Campus Santa Fe

Materia: Análisis y Diseño de Algoritmos Avanzados (Gpo 602)

Evidencia 1: Actividad Integradora 1 (Reflexión)

Shaul Zayat Askenazi | A01783240

Domingo Mora Perez | A01783317

Sylvia Fernanda Colomo | A01781983

Juan Pablo Moreno Robles Arenas | A01374091

Fecha de Entrega: 07/09/2023

Para esta evidencia desarrollamos un proyecto que incluye 3 partes diferentes. La evidencia se divide en 3 algoritmos diferentes los cuales son: KMP, Buscador de palíndromos, y la subcadena más larga dentro de un texto. Para poder trabajar eficazmente decidimos dividir los componentes entre los 4 integrantes de nuestro equipo. El trabajo se dividió de la siguiente forma; antes del desarrollo de los programas tuvimos una junta donde las partes del proyecto se distribuyeron así: Fernanda (KMP), Shaul y Domingo (Palindromos), y finalmente subcadenas entre Juan Pablo y Fernanda. El desarrollo a lo largo de las semanas, nos juntamos después de clase para ver el estado del entregable y ver cómo nos ayudabamos para acabar en forma.

Parte 1: KMP es un algoritmo que busca patrones dentro de cadenas de texto. Lo que hace es que dentro de la cadena de texto encuentra el patrón que dicta el usuario. El código utilizado fue parte del KMP ya hecho previamente para la clase. La base de este código es la función hecha para regresar el vector con los prefijos, es decir las ocurrencias del patrón dentro del texto dado. Solo falta usar ese resultado dentro de la función llamada "kmp" que tiene como misión usar ese vector solo si hay coincidencias, si las hay imprime el lugar dentro del texto en el cual está el patrón dado. De lo contrario se le da un false. En nuestra opinión no hubo tanta complejidad en esta parte dado que ya teníamos parte del código avanzado, solo fue hacer una adaptación del mismo como se explicó previamente.

Parte 2: Para esta parte dividimos el algoritmo de Manacher en 3 funciones. La primera recibe la cadena de texto original y se encarga de transformarla en una nueva cadena que va estar intercalada con el símbolo de '$' que tienen la tarea de actuar como delimitadores. La siguiente función toma la nueva cadena que creamos y la recorre para verificarla. Después se toma un valor como el centro del string y se compara. Con esto vemos que mientras que el carácter de la derecha es igual que el de la izquierda, y mientras este proceso no sobrepase la cadena en búsqueda, aumenta el array de longitud como un contador. Finalmente nuestra última función, toma la lista de longitudes agarrando el valor más grande y la máxima longitud. A partir de esto crea una variable que almacena en un valor inicial de búsqueda y encuentra el palíndromo sobre la cadena original y lo devuelve. El proceso que más nos costó al hacer estas funciones para el algoritmo fue desarrollar la lógica de búsqueda para el palíndromo y también que nuestro while funcionara de manera correcta, la cual llevo varios actos de debugeo.

Parte 3: Para está parte nos decidimos basar en el algoritmo de sufijos, en cuanto a la lógica de cómo comparar nuestras cadenas que utilizamos para desarrollar el código. Nosotros, en vez de hacer un vector por cada concatenación, comparamos ambos strings desde la posición 0 y a si irá incrementado hasta encontrar una posición que coincida con el carácter de la cadena y este se guarda en la variable que guarda la cadena resultante. Así podemos ir comparando cada una de las posibilidades. Utilizamos esta manera de resolverlo, por qué nos permite optimizar la búsqueda de coincidencias sin tener que guardar subcadenas y poder hacer la comparación de manera más directa. Esto nos causó ciertos problemas en el while ya que implementar uno que

pueda hacer justamente lo que buscábamos sin que hubiera problemas al compilar la función fue algo complicado y resultó en que tuviéramos que meterle varios parámetros.

En general:

Lo que nosotros como equipo pudimos aprender de esta actividad fue la importancia de conocer diferentes tipos de algoritmos dado que así pudimos llegar a la conclusión de cuál era mejor usar en base a su optimización. También reforzamos la idea de que tener un algoritmo que sea "el mejor" siempre va a variar dependiendo del problema por lo que siempre es importante entender las diferentes maneras de resolverlo o al menos los diferentes tipos de algoritmos para buscar diferentes soluciones a un problema.