Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Lenguajes Formales y de Programacion
Sección A-
Inga. Vivian Damaris Campos Gonzales
Aux. Mario Josué Solis Solórzano
Primer semestre 2023

MANUAL TECNICO

APLICACIÓN DE LECTURA DE ARCHIVO DE ENTRADA MEDIANTE UN AFD

Nombre:	Eduardo Misael Lopez Avila	Registro académico:	202100147
Fecha:	27/03/2022	Sección:	A-

1. Descripción

Se requiere el desarrollo de una aplicación que permita la lectura de un archivo de texto que contiene diferentes listas de valores con una estructura definida, separados por una coma. Estos valores contienen información de operaciones matemáticas básicas. Lo que se solicita realizar con la información contenida en cada línea de texto, es leer cada carácter mediante un AFD y realice operaciones matemáticas según se índique en el archivo de carga..

2. Aplicación de lectura de archivo de carga.

2.1. Librerías en uso.

Se utilizan las siguientes librerías:

Os: Esta librería sirve para verificar dentro de la aplicación mediante comandos del sistema si el archivo a cargar existe en la dirección correcta.

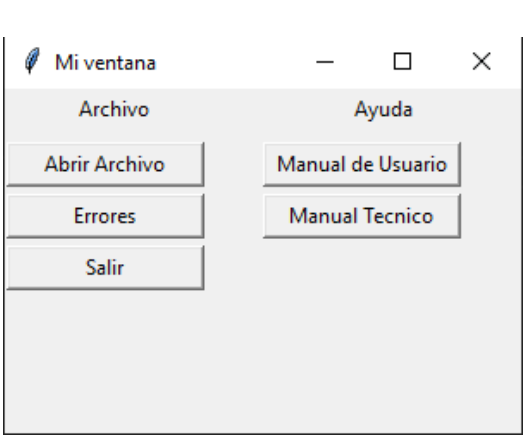
Graphviz: Esta librería nos sirve para generar un grafo con una estructura definida a partir de la información cargada al sistema durante la ejecución.

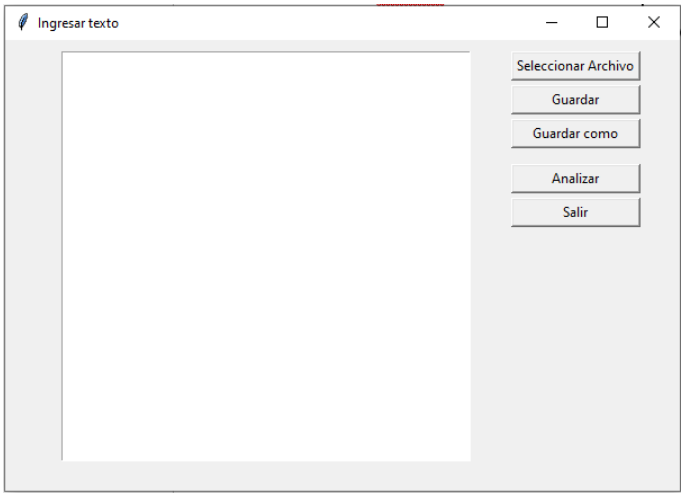
Uuid: Esta librería nos permite definir una variable universal, la cual nos sirve para apoyarnos en generar grafos mediante graphviz.

Tkinter: Esta librería nos permite realizar interfaces graficas modificables según se indique, el cual nos sirve para imprimir en una ventana la información a utilizar durante el proyecto.

2.2. Vista general de la aplicación

Se inicializa la aplicación.





2.3. Cargar archivo de entrada

Se debe ingresar una carga de datos de un archivo, el cual contiene los datos con una determinada estructura para operar. Para esto, se debe ingresar la opción ”Abrir Archivo”el cual abrirá una nueva ventana, para seleccionar el archivo a cargar, se debe presionar el botón “Seleccionar_archivo”. Estos se agregan a un text_area, el cual se puede modificar dentro de la aplicación y realizar las diferentes funciones: Guardar, Guardar como, Analizar.

```
def abrir_archivo():
    # Crea una nueva ventana emergente
    new_window = tk.Toplevel()
    new_window.title("Ingresar texto")
    new_window.geometry("600x400")
    text_area = tk.Text(new_window, width=40, height=20, font="Arial")
    text_area.place(x=50, y=10)
    def seleccionar_archivo():
            file_path = filedialog.askopenfilename()

            if file_path:
                with open(file_path, 'r') as file:
                    contenido = file.read()
                    text_area.delete(1.0, END)
                    text_area.insert(END, contenido)
                    global archivo_seleccionado
                    archivo_seleccionado = file_path
    button2 = tk.Button(new_window, text="Seleccionar Archivo", width=15, command=seleccionar_archivo)
    button2.place(x=450, y=10)
```

def abrir_archivo

Abre la ventana donde se encuentra el text_area y las funciones que puede realizar con el texto, asimismo, se indica el formato que debe tener la ventana, atributos,etc.

def seleccionar_archivo

Mediante una ventana emergente filedialog.asopenfilename, se abre una ventana para selección de archivo, se selecciona el archivo, se lee y se inserta en el text_area y guarda la direccion del archivo

2.4. Funciones de la aplicación

```
def guardar_archivo():
    global archivo_seleccionado
    if archivo_seleccionado:
        with open(archivo_seleccionado, 'w') as file:
            contenido = text_area.get("1.0", "end-1c")
            file.write(contenido)
button3 = tk.Button(new_window, text="Guardar", width=15, command=guardar_archivo)
button3.place(x=450, y=40)
def guardar_como():
    global archivo_seleccionado
    file_path = filedialog.asksaveasfilename(defaultextension=".txt", filetypes=[("Text Files", "*.txt"), ("All Files", "*.*")])
    if file_path:
        with open(file_path, 'w') as file:
            contenido = text_area.get("1.0", "end-1c")
            file.write(contenido)
            archivo_seleccionado = file_path
button4 = tk.Button(new_window, text="Guardar como", width=15, command=guardar_como)
button4.place(x=450, y=70)
```

def guardar_archivo

Sobrescribe el archivo que se selecciono anteriormente por el nuevo editado desde el text_area de la ventana de funciones.

def guardar_como

Guarda el archivo con un nombre nuevo (o el mismo) indicado por el usuario en cualquier directorio existente dentro del sistema.

```
def boton_analizador():
    global archivo_seleccionado
    autom = Automata()
    cadena = open(archivo_seleccionado, 'r').read()
    resultado = autom.analizar(cadena, Operacion(''))
    #autom.imprimir_tokens()
    if len(resultado[2])>0:
        contador = 0
        cadenatodo='{\n'
        for i in resultado[2]:
            if len(resultado[2])-1==contador:
                cadenatodo+=i.generar_cadena(contador)+'\n'
            else:
                cadenatodo+=i.generar_cadena(contador)+',\n'
            contador+=1
        cadenatodo+='}\n'
        f = open("archivo_errores.txt", "w")
        f.write(cadenatodo)
        f.close()

    if autom.estado_actual in autom.estados_aceptacion:
        for oper in resultado[1]:
            resultado_operacion = oper.operar()
            print(resultado_operacion[0], "=", resultado_operacion[1])
            grafo = oper.generar_grafo()
            # Remover caracteres no permitidos del nombre del archivo
            nombre_archivo = f'{resultado_operacion[0]}_{resultado_operacion[1]}'
            nombre_archivo = re.sub(r'[^\w\-_\. ]', '', str(resultado_operacion[1]))
            grafo.render(f'grafosgenerados/grafo_operacion_{nombre_archivo}', format='png', view=True)
```

def boton_analizador

Este botón, ejecuta la clase Automata(), abre el archivo seleccionado como lectura y comienza a analizar la estructura de datos (cadena=open(archivo_seleccionado,’r’)), si la estructura de datos es mayor a

0, entonces se comienza a leer el archivo con ciertas condiciones y contadores, si dentro de la estructura se verifica que tiene caracteres erróneos, se almacenan en un archivo_errores, si la estructura es la correcta, comienza a realizar las operaciones y se muestra en consola (if autom.estado_actual in autom.estados_aceptacion:

for oper in resultado[1]:

resultado_operacion = oper.operar()

print(resultado_operacion[0], "=", resultado_operacion[1])), si se detecta que ya no existen mas operaciones, finaliza la ejecución de la lectura de datos y se imprime un grafo tipo árbol (

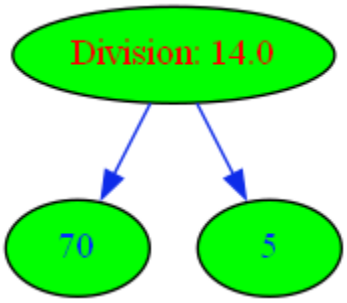
grafo = oper.generar_grafo()

nombre_archivo=f'{resultado_operacion[0]}_{resultado_operacion[1]}'

nombre_archivo = re.sub(r'[^\w\-_\.]', '', str(resultado_operacion[1]))

grafo.render(f'grafosgenerados/grafo_operacion_{nombre_archivo}', format='png', view=True))

con sus respectivas operaciones según la operación detectada dentro de la estructura de datos.




```
def cerrar_ventana():
    text_area.delete(1.0, END)
    new_window.destroy()
button9 = tk.Button(new_window, text="Salir", width=15, command=cerrar_ventana)
button9.place(x=450, y=140)
```

def cerrar_ventana

Únicamente cierra la ventana de “Abrir archivo” y borra el contenido dentro del text_area

```
def abrir_errores():
    window_errores = tk.Toplevel()
    window_errores.title("Errores generados")
    window_errores.geometry("550x400")
    text_area_errores = tk.Text(window_errores, width=40, height=20, font="Arial")
    text_area_errores.place(x=50, y=10)
    file_path = 'D:\\USAC\\2023\\Primer Semestre\\LFP\\LAB-LFP\\Proyecto1\\Proyecto1\\archivo_errores.txt'
    if file_path:
        with open(file_path, 'r') as file:
            contenido = file.read()
            text_area_errores.delete(1.0, END)
            text_area_errores.insert(END, contenido)
```

Def abrir_errores

Aquí se abre una nueva ventana (window_errores), la cual contiene un text_area para los errores, se definen los atributos de ambas cosas y se insertan los errores con el formato indicado en el text_area para poder visualizarlos.

3. Clases en uso

3.1. Operación

La clase Operación, contiene todos los métodos que utiliza el automata para la lectura del archivo de entrada, los cuales son, suma, resta, multiplicación, división, raíz, potencia, residuo, inverso, seno, coseno y tangente.

Ejemplo para suma:

```
if self.tipo.lower() == 'suma':
    for operando in self.operandos:
        if type(operando) is not Operacion:
            res += operando + ' + '
            resnum += float(operando)
        else:
            operado = operando.operar()
            if type(operado) is list:
                res += "(" + operado[0] + ") + "
                resnum += operado[1]
            else:
                res += "(" + operado + ") + "
                resnum += float(operado)
```

Primero, comprueba si el tipo de operación es "suma" y luego itera sobre los operandos de la operación, que pueden ser números o sub-operaciones de la clase Operación (como operaciones anidadas). Si el operando es un número, se agrega

a la cadena de resultados res y se suma al resultado numérico resnum. Si el operando es una sub-operación, se llama al método operar() de esa sub-operación para obtener su resultado. Si el resultado es un número, se agrega a la cadena de resultados y se suma al resultado numérico. Si el resultado es una lista, el primer elemento de la lista es la cadena del resultado y el segundo elemento es el resultado numérico.

En cualquier caso, se agrega la cadena del resultado a res y se suma el resultado numérico a resnum. La cadena del resultado se construye mediante la concatenación de los resultados de los operandos con el signo "+" entre ellos y los paréntesis para las sub-operaciones. Al final, se devuelve una lista que contiene la cadena del resultado y el resultado numérico.

3.2. Token

La clase Token:

```
class Token:
    def __init__(self, fila, columna, lexema):
        self.fila = fila
        self.columna = columna
        self.lexema = lexema
```

Únicamente almacena un dato “Token” proveniente del lexema del automata, es decir, si el automata detecta una operación correcta, se almacena en token según su fila, columna y se inserta en una lista tabla_tokens.

3.3. Error

La clase Error:

```
class Error:
    def __init__(self, fila, columna, lexema):
        self.fila = fila
        self.columna = columna
        self.lexema = lexema
    def generar_cadena(self, posicion):
      return  "\t{\n"\
                '\t\t"No.":'+str(posicion)+'\n'\
                '\t\t"Descripcion-Token":{\n'\
                '\t\t\t"Lexema":'+str(self.lexema)+'\n'\
                '\t\t\t"Tipo": Error\n'\
                '\t\t\t"Columna":'+str(self.fila)+'\n'\
                '\t\t\t"Fila":'+str(self.columna)+'\n'\
                '\t\t}\n'\
        '\t}'
```

Dentro de esta clase, se define igualmente un dato “Error” que contiene una fila, columna y lexema, estos atributos se obtienen y registran desde el Automata y se almacenan en una lista de errores.

def generar_cadena

Se define el método generar_cadena, que genera una estructura de información según el error que el automata encuentre al analizar el archivo de entrada

3.4. Automata

La clase Automata:

```
class Automata:
    letras = ["a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q","r","s", "t", "u","v","w","x","y","z", "-"]
    numeros = ["1","2","3","4","5","6","7","8","9","0", "."]
    tabla_tokens = []
    cadena = ''
    fila = 0
    columna = 0
    estado_actual = 0
    estado_anterior = 0
    estados_aceptacion = [9]
```

Dentro de esta clase, se definen las variables, listas a utilizar, se define un conjunto de letras que únicamente se pueden utilizar dentro del analizador de texto, se define un conjunto de números que únicamente se pueden utilizar dentro del analizador de texto, la variable tabla_tokens que indica una lista, la cadena=’’ que indica una variable de texto, fila que es una variable de enteros, columna que es

una variable de enteros, estado_actual una variable de enteros, estado_anterior una variable de enteros, y estados_aceptacion, una variable tipo lista de tamaño 9.

```
def analizar(self, cadena, operacion:Operacion):
    operandos = []
    token = ''
    tipo_operacion = ''
    tabla_errores=[]

    while len(cadena) > 0:
        char = cadena[0]

        # ignorar espacios en blanco o saltos de linea
        if char == '\n':
            self.fila += 1
            self.columna = 0
            cadena = cadena[1:] #abaab -> #baab
            continue
        elif char == '\t':
            self.columna += 4
            cadena = cadena[1:]
            continue
        elif char == ' ':
            self.columna += 1
            cadena = cadena[1:] #abaab -> #baab
            continue

        if self.estado_actual == 0:
            if char == '{':
                self.guardar_token(char)
                self.estado_anterior = 0
                self.estado_actual = 1
            else:
                nuevoError = Error(self.fila, self.columna, char)
                tabla_errores.append(nuevoError)
```

Def analizar

El primer argumento “cadena” es una cadena de texto que se va a analizar. El segundo argumento “operación” es una instancia de la clase Operacion que se va a utilizar para operar sobre la cadena de texto. Esto nos sirve para analizar texto y realizar operaciones.

Se definen variables:

operandos es una lista, token es una variable de texto, tipo_operacion es una variable de texto y tabla_errores es una lista.

```
while len(cadena) > 0:
    char = cadena[0]

    # ignorar espacios en blanco o saltos de linea
    if char == '\n':
        self.fila += 1
        self.columna = 0
        cadena = cadena[1:] #abaab -> #baab
        continue
    elif char == '\t':
        self.columna += 4
        cadena = cadena[1:]
        continue
    elif char == ' ':
        self.columna += 1
        cadena = cadena[1:] #abaab -> #baab
        continue

    if self.estado_actual == 0:
        if char == '{':
            self.guardar_token(char)
            self.estado_anterior = 0
            self.estado_actual = 1
        else:
            nuevoError = Error(self.fila, self.columna, char)
            tabla_errores.append(nuevoError)
```

Se implementa un ciclo while que recorre la entrada carácter por carácter.

La variable cadena es la cadena de entrada que se va a analizar, y self.estado_actual es una variable que representa el estado actual del analizador.

La línea char = cadena[0] asigna el primer carácter de la cadena de entrada a la variable char.

Las siguientes líneas verifican si el carácter es un espacio en blanco, un salto de línea o una tabulación, y actualizan la posición del cursor en la fila y columna correspondientes en consecuencia. Si el carácter es alguno de estos caracteres de espacio, la línea cadena = cadena[1:] elimina ese carácter de la cadena y continúa con el siguiente carácter.

Si el carácter no es un espacio en blanco, se comprueba el estado actual del analizador. Si el estado actual es 0, se comprueba si el carácter es una llave abierta {. Si es así, se guarda el token en la lista de tokens y se actualizan los estados self.estado_anterior y self.estado_actual para que reflejen que se ha encontrado un token válido.

Si el carácter no es una llave abierta, se crea un nuevo objeto Error que indica la fila, columna y carácter en el que se encontró el error, y se agrega a la lista tabla_errores.

```
elif self.estado_actual == 1:
    if char == '"':
        self.guardar_token(char)
        self.estado_anterior = 1
        self.estado_actual = 2
    elif char == '{':
        self.guardar_token(char)
        self.estado_anterior = 1
        self.estado_actual = 10
    else:
        nuevoError = Error(self.fila, self.columna, char)
        tabla_errores.append(nuevoError)
```

Entonces, al cambiar de estado actual, se pasa al siguiente que seria el 1. Se realiza la misma operación que el estado anterior, por lo que verifica si la variable char equivale a algún carácter definido (“, {), si es correcto, se almacena un nuevo Token, si es incorrecto, se crea un objeto tipo Error y se almacena en la tabla_errores.

Así es como se definieron todos los estados.

```
elif self.estado_actual == 2:
    if char.lower() in self.letras:
        token += char
        self.estado_anterior = 2
        self.estado_actual = 3
    else:
        nuevoError = Error(self.fila, self.columna, char)
        tabla_errores.append(nuevoError)

elif self.estado_actual == 3:
    if char.lower() in self.letras:
        token += char
        self.estado_anterior = 3
        self.estado_actual = 3
    elif char == '"':
        self.guardar_token(token)
        token = ''
        self.guardar_token(char)
        self.estado_anterior = 3
        self.estado_actual = 4
    else:
        nuevoError = Error(self.fila, self.columna, char)
        tabla_errores.append(nuevoError)
elif self.estado_actual == 4:
    if char == ':':
        self.guardar_token(char)
        self.estado_anterior = 4
        self.estado_actual = 5
    else:
        nuevoError = Error(self.fila, self.columna, char)
        tabla_errores.append(nuevoError)
```

```
elif self.estado_actual == 5:
    if char == '"':
        self.guardar_token(char)
        self.estado_anterior = 5
        self.estado_actual = 6
    else:
        nuevoError = Error(self.fila, self.columna, char)
        tabla_errores.append(nuevoError)
elif self.estado_actual == 6:
    if char.lower() in self.letras:
        token += char
        self.estado_anterior = 6
        self.estado_actual = 7
    else:
        nuevoError = Error(self.fila, self.columna, char)
        tabla_errores.append(nuevoError)
elif self.estado_actual == 7:
    if char.lower() in self.letras:
        token += char
        self.estado_anterior = 7
        self.estado_actual = 7
    elif char == '"':
        self.guardar_token(token)
        token = ''
        self.guardar_token(char)
        self.estado_anterior = 7
        self.estado_actual = 8
    else:
        nuevoError = Error(self.fila, self.columna, char)
        tabla_errores.append(nuevoError)
```

```
elif self.estado_actual == 8:
    if char == '}':
        self.guardar_token(char)
        self.estado_anterior = 8
        self.estado_actual = 9
    elif char == ',':
        self.guardar_token(char)
        self.estado_anterior = 8
        self.estado_actual = 1
    else:
        nuevoError = Error(self.fila, self.columna, char)
        tabla_errores.append(nuevoError)
elif self.estado_actual == 10:
    if char == '"':
        self.guardar_token(char)
        self.estado_anterior = 10
        self.estado_actual = 11
    else:
        nuevoError = Error(self.fila, self.columna, char)
        tabla_errores.append(nuevoError)
elif self.estado_actual == 11:
    if char.lower() in self.letras:
        token += char
        self.estado_anterior = 11
        self.estado_actual = 12
    else:
        nuevoError = Error(self.fila, self.columna, char)
        tabla_errores.append(nuevoError)
```

```
operacion.operandos = operacion
return [cadena, operandos, tabla_errores]
```

Estas líneas de código están en una función que tiene como objetivo analizar una cadena de texto y retornar una lista con información sobre la operación a realizar, los operandos y los errores encontrados en la cadena.

En la primera línea, se asigna la operación actual como único operando de la misma operación, esto sucede porque en la cadena analizada no se encontró ningún operando válido. Es importante mencionar que esto puede causar errores al momento de intentar operar los valores.

En la segunda línea, se retorna una lista con tres elementos:

La cadena que se analizó.

Los operandos encontrados durante el análisis.

Una lista con los errores encontrados durante el análisis (si es que existen).

```
def guardar_token(self, lexema):
    nuevo_token = Token(self.fila, self.columna, lexema)
    self.tabla_tokens.append(nuevo_token)
```

Def guardar_token

Este método se utiliza para crear y guardar objetos Token a una tabla de tokens (tabla_tokens), el método toma un argumento lexema que representa el valor del token que se va a crear, se crea el objeto y se inserta en la lista.

3.4.1. AFD utilizado