomar en cuenta palíndromos de tamaño par.

Una vez con este nuevo string, utilizamos un arreglo que vaya guardando el tamaño o ancho de un palíndromo en posición *i*. También requerimos ayuda de otras variables que llevan un seguimiento del centro de un palíndromo detectado (*centro)*, la posición extrema máxima derecha de un palíndromo detectado (*right*) y el tamaño del palíndromo más largo detectado (*maxP*). Con esta información, podemos actualizar la ubicación de la variable

centro y *right* al momento de ir recorriendo el string. Para obtener el tamaño más largo, debemos de recorrer este arreglo auxiliar. Para regresar la información útil para el reto, regresamos el tamaño del palíndromo más largo y en qué índice que encuentra se puede obtener con las variables de *center* y *maxP*. La complejidad temporal de este algoritmo es de O(n). Pero la complejidad de espacio para en este caso es de O(n*m), ya que se utiliza un arreglo de tamaño doble o triple a la cantidad de caracteres del string (por la función auxiliar).