UA

Tokenizador

DanielAsensiRochDNI : **48776120C**

March 9, 2022

Contents

1 Introducción:

En la asignatura de Explotación de la información estudiamos diversos problemas relacionados a como se organiza la información que llega de manera masiva a los computadores y los diferentes algoritmos para procesarla, organizarla y presentarla, de manera que esta sea lo más accesible posible para los algoritmos de búsqueda y por consecuencia para el usuario final.

1.1 Presentación del problema

En esta práctica se nos presenta la problemática de realizar un tokenizador capaz de pasada una cadena de texto y una serie de delimitadores obtener los tokens pertinentes de la cadena, pero a su vez este tokenizador deberá ser capaz de distinguir los siguientes casos especiales:

1. Detección de URLs
2. Detección de números reales más adición de información
3. Detección de acrónimos
4. Detección de Emails
5. Detección de multiplabras (guiones)

El orden de reconocimiento de estos casos especiales será muy importante para el correcto funcionamiento del algoritmo, siendo el mismo una problemática a resolver.

1.2 Hipótesis de partida

La hipótesis de partida de nuestro algoritmo vendrá dada por el orden de los pasos que habrán que realizar para la obtención correcta de todos los tokens, para ello deberemos analizar en que casos se puede encontrar nuestro algoritmo.

En primer lugar deberemos saber es que nuestro tokenizador posee un parámetro denominado *delimitadores* este será el que almacene los caracteres de parada para la formación de los tokens. Por otro lado tenemos un parámetro que nos indicará si debemos quitar los acentos y pasar a minúsculas nuestros tokens antes de formarlos.

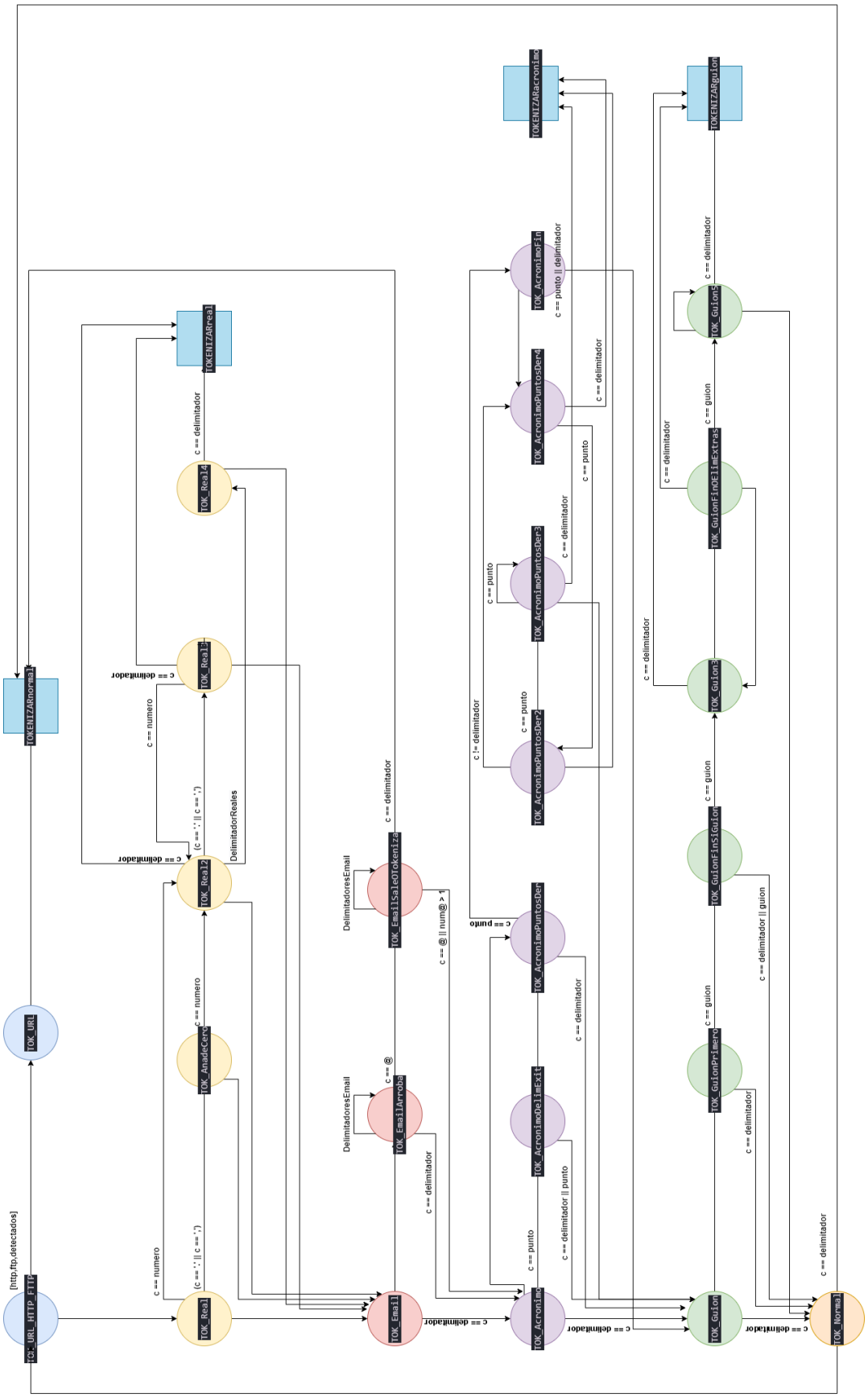
En cuanto a los casos especiales anteriormente mencionados cabe recalcar que la detección de los mismos puede estar activada o no, al igual que con el paso a minúsculas y la eliminación de acentos, si esta tokenización mediante casos especiales estuviera activada deberemos incluir en nuestros *delimitadores* los caracteres del espacio y el salto de linea como parámetros por defecto.

Una vez explicados los parámetros importantes, pasaremos a recalcar los comportamientos básicos que debe de seguir nuestro programa siendo los siguientes:

- Pasada una cadena de caracteres por parámetro el tokenizador deberá devolver una lista con los tokens pertinentes a la misma
- Pasado un fichero de entrada y un fichero de salida como parámetro nuestro tokenizador este deberá escribir en el fichero de salida todos los tokens (uno por linea) el resultado de la tokenización de todo el archivo de entrada
- Pasado un archivo que poseea las direcciones de un directorio con sus respectivos ficheros, nuestro tokenizador deberá tokenizar el contenido de todos los archivos del directorio y escribir el resultado en unos nuevos archivos generados con el nombre de los anteriores más la extensión .tk

2 Análisis de la solución

La solución implementada en el tokenizador con los casos especiales activados ha sido la de un árbol de decisión, el cual analizará el string pasado como parámetro letra a letra saltando entre los diferentes estados definidos en los cuales se tomará la decisión de cual es el siguiente estado y en que momento se termina la formación de un token, además tendremos diferentes estados finales los cuales formarán el token final mediante las variables pos y npos las cuales almacenarán la posición inicial y en la que nos encontramos analizando. El árbol de decisión implementado ha sido el siguiente:


TOKENIZARAcronimo
TOKENIZARguion
TOKENIZARreal
TOKENIZARnormal
TOK_AcronimoFin
TOK_AcronimoPuntosDer4
TOK_AcronimoPuntosDer3
TOK_AcronimoPuntosDer2
TOK_AcronimoPuntosDer
TOK_AcronimoDelimExit
TOK_Acronimo
TOK_GuionS
TOK_GuionFinDelimExtras
TOK_GuionS
TOK_GuionFinSiGuion
TOK_GuionPrimero
TOK_Guion
TOK_Real4
TOK_Real1
TOK_Real2
TOK_AnadeCero
TOK_Real
TOK_EmailSaleOTokeniza
TOK_EmailArroba
TOK_Email
TOK_URL
TOK_URL_HTTP_FTP
TOK_Normal
[http,ftp,detectados]
c == delimitador
c == punto || delimitador
c == punto
c != delimitador
c == guion
c == delimitador || guion
c == numero
(c == '.' || c == ',')
c == delimitador
DelimitadorReales
DelimitadoresEmail
c == @
c == @ || num@ > 1
c == punto
c == delimitador || punto

3 Justificación de la solución

El motivo de la implementación de esta solución ha sido por la sencillez, velocidad y cantidad de kilobytes utilizados por la misma.

En mi caso esta solución ha sido compleja, pero a su vez bastante maleable a la hora de implementarla, permitiendo la misma una gran escalabilidad a la hora de añadir casos intermedios o nuevos a la implementación del grafo en el código. Los estados del mismo son variables que se generan con el objeto siendo estas globales y constantes con un valor numérico asociado, además durante la implementación he procurado que estas variables tengan un nombre sencillo para que a la hora de debuggear podamos identificar perfectamente en que caso nos encontramos.

Además de las variables globales que representan nuestros estados, se han inicializado los delimitadores comunes en los emails, URLs y reales permitiéndonos estos identificar de manera más sencilla que caso nos encontramos analizando y cuales de los caracteres tendremos *"absorber"* dentro del token que genera un caso especial.

Durante la implementación de la solución se ha procurado que el manejo de la memoria fuera lo más eficiente posible, eliminando por completo el paso valor de variables, lo que evitaría así el consumo innecesario de memoria durante la ejecución del programa.

3.1 Soluciones descartadas

Antes de realizar el árbol de decisiones finalmente implementado se probaron dos alternativas.

3.1.1 Alternativa de Regex y 5 vueltas

La primera implementación que realizada y finalmente descartada fue el uso de *"flags"* para cada uno de los términos a analizar, de esta manera cada una de las palabras separadas por espacios en el texto a analizar sería un termino, estos términos serían analizados utilizando el sistema de Regex, esto haría que los términos se analizaran una vez en el mejor caso y 5 en el peor caso. Cuando uno de los términos diera positivo sería tokenizado de manera completa y analizando de nuevo los posibles delimitadores que contuviera.

Esta solución fue completamente descartada porque, tras haber implementado la mitad de los caso, el tiempo total de ejecución de la tokenización para el corpues alcanzaba los 10 segundos y más de 8000kb de espacio en memoria ocupados.

3.1.2 Autómatas de estado finito

Esa solución fue implementada de manera parcial durante el desarrollo de la práctica, el motivo final del descarte de la misma fue la complejidad de la implementación de esta y la dificultad en la escalabilidad y maleabilidad en la adición de más casos especiales, aunque esta solución era la más rápida en la tokenización del corpus mi decisión final fue descartarla ya que en mi caso no pude terminar de implementarla por su dificultad, además cada vez que finalizaba la creación del grafo y las matrices de adyacencia quedaban implementadas aparecían nuevos casos especiales que no podía manejar con la implementación anterior.

4 Análisis de las mejoras realizadas en la práctica

En este apartado explicaré los diferentes sistemas que he utilizado para que mi práctica fuera lo más eficiente posible y ocupara la menor cantidad de espacio posible en memoria.

4.0.1 Paso de valores por referencia

De esta manera el código no pasa ningún variable por valor para no ocupar espacio innecesario en memoria de manera que no se realizará la copia de ninguna de las variables.

4.0.2 Casteado de caracteres a enteros

Ante la problemática de que cada vez que abría el código en mi editor y siendo este casteo un pequeño incentivo en la aceleración de la formación de strings, decidí en mi método que transformaba los tokens formados a minúsculas y eliminaba los acentos transformar todos los valores que debían ser cambiados a enteros y luego sumarlos en el string final.

4.0.3 Recorrido carácter a carácter

Al haber realizado la implementación de un árbol de decisión y para mejorar la eficiencia, complejidad y velocidad de mi código decidí recorrer la cadena de caracteres a tokenizar una sola vez, este recorrido lo haríamos guardándonos el inicio del posible token y recorriendo sus componentes iteración a iteración en los diferentes analizadores implementados.

5 Análisis de complejidad (teórica y práctica) del tokenizador.

5.1 Complejidad Temporal

La complejidad implementada en el tokenizador de casos especiales, al haber utilizado un árbol de decisión para la lectura carácter a carácter. Obtendremos que **"D"** será la cantidad de delimitadores que tenemos, y **"N"** la cantidad de caracteres de la cadena a tokenizar, de esta manera el peor caso lo obtendremos cuando todos los caracteres de la cadena sean delimitadores siendo este **O(n*d)** y el mejor caso cuando la cadena de delimitadores se encuentre vacía.

5.2 Complejidad Espacial

Nuestro tokenizador de casos especiales almacenará el string recibido por parámetro en una lista de tokens de espacio lineal en función del tamaño de entrada siendo **O(N)**