MANUAL TÉCNICO

1. Clase Gramática y AFD

Son los modelos principales a utilizar durante el programa, generalmente se está interactuando con objetos de este tipo. La clase contiene las partes que le pertenecen a un AFD o una gramática regular, además de agregar un atributo nombre, que es muy importante porque es el identificador de cada objeto en la realización de las distintas opciones de menús.

```python
# CLASE GRAMATICA
class GramaticaRegular():
    def __init__(self,nombre,No_Terminales,terminales,NT_Inicial,producciones,estadosAceptacion):
        self.nombre=nombre
        self.No_Terminales=No_Terminales
        self.terminales=terminales
        self.NT_Inicial=NT_Inicial
        self.producciones=producciones
        self.estadosAceptacion=estadosAceptacion


#CLASE AUTOMATA
class Automata():
    def __init__(self,nombre,estados,alfabeto,estado_Inicial,estados_Aceptacion,transiciones):
        self.nombre = nombre
        self.estados = estados
        self.alfabeto = alfabeto
        self.estado_Inicial=estado_Inicial
        self.estados_Aceptacion =estados_Aceptacion
        self.transiciones=transiciones

    def editar_ListaEstados(self,nuevaLista):
        self.estados = nuevaLista
```

2. Guardar transiciones o producciones

```python
class Estado():
    def __init__(self):
        self.nombre=""
        self.transiciones=[]

    def __str__(self):
        return self.nombre

    def agregarNombre(self,nombre):
        self.nombre=nombre

    def definirTransicion(self,hacia,simbolo):
        transicion_Temporal=Transicion()
        transicion_Temporal.Estado=hacia
        transicion_Temporal.simbolo=simbolo
        self.transiciones.append(transicion_Temporal)

    def get_Nombre(self):
        retu
    def get_
        for
```

```python
class Transicion():
    def __init__(self):
        self.Estado=""
        self.simbolo=""

    def __str__(self):
        return "Hacia : "+self.Es

    def get_Estado_Hacia(self):
        return self.Estado

    def get_Simbolo(self):
        return self.simbolo
```

Para guardar las transiciones o producciones de una gramática o un AFD se necesitan de dos clases más llamadas Estado y Transición.

La clase Estado contiene el nombre del estado de donde empieza la transición además de una lista de objetos tipo Transición, se hace de esta forma para poder guardar para un mismo estado varias transiciones y no crear nuevos estados con el mismo estado donde empieza la transición.

La clase Transición almacena el nombre del estado hacía donde se dirige la transición y el símbolo del alfabeto.

Guardarlos de esta forma optimiza el proceso de encontrar símbolos o estados hacia donde se dirige la transición.

3. Ingresar estados o No terminales

```
print("Ingresando estados a: "+nombre_AFD+"\n")
estado= input("Estado No."+str(contador1)+": ")
estado= estado.upper()

if verificar_Lista_Vacia(estado)==True:
    print("\n******** DEBE INGRESAR UN ESTADO VALIDO, INTENTE NUEVAMENTE ********")
elif estado in listaTemporal_Estados:
    print("\nEL ESTADO INGRESADO YA EXISTE, INGRESE UNO DISTINTO")
elif estado in listaTemporal_Alfabeto:
    print("\nEL ESTADO INGRESADO COINCIDE CON UN SIMBOLO DEL ALFABETO, INTENTE NUEVAMENTE")
else:
    listaTemporal_Estados.append(estado)
    contador1=contador1+1
```

Verifica que la cadena no esté vacía y también si es un estado repetido envía un mensaje en consola advirtiendo que esa acción no es posible realizarla.
Después de pasar por las validaciones anteriores significa que la cadena es valida, entonces lo agrega a la lista de estados (atributo de un AFD y gramática).

4. Ingresar alfabeto o terminales

```
elif opcion1==2:
    os.system('cls')
    print("Ingresando alfabeto a: "+nombre_AFD)
    terminal=input("Terminal No. "+str(contador2)+":") # TERMINAL ES UN SIMBOLO DEL ALFABETO

    if verificar_Lista_Vacia(terminal)==True:
        print("\n******** DEBE INGRESAR UN SIMBOLO VALIDO, INTENTE NUEVAMENTE ********")
    else:
        if terminal in listaTemporal_Estados:
            os.system('cls')
            print("\nEL SIMBOLO INGRESADO COINCIDE EN LA LISTA DE ESTADOS")
        elif terminal in listaTemporal_Alfabeto:
            print("\nEL SIMBOLO INGRESADO YA EXISTE")
        else:
            os.system('cls')
            listaTemporal_Alfabeto.append(terminal)
            contador2=contador2+1
#TERMINA INGRESAR ALFABETO
```

Además de hacer las mismas validaciones que el ingreso de estados (si la cadena está vacía y si ya existe símbolo en la lista) se agrega el símbolo a la lista de terminales o lista alfabeto.

5. Asignar estado inicial

```
print("Ingresando estado inicial a: "+nombre_AFD)
inicial=input("Estado inicial: ")

if inicial in listaTemporal_Estados:
    pass
else:
    inicial=""
    print("\nEL ESTADO INICIAL INGRESADO NO EXISTE EN LA LISTA DE ESTADOS")
inicial=inicial.upper()
estadoTemporal_Inicial=inicial
```

Para AFD y gramática existe únicamente un estado inicial, por lo tanto, lo primero que se hace es verificar que el estado inicial exista en la lista de estados antes ingresados.
Sino se cumple esta condición envía un mensaje de alerta y se vacía la variable para estado inicial, por último, se asigna el estado ingresado a la variable propiamente del objeto, ya sea de autómata o gramática.

6. Ingresar estados de aceptación

```
print("Ingresando estados de aceptacion a: "+nombre_AFD)
E_aceptacion= input("Estado de aceptacion No. "+str(contador4)+":")
E_aceptacion=E_aceptacion.upper()

if E_aceptacion in listaTemporal_Estados:
    listaTemporal_EstadosAceptacion.append(E_aceptacion)
    contador4=contador4+1
else:
    print("\nEL ESTADO DE ACEPTACION INGRESADO NO EXISTE EN LA LISTA DE ESTADOS")
```

Para que un estado sea de aceptación, se necesita que exista en la lista de estados. Esta validación se hace en el if, si es así se ingresa a la lista de estados de aceptación.
Si no se cumple esta condición envía un mensaje de alerta mencionando que el estado ingresado no existe en la lista de estados.

7. Guardar transiciones de un AFD en el modo 1

```
existe=False
os.system('cls')
print("----- Se encuentra en el modo 1 -----")
print("NOTA: La notacion correcta para ingresar una transicione es: A,B;0\n")
transicion=input("Transicion No. "+str(contador5)+": ") #A,B;0
separacion1=transicion.split(";") #["A,B","0"]
separacion2=separacion1[0].split(",") #["A","B"]
terminal = separacion1[1] #0

#VERIFICANDO SI EXISTE EL ESTADO EN LA LISTA
for x in range(0, len(listaTemporal_Transiciones)):
    if separacion2[0] == listaTemporal_Transiciones[x].nombre:
        existe=True
    else:
        continue

#SI EXISTE, BUSCAR EL ESTADO Y SOLO AGREGAR LA TRANSICION
if existe==True:
    print("EL ESTADO -"+separacion2[0]+"- YA EXISTE")
    buscarEstado(separacion2[0],listaTemporal_Transiciones).definirTransicion(se
#SINO EXISTE, CREAR UN NUEVO ESTADO Y AGREGAR LA TRANSICION
else:
    print("EL ESTADO -"+separacion2[0]+"- NO EXISTE")
    estado_Temporal=Estado()
    estado_Temporal.agregarNombre(separacion2[0])
    estado_Temporal.definirTransicion(separacion2[1],terminal)
    listaTemporal_Transiciones.append(estado_Temporal)

contador5=contador5+1
```

La notación para ingresar transiciones en el modo 1 es: A, B; 0 (El estado que inicia la transición, el estado hacía donde se dirige y el símbolo).
Entonces al ingresar esta cadena, lo primero que se hace es utilizar el método split para separar la cadena por “;”. Luego se utiliza nuevamente el método split para obtener los dos estados.
El ciclo for se utiliza para comprobar si el estado del que inicia la transición existe. Esto se hace con el objetivo de no crear estados con el mismo nombre, es decir, tener solamente un estado “A” por ejemplo que contenga la lista de transiciones (estado hacia donde se dirige la transición y el símbolo).
De esa forma se optimiza la búsqueda de símbolos o la evaluación de cadenas, que se explica más adelante.

8. Guardar transiciones de un AFD en el modo 2

```
#SEPARANDO CADENA1 Y ALMACENANDO EN LISTA_ALFABETO
lista_Alfabeto = cadena1.split(",")
#SEPARANDO CADENA2 Y ALMACENANDO EN LISTA_ESTADOS_DE
lista_Estados_De = cadena2.split(",")
#SEPARANDO CADENA3 Y ALMACENANDO EN LISTA_TEMPORAL
lista_Temporal = cadena3.split(";")

contador = 0
for estado_De in lista_Estados_De:
    estadoTemporal = Estado()
    estadoTemporal.agregarNombre(estado_De)
    for indice in range(0,len(lista_Alfabeto)):
        aux = lista_Temporal[contador].split(",")
        estadoTemporal.definirTransicion(aux[indice],lista_Alfabeto[indice])

    contador+=1
    lista_Transiciones.append(estadoTemporal)

listaTemporal_Transiciones = lista_Transiciones.copy()
```

Para ingresar transiciones por le método 2, se ingresan mediante 3 cadenas de texto, la primera necesita el alfabeto, la segunda el nombre de los estados, y por último se piden las transiciones de la forma: A, C que significa los estados hacia donde se dirige la transición, dependiendo la cantidad de símbolos del alfabeto así mismo se ingresaran grupos de la misma cantidad en esta última.

9. Guardar objeto autómata y gramática correspondiente

```
#CREANDO OBJETO AUTOMATA
estadoTemporal_Inicial: str nbre_AFD,listaTemporal_Estados,listaTemporal_Alfabeto,
estadoTemporal_Inicial,listaTemporal_EstadosAceptacion,listaTemporal_Transiciones)
#AGREGANDO A LA LISTA AUTOMATAS
listaAutomatas.append(automata_Temporal)

#GUARDANDO SU GRAMATICA CORRESPONDIENTE
nombre_Gramatica=nombreaux+" Gramatica"
gramatica_Temporal=GramaticaRegular(nombre_Gramatica,listaTemporal_Estados,listaTemporal_Alfabeto,
estadoTemporal_Inicial,listaTemporal_Transiciones,listaTemporal_EstadosAceptacion)
#AGREGANDO A LA LISTA
listaGramaticas.append(gramatica_Temporal)

print("AFD guardado con exito")

break
```

Un AFD y una gramática lo forma las mismas partes, a excepción que en un AFD se necesitan conocer los estados de aceptación de forma directa, en cambio en una gramática se entiende que un estado es de aceptación si aparece la siguiente notación ESTADO > épsilon. Debido a esto simplemente se crean el objeto gramática y AFD y se guardan las partes que requiere cada uno.

10. Validar cadena

```
inicio = estado_Inicio
actual = inicio
cadena = cadena_A_Evaluar
fin = False
contador = 0
while fin == False:

    if contador > (len(cadena)-1):
        fin = True
        break
    else:
        print("---Contador---"+str(contador))
        if actual in lista_Estados:
            if Existe_EstadoDe_Boolean(actual, lista_General_Transiciones) == True:
                if Existe_Simbolo_En_EstadoHacia(actual, cadena[contador], lista_General_Transiciones, lista_Estad
                    actual = Existe_Simbolo_En_EstadoHacia_Estado_Hacia(actual, cadena[contador], lista_General_Tra

        contador = contador+1
        print("ACTUAL: "+actual)
        print()


if actual in lista_Estados_Aceptacion:
    print("---CADENA CORRECTA---")
else:
    print("---CADENA INCORRECTA")
```

Para el ciclo while se necesitan de 4 métodos más. Un método que busca en la lista de transiciones el estado donde comienza la transición y retorna un valor booleano. El segundo método realiza la misma función que el primero, es decir, busca en la lista de transiciones el estado donde inicia la transición con la diferencia que este método retorna la lista de transiciones del estado. El tercer método busca si existe cierto símbolo en la lista de transiciones que retorna el segundo método mencionado. Por último, el cuarto método verifica si existe el símbolo y si es así, retorna el nombre del estado hacia donde se dirige, esto se hace para tener la referencia del estado en que se quedó la cadena.
El ciclo while hace referencia a tres de los métodos antes mencionados, que lo único que hace es llevar el control del tamaño de la cadena, ir comparando el símbolo y si es así actualizar la variable actual.
Para determinar si la cadena es correcta se compara si “actual” está la lista de estados de aceptación, esto significa que el símbolo termino en un estado de finalización lo que indica que la cadena es correcta, sino automáticamente se dice que es incorrecta.

11. Ruta en AFD

Se utilizan el mismo proceso que en “validar cadena”, con la única diferencia que se en la ejecución se va imprimiendo el valor de actual y el símbolo, esto para que aparezca en consola del estado donde viene hacia el estado donde se dirige y el símbolo, ejemplo: A, B;0

12. Expandir gramática

Mismo proceso que los dos métodos anteriores (validar cadena y ruta en AFD), con la diferencia que en este método existe una variable “gramática expandida” que va acumulando el símbolo o terminal en cada pasada del ciclo, esto para que al final aparezca el total de la cadena evaluada y si es correcta o no.

12. Buscar gramática o buscar AFD

```
def buscar_Gramatica(nombreGramatica):
    gramatica_Temporal= GramaticaRegular("","","","","","")
    for x in range(0, len(listaGramaticas)):
        if nombreGramatica == listaGramaticas[x].nombre:
            gramatica_Temporal=listaGramaticas[x]
        else:
            continue
    return gramatica_Temporal
```

```
def buscar_AFD(nombreAFD):
    AFD = Automata("","","","","","")
    for x in range(0, len(listaAutomatas)):
        if nombreAFD == listaAutomatas[x].nombre:
            AFD = listaAutomatas[x]
        else:
            continue
    return AFD
```

Existe una lista general de objetos de tipo gramática y tipo autómata, para buscar cierto objeto se envía como parámetro el nombre del objeto a buscar y se crea un objeto vacío que es donde se almacenará si el objeto se encuentra, con un for se recorre la lista de objetos y si el nombre enviado como parámetro es igual al atributo nombre de algún objeto que esté en la lista, se procede a almacenar todos sus atributos en el objeto creado al inicio. Por último, se retorna el objeto encontrado.

13. Generar autómata en graphv

```
#GENERANDO GRAPHVIZ
f = Digraph('finite_state_machine', filename='fsm.gv', format="png")
f.attr(rankdir='LR', size='8,5')

f.attr('node', shape='doublecircle')
for estadoAceptacion in ObjetoGramatica.estadosAceptacion:
    f.node(estadoAceptacion)

f.attr('node', shape='circle')

for z in ObjetoGramatica.producciones:
    estadoDe=z.get_Nombre()
    for p in z.transiciones:
        f.edge(estadoDe.strip(), (p.get_Estado_Hacia()).strip(), label=(p.get_Simbolo()).strip())

f.node("invisible", style="invis")
f.edge("invisible", ObjetoAFD.estado_Inicial)

f.view()
```

Lo que realiza la vista del autómata es el ciclo for, en éste se crea una figura por cada transición que existe en la lista de transiciones que tiene un estado.

DIAGRAMA DE CLASES