Evidencia 1

Integrantes

Carla Oñate Gardella A01653555

Octavio Augusto Aleman Esparza A01660702

Profesores

Jose Daniel Azofeifa Ugalde

Introducción

En esta evidencia se hizo uso de varios algoritmos como: Z algorithm, “manacher algorithm” and “longest substring”. En este documento se explicará un poco sobre cada uno de estos algoritmos, así como sus complejidades. En caso de que el archivo cpp no se pueda abrir se hizo un replit donde se puede correr el archivo y probar esta evidencia: https://replit.com/@AugustoAleman/Ejercicio-3#main.cpp

Main

La función main es la principal en nuestro programa, en donde se llaman las funciones correspondientes para hacer la parte 1, 2 y 3 de la evidencia. Dentro de las funciones que se llaman dentro de main se hacen las validaciones para revisar primeramente que los archivos mcode1.txt, mcode2.tx, mcode3.txt, transmision1.txt y transmision2.txt existan y posteriormente que dichos archivos contengan solo caracteres de A - F y de 0 - 9 con saltos de línea.

Primero se hace una revisión con la función en la clase “util” para comprobar que los archivos se pueden abrir correctamente. En caso de que los archivos no estén directamente en la carpeta del .cpp entonces mandará un error.

Después se llaman funciones para leer los archivos de transmisión. La función se llama **readFileToVectorAndString** en donde se van leyendo un archivo en especifico y se va guardando en un string separado por un $ en cada salto de línea y en un vector. Al final se regresa una tupla con las dos estructuras para poder ser utilizadas en las siguientes partes de la evidencia. En esta misma función al ir leyendo las líneas se hace la validación usando otra función de la clase “util” en donde usando expresiones regulares se revisa si los caracteres son los permitidos.

Parte 1

SearchPatternInFile

Esta función se llama desde main con el archivo de transmisión 1 y 2. Aquí se lee con una función similar a la explicada anteriormente los archivos de mcode para poder buscar ese patrón en las transmisiones. Lo único que se hace en esta función

es llamar al Z algorithm para poder buscar la lista de patrones en ese archivo de transmisión.

Z algorithm

Este algoritmo sirve para poder encontrar patrones en una cadena de texto. Esto lo hace creando una ventana dentro de la cadena de texto donde se busca el patrón. Originalmente al implementar este algoritmo se concatena el patrón a buscar en el texto para poder ahorrar espacio y evitar usar alguna estructura externa a la cadena de texto. Sin embargo en esta aplicación del Z algorithm se hace uso de un string separado con el patrón a buscar, esto porque en una misma iteración del texto se están buscando varios patrones. En nuestro caso, hicimos algunas modificaciones al algoritmo para evitar la búsqueda en el Zarr y al igual poder encontrar muchos patrones en una misma cadena de texto. En nuestro caso tenemos una estructura auxiliar la cual es una matriz de tuplas en donde se va guardando qué patrones se encontraron dentro del texto mientras lo va revisando. Se tiene un while donde se va ciclando el string que es nuestro archivo de transmisión y dentro del mismo se tiene un for el cual es para ir ciclando los patrones a comparar desde el punto izquierdo de nuestra ventana. Con cada patrón se revisa si se encuentran coincidencias más adelante aumentando nuestra ventana derecha, así como el contador para saber cuántas coincidencias se tienen. En caso de que no hay más coincidencias con el patrón o que ya se extendió el tamaño del patrón, se termina el ciclo y se revisa si hay algún resultado que se tenga que guardar en la matriz de tuplas. Las tuplas que se guardan en nuestra matriz de resultados contiene el patrón el cual fue encontrado, la línea donde fue encontrado y el índice inicial. La línea se calcula con cada movimiento de la ventana izquierda revisando si el carácter es un $ el cual indica un salto de línea en nuestro archivo.

Es importante mencionar esto ya que la complejidad del Z algorithm es de O(n+m) esto porque al concatenar el patrón (m) a la cadena de texto se recorre todo el texto más los caracteres del patrón. La complejidad del algoritmo cambia ya que se hace búsqueda de múltiples patrones dentro de la misma función. Esto nos da una complejidad de O(n*p), **n** representa la cadena de texto y **p** la lista de patrones a recorrer. En caso de que la lista de patrones sea 1 elemento entonces la

complejidad es de O(n) ya que no concatenamos la cadena de texto y usamos una estructura auxiliar.
Al final del algoritmo se llama otra función en donde se imprime la lista de resultados que se encontraron en ese archivo de transmisión así como los no encontrados.

Parte 2

largestPalindrome

Esta función se llama de main con la lista de líneas que se tomaron del archivo tranmision 1 o 2 respectivamente. Se va ciclando cada línea de la lista y con cada una se llama la función de “manacher”, la cual regresa una tupla con el palíndromo línea e índice donde se encontró. Dentro de esta función se revisa si ese palíndromo es mejor y se va actualizando un valor. Al terminar todo el archivo entonces se imprime el palíndromo más grande que haya sido guardado en la variable de bestPalindrome.

Manacher algorithm

En la segunda parte de la evidencia se hace uso del algoritmo de manacher para encontrar el palíndromo más largo dentro de los archivos de transmisión. Este algoritmo usa en centro para poder comparar desde ese centro los caracteres derechos e izquierdos y analizar si es un palíndromo. Ya que para que funcione se necesita un centro, se hizo una función auxiliar que hace el string del texto encontrado a impar con # entre las letras, esto para asegurarnos que siempre exista un centro. Cada coincidencia que indique que es un palíndromo se va guardando en un array auxiliar “p”. Este array auxiliar nos sirve para encontrar el palíndromo más grande pero para optimizar el proceso y usar a nuestro favor algunas características de los palíndromos. En este algoritmo no se hace ninguna modificación especial a manacher ya que este algoritmo ya hace lo que queremos conseguir. Lo único extra es que al momento de ir comparando palíndromos se va guardando en variables auxiliares el centro del palíndromo más grande encontrado hasta el momento y la

longitud del mismo. Esto para después poder calcular el inicio del palíndromo y mostrarlo en consola. Este algoritmo tiene una complejidad de O(n). Es importante mencionar que para que este algoritmo funcione con todos los palíndromos, tanto pares como impares, se agrega a la cadena de texto un # entre cada carácter y se convierte en impar, esto para que siempre se tenga un centro. Ya que no se hicieron cambios a este algoritmo la complejidad sigue siendo de O(n). Este algoritmo es mejor que la solución de fuerza bruta ya que usa el array auxiliar de P para poder optimizar el proceso y recorrer menos del array en ciertas ocasiones. Es por lo mismo que decidimos usar el algoritmo de Manacher y no hacer una solución aparte que pueda tener una complejidad mayor a O(n). Al final de la función se llama a **getPalindrome** la cual recibe los índices del palíndromo y el texto para regresar una tupla con esos valores que se usa como se explicó anteriormente para ir comparando los palíndromos encontrados y ver cual es el mejor de entre todas las líneas del archivo.

Parte 3

findCommon

La función findCommon recibe como parámetro dos strings a comparar, así como la posición de dichos strings dentro de sus archivos de origen, en este caso, transmision1.txt y transmision2.txt. Para ello, la función mide los tamaños de los strings recibidos, creando a partir de ello una matriz de tamaño [n +1] * [m + 1] e igualando todas las posiciones a 0. Adicionalmente, se crea una segunda matriz de tamaño [2][3], en la cual se registran los valores y posiciones en ambos archivos de la mayor coincidencia encontrada.

Haciendo uso de una adaptación del algoritmo de longest common substring, se cicla por los caracteres de dos líneas de cada documento y en caso de encontrar alguna coincidencia, se registra su valor en una matriz, así como en una variable. Si se siguen encontrando coincidencias en líneas posteriores en las que sea invocada la función, se compara el valor de la coincidencia actual con la mayor registrada anteriormente. Si la coincidencia actual fuera mayor al valor de referencia, entonces se guarda su tamaño y posiciones dentro de la segunda matriz.

findLongestCS

La funcion findLongestCS abre los archivos transmision1.txt y transmision2.txt, después, dentro de dos ciclos for anidados se toma una línea del primer archivo y se compara dentro del siguiente for con todas las líneas del segundo archivo al llamar a la función findCommon. Cuando se terminen de comparar todas las líneas del primer archivo con todas las líneas del segundo archivo, se retoman los valores de la matriz de mayor coincidence y se imprimen los resultados.

Longest common substring

Para el funcionamiento de las funciones anteriores se hizo uso de una adaptación del algoritmo de Longest Common Substring. A partir de los tamaños de cada string recibido, se crea una matriz de tamaño [n + 1] * [m + 1] y se igualan todas las pocisiones a 0. Después, en un ciclo de dos fors anidados se comparan uno a uno los caracteres del primer string con todos los caracteres del segundo, sumando en la casilla correspondiente 1 si se encuentra una coincidencia, además de los valores acumulados en la casilla en diagonal anterior.

Los valores de coincidencia se van registrando en una variable temporal, si el valor de coincidencia actual es mayor al valor mayor registrado, entonces se actualiza el mayor valor y se registran en una lista las posiciones y tamaño de la coincidencia.

Cabe mencionar que la complejidad de este algoritmo en el peor caso es igual a O(n^2), pues se hace uso de dos ciclos for anidados.