PROYECTO 2

Lenguajes Formales y de Programación

Manual Técnico

2021

Nombre: Gerson Sebastian Quintana Berganza

Carné: 201908686

DESCRIPCIÓN DE LA APLICACIÓN

La aplicación fue desarrollada utilizando como lenguaje de programación *Python* y utilizando la herramienta *Graphviz* para la generación de imágenes, la cual tiene como objetivo principal profundizar acerca de los lenguajes independientes del contexto, el cual permite navegar entre diferentes opciones las cuales agrupan funciones de mucha utilidad para poder comprender mucho mejor el funcionamiento de los lenguajes libres del contexto.

El flujo de la aplicación va desde la lectura de un archivo de texto que contiene una determinada cantidad de gramáticas con cierta estructura, que con opciones de la aplicación se pueden ver desglosadas cada una de las gramáticas si así se desea. Además, con estas gramáticas ingresadas se pueden realizar un conjunto de operaciones que son el corazón de la aplicación, ya que dentro de una de las operaciones que se pueden realizar en la aplicación esta poder ingresar una cadena de texto que será reconocida por un determinado autómata y verificar si esta cadena es válida.

Además, la aplicación es capaz de generar reportes en html, estos reportes son: la representación gráfica de las iteraciones realizadas por un autómata para reconocer o no una cadena de texto; la representación gráfica de una tabla que muestra las iteraciones realizadas por el autómata para validar o no la cadena; y no reporte que muestra todas las gramáticas que no fueron cargadas a la aplicación por no ser exclusivamente independientes del contexto.

DATOS TÉCNICOS

Lenguaje de programación utilizado: Python 3.8.0

IDE utilizando: Visual Studio Code 1.48.2

Sistema operativo: Windows 10 (64 bits)

Lenguaje utilizado para la generación de reportes: HTML

Herramienta utilizada para generar grafos: Graphviz

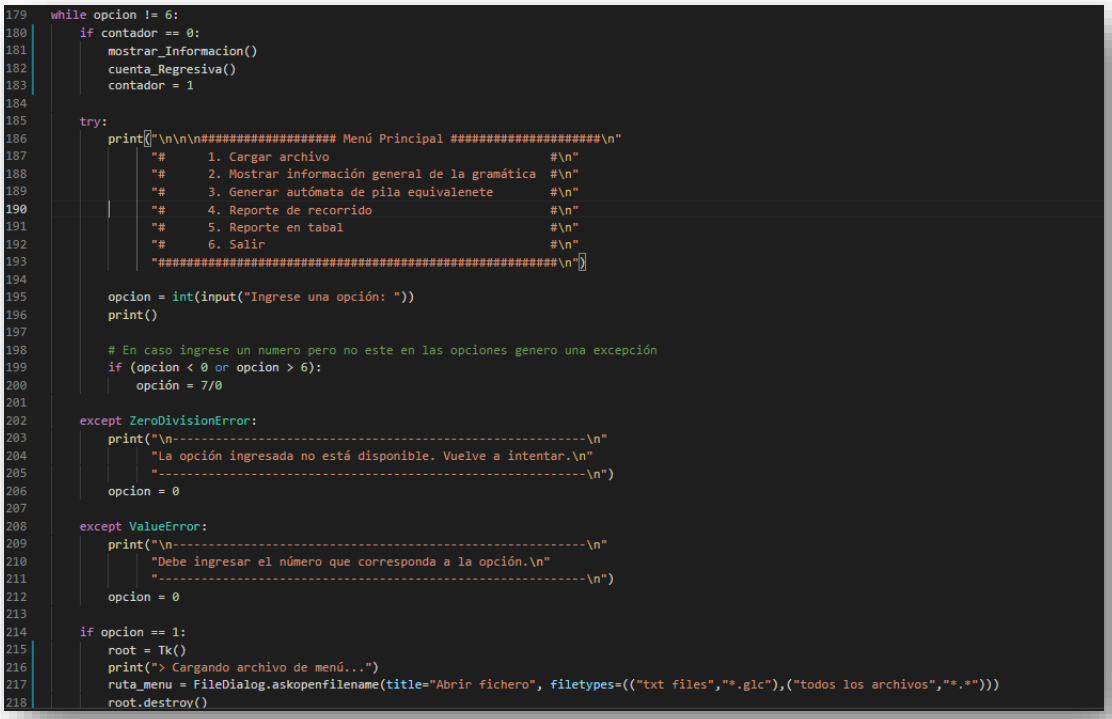
LÓGICA DEL PROGRAMA

FUNCIONES UTILIZADAS EN LOS MÓDULOS

➢ Main.py

Este módulo es el encargado de controlar el flujo de todo el programa, ya que es en donde se eligen las opciones del menú y se manejan diferentes excepciones.

Lo primero que se ejecutará en este módulo cuando el usuario ingrese será una estructura repetitiva while, la cual se ejecuta siempre y cuando el usuario no ingrese la *opción 6*. En cada ejecución se mostrará el menú de opciones.



```
while opcion != 6:
    if contador == 0:
        mostrar_Informacion()
        cuenta_Regresiva()
        contador = 1

    try:
        print("\n\n\n################### Menú Principal ####################\n"
              "#      1. Cargar archivo                              #\n"
              "#      2. Mostrar información general de la gramática  #\n"
              "#      3. Generar autómata de pila equivalenete       #\n"
              "#      4. Reporte de recorrido                        #\n"
              "#      5. Reporte en tabal                            #\n"
              "#      6. Salir                                       #\n"
              "#########################################################\n")

        opcion = int(input("Ingrese una opción: "))
        print()

        # En caso ingrese un numero pero no este en las opciones genero una excepción
        if (opcion < 0 or opcion > 6):
            opción = 7/0

    except ZeroDivisionError:
        print("\n-------------------------------------------------------------\n"
              "La opción ingresada no está disponible. Vuelve a intentar.\n"
              "-------------------------------------------------------------\n")
        opcion = 0

    except ValueError:
        print("\n-------------------------------------------------------------\n"
              "Debe ingresar el número que corresponda a la opción.\n"
              "-------------------------------------------------------------\n")
        opcion = 0

    if opcion == 1:
        root = Tk()
        print("> Cargando archivo de menú...")
        ruta_menu = FileDialog.askopenfilename(title="Abrir fichero", filetypes=(("txt files","*.glc"),("todos los archivos","*.*")))
        root.destroy()
```

- Al inicio de este ciclo y cuando la variable *contador* sea cero se ejecutarán las funciones *mostrar_Informacion() y cuenta_Regresiva().*

    - **mostrar_informacion():** Muestra la información de quien desarrollo el programa.

```python
def mostrar_Informacion():

    print("\n\n------------------------- Proyecto 2 - LFP -------------------------")
    print("Lenguajes Formales y de Programación,  sección B+")
    print("Gerson Sebastian Quintana Berganza, 201908686")
    print("--------------------------------------------------------------------\n\n")

```

- **cuenta_Regresiva():** Consta de una estructura repetitiva que imprime los número del 1 al 5 pero de forma inversa. Para esto concatena cada uno de esos números en una variable, utilizando un retorno de carro para posicionarse de nuevo al inicio de la línea y nuevamente imprimir lo que había concatenado. Así mismo, utiliza el módulo *sleep* para dar un intervalo de tiempo antes de mostrar lo concatenado nuevamente. Finalizado el ciclo muestra la palabra *¡Bienvenido!.*

```python
def cuenta_Regresiva():
    cuenta = ""
    i = 5
    while i >= 0:
        # Si ya es la ultima vez que entra al ciclo que no imprima una ',' sino '...'
        if i == 0:
            cuenta += str(i)
            print(cuenta + " ", end="...")
            sleep(0.5)
        else:
            cuenta += str(i)
            # Se hace un retorno al carro para que quite lo que estaba anteriormente y ponga lo que la variable 'cuenta' lleva concatenado
            print(cuenta + "\r", end=" ")
            cuenta += ", "
        sleep(1)
        i -= 1
    print(" ¡Bienvenido!")
    sleep(1)
```

- Al seleccionar la *opción 1*, abre una ventana en donde se debe de seleccionar el archivo de texto con extensión *.glc* con todas las gramáticas. Además, se valida que si se haya seleccionado un archivo.

  Si se llega a encontrar alguna gramática que no sea exclusivamente libre del contexto durante el análisis del archivo se mostrará en un reporte que podrá ser generado si ingresa la letra *s*. Dicho reporte se abrirá automáticamente.

```python
if opcion == 1:
    root = Tk()
    print("> Cargando archivo de menú...")
    ruta_menu = FileDialog.askopenfilename(title="Abrir fichero", filetypes=(("txt files","*.glc"),("todos los archivos","*.*")))
    root.destroy()


    if ruta != "":

        print("> Archivo cargado con éxito...")
        analizar_archivo = Analizar_Archivo.Analizar_Archivo(ruta)              # Enviando la ruta del archivo

        # Analizando el archivo
        analizar_archivo.analizar_file()
        gramaticas_no_cargadas = analizar_archivo.getGramaticas_no_cargadas()

        if gramaticas_no_cargadas != 0:
            opcion = input("> Algunas gramáticas no se cargaron.¿Desea ver el reporte? (s/n): ")
            if opcion.lower() == "s":
                os.system("reportes\\Reporte.html")

    else:
        print("\n-----------------------------------------------------------------")
        print("No se ingresó ruta de archivo.")
        print("-----------------------------------------------------------------\n")

```

- Al seleccionar la *opción 2,* se ejecuta una función con nombre *mostrar_informacion_gramaticas()*.

```python
elif opcion == 2:
    mostrar_informacion_gramaticas()
```

- **mostrar_informacion_gramaticas:** Obtiene el nombre de todas las gramáticas cargadas y almacenadas en el objeto *analizar_archivo* y genera una estructura de modo que se muestren con la siguiente estructura:

```
No terminal -> Expresión
            | Expresión
No terminal -> Expresión
No terminal -> Expresión
```

```python
def mostrar_informacion_gramaticas():

    global analizar_archivo

    # Obteniendo la lista que tiene el nombre de todas las gramaticas que estan en el archivo de entrada
    nombres_gramaticas = analizar_archivo.getNombres_Gramaticas()
    # Tamaño de la lista de nombres
    len_nombres_gramaticas = len(nombres_gramaticas)
    # Guardara lo que esta al lado derecho de cada produccion
    expresion = ""

    print("\n---------- Gramáticas Cargadas ------------\n")
    for i in range(len_nombres_gramaticas):
        print("  " + str(i+1) + ". " + nombres_gramaticas[i])
    print()

    try:

        opcion_gramatica = int(input("Ingrese el número correspondiente a la gramática: "))

        if opcion_gramatica > 0 and opcion_gramatica <= len_nombres_gramaticas:

            # Donde se guardaran las posiciones que ya fueron obtenidas
            nombre_gramatica = nombres_gramaticas[opcion_gramatica-1]

            # Buscando el nodo de la lista circular con el nombre de la gramatica
            gramatica = analizar_archivo.obtener_objeto_gramatica(nombre_gramatica)

            # Obteniendo todas las producciones de la gramatica que se acaba de buscar (devuelve una lista)
            producciones = gramatica.listaProducciones
            nombre_gramatica = gramatica.nombre
            lista_no_terminales = gramatica.lista_NoTerminales
            lista_terminales = gramatica.listaTerminales
            no_terminal_inicial = gramatica.NoTerminalInicial
            terminales = ""
            no_terminales = ""
            terminales = estructurar_Lista_Terminales(lista_terminales)
            no_terminales = estructurar_Lista_No_Terminales(lista_no_terminales)

            print("\n\nNombre de la gramática = " + nombre_gramatica)
            print("No terminales = " + "{" + no_terminales + "}")
            print("Terminales = " + "{" + terminales + "}")
            print("No terminal inicial = " + no_terminal_inicial)
            print("Producciones: \n")
```

- Al seleccionar la *opción 3,* obtiene el nombre de todas las gramáticas cargadas en el sistema y simplemente las muestra al usuario. Luego de mostrarlas le pide al usuario que ingrese el numero de la gramática de la cual desea generar el autómata de pila equivalente. Al ingresar el numero en realidad, lo que se va a enviar es nombre de la gramática que se encuentra en esa posición.

  Una vez generado el autómata, el nombre de este se guarda en una lista con nombre *automatas_de_pila*. Asimismo se generan las funciones de transición de la forma *(q,$,A; q,aAa)*.

```
245     elif opcion == 3:
246
247         nombres_gramaticas = analizar_archivo.getNombres_Gramaticas()
248         len_nombres_gramaticas = len(nombres_gramaticas)
249
250         for i in range(len_nombres_gramaticas):
251             print(str(i+1) + ". " + nombres_gramaticas[i])
252         print()
253
254         try:
255             opcion_gramatica = int(
256                 input("Ingrese el número correspondiente a la gramática: "))
257
258             if opcion_gramatica > 0 and opcion_gramatica <= len_nombres_gramaticas:
259                 nombre_gramatica = nombres_gramaticas[opcion_gramatica-1]
260                 automatas_de_pila.append(nombre_gramatica)
261                 gramatica = analizar_archivo.obtener_objeto_gramatica(
262                     nombre_gramatica)
263                 grafo = Graphviz.Graphviz(gramatica)
264                 grafo.generar_funciones()
265                 grafo.generar_grafo(True)
266
267                 print("\n----------------------------------------------------------")
268                 print("> Autómata de pila equivalente generado con éxito.")
269                 print("----------------------------------------------------------")
270
271             else:
272                 print("\n----------------------------------------------------------")
273                 print("Únicamente puede eligir entre las opciones disponibles.")
274                 print("----------------------------------------------------------")
275
276         except ValueError:
277             print("\n----------------------------------------------------------")
278             print("Debe ingresar el número que corresponde a la gramática.")
279             print("----------------------------------------------------------")
280
```

- Al seleccionar la *opción 4*, aquí toma la lista que guarda el nombre de todas las gramáticas de las cuales se generó un autómata de pila equivalente en la *opción 3* (*automatas_de_pila*) y muestra todos los nombres anteponiendo la cadena *AP_*, generando asi un lista de todos los autómatas generados en la *opción 3*. Una vez elegido el automata, se le pide al usuario ingresar una cadena que será validada.

  En la línea 297 obtiene el objeto *gramática* para que con la clase *Graphviz* generar las transiciones.

  Una vez validada la cadena se genera un reporte html con todas las transiciones representadas con imágenes de grafos generados con *Graphviz*.

```python
elif opcion == 4:

    len_automatas_de_pila = len(automatas_de_pila)
    expresion = ""

    print("\n---------- Gramáticas Cargadas ------------\n")
    for i in range(len_automatas_de_pila):
        print("  " + str(i+1) + ". AP_" + automatas_de_pila[i])
    print()

    try:

        opcion_automata = int(input("Ingrese el número correspondiente a la gramática: "))
        cadena = input("Ingrese una cadena: ")

        if opcion_gramatica > 0 and opcion_automata <= len_automatas_de_pila:

            # Donde se guardaran las posiciones que ya fueron obtenidas
            nombre_gramatica = automatas_de_pila[opcion_automata-1]
            gramatica = analizar_archivo.obtener_objeto_gramatica(nombre_gramatica)
            grafo = Graphviz.Graphviz(gramatica)
            grafo.generar_funciones()
            grafo.generar_grafo(False)
            transiciones = grafo.getTransiciones()
            no_terminal_inicial = grafo.getNoTerminalInicial()
            terminales = grafo.getTerminales()
            no_terminales = grafo.getNoterminales()


            recorrido, razon = Automata_de_Pila.analizar_Cadena(grafo, cadena, transiciones, terminales, no_terminales, no_terminal_inicial)

            if recorrido == True:   # La cadena fue valida
                file = open("reportes/Recorrido_Cadena.html","a", encoding="UTF-8")
                file.write("\n\t<div class=\"estado\">¡La cadena ingresada es válida!</div>\n\t</body>\n</html>\n")
            else:   # La cadena no fue valida
                file = open("reportes/Recorrido_Cadena.html","a", encoding="UTF-8")
                file.write("\n\t<div class=\"estado\">CADENA NO VÁLIDA.\n" + razon + "</div>\n\t</body>\n</html>\n")
            file.close()
            os.system("reportes\\Recorrido_Cadena.html")

        else:
            print("\n-------------------------------------------------------")
            print("Únicamente puede eligir entre las opciones disponibles.")
            print("-------------------------------------------------------")

    except ValueError:
        print("\n-------------------------------------------------------")
        print("Debe ingresar el número que corresponde a la gramática.")
        print("-------------------------------------------------------")
```

- Al seleccionar la *opción 5,* la lógica es básicamente la misma que en la *opción 4* ya que se muestran todos los nombres de los autómatas generados en la *opción 3* para que el usuario seleccione uno, ingrese una cadena de texto que será validada pero aquí en vez de generarse un reporte con todas las iteraciones, se muestra una tabla que muestra el número de iteración, la cadena en lectura y la transición utilizada.

  Una vez finalizado, abre automáticamente dicho reporte.

```python
elif opcion == 5:
    # Obteniendo la lista que tiene el nombre de todas las gramaticas que estan en el archivo de entrada
    nombres_gramaticas = analizar_archivo.getNombres_Gramaticas()
    # Tamaño de la lista de nombres
    len_nombres_gramaticas = len(nombres_gramaticas)
    # Guardara lo que esta al lado derecho de cada produccion
    expresion = ""

    print("\n---------- Gramáticas Cargadas ------------\n")
    for i in range(len_nombres_gramaticas):
        print("  " + str(i+1) + ". " + nombres_gramaticas[i])
    print()

    try:

        opcion_gramatica = int(input("Ingrese el número correspondiente a la gramática: "))
        cadena = input("Ingrese una cadena: ")

        if opcion_gramatica > 0 and opcion_gramatica <= len_nombres_gramaticas:

            # Donde se guardaran las posiciones que ya fueron obtenidas
            nombre_gramatica = nombres_gramaticas[opcion_gramatica-1]
            gramatica = analizar_archivo.obtener_objeto_gramatica("Grm1")  # Retorna el objeto 'gramatica'
            grafo = Graphviz.Graphviz(gramatica)
            grafo.generar_funciones()
            grafo.generar_grafo(False)
            transiciones = grafo.getTransiciones()
            no_terminal_inicial = grafo.getNoTerminalInicial()
            terminales = grafo.getTerminales()
            no_terminales = grafo.getNoterminales()

            recorrido, razon = Automata_de_Pila.analizar_Cadena_con_tabla(grafo, "abzba", transiciones, terminales, no_terminales, no_terminal_inicial)

            if recorrido == True:
                file = open("reportes/Reporte_Tabla.html", "a", encoding="UTF-8")
                fin_HTML = """</tbody>\n</thead>\n</table>\n</div>\n\t<div class=\"estado\">¡La cadena ingresada es válida!</div>\n\t"""
                file.write(fin_HTML)
            else:
                file = open("reportes/Reporte_Tabla.html", "a", encoding="UTF-8")
                fin_HTML = "</tbody>\n</thead>\n</table>\n</div>\n\t<div class=\"estado\">CADENA NO VÁLIDA.\n" + razon + "</div>\n\t"
                file.write(fin_HTML)
            file.close()

            os.system("reportes\\Reporte_Tabla.html")

        else:
            print("\n-------------------------------------------------------")
            print("Únicamente puede elígir entre las opciones disponibles.")
            print("-------------------------------------------------------")

    except ValueError:
        print("\n-------------------------------------------------------")
        print("Debe ingresar el número que corresponde a la gramática.")
        print("-------------------------------------------------------")

```

- La opción 6, lo que realiza es simplemente romper con el ciclo, haciendo que la condición que lo hace seguir ejecutándose ya no se cumpla.

➢ Lista_Cirular.py

Modulo que implementa tres clases: *Gramatica, Nodo y Lista_Circular*, en los cuales almacenará todas las partes necesarias de la gramática para realizar todas las operaciones posteriores.

- **Clase Gramática:** Esta clase únicamente cuenta con su método constructor, que va a inicializar cada gramática libre del contexto. Este método recibe como parámetros, el nombre de la gramática, todos los no terminales de la gramática, todos los terminales de la gramática, el no terminal inicial y la lista de todas las producciones.

```python
class Gramatica:
    def __init__(self, nombre=None, lista_NoTerminales=None, lista_Terminales=None, NoTerminalInicial=None, listaProducciones=None):
        self.nombre = nombre
        self.lista_NoTerminales = lista_NoTerminales
        self.listaTerminales = lista_Terminales
        self.NoTerminalInicial = NoTerminalInicial
        self.listaProducciones = listaProducciones

```

- **Clase Nodo:** Cuenta con el método constructor, el cual contendrá el objeto de la clase gramática, y un apuntador al siguiente nodo.

```python
class Nodo:
    def __init__(self, gramatica=None, siguiente=None):
        self.gramatica = gramatica
        self.siguiente = siguiente

```

- **Clase Lista_Circular:** Cuenta con tres funciones: *__init__()*, que inicializa la nueva cabecera de la lista circular y una variable *tamano*, que llevara la cuenta de cuantos nodos hay en la lista; *insertar_gramatica(),* que recibe los mismos parámetros que el constructor de la clase Gramatica; *buscar_gramatca()*, recibe como parámetro el nombre de la gramática que se quiere encontrar, retornando el objeto gramática que tiene ese nombre.

```python
class Lista_Circular:
    def __init__(self, head=None):
        self.head = head
        self.tamano = 0

    def insertar_gramatica(self, nombre_gramatica=None, lista_NoTerminales=None, lista_Terminales=None, NoTerminalInicial=None, listaProducciones=None):
        if self.tamano == 0:
            gramatica = Gramatica(nombre=nombre_gramatica, lista_NoTerminales=lista_NoTerminales, lista_Terminales=lista_Terminales, NoTerminalInicial=NoTerminalInicial, listaProducciones=listaProduccion
            self.head = Nodo(gramatica=gramatica)
            self.head.siguiente = self.head
        else:
            gramatica = Gramatica(nombre=nombre_gramatica, lista_NoTerminales=lista_NoTerminales, lista_Terminales=lista_Terminales, NoTerminalInicial=NoTerminalInicial, listaProducciones=listaProduccion
            nuevo_nodo = Nodo(gramatica=gramatica, siguiente=self.head.siguiente)
            self.head.siguiente = nuevo_nodo
        self.tamano += 1

    def buscar_gramatica(self, nombre_gramatica):
        if self.tamano == 0:
            return False

        nodo = self.head
        for i in range(self.tamano):
            if nodo.gramatica.nombre == nombre_gramatica:
                return nodo.gramatica
            nodo = nodo.siguiente

```

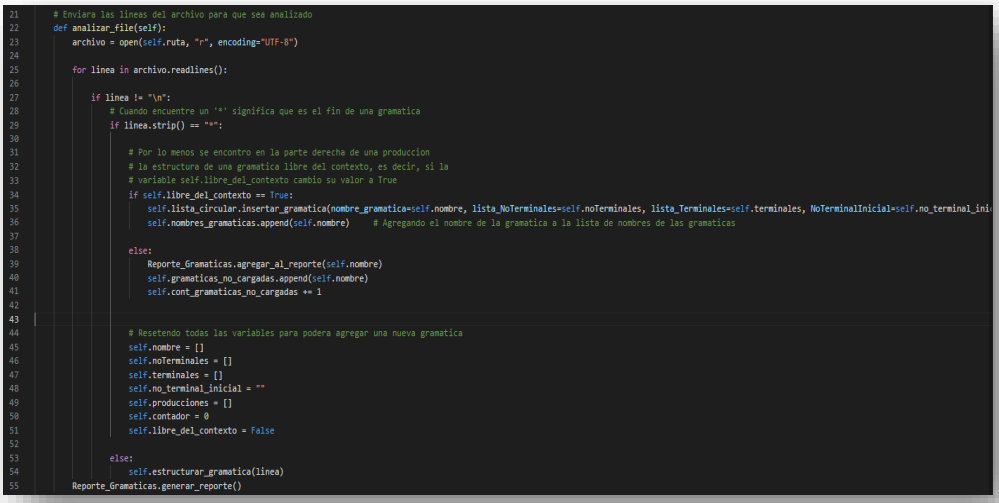
➢ Analizar_Archivo.py

Es en este modulo donde se analiza la estructura del archivo de entrada y en base a ella almacenar los datos de tal modo que puedan ser fácilmente accedidos. Cuenta con las siguientes funciones:

- **__init__():** Recibe por parámetro la ruta del archivo que contiene a todas las gramáticas y es donde además se crea el objeto de la clase *Lista_Circular*, en el cual se guardarán todas las gramáticas del archivo.

```
6     def __init__(self, ruta):
7         self.ruta = ruta    # Ruta del archivo
8         self.lista_circular = Lista_Circular.Lista_Circular()
9         self.nombre = ""    # Nombre de la gramatica
10        self.noTerminales = []  # Guarda todos los no terminales de la gramatica
11        self.terminales = []    # Guardara todos los terminales de la gramatica
12        self.no_terminal_inicial = ""   # Guardara el no terminal inicial de la gramatica
13        self.producciones = []  # Se guardaran las producciones de una gramatica
14        self.nombres_gramaticas = []    # Guardar todos los nombre de las gramaticas
15        self.contador = 0               # Para guiar que linea de la gramatica estoy
16        self.libre_del_contexto = False  # Para validar que la gramatica sea libre del contexto
17        self.gramaticas_no_cargadas = []    # Se guardara el nombre de las gramaticas que no sean libres del contexto
18        self.cont_gramaticas_no_cargadas = 0    # Llevara el numero de gramaticas no cargadas
19
```

- **analizar_file():** Abre el archivo del cual se recibió la ruta en el constructor y lo lee línea por línea siempre y cuando esta no este vacía. Lo que realiza durante este proceso es ir guardando en la lista *lista_Producciones* todas las producciones de esa gramatica, en la *lista_NoTerminales* todos los no terminales, en *lista_Terminales* todos los terminales, y el nombre de la gramática. Esto lo hace hasta que encuentra un '*', cuando lo encuentra, inserta una nueva gramática en la lista circular y 'resetea' las variables utilizadas para que nuevamente vuelva a realizar el mismo proceso.



```
# Enviara las lineas del archivo para que sea analizado
def analizar_file(self):
    archivo = open(self.ruta, "r", encoding="UTF-8")

    for linea in archivo.readlines():

        if linea != "\n":
            # Cuando encuentre un '*' significa que es el fin de una gramatica
            if linea.strip() == "*":

                # Por lo menos se encontro en la parte derecha de una produccion
                # la estructura de una gramatica libre del contexto, es decir, si la
                # variable self.libre_del_contexto cambio su valor a True
                if self.libre_del_contexto == True:
                    self.lista_circular.insertar_gramatica(nombre_gramatica=self.nombre, lista_NoTerminales=self.noTerminales, lista_Terminales=self.terminales, NoTerminalInicial=self.no_terminal_ini
                    self.nombres_gramaticas.append(self.nombre)     # Agregando el nombre de la gramatica a la lista de nombres de las gramaticas

                else:
                    Reporte_Gramaticas.agregar_al_reporte(self.nombre)
                    self.gramaticas_no_cargadas.append(self.nombre)
                    self.cont_gramaticas_no_cargadas += 1


                # Resetendo todas las variables para podera agregar una nueva gramatica
                self.nombre = []
                self.noTerminales = []
                self.terminales = []
                self.no_terminal_inicial = ""
                self.producciones = []
                self.contador = 0
                self.libre_del_contexto = False

            else:
                self.estructurar_gramatica(linea)
    Reporte_Gramaticas.generar_reporte()
```

- **estructurar_gramatica():** Esta función recibe por parámetro la línea del archivo en lectura en el base al valor almacenado en la variable *contador,* realiza una determinada tarea.

    1. Si el contador es igual a 1, almacena en la variables *nombre* el nombre de la gramática, ya que es lo primero que está.
    2. Si el contador es igual a 2, almacena la lista de terminales, no terminales y el no terminal inicial, ya que esta es la segunda parte de la gramática.
    3. Si no es ni 1 ni 2, entonces significa que lo que va a estar leyendo son todas las producciones de la gramática.

    Y este proceso continuo hasta encontrar un ‘*’ en el archivo el cual indica el fin de la gramática, y el contador vuelve a 0.

    Además, valida que la gramática sea libre del contexto, ya que la cantidad de terminales y de no terminales era 1 en todas las expresiones del lado derecho de la producción significa no es exclusivamente independiente del contexto, o si hubo como máximo un terminal y ningún terminal en la expresión del lado derecho de la producción o si no hubo ningún terminal o no terminal. Todos los casos anteriores hacen que esa gramática no se cargue al sistema.

```python
# Va a ir guardando las listas necesarias para guardar una gramatica
def estructurar_gramatica(self, linea):
    if self.contador == 0:              # Estoy leyendo la primera linea de la estructura de la gramatica: LEER EL NOMBRE DE LA GRAMATICA
        self.nombre = linea.strip()
        self.contador += 1

    elif self.contador == 1:            # Estoy leyendo la segunda linea de la estructura de la gramatica: LEER 'NO TERMINALES', 'TERMINALES' Y 'NO TERMINAL INICIAL'

        lista = linea.split(";")        # Separandolos por el ';'
        # Se creara un lista con 3 posiciones
        lista[0] = self.limpiar_Cadena(lista[0])
        self.noTerminales = lista[0].split(",")
        lista[1] = self.limpiar_Cadena(lista[1])
        self.terminales = lista[1].split(",")
        self.no_terminal_inicial = lista[2].strip()
        self.contador += 1

    else:                               # Estoy leyendo las lineas que siguen: LEER PRODUCCIONES
        self.producciones.append(linea.strip())

        lista = linea.strip().split("->")
        # En la segunda posicion de 'lista' contendra las expresiones que se producen

        # Aqui sera de validar que la produccion sea de una gramatica libre de contexto
        contador_terminales = 0
        contador_no_terminales = 0
        derecha_produccion = lista[1]

        for i in range(len(derecha_produccion)):
            caracter = derecha_produccion[i]
            if caracter != " " and caracter != "\t":
                if caracter in self.terminales:
                    contador_terminales += 1
                elif caracter in self.noTerminales:
                    contador_no_terminales += 1

        # Los siguientes 'if' solo son de las dos formas en que puede el lado derecho de una
        # produccion de una gramatica regular, por lo que simplemente no se hace nada, ya que solo se cambiara
        # cuando encuentre una estructura de una gramatica libre del contexto
        if contador_terminales == 1 and contador_no_terminales == 1:    # Significa que esta produccion no pertenece a una gramatica regular
            print("", end="")
        elif contador_terminales == 1 and contador_no_terminales == 0:  # Significa que esta produccion no pertenece a una gramatica regular
            print("", end="")
        elif contador_terminales == 0 and contador_no_terminales == 0:  # Significa que no se reconocio ningun terminal o no terminal definido al principio de la gramatica
            # print("Los caracteres no forman parte del alfabeto")
            # print(self.nombre)
            # print(caracter)
            print("", end="")
        else:
            self.libre_del_contexto = True                          # Significa que es una gramatica libre del contexoto
```

- **getNombres_Gramaticas():** Retorna el nombre de todas las gramáticas cargadas en el sistema.

```python
110     def getNombres_Gramaticas(self):
111         return self.nombres_gramaticas
```

- **obtener_objeto_gramatica():** Recibe el nombre de la gramática que se desea encontrar y retorna el objeto gramática.

```python
110     def getNombres_Gramaticas(self):
111         return self.nombres_gramaticas
```

- **limpiar_Cadena():** Recibe por parámetro una cadena y lo que realiza es eliminar los espacios en blanco o tabulaciones de la cadena, retornando así, una cadena en donde todo este ‘junto’, sin espacios.

```python
    # Limpia la cadena enviada para que quede sin espacios o tabulaciones
    def limpiar_Cadena(self, cadena):
        cadenaLimpia = ""
        for i in range(len(cadena)):
            if cadena[i] != " " and cadena[i] != "\t":
                cadenaLimpia += cadena[i]
        return cadenaLimpia
```

- **getGramaticas_no_cargadas():** Retorna el nombre de todas las gramáticas no cargadas en el sistema.

```python
    # Si hay gramaticas que no se cargaron, entonces de retornará la lista de los nombres de esas gramaticas, sino solo un 0
    def getGramaticas_no_cargadas(self):
        if self.cont_gramaticas_no_cargadas != 0:
            return self.gramaticas_no_cargadas
        return 0
```

➢ Automata_de_Pila

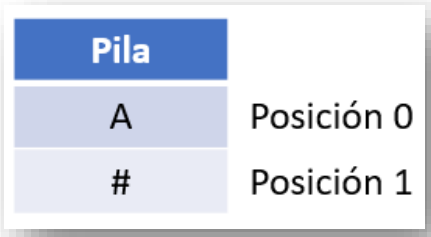
En este módulo se implementa un autómata de pila capaz de reconocer cadenas que se derivan de las gramáticas más comunes.

- **analizar_Cadena():** Recibe por parámetro un objeto de la clase *Graphviz* (*obj_grafo*), la cadena a reconocer, las transiciones, los terminales, no terminales y el no terminal inicial. Esta función trabaja en base a estados, en donde los estas ‘i’ y ‘p’ nunca cambian, mientras que es en el estado ‘q’ en donde dependiendo del carácter en lectura y lo que esta en la cima de la pila, el automata actuara de forma distinta.

  Para este autómata se consideraron los siguientes casos:

  1. Caso 1: Si en la cima de la pila (posición 0) hay no terminal y en la posición 1 está el símbolo ‘#’ y aun no se termina de leer la cadena, entonces, se busca una producción que en el lado derecho tenga un dos o más terminales y no terminales que empiecen con el carácter que estoy leyendo, en este caso ‘a’.

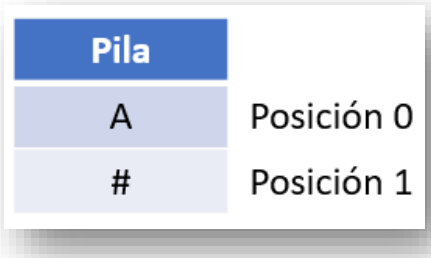
     Cadena en lectura: aabz




```
# CASO 1: La pila esta vacia (tiene #) pero son se ha llegado al final de la cadena
if pila[1] == "#" and n + 1 != len(cadena):    # Si se cumple deberia de reemplazar por la expresion mas 'grande' o con el que tenga un no terminal despues del terminal de su izquierd
    count = 0

    # Recorriendo todas las posiciones guardadas en la lista 'pos_inicioTerminal'
    for posicion in pos_inicioTerminal:  # Recorriendo todas las posiciones en que hay producciones que empiezan con el caracter que se esta leyendo

        # Convieriendo a lista la expresion que esta en la esa posicion
        produccion = producciones[posicion]
        lista_expresion = str_a_lista(produccion)

        # Va a acceder solo cuando hayan dos o mas terminales o no terminales en la expresion del lado derecho (si es asi, se asume que en la posicion 1 hay no terminal)
        # Esto porque hay que elegir la expresion que sea más 'grande' o que contenga por lo menos un no terminal, para seguir generando producciones
        if len(lista_expresion) != 1:

            #if lista_expresion[1] in no_terminales:
            transicion = "λ, " + pila[0] + "; " + produccion
            objeto_grafo.realizarRecorrido("q", char, pila, transicion, 1)
            pila.pop(0)
            pila = lista_expresion + pila
            break
```

2. Caso 2: Si en la cima de la pila hay un no terminal y en la posición 1 esta el símbolo ‘#’ y la cadena en lectura es ‘a’, entonces que busque producciones del no terminal de la cima que solo produzcan el carácter en lectura, en este caso la letra ‘a’.

   Cadena en lectura: a



```
# CASO 2: La pila esta vacia (tiene en la posicion 0 a 'S' y en la 1 a '#') y estoy en el ultimo caracter de la cadena
elif pila[1] == "#" and n + 1 >= len(cadena):

    # Recorriendo todas las posiciones guardadas en la lista 'pos_inicioTerminal'
    for posicion in pos_inicioTerminal:

        # Convieriendo a lista la expresion que esta en la esa posicion
        produccion = producciones[posicion] # Obteniendo la expresion del lado derecho de la produccion
        lista_expresion = str_a_lista(produccion)


        # Debido a que solo falta leer un caracter, lo debo reemplazar por la expresion que tenga longitud 1, es decir, ya que solo se encontraron
        # las expresiones que iniciaban con el caracter que estoy leyendo, entonces en este caso la lista solo va a tener el caracter que se esta leyendo
        if len(lista_expresion) == 1:   # Solo tiene un elemeto de la lista
            transicion = "λ, " + pila[0] + "; " + produccion
            objeto_grafo.realizarRecorrido("q", char, pila, transicion, 1)
            pila.pop(0)
            pila = lista_expresion + pila
            break
```

3. Caso 3: Si hay un no terminal en la cima de la pila, hay más contenido en la pila, pero la cadena aún no se lee por completo, entoces, hay que sustituir ese no terminal por una producción que genere una producción que tenga solo que carácter que se está leyendo.

   Cadena: aabz

Pila	
A	Posición 0
a	Posición 1
b	Posición 2
c	Posición 3
#	Posición 4

```
# Si el elemento posterior a la cima de pila no es un '#' y no estoy leyendo el ultimo caracter de la cadena
elif pila[1] != "#" and n + 1 != len(cadena):

    contador = 0

    # Recorriendo todas las posiciones guardadas en la lista 'pos_inicioTerminal'
    for posicion in pos_inicioTerminal:  # Recorriendo todas las posiciones de las producciones que tiene al inicio de la produccion el caracter que estoy leyendo (Ej. A -> z B ó C ->

        contador = contador + 1

        # Convieriendo a lista la expresion que esta en la esa posicion
        produccion = producciones[posicion]
        lista_expresion = str_a_lista(produccion)

        # Los siguientes casos se aplican cuando se encuentran dos o mas producciones que inician con el caracter que estoy leyendo
        if len(pila) - 1 == len(cadena) - n:    # Si en la pila tengo la misma cantidad de elementos (sin contar el '#') que el tamaño del resto de la cadena que estoy leyendo. Ej. pi

            # Cuando se presenta este caso, solamente se reemplaza el no terminal con la produccion que tenga como expresion solo cadena que estoy leyendo, es decir, cuando la lista d
            # Ya que ya esta implicito que sea el caracte que se esta leyendo
            if len(lista_expresion) == 1:   # Si la lista de expresiones unicamento tienen el elemento que estoy leyendo, lo debo de insertar
                transicion = "λ, " + pila[0] + "; " + produccion
                objeto_grafo.realizarRecorrido("q", char, pila, transicion, 1)
                pila.pop(0)
                pila = lista_expresion + pila
                break

        # Si lo que lo que la cima tiene justo abajo es igual al siguiente caracter de la cadena,
        # entonces cuando encuentre la produccion que tenga como inicial de la expresion el caracter que se esta leyendo y como siguinete de la pila y cadena un mismo caracter,
        # entonces se puede remplazar el no terminal con el la produccion que genera la expresion que solo tiene el caracter que se esta leyendo
        elif pila[1] == cadena[n+1]:

            bandera = False # Si toma el valor de True, significa que dentro de la pila si hay un no terminal, sino, no hay no no terminal
            for i in range(1, len(pila)):   # Buscando si hay no terminal dentro de la pila
                if pila[i] in no_terminales:
                    bandera = True
                    break

            # Si en el lado derecho de la produccion solo esta el caracter que estoy buscando y ademas dentro de la pila hay un no terminal,
            # esa me da la confianza de poder reemplazarlo por un terminal para asegurarme de que la cadena se siga leyendo, ya que ese no terminal generara mas producciones
            if len(lista_expresion) == 1 and bandera == True:
                transicion = "λ, " + pila[0] + "; " + produccion
                objeto_grafo.realizarRecorrido("q", char, pila, transicion, 1)
                pila.pop(0)
                pila = lista_expresion + pila
                break
```

- **buscar_transicion():** Recibe por parámetro el no terminal y la producción del cual se quiere encontrar sus transiciones, retornado la transición encontrada.

```
693  def buscar_transicion(no_terminal, produccion):
694      global transiciones_i
695
696      for i in range(len(transiciones_i)):
697
698          if transiciones_i[i][1] == no_terminal and transiciones_i[i][2] == produccion:
699              return transiciones_i[i]
700
```

- **buscar_producciones_con_no_terminal():** Recibe por parámetros el no terminal del cual se desea buscar sus producciones y el carácter con el que debe empezar esa producción. Retorna todas las producciones que empiecen con ese carácter.

```
671  # Buscara todas las producciones en donde el caracter al prinicipio de la expresion sea igual al caracter que se esta leyendo
672  # Ejemplo: caracter en lectura: a; S -> a B ó  S -> a M a
673  def buscar_producciones_con_caracter(no_terminal, caracter):
674
675      global transiciones_i
676      producciones = []
677
678      for i in range(len(transiciones_i)):
679
680          if transiciones_i[i][1] == no_terminal:
681
682              expresion = transiciones_i[i][2]
683
684              if expresion[0] == caracter:
685                  producciones.append(transiciones_i[i][2])
686
687      return producciones
```

- **str_a_lista():** Recibe la cadena que se quiere convertir a lista.

```python
def str_a_lista(cadena):

    cad = ""
    lista = []

    for i in range(len(cadena)):

        if cadena[i] == " " or cadena[i] == "\t":

            if cad != "":
                lista.append(cad)
                cad = ""

        else:
            cad += cadena[i]

        if i == len(cadena) - 1 and cad != "":
            lista.append(cad)

    return lista
```

➢ Graphviz.py

Realiza toda la estructura para que pueda ser representado en un grafo todas las iteraciones.

- **__init__():** Recibe por parámetro el objeto gramática y obtiene de este toda la información necesaria para generar el grafo.

```python
    # Recibe el objeto gramatica
    def __init__(self, gramatica):
        self.gramatica = gramatica
        self.listaTerminales = gramatica.listaTerminales
        self.listaNoTerminales = gramatica.lista_NoTerminales
        self.transisiones = []
        self.transisiones_automata = []
        self.pila_original = []
        self.count = 0
```

- **generar_funciones():** Genera la estructura de todas las transiciones de la forma: (q,$,A; q,aAa).

```python
    # GENERA LAS TRANSICIONES QUE ESTARAN EN LAS ARISTAS DE CADA ESTADO
    def generar_funciones(self):
        producciones = self.gramatica.listaProducciones
        lista_terminales = self.gramatica.listaTerminales
        lista_no_terminales = self.gramatica.lista_NoTerminales

        posiciones = []

        for i in range(len(producciones)):
            produccion = producciones[i]
            lista = produccion.split("->")
            izquierda_produccion = lista[0].strip()
            derecha_produccion = lista[1].strip()

            cadena = ""
            estado = 0
            pos = 0

            lista = ["q,λ," + izquierda_produccion, "q," + derecha_produccion]
            self.transisiones.append(lista)

        for j in range(len(lista_terminales)):
            terminal = lista_terminales[j]
            lista = ["q," + terminal + "," + terminal, "q,λ"]
            self.transisiones.append(lista)

```

- **generar_grafo():** Recibe por parámetro una variable por nombre ‘imprimir’, la cual, al tener el valor de True, generará el grafo y retornara la imagen para se colocada en el reporte html, si es False, únicamente genera la imagen pero no retorna nada.

```python
    def generar_grafo(self, imprimir):

        nombre_grafo = "AP_" + self.gramatica.nombre + ".dot"
        f = Digraph('grafo', filename=nombre_grafo, node_attr={'height': '1.1', 'fontsize':"18"}, format='png')
        f.attr(rankdir="LR", diredgeconstraints="True")

        no_terminal_inicial = self.gramatica.NoTerminalInicial

        f.attr('node', shape="none", fontcolor='black')
        f.node('')
        f.attr('node', shape='circle')
        f.node('i', fontsize='30')
        f.node('p', fontsize='30')
        f.attr('node', shape='doublecircle')
        f.node('f', fontsize='30')
        f.attr('node', shape='circle')
        f.edge('', 'i', label="")
        f.edge('i', 'p', label='λ , λ; #')
        f.edge('p', 'q', label='λ , λ; '+ "S")

        etiqueta_NT = ""
        etiqueta_T = ""
```

➢ Recorrido_Cadena.py

Genera la estructura de html en donde se mostrará todo el recorrido para validar una cadena ingresada por el usuario.

```python
def realizar_iteracion(ruta_imagen, contenido_pila, entrada, comenzar):
    global contador

    inicio_HTML = ""
    fin_HTML = ""
    inicio_body = ""
    fin_body = ""
    titulo = ""

    ruta_imagen = ruta_imagen.replace("\\", "/")

    if comenzar == 0:
        contador = 0
        inicio_HTML = """<!DOCTYPE html>
    <html lang="en">

    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Reporte de recorrido</title>
        <link rel="stylesheet" href="css/estilos_recorrido.css">
        <!-- Bootstrap CSS -->
        <link rel="stylesheet" href="css/bootstrap.min.css">
    </head>\n"""

        fin_HTML = "</html>\n"
        inicio_body = "<body>\n"
        fin_body = "</body>\n"
        titulo = """<div class="title">
    <h1>REPORTE DEL RECORRIDO</h1>
</div>"""
        file = open("reportes/Recorrido_Cadena.html", "w", encoding="UTF-8")
        file.write(" ")
        file.close()

```

➢ Reporte_Tabla.py

Genera la estructura de html en donde se mostrará todo el recorrido para validar una cadena ingresada por el usuario, pero esta vez representado en una tabla.

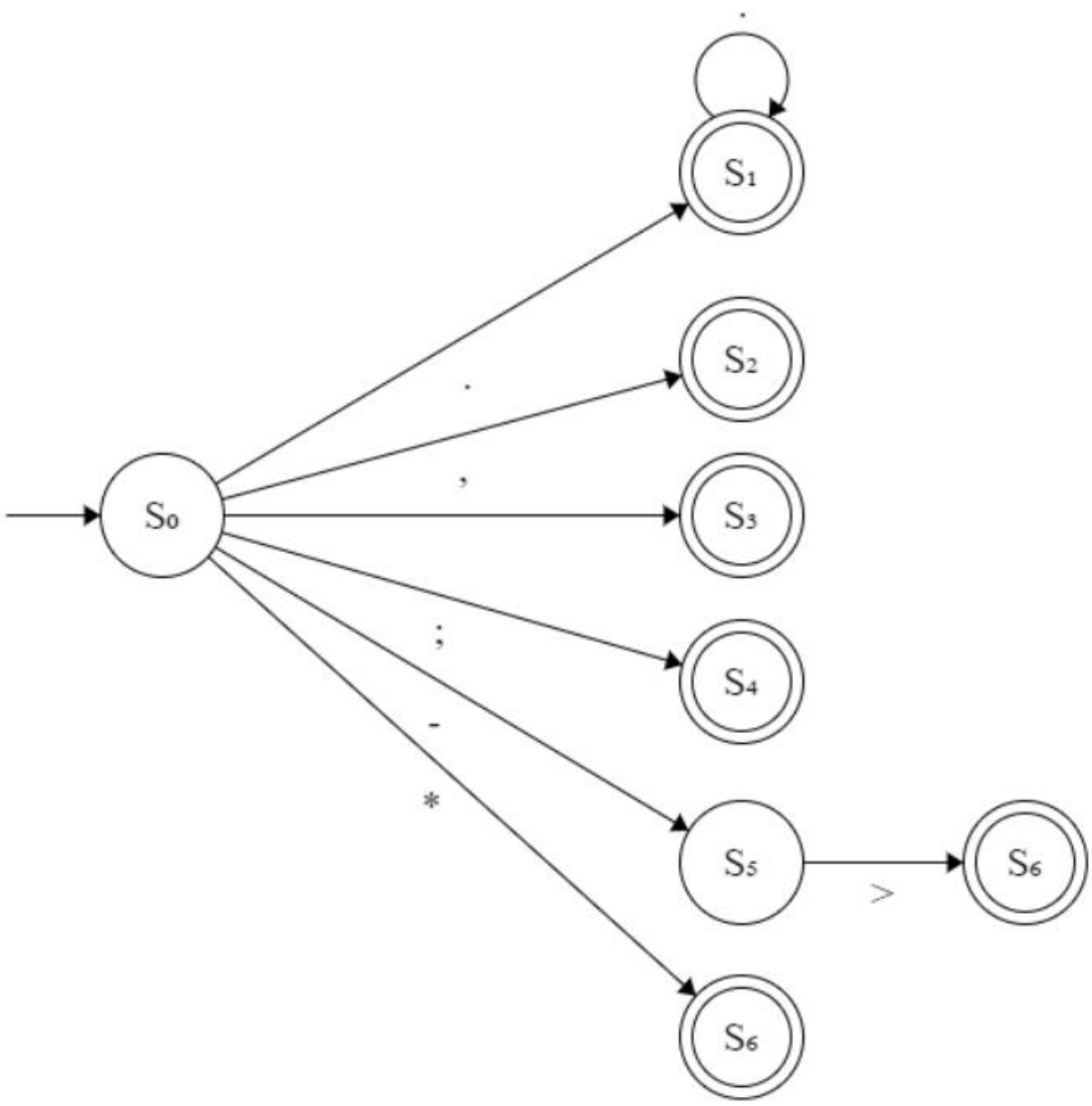
```python
class Reporte_en_Tabla():

    def __init__(self):
        self.contador = 0

    def realizar_fila(self, contenido_pila, entrada, transicion, cadena):
        inicio_HTML = ""
        fin_HTML = ""
        inicio_body = ""
        fin_body = ""
        inicio_tabla = ""
        titulo = ""

        if self.contador == 0:
            inicio_HTML = """<!DOCTYPE html>
    <html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Reporte en tabla</title>
        <link rel="stylesheet" href="css/bootstrap.min.css">
        <link rel="stylesheet" href="css/estilos_reporte_tabla.css">
    </head>\n"""
            titulo = """<div class="title">
    <h1>REPORTE EN TABLA</h1>
</div>"""

            fin_HTML = "</html>\n"
            inicio_body = "<body>\n"
            fin_body = "</body>\n"
            inicio_tabla = """<div class="contenido">
            <table class="table table-hover">
                <thead class="table-dark">
                    <tr>
                        <th>Iteración</th>
                        <th>Resto de la cadena</th>
                        <th>Pila</th>
                        <th>Entrada</th>
                        <th>Transiciones</th>
                    </tr>
                    <tbody>"""

            file = open("reportes/Reporte_Tabla.html", "w", encoding="UTF-8")
            file.write(" ")
            file.close()
```

➢ Reporte_Gramaticas.py

Genera la estructura de html en donde se mostrará una tabla de todas las gramáticas que no fueron cargadas en el sistema al no ser independientes del contexto.

```python
def agregar_al_reporte(nombre_gramatica):
    global row
    razon = "No posee ninguna producción que la convierta exclusivamente en libre del contexto."
    row += f"""<tr>
                    <td>{nombre_gramatica}</td>
                    <td>{razon}</td>
                </tr>\n"""

def generar_reporte():
    global row
    i_html = """<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="css/bootstrap.min.css">
    <title>Document</title>
    <link rel="stylesheet" href="css/estilos.css">
</head>
<body>\n"""

    f_html = """</body>
</html>\n"""

    tabla = f"""<div class="container">
        <h1>Reporte de Gramáticas No Aceptadas</h1>
        <table class="table table-hover">
            <thead class="table-dark">
                <th width="400">Nombre de la gramática</th>
                <th>Razón</th>
            </thead>
            <tbody>
                {row}
            </tbody>
            </table>
        </div>\n"""

    HTML = i_html + tabla + f_html
    file = open("reportes/Reporte_Gramaticas.html", "w", encoding='UTF-8')
    file.write(HTML)
    file.close()
    row = ""
```

AUTÓMATA FINITO DETERMINISTA



El autómata puede recibir un carácter ‘n’ número de veces y este sería aceptado, asi como también un ‘,’, ‘;’, ‘*’ y una secuencia que forme ‘->’, los cuales estarán es un estado de aceptación, con esto se valida que el archivo tenga esto y únicamente esto para la lectura del archivo.