Manual Tecnico

> Este programa se desarrollo con el lenguaje de programacion python en su version 3.10.6 y haciendo uso de algunos modulos como os, re, io, graphviz.

> Se utilizo programacion orientada a objetos, para poder crear varios objetos que nos seran utiles para ejecutar funciones en cualquier parte del programa

> Este programa tiene un ejecutable para iniciar a ejecutar

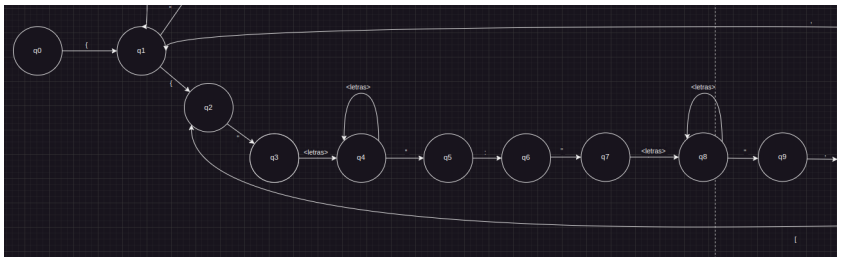
1. AUTOMATA

Se necesitaba crear un Automata Finito Determinista el cual fuera capaz de reconocer una estrutura de datos similiar a la siguiente:

```
{
  {
    "Operacion":"Suma",
    "Valor1":4.5,
    "Valor2":5.32
  },
  {
    "Operacion":"Resta",
    "Valor1":4.5,
    "Valor2":
    [
      "Operacion":"Potencia",
      "Valor1":10,
      "Valor2":3
    ]
  },
  {
    "Operacion":"Suma",
    "Valor1":
    [
      "Operacion":"Seno",
      "Valor1":90
    ]
    "Valor2":5.32
  }
  "Texto":"Realizacion de Operaciones",
  "Color-Fondo-Nodo":"Amarillo",
  "Color-Fuente-Nodo":"Rojo",
  "Forma-Nodo":"Circulo"
}
```

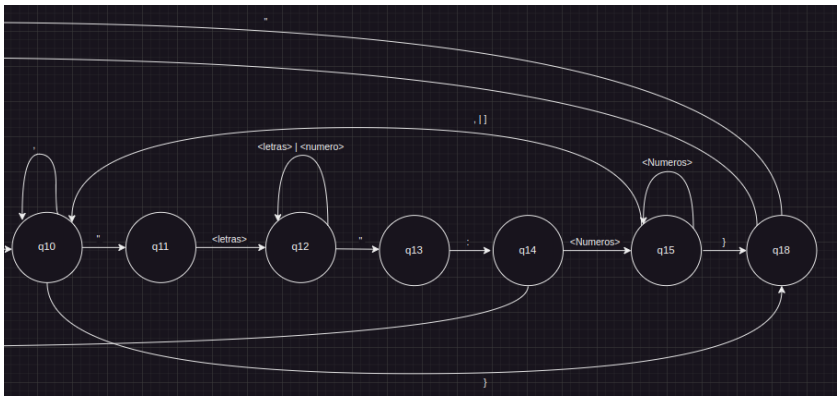
Y dependiendo de lo que venga escrito en el documento que realice esas operaciones. El primer paso como tal fue crear este automota capaz de reconocer esta estrurtura y se establecio de la siguiente manera:

1. Hasta este punto (q9) del automa es capaz de reconocer las llaves de inicio y la primera linea de caracteres y simbolos especiales, donde se indica el tipo de operacion que se desea realizar



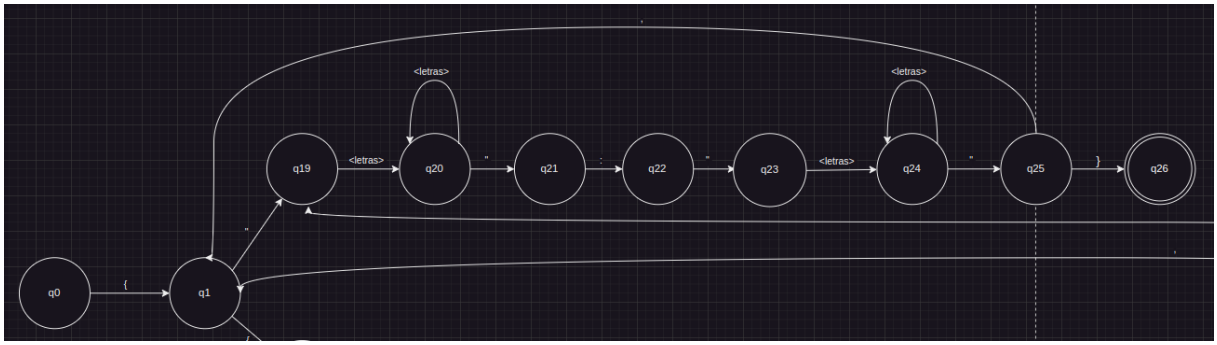
Funte: elaboracion propia 2023

2. Apartir del estado q10 el automata se encargara de recibir cadenas que indican que valor se esta almacenando y dependiendo de los caracteres especiales que vengan realizar transiciones hacia estados especificos para poder seguir recolectando valores o pasar a la siguiente operacion.



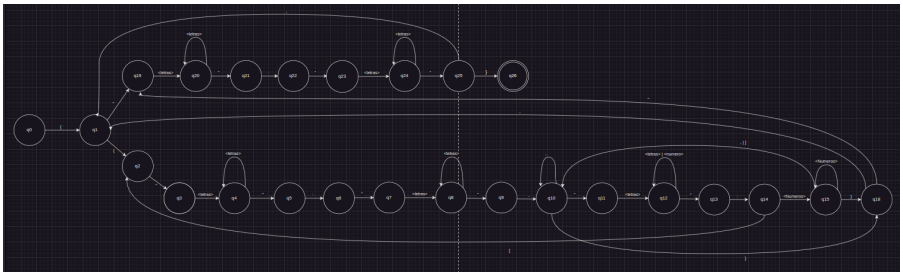
Fuente: Elaboracion propia

3. En en el camino superior del automata se tiene un algoritmo muy similar al de la primer imagen explicada del automata, en esta parte se encargara de recolectar informacion que sera util para darle atributos a ciertas graficas que se crearan al momento de realizar las operaciones leidas. En esta parte del automata es donde se encuentra el estado de aceptacion del automata.



Fuente: Elaboracion propia, 2023

4. Vista general del Automata creado



Fuente: Elaboracion propia, 2023

Terminadando esto se procede a pasarlo a codigo y empezar a guardar informacion util para poder resolver el enunciado.

2. Automata.py

En esta clase escrebiremos el automata que se diseno previamente se declaro una clase la cual se llamo ‘Automata’ el cual tendra los siguiente atributos:

```
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 = []
tabla_errores = []
cadena = ''
fila = 0
columna = 0
estado_actual = 0
estado_anterior = 0
estados_aceptacion = [26] #El unico estado de aceptacion
inidiceError = 0
```

1. El primer atributo contendra el listado de caracteres posibles que podra reconocer el automata al igual que los numeros y algunos caracteres especiales.
2. Se tiene dos listas, las cuales se utilizaran para almacenar informacion importante como las palabras, numeros y errores que se encuentren durante la lectura
3. La variable Cadena simplemente almacena todo el contenido que trae el documento de prueba o que se esta analizando
4. Las variables de tipo fila y columna se utilizan para tener una especie de coordenada donde se guardo cierto caracter
5. Se tienen dos variables denominadas como ‘estados’. Con estas variables estaremos jugando para ir avanzando en nuestro automata o regresando dependiendo de nuestra necesidad
6. Estados de aceptacion indica el numero o nombre del estado en donde tendriamos un archivo aceptado, en caso el archivo termine y no se encuentra en alguno de estos estados almacenados en esta parte, se tomaria como una cadena no aceptada por el automata.

1.1 def Analizar(cad, Operacion):

Esta es la funcion que almacena nuestro automata, en la cual se definen algunas variables al inicio:

```
        operandos = []
        tipo_operacion = ""
        token = ''
```

1. La primera lista se utilizo para poder almacenar cadenas donde se indica los valores numericos de las operaciones a realizar
2. la variable tipo de operacion guardara la cadena en donde el documento indique que tipo de operacion se procedera a realizar
3. En la variable token almacenara de forma temporal el caracter que se este estudiando o una concatenacion de los mismos.

Posteriormente se encuentra un ciclo while el cual se encargara de recorrer todo el caracter del texto que se desea analizar, para esto el ciclo se encarga de eliminar el caracter ubicado en el primer indice ya que este caracter ya fue estudiado.

Dentro del ciclo se encuentran muchos condicionales los cuales haran que las variables estados vayan cambiando dependiendo de la condicion que se encuentre, se muestra algunos de los codicionales ya que este codigo es demasiado extenso:

```
#Inicia el Automata
            if self.estado_actual == 0:
                if char == '{' :
                    self.guardar_token(char)
                    self.estado_anterior = 0
                    self.estado_actual = 1
                else:
                    self.inidiceError +=1
                    self.guardar_error(char, "Error Sintactico" ,self.inidiceError)

            elif self.estado_actual == 1:     # q1
                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 = 19
                else:
                    self.inidiceError +=1
                    self.guardar_error(char, "Error Sintactico" ,self.inidiceError)


            elif self.estado_actual == 2:     #q2
                if char == '"' :
                    self.guardar_token(char)
                    self.estado_anterior = 2
                    self.estado_actual = 3
                else:
                    self.inidiceError +=1
                    self.guardar_error(char, "Error Sintactico" ,self.inidiceError)

            elif self.estado_actual == 3:     #q3
                if char.lower() in self.letras :
                    token += char
                    self.estado_anterior = 3
                    self.estado_actual = 4
                else:
                    self.inidiceError +=1
                    self.guardar_error(char, "Error Lexico" ,self.inidiceError)

            elif self.estado_actual == 4:     #q4
                if char.lower() in self.letras :
                    token += char
                    self.estado_anterior = 4
                    self.estado_actual = 4
                elif char == '"' :
                    self.guardar_token(token)
                    token = ''
                    self.guardar_token(char)
                    self.estado_anterior = 4
                    self.estado_actual = 5
                else:
                    self.inidiceError +=1
                    self.guardar_error(char, "Error Lexico" ,self.inidiceError)
            #Para finalizar ciclo
            self.columna += 1  #aumenta columna
            cadena = cadena[1:] #Elimina el caracter estudiado
        operacion.operandos = operacion
        return [cadena, operandos]
```

De la misma manera se realizan los condicionales con todos los estados establecido en nuestro automata.

Dentro de estos condicionales hacemos algunas operaciones para poder almacenar la informacion deseada y mandarla a funciones donde los manipularemos.

def guardar_token(n)

Esta funcion es con tipos de datos abstractos esta sirve para poder guardar, cadenas de caracteres concatenadas de manera que se pudieran utilizar esta informacion mas adelante con las operaciones.

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

Esta funcion es utili para poder agregarle a las listas de la clase un nodo con toda es informacion contendia, para poder utilizarlos y manipularlos con mas facilidad

def guardar_error()

De la misma manera esta es una funcion de tipo de dato abstracto, esta sirve para poder almacenar los errores capturados en el texto analizado

```
def guardar_error(self, lexema, tipoError, id):
        nuevo_token = Error(self.fila, self.columna, lexema, tipoError, id)
        self.tabla_errores.append(nuevo_token)
```

Esta funcion es utili para poder agregarle a las listas de la clase un nodo con toda es informacion contendia, para poder utilizarlos y manipularlos con mas facilidad

def imprirResultados( )

En esta funcion se crea un archivo .dot este contendra los grafos de la operaciones realizadas pero para esto, la funcion llama a una funcion de otra clase la cual se detallara mas adelante.

```
def imprimrResultados(self, operandos):
        # lista_Operaciones = []
        # lista_Resultados = []
        num = 0
        texto = ''
        texto +="digraph {\n"
        texto += "\trankdir=TB\n"
        if self.estado_actual in self.estados_aceptacion:
            for oper in operandos[1]:
                num += 1
                resultado = oper.operar(num) #se crea objeto Operaciones
                texto += resultado[2]
                print(resultado[0], "=", resultado[1])  #tendra una lista de dos elementos [string = int
        texto += "}\n"
        file = open("grafo.dot", "w")
        file.write(texto)
        file.close()
        os.system("dot -Tpdf grafo.dot -o  grafo.pdf")
```

La ultima linea de esta funcion se encarga de hacer uso del modulo os.system para convertir el archivo .dot creado anteriormente en un archivo de formato pdf.

2. Tonken.py

Esta clase simplemente almacena la informacion del cualquier nodo y tiene la siguiente estructura

```
class Tonken:
    def __init__(self, fila, columna, lexema):
        self.fila = fila  #Fila del caracter
        self.columna = columna  #Columna del caracter
        self.lexema = lexema    #El caracter
```

3. Error.py

Esta clase simplemente almacena la informacion del cualquier nodo y tiene la siguiente estructura

```
class Error:
    def __init__(self, fila, columna, lexema, tipoError, id):
        self.fila = fila  #Fila del caracter
        self.columna = columna  #Columna del caracter
        self.lexema = lexema    #El caracter
        self.tipoError = tipoError
        self.id = id
```

4. Operaciones.py

Se creo una clase denominada ‘Operacion’ la cual tiene los siguientes atributos:

```
def __init__(self, tipo) :
        self.tipo = tipo
        self.operandos = []  #lista los valores de la operacion
        self.texto = ''
        self.aux = 0
```

El primer atributo se encarga de almacenar el tipo de operacion como un string.

dentro de esta clase se encuentra una funcion llamada ‘Operar’ la cual es donde se realizaran todas las operaciones reconocidas en el analisis.

def Operar(id)

Esta operacion recibe un id para poder identificar el orden de las operaciones y asi poder crear los grafos con una relacion correcta. dentro de esta operacion se tienen dos variables y varios condicionales. Una de estas variables se encargara de almacenar una cadena con la estructura de la operacion, en caso haya operaciones anidadas y la segunda variable almacenara el valor total resultado de realizar las operaciones. Cada condicional representa una operacion y dentro de este se encuentra un ciclo el cual recorre una lista que contendra toda la operacion recolectada del analisis. Uno de los condicionales puede verse de la siguiente manera.

```
if self.tipo.lower() == 'suma':
            for operando in self.operandos:
                if type(operando) is not Operacion:  #significa que es algo simple como un NUMERO
                    #resultado numerico
                    res += operando + ' + '
                    resnum += float(operando) #con flotante para evitar clavos

                    #grafica
                    self.texto += f"\t{str(operando)} [shape=circle style=filled color = blue]\n "
                    self.texto += f"\t{str(self.tipo.lower())+str(id)} -> {str(operando)} [shape=record color=red]\n"
                else:
                    operado = operando.operar(id)     #Recursividad en caso la operacion venga anidada con el else llamamos de nuevo para t
                    # regresa con valores - [cadena, resultado]
                    #en caso venga anidada sera aux para obtener nodos internos
                    res += "(" + operado[0] + ") + "# y asignarlos en el tipo de operacion.  Par identificar que era operacion concatenada
                    resnum += operado[1]  #Se suma el resultado de la operacion anidada al total

                    #Descomponer operacion anidada
                    operado[0] = re.sub("\+","",operado[0]) #quita el signo
                    operado[0] = re.sub("\-","",operado[0]) #quita el signo
                    operado[0] = re.sub("\(|\)","",operado[0]) #quita el signo
                    anidada = operado[0].split()    #se descompone
                    for i in anidada:   #se itera
                        self.aux += int(i)   #se crea el total interno
                        self.texto += f"\t{str(i)} [shape=circle style=filled color = blue]\n " #nodo operacion anidada
                        self.texto += f"\t{str(self.tipo.lower())+str(id+100)} -> {str(i)} [shape=record color=red]\n"  #coneccion con subn
                    self.texto += f"\t{str(self.tipo.lower())+str(id+100)} [shape=circle style=filled color = blue, label=<{'suma: '+ str(s
                    self.texto += f"\t{str(self.tipo.lower())+str(id)} -> {str(self.tipo.lower())+str(id+100)} [shape=record color=red]\n"
            #finaliza For
            self.texto += f"\t{str(self.tipo.lower())+str(id)} [shape=circle style=filled color = blue, label=<{'suma: '+ str(resnum)}>]\n
        #RESTA
```

dentro del ciclo mencionado se encuentran dos opciones,

1. Que los valores que trae la lista sean simples, es decir no son otra operacion anidada, en este caso el bloque if se encarga de concatenar los valores con su respectivo signo y de sumar cada valor que contenga la operacion
2. En caso contrario, la operacion seria anidada y se procedera a llamar a la funcion nuevamente (Recursivida) ya que la funcion retorna la cadena de una operacion simple y su total, una vez obtenidos estos totales ya simplemente es de concatenar de nuevo y de sumar los datos obtenidos

Se encuentra una variable de tipo texto, esta tambien sera retornada al lugar donde es llamada la funcion, esta variable almacena data importante para la creacion de los nodos del grafo y asi mismo hace las relaciones entre ellos para poder general el pdf con los resultados.

En esta clase se encuentra mas funciones las cuales son complementarias para poder realizar todas las operaciones que describe el enunciado del proyecto.

5. VentanaPrincipal.py

En esta clase se creo toda la interfaz grafica y las funcionalidades de sus botones y menus. Cabe recalcar que se utilizo Tkinter para la realizacion de la misma y esta se encuentra compuesta por las siguientes funciones:

def AbrirArchivo( )

En esta parte se crea una variable global para almacenar la ruta de un archivo y posteriormente haciendo uso de ‘Filedialog’ se abre una ventana en donde se podra seleccionar algun archivo con extension .txt y es aqui donde se almacena la ruta.

Posteriormente con un condicional se valida si la ruta esta vacia o no, en caso no este vacia procede a abrir el archivo y a mostralo en la caja de texto que contiene la ventana.

Se implemento la misma idea para poder abrir documentos en pdf dentro de la aplicacion y para abrir los archivos de errores

El codigo se ve de la siguiente manera:

```
def AbrirArchivo():
    global ruta

    ruta = FileDialog.askopenfilename(
        initialdir='.',
        filetypes=(
        ("Ficheros de texto", "*.txt"),
        ),
        title= "Abrir un fichero"
    )

    if ruta != "":
        fichero = open(ruta, 'r')
        contenido = fichero.read()
        caja_texto.delete(1.0, 'end')
        caja_texto.insert('insert', contenido)
        fichero.close()
        root.title(ruta + " - Mi editor")
```

def Guardar( )

Esta funcion se es muy similar al ultimo condicional de la funcion abrir archivo ya que valida si la ruta no esta vacia y si se cumple entonces recolecta el texto que se encuentra en la ventana de la interfaz, abre el archivo de la misma ruta pero con el atributo ‘w+’ esto para poder sobreescribirlo con lo que ya tiene. y en caso contrario se llama a la funcion GuardarComo. se ve de la siguiente manera:

```
def Guardar():
    if ruta != "":
        contenido = caja_texto.get(1.0, 'end-1c')
        fichero = open(ruta, 'w+')
        fichero.write(contenido)
        fichero.close()
    else:
        savefileas()
```

def savefiles( )

Esta funcion nos abre una ventana en donde podres escribir el nombre con que queremos guardar el texto que se encuentra la caja de texto, esto para poder almacenarlo en una nueva ruta y asi poder llamarlo un nuevo archivo. se ve de la siguiente manera:

```
def savefileas():
    try:
        path = FileDialog.asksaveasfile(filetypes = (("Text files", "*.txt"), ("All files", "*.*"))).name
        root.title('Notepad - ' + path)

    except:
        return

    with open(path, 'w') as f:
        f.write(caja_texto.get('1.0', END))
```

def Analizar( )

Esta es la funcion encarga de llamar y manipular la informacion que regresen todos los metodos de la clase automata.

Primero crea un objeto de tipo Automata para poder llamar a sus funciones. posteriormente se procede a llamar al metodo analizar el cual devolvera los valores de cada operacio y esta funcion se encarga de mandarlos a la otra clase llamada Operaciones.

Esta funcion tambien crea un archivo con formato json de los errores que se lograron recolectar durante el analisis del texto .

el codigo se ve de la siguiente manera:

```
def Analizar():
    funcion = Automata()  #objeto tipo Automata
    contenido = caja_texto.get(1.0, 'end-1c') #Extrae contenido de la caja (Interfaz)

    resultado = funcion.analizar(contenido, Operacion('suma'))  #Se llama al automata y se le pasa lo extraido.  REtorna valores
    lista_errores = funcion.imprimir_errores()  #REtorna la lista de errores

    #Borrar datos del doc
    texto = ''
    os.remove("/home/jhonatan/Documentos/Universidad_USAC/Semestre5/01_LenguajesFormales/Proyecto1/Errores.txt")
    #Si hay errores -> no pdf
    if len(lista_errores) == 0:
        print("No hay errores")
        funcion.imprimrResultados(resultado)
    else:
        print("Si hay errores")
        caja_texto.delete(1.0, END)
        caja_texto.insert(1.0, "ALETA: Existen errores en el documento, De clic en el boton 'Errores' para seleccionar el archivo que los c

        for val in lista_errores:
            texto += '{\n'
            texto += '\t{\n'
            texto += f'\t\t"No.":{val.id}\n'
            texto += '\t\t"Descripcion-Token":{ \n'
            texto += f'\t\t\t"Lexema": {val.lexema}\n'
            texto += f'\t\t\t"Tipo": {val.tipoError}\n'
            texto += f'\t\t\t"Columna": {val.columna}\n'
            texto += f'\t\t\t"Fila": {val.fila}\n'
            texto += '\t\t}\n'
            texto += '\t},\n'
        texto += '}\n'
        print(texto)
        file = open("Errores.txt", "w")
        file.write(texto)
        file.close()
```