

Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Escuela de Ciencias y Sistemas

Lenguajes Formales y de Programación

Catedrática:

Inga. Asunción Mariana Sic Sor

Tutor académico:

Enrique Alejandro Pinula Quiñonez



“MANUAL TECNICO PROYECTO 1”

Melvin Eduardo Ordoñez Sapón RA | 3256215860802

Guatemala, junio del 2024

“Id y enseñad a todos”

INDICE

Contenido

Universidad de San Carlos de Guatemala	1
"MANUAL TECNICO PROYECTO 1"	1
INDICE.....	2
LOGICA DE LA SOLUCION	4
Análisis léxico	4
AFD	5
Almacenamiento de cada token	6
Reconocimiento de la cantidad de grafos.....	6
Extracción de la información que describe cada grafo	6
Creación de grafos a partir de la información obtenida en el archivo de entrada	7
CLASES Y METODOS IMPORTANTES.....	7
Métodos importantes	7
SINTAXIS DEL ARCHIVO DE ENTRADA	8

ACERCA DEL PROGRAMA:

IDE utilizado: Visual Studio Code

Versión de Python: Python 3.12.3

Programado en Sistema Operativo: Windows 11 Pro

Terminal Utilizada: Git Bash

Lenguaje de Programación Utilizado: Python

Compatibilidad: Windows, Linux, macOS

Librerías Utilizadas: Tkinter, Graphviz

Versión Del Programa: 1.0

LOGICA DE LA SOLUCION

La lógica que describe el comportamiento del programa a través de un archivo de entrada es el siguiente:

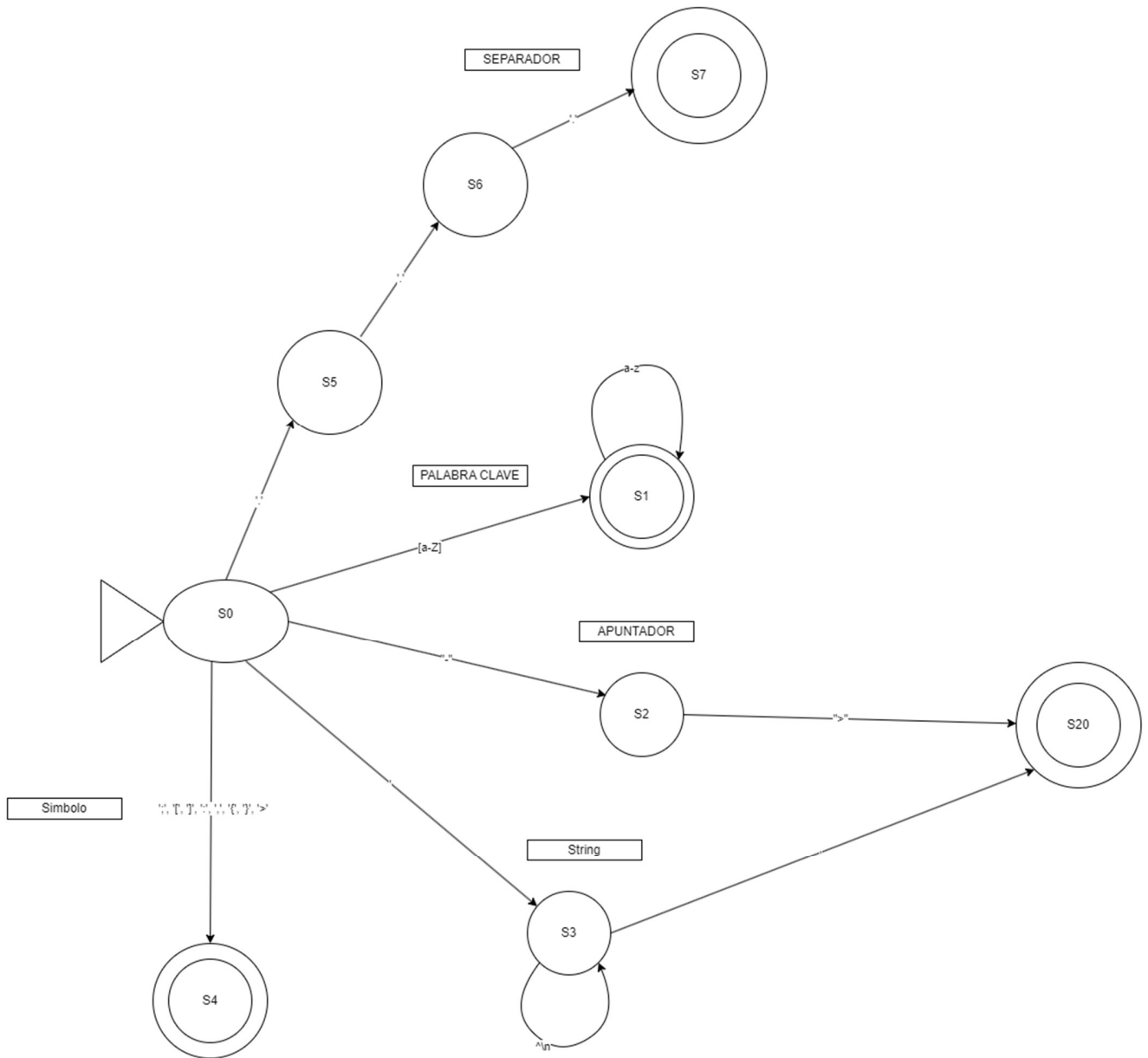
1. Análisis léxico
2. Almacenamiento de cada token encontrado
3. Reconocimiento de la cantidad de grafos encontrados
4. Extracción de la información que describe cada grafo
5. Creación de grafos a partir de la información obtenida en el archivo de entrada

Análisis léxico

En la parte del análisis léxico el programa se basa en una serie de estados en donde cada uno representa un carácter válido dentro de nuestro lenguaje especificado en el archivo de entrada y así poder identificar cuáles son los tokens válidos e identificar los errores léxicos.

Para el AFD se utilizaron estados de transición y estados de aceptación en donde se toma como válido el lexema formado, el Autómata que describe el comportamiento de los estados es el siguiente:

AFD



Almacenamiento de cada token

Para el almacenamiento de los tokens el lexer se encarga de ir almacenando en un arreglo que tokens son válidos, al momento de reconocer un token como valido se guarda y el estado regresa al inicio para analizar el carácter siguiente en la cola del archivo de entrada.

Reconocimiento de la cantidad de grafos

Para reconocer la cantidad de grafos que contiene el archivo de entrada se utiliza una lista que almacena una sublista de tokens válidos, cada uno separado por el símbolo que indica que hay mas de un grafo (...).

Extracción de la información que describe cada grafo

Para reconocer la cantidad de nodos y conexiones que tiene un grafo descrito en el archivo de entrada, se utiliza una lista que contiene dos sublistas: una que contiene la información de los nodos y otra que contiene la información de las conexiones. Esta información se obtiene a través de métodos específicos encargados de extraer cada tipo de dato. (Ver sección de métodos importantes)

Creación de grafos a partir de la información obtenida en el archivo de entrada

Para crear un grafo el programa utiliza la librería graphviz que es la encargada en reconocer cada nodo y sus respectivas conexiones a través de un método pasa la información de los nodos y conexiones a lenguaje DOT. (Ver sección de métodos importantes)

CLASES Y METODOS IMPORTANTES

La clase encargada de crear la interfaz grafica y conectarla con los métodos correspondientes se encuentra en el archivo ‘VentanaAplicacion.py’ en donde se crea la GUI utilizando la librería Tkinter

Métodos importantes

Los métodos principales del programa son los siguientes:

analizar:

Este método recibe el texto que se le pasa al programa a través de un archivo de entrada y analiza cada carácter del texto recibido para irlos clasificando como tokens validos o errores léxicos (Método en la clase Lexer).

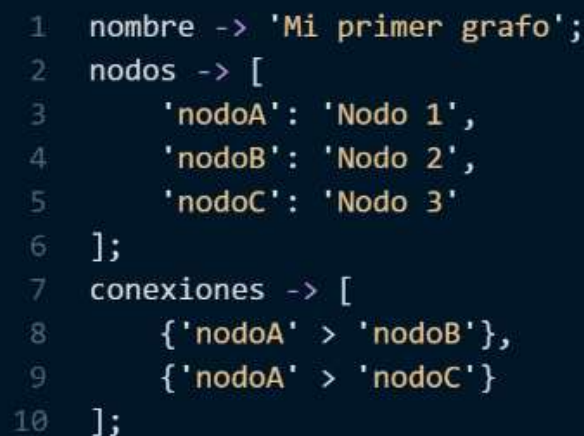
escanear_tokens:

Este método es el que se encarga de extraer la información de los nodos y conexiones de un grafo encontrado en el archivo de entrada, utiliza los métodos de extraer_nodos y extraer_conexiones para lograr este objetivo y almacenar los grafos ya procesados en un arreglo listo para ser convertida en imagen

mostrar_grafo:

Este método recibe como parámetro un índice dentro del rango de imágenes procesadas y lo convierte a lenguaje DOT utilizando la información del arreglo solicitado y luego se muestra el resultado en un label que puede ser visualizado en la interfaz gráfica.

SINTAXIS DEL ARCHIVO DE ENTRADA



```
1 nombre -> 'Mi primer grafo';
2 nodos -> [
3     'nodoA': 'Nodo 1',
4     'nodoB': 'Nodo 2',
5     'nodoC': 'Nodo 3'
6 ];
7 conexiones -> [
8     {'nodoA' > 'nodoB'},
9     {'nodoA' > 'nodoC'}
10  ];
```