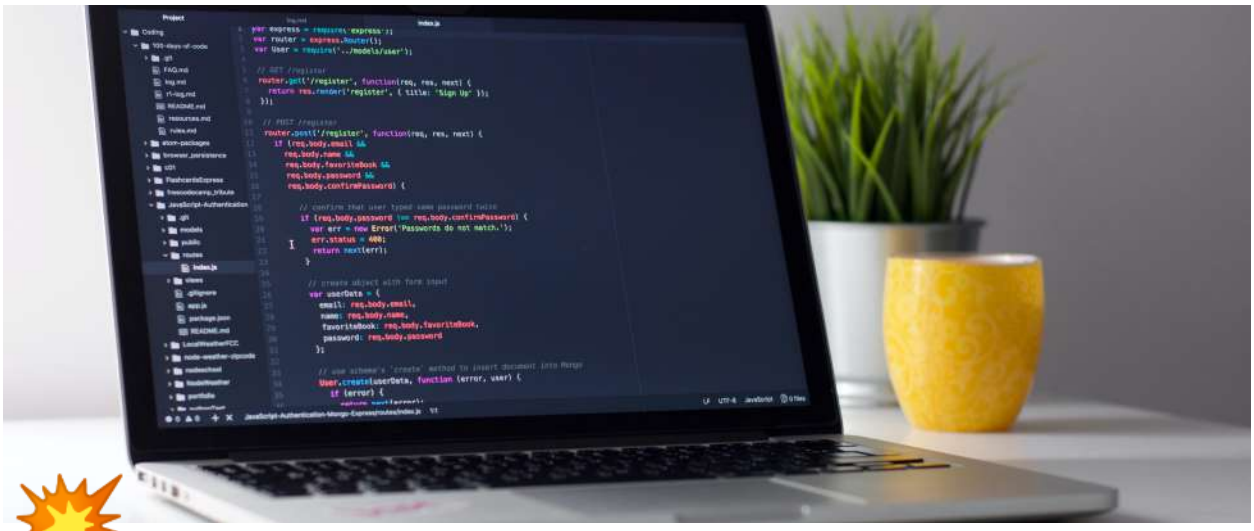


Manual Técnico Proyecto#01 - LFP

Compatibilidad

-Windows 10

-Linux

Lenguajes:

Este proyecto fue desarrollado en su totalidad por el lenguaje de programación python, donde se implementaron matrices, analizadores léxicos y otros conceptos propios del lenguaje.

Tabla de tokens

Tabla de Tokens

Aa TOKEN	EXPRESIÓN REGULAR
TITULO	TITUTLO
CELDAS	CELDAS
ALTO	ALTO
ANCHO	ANCHO
FILTROS	FILTROS
FILAS	FILAS
COLUMNAS	COLUMNAS
SEPARATOR	@@@@
Closed_key	}

Aa TOKEN	EXPRESIÓN REGULAR
Open_key	{
two_points	:
semicolon	;
comma	;
equals	=
quotation_mark	"
open_square_bracket	[
closed_square_bracket	]
int_value	[0-9]*
boolen_value	TRUE | FALSE

Librerías o importaciones utilizadas

```
import os
from Token_DAO import Token_DAO
from os import system
from tkinter import *
from tkinter.ttk import Combobox
from Lexical_Analyzer import Lexical_Analyzer
from tkinter.filedialog import askopenfilename
from Token import Token
from tkinter import messagebox
from Lexical_Analyzer import token_handler,error_handler
from Image_DAO  import Image_DAO
import re
```

Creación de los Frame

Creación de la Raíz

```
#Creación de la raíz aplicando algunos atributos
root=Tk()
root.title("Image Maker v1.0")
root.resizable(False,False)
root.iconbitmap("img/cobra_kai.ico")
root.config(cursor="hand2")
root.geometry("1200x600")
root.config(bg="Gold")
root.config(relief="groove")
root.config(bd="30")
```

Frame de inicio

```
#Creación del frame principal aplicando algunos atributos y elementos
main_frame=Frame(root,width="650",height="430")
main_frame.pack()
main_frame.config(bg="black")
main_frame.config(bd="20")
main_frame.config(relief="groove")
main_frame.config(cursor="hand2")
#Label del titulo del frame principal
Label(main_frame,text="IMAGE MAKER",fg="gold",
font=("Comic Sans MS",30),bg="black").place(x=170,y=20)
#Label que contendrá una imagen del logo principal y el botón de inicio
logo_image=PhotoImage(file="img/cobra.png")
Label(main_frame,image=logo_image,bg="black").place(x=170,y=80)
#Botón para iniciar la aplicación
principal_Button=Button()
principal_Button.config(text="Iniciar",
width="25",height="1",bg="black",fg="gold",font=("Comic Sans MS",20))
principal_Button.pack()
```

Frame de Funciones

```
#Creación del frame con todas las funciones
master_frame=Frame(root,width="650",height="100")
master_frame.config(bg="red")
master_frame.config(bd="20")
master_frame.config(relief="groove")
```

```
master_frame.config(cursor="hand2")
upload_Button=Button(master_frame,text="Cargar Archivo",
font=("Comic Sans MS",10),bg="black",fg="yellow")
upload_Button.grid(row=0,column=0,padx=10)
analyze_Button=Button(master_frame,text="Analizar Archivo",
font=("Comic Sans MS",10),bg="black",fg="yellow")
analyze_Button.grid(row=0,column=1,padx=10)
reports_Button=Button(master_frame,text="Reportes",
font=("Comic Sans MS",10),bg="black",fg="yellow")
reports_Button.grid(row=0,column=2,padx=10)
images_Button=Button(master_frame,text="Generar Imágenes",
font=("Comic Sans MS",10),bg="black",fg="yellow")
images_Button.grid(row=0,column=3,padx=10)
exit_Button=Button(master_frame,text=" Salir",
font=("Comic Sans MS",10),bg="black",fg="yellow")
exit_Button.grid(row=0,column=4,padx=10)
```

Frame de los Filtros

```
#Creación del frame donde se muestran todos los filtros
filter_frame=Frame(root,width="650",height="430")
filter_frame.config(bg="gold")
original_Button=Button(filter_frame,text="Original",
font=("Comic Sans MS",20),bg="black",fg="yellow")
original_Button.grid(row=0,column=0,pady=10)
original_Button.config(width=20)
mirrorx_Button=Button(filter_frame,text="Mirror X",
font=("Comic Sans MS",20),bg="black",fg="yellow")
mirrorx_Button.grid(row=1,column=0,pady=10)
mirrorx_Button.config(width=20)
mirrory_Button=Button(filter_frame,text="Mirror Y",
font=("Comic Sans MS",20),bg="black",fg="yellow")
mirrory_Button.grid(row=2,column=0,pady=10)
mirrory_Button.config(width=20)
double_mirror_Button=Button(filter_frame,text="Double Mirror",
font=("Comic Sans MS",20),bg="black",fg="yellow")
double_mirror_Button.grid(row=3,column=0,pady=10)
double_mirror_Button.config(width=20)
#Variable que almacenará los valores del combobox
options_list=[]
#Creación del combobox para elegir la imagen
images_combobox=Combobox(filter_frame,width="50",state="readonly")
#Botón para seleccionar la imagen del combobox
image_selected_button=Button(filter_frame,text="Buscar Imagen",
font=("Comic Sans MS",20),bg="black",fg="yellow")
```

Frame donde se muestra la imagen

```
#Creación del botón donde se muestra la imagen
image_frame=Frame(root,width="650",height="470")
image_frame.grid_propagate(FALSE)
image_frame.config(relief="groove")
image_frame.config(cursor="hand2")
img=PhotoImage(file="")
image_label=Label(image_frame,bg="red")
image_label.grid(row=2,column=2)
size_label=Label(bg="gold",text="Tamaño de la imagen :",font=("Comic Sans MS",30))
```

Funciones utilizadas

master_mind_window()

Con está función pasamos del frame de inicio al frame donde están todas las funciones.

```
def master_mind_window():
    main_frame.destroy()
    principal_Button.destroy()
    root.geometry("1400x600")
    master_frame.grid(row=0,column=0)
    filter_frame.grid(row=1,column=0)
    division_frame.grid(row=1,column=1)
    image_frame.grid(row=1,column=2)
    size_label.grid(row=0,column=2)
#Con este botón inicializamos esta función
principal_Button.config(command=master_mind_window)
```

exit_application():

Función que termina con la ejecución de la aplicación.

```
def exit_application():
    messagebox.showinfo(title="Image Maker V1.0",message="Vuelve pronto Crack!!")
    exit()
exit_Button.config(command=exit_application)
```

upload_file():

Función para cargar el archivo con extensión "pxla" a memoria.

```
def upload_file():
    global route,original_text,images_counter,options_list,images_combobox,
image_selected_button,temp_text
    route = askopenfilename()
    if route.endswith("pxla"):
        archive = open(route,"r")
        original_text=archive.read()
        archive.close()
        temp_text=original_text
        messagebox.showinfo(title="Image Maker V1.0",
        message="El archivo fue cargado con éxito al sistema!!")
    else:
        messagebox.showerror(title="Image Maker V1.0",
        message="No es un archivo con extensión 'pxla', intenta de nuevo!!")
        upload_file()
upload_Button.config(command=upload_file)
```

images_generator():

Función para generar los objetos de tipo imagen

```
def images_generator():
    global temp_text,images_counter,options_list,images_data,
    images_combobox,image_selected_button
    counter=0
    temp_text.replace("\t","")
    images_counter=(temp_text.count("@@@@"))+1
    for x in range(images_counter):
        if temp_text.count("@@@@")>=1:
            temp=temp_text[counter:temp_text.find("@@@@")+4]
            images_data.append(temp)
            temp_text=temp_text[temp_text.find("@@@@")+4:len(temp_text)]
        else:
            temp=temp_text
            images_data.append(temp)
    image_attributes(images_data)
    image_dao_handler.images_name_collector()
    options_list+=image_dao_handler.name_images_list
    images_combobox.grid(row=4,column=0)
    image_selected_button.grid(row=5,column=0,pady=10)
    images_combobox.config(values=options_list)

    messagebox.showinfo(title="Image Maker V1.0",
    message="Archivos Html de las imágenes Generados!!")
images_Button.config(command=images_generator)
```

analyze_file():

Función para analizar el archivo.

```
def analyze_file():
    global original_text
    try:
        lexical_analyzer_handler.analyze_file(original_text)
        print("\n\n\n\n\n\n\n")
        lexical_analyzer_handler.print_errors()
        print("\n\n\n\n\n\n\n")
        lexical_analyzer_handler.print_tokens()
        messagebox.showinfo(title="Image Maker V1.0",
        message="El archivo se analizó con éxito!!")

    except:
        messagebox.showerror(title="Image Maker V1.0",
        message="El archivo No pudo analizarse, intenta de nuevo!!")
        analyze_file()

analyze_Button.config(command=analyze_file)
```

search_information():

Función para recolectar información de la imagen solicitada.

```
def search_information():
    global images_combobox,image_selected_button,current_image,size_label
    temp_dimensions=""
    if images_combobox.get()!="":
        current_image=images_combobox.get()
        temp_dimensions=image_dao_handler.dimensions_by_name(current_image)
        image_dao_handler.validate_image(current_image)
        messagebox.showinfo(title="Image Maker V1.0",
        message="La información de la imagen: "+images_combobox.get()
        +"\nHa sido procesada!!\nPuedes continuar seleccionando los filtros!!")
        size_label.config(text=temp_dimensions)

    else:
        messagebox.showerror(title="Image Maker V1.0",
        message="No has seleccionado ninguna imagen!!")
image_selected_button.config(command=search_information)
```

reports_view():

Función para abrir automaticamente los reportes de tokens y errores.

```
def reports_view():
    token_handler.tokens_html_report()
    error_handler.errors_html_report()
    messagebox.showinfo(title="Image Maker V1.0",
    message="Se Abrirán Los siguientes Reportes:\n-Reporte De Tokens\n"
    "-Reporte De errores")
    os.system("C:/Users/Erwin14k/Documents/Image_Maker_Automaton/Reportes/TOKENS.html")
    os.system("C:/Users/Erwin14k/Documents/Image_Maker_Automaton/Reportes/ERRORS.html")
reports_Button.config(command=reports_view)
```

original_view():

Función para ver la imagen en su forma orignal.

```
def original_view():
    global current_image,original_Button,img,image_label,image_frame
    print(current_image)
    img.config(file="jpg/"+current_image+"ORIGINAL.png")
    image_label.config(image=img)
    image_frame.grid_propagate(TRUE)
original_Button.config(command=original_view)
```

mirrory_view():

Función para visualizar la imagen con el filtro "MIRRORY"

```
def mirrory_view():
    global current_image,mirrory_Button,img,image_label,image_frame
    temp_filt=""
    temp_filt=image_dao_handler.filters_by_name(current_image)
    if temp_filt.count("MIRRORY") >=1:
        print(current_image)
        img.config(file="jpg/"+current_image+"MIRRORY.png")
        image_label.config(image=img)
        image_frame.grid_propagate(TRUE)
    else:
        messagebox.showerror(title="Image Maker V1.0",
        message="Filtro no disponible para esta imagen!!")
mirrory_Button.config(command=mirrory_view)
```

mirrorx_view():

Función para visualizar la imagen con el filtro "MIRRORX"

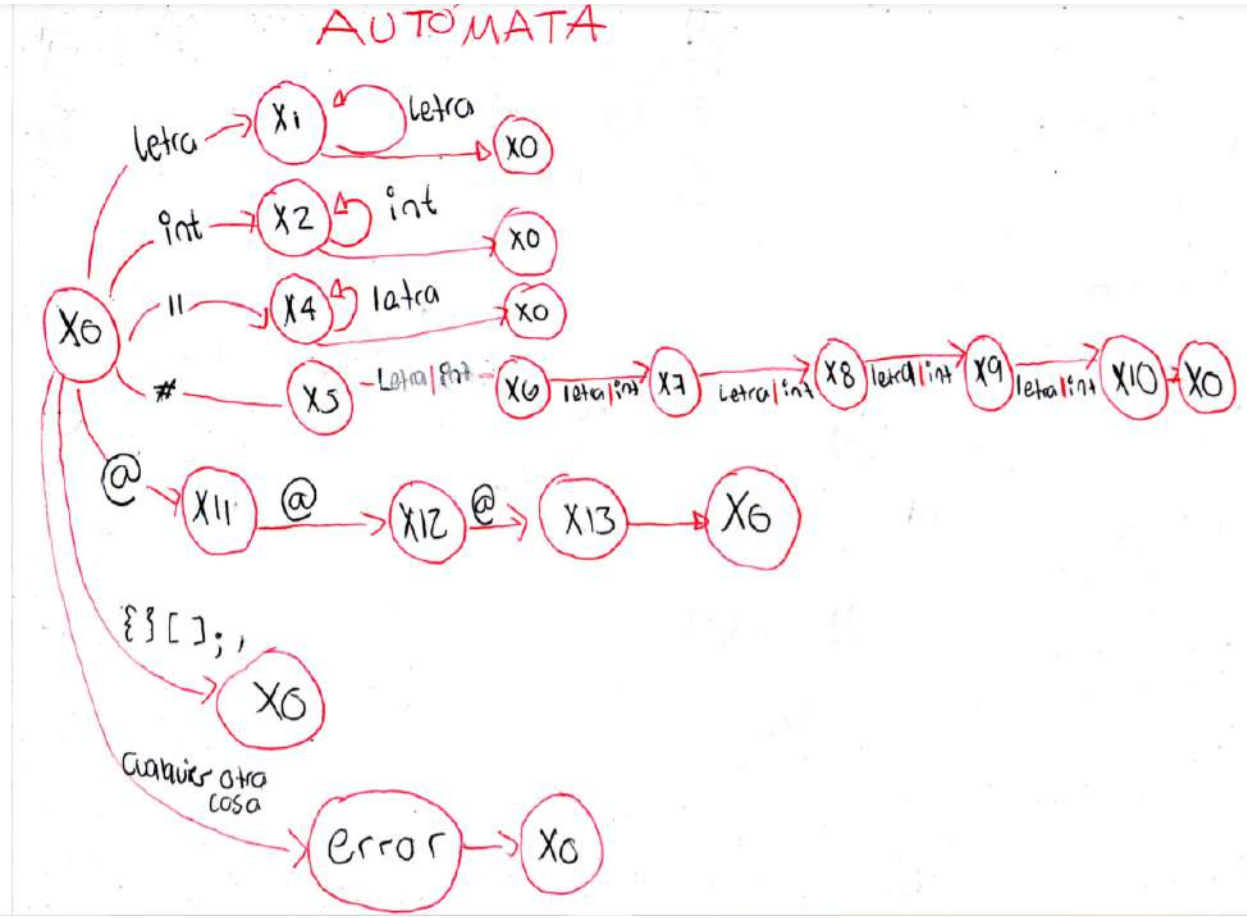
```
def mirrorx_view():
    global current_image,mirrorx_Button,img,image_label,image_frame
    temp_filt=""
    temp_filt=image_dao_handler.filters_by_name(current_image)
    if temp_filt.count("MIRRORX") >=1:
        print(current_image)
        img.config(file="jpg/"+current_image+"MIRRORX.png")
        image_label.config(image=img)
        image_frame.grid_propagate(TRUE)
    else:
        messagebox.showerror(title="Image Maker V1.0",
        message="Filtro no disponible para esta imagen!!")
mirrorx_Button.config(command=mirrorx_view)
```

double_view():

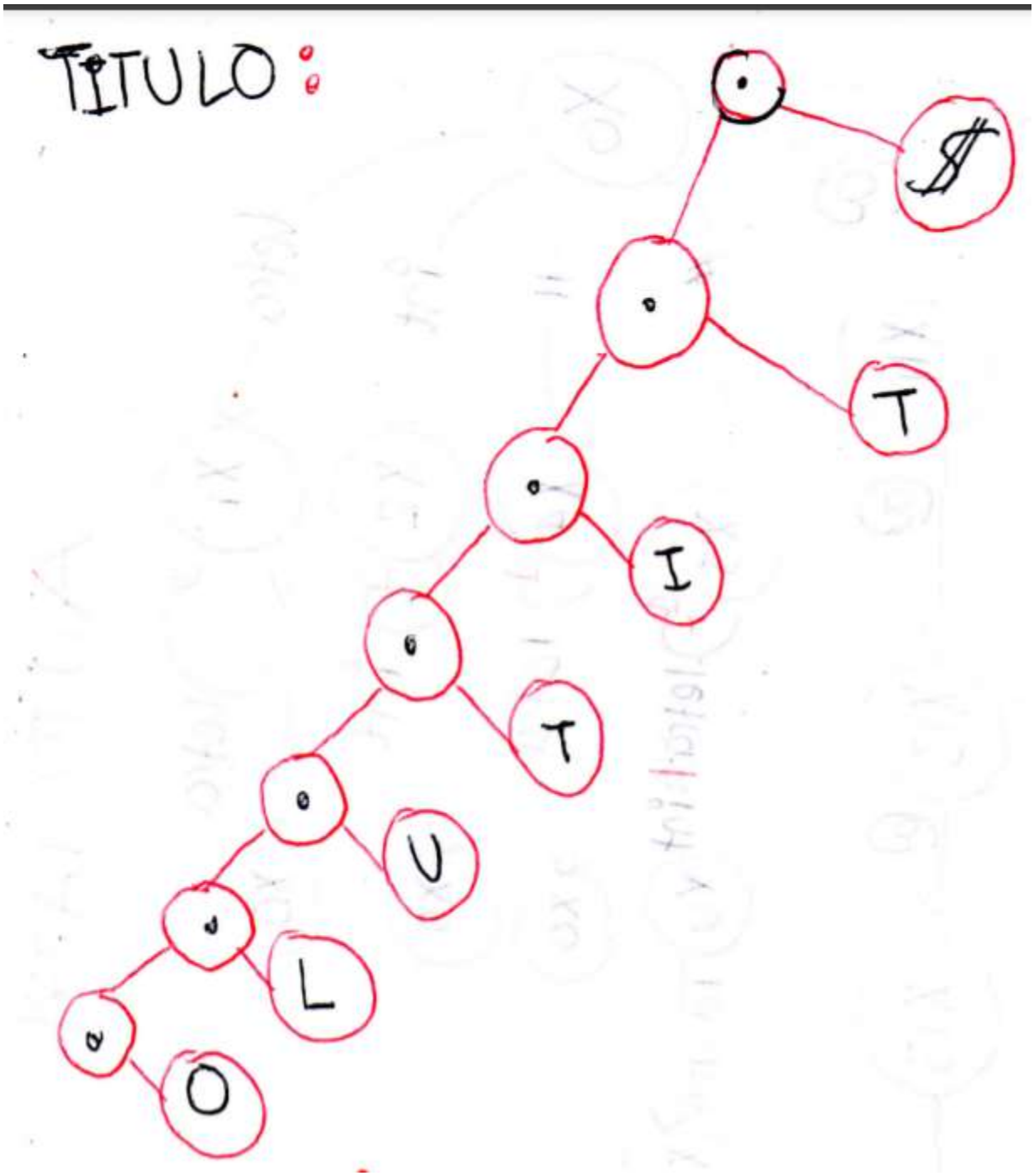
Función para visualizar la imagen con el filtro "DOUBLEMIRROR"

```
def double_view():
    global current_image,double_mirror_Button,img,image_label,image_frame
    temp_filt=""
    temp_filt=image_dao_handler.filters_by_name(current_image)
    if temp_filt.count("DOUBLEMIRROR") >=1:
        print(current_image)
        img.config(file="jpg/"+current_image+"DOUBLE.png")
        image_label.config(image=img)
        image_frame.grid_propagate(TRUE)
    else:
        messagebox.showerror(title="Image Maker V1.0",
        message="Filtro no disponible para esta imagen!!")
double_mirror_Button.config(command=double_view)
```

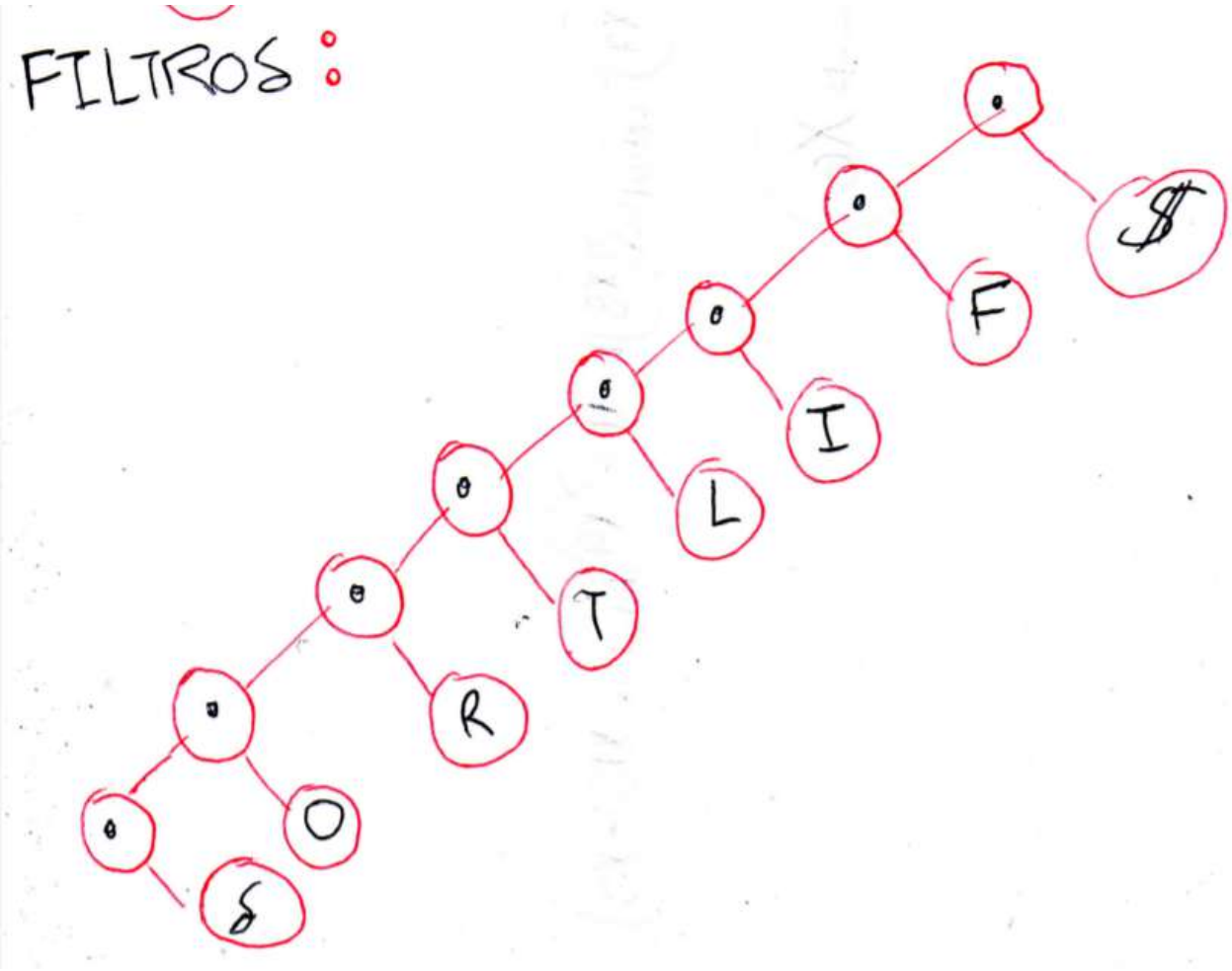
Autómata Implementado



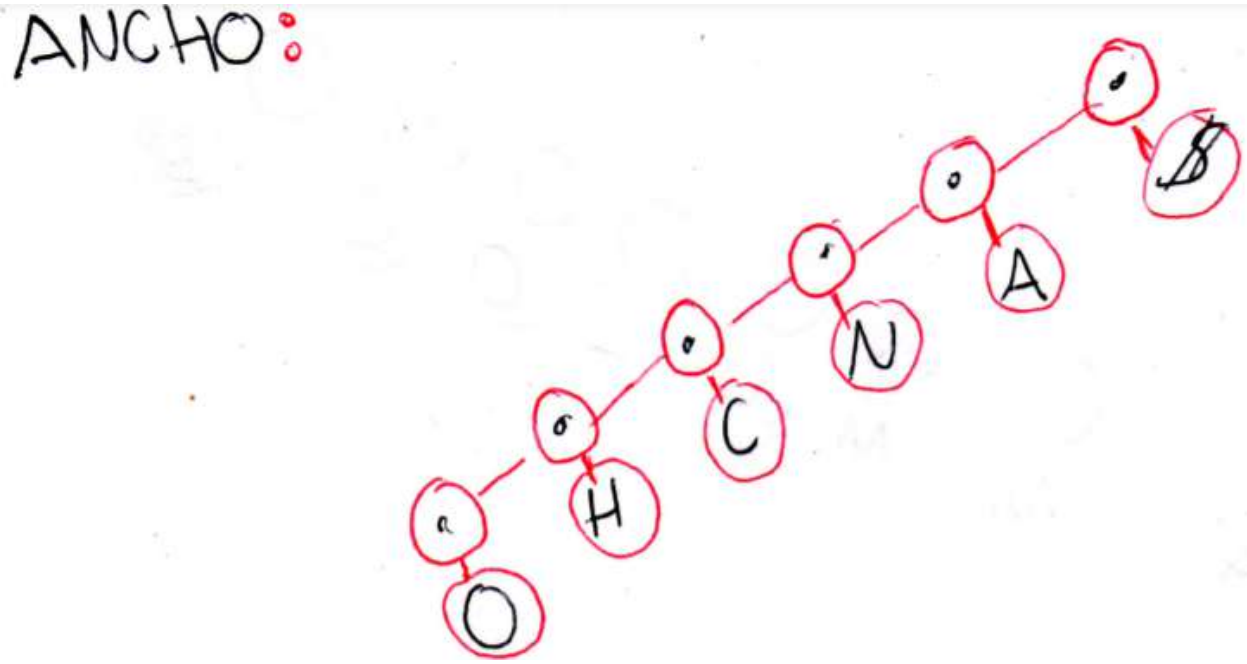
Métodos de Árbol para expresiones regulares

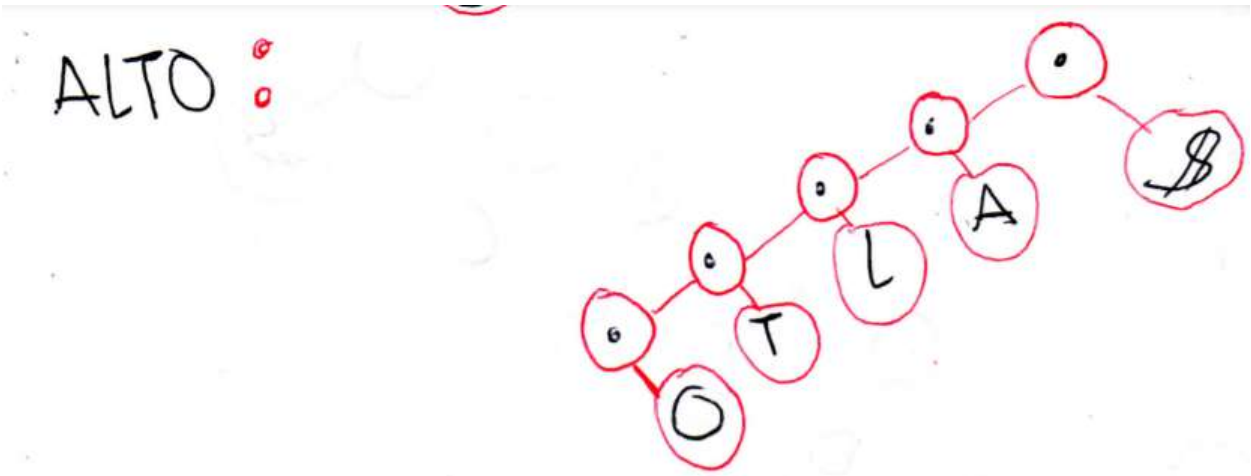

TITULO:
.
$
.
T
.
I
.
T
.
U
.
L
.
O

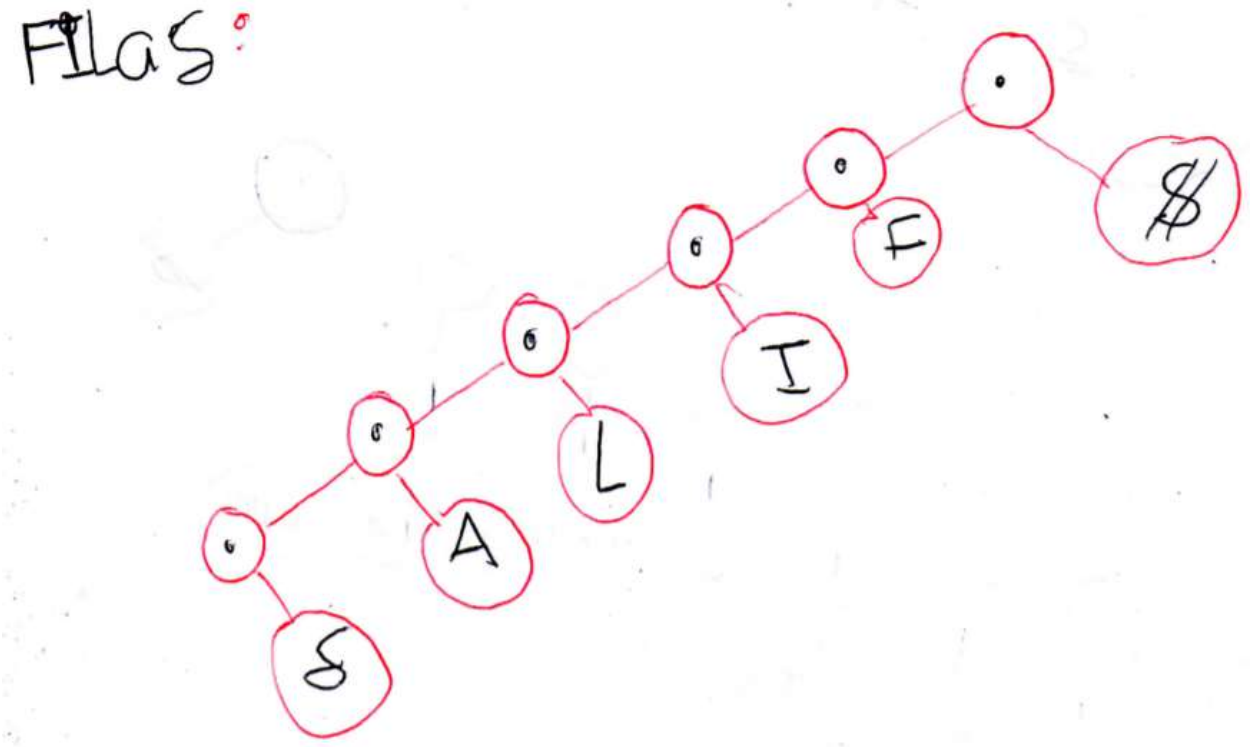
FILTROS:

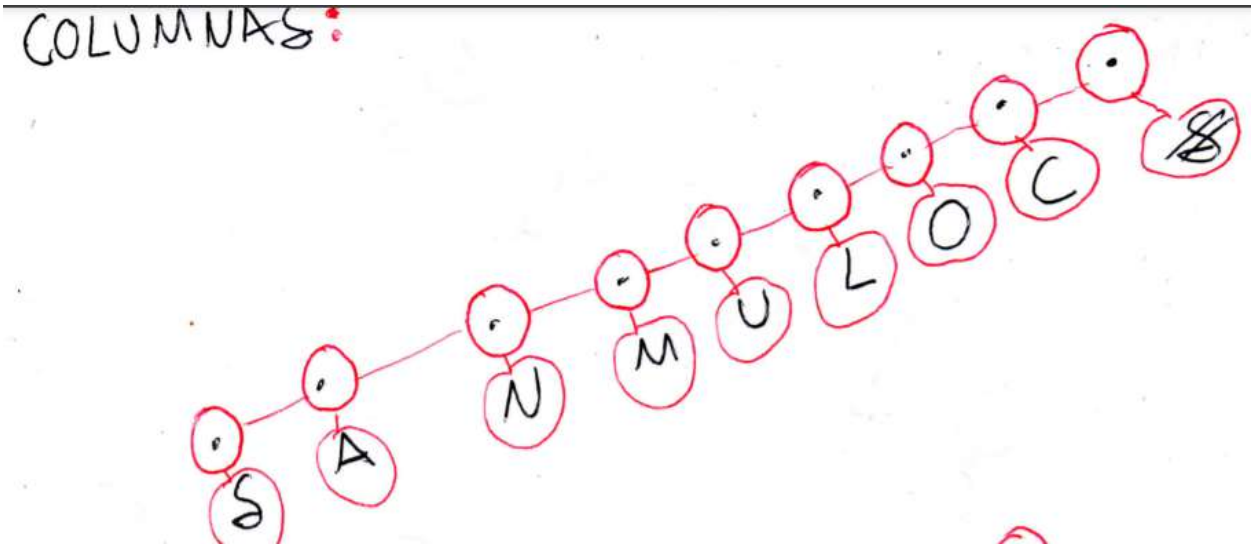


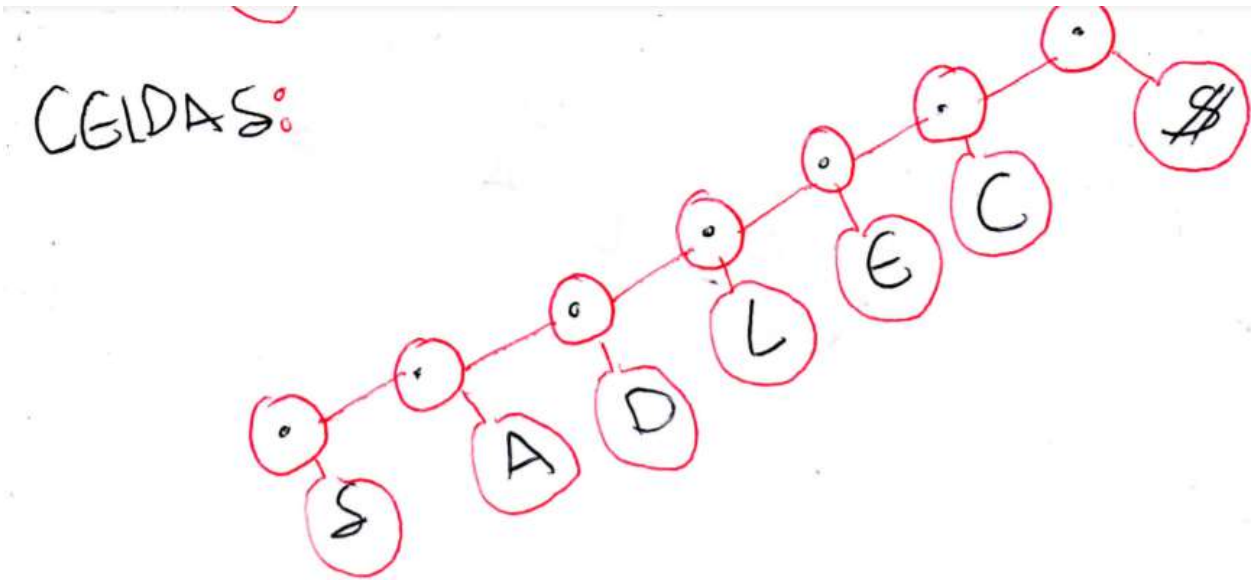
ANCHO:

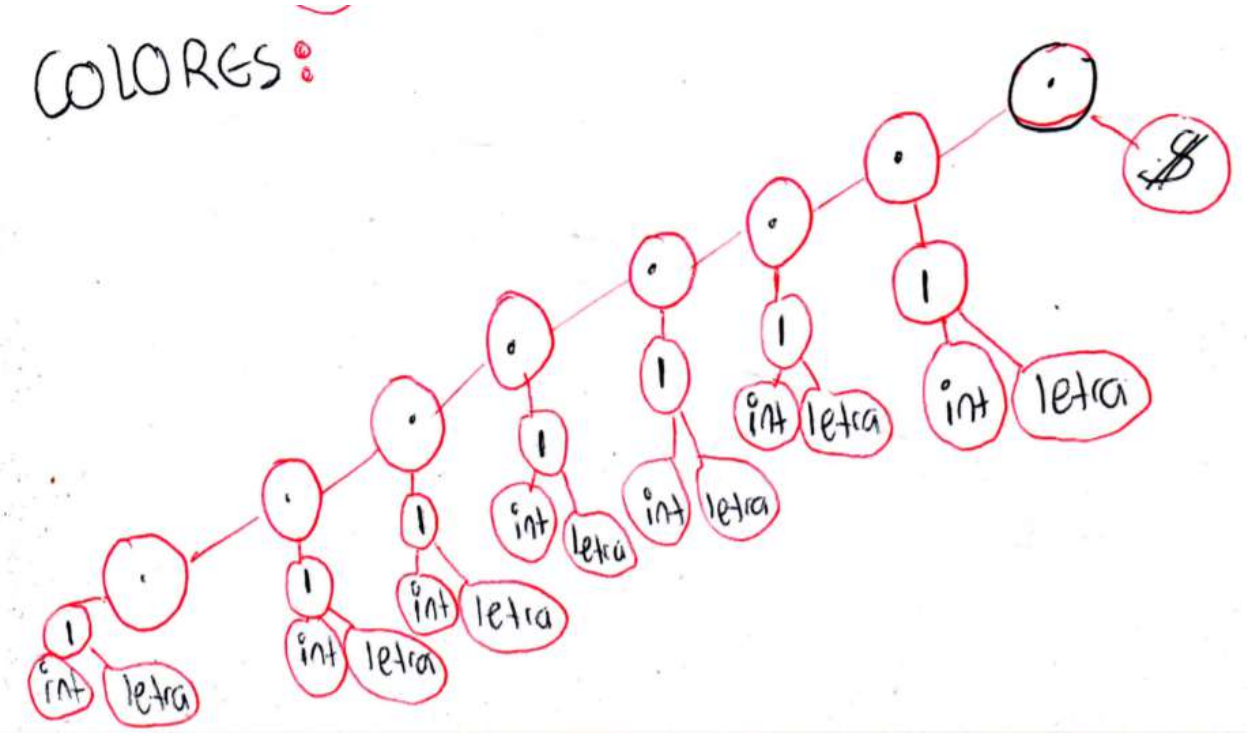



ALTO:
A
L
T
O

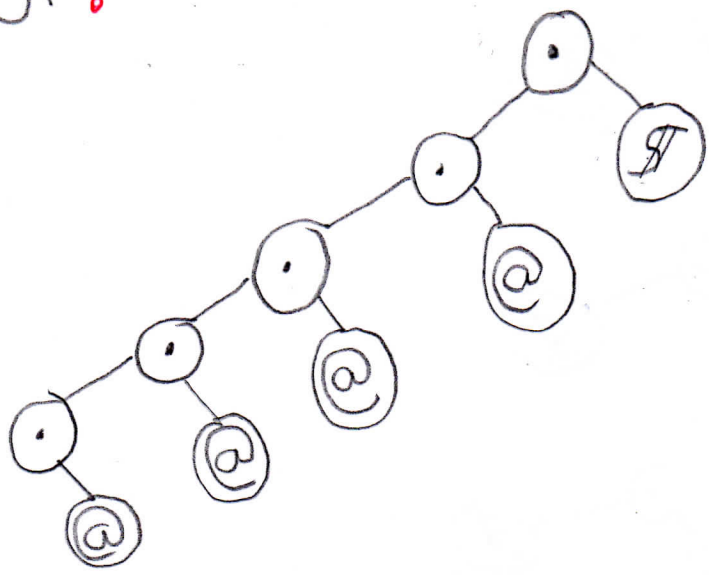

FILAS:
F
I
L
A
S


COLUMNAS:
S
A
N
M
U
L
O
C
$

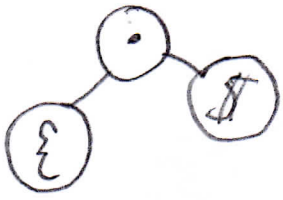

CELDAS:
S
A
D
L
E
C
$


COLORES:
$
int
letra

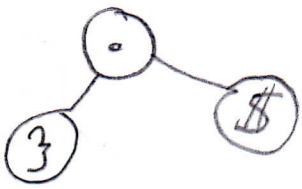
Separator:



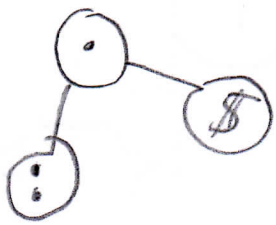
Open_key



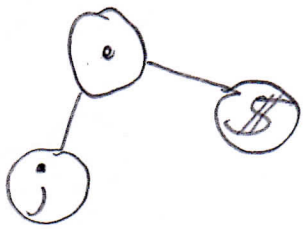
Closed_key



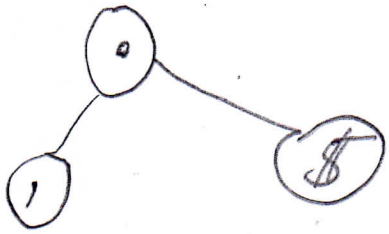
two_points



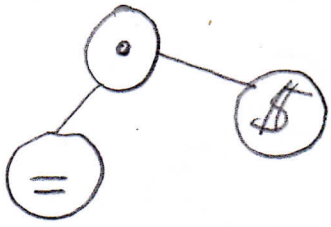
semicolon



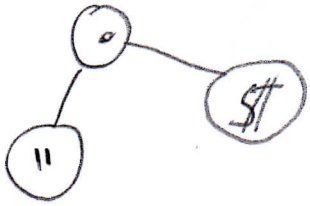
comma



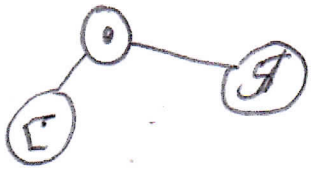
equals



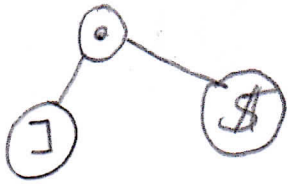
quotation_mark



open_square_bracket



closed_square_bracket



boolean_value