Universidad Tecnológica de la Mixteca

Ingeniería en Computación

Compiladores

Práctica:

Reporte de la implementación del algoritmo de Thompson e interfaz gráfica en Python

Profesor:

Ing. David Martínez Torres

Grupo:

502-A

Integrantes del equipo:

Alan Yael Lopez Rojas

Elorza Velásquez Jorge Aurelio

García Ramírez Fernando

Martinez Lorenzo Frida Ximena

Ángel de Jesús Alvarado Torres

José Guadalupe Robles Jiménez

Fecha de entrega:

20 / 10 / 2023

Índice:

Introducción:

Con el objetivo de la realización de la implementación del algoritmo de thompson en un IDE de desarrollo a nuestra elección, logramos representar a los autómatas finitos no deterministas (AFN) los cuales a diferencia de los AFD, no tienen restricciones en cuanto a las etiquetas de sus líneas. Un símbolo puede etiquetar a varias líneas que surgen del mismo estado, y ϵ, la cadena vacía, es una posible etiqueta. Así mismo, lograr lo primordial del proyecto el de definir la metodología del algoritmo de McNaughton Yamada Thompson para convertir una expresión regular en un AFN.

Desarrollo:

Este proyecto constó de varias etapas desde una metodología ágil para la planeación, búsqueda de requerimientos, análisis, diseño, evolución y verificación, para finalmente llegar a una entrega.

Mediante la metodología SCRUM logramos encontrar los principios para una sinergia que resultara en la eficacia, eficiencia y efectividad de nuestro equipo de desarrollo, mediante la adquisición de las siguientes. Durante estos días para el desarrollo del proyecto se realizaron los sprints correspondientes a la planeación (Sprint Planning Meeting) de las siguientes actividades:

Comenzando con nuestras actividades previas correspondientes, comenzamos con una pequeña presentación del equipo, al término de esta reunión se optó por elegir un Scrum Master, nuestra compañera líder del equipo Frida Martinez, para acto seguido hacer nuestro **Backlog del producto:**

- Exploración del algoritmo de Thompson.
- Análisis de AFN y expresiones regulares.
- Elección del IDE de desarrollo y paradigma de programación a usar.
- Indagación sobre el diseño de Interfaces en Python y manejo de archivos.
- Diseño del AFN en Python y Creación de funciones para el algoritmo de Thompson, cerradura de kleene, cerradura positiva, cerradura opcional, etc.
- Unión de las funciones del algoritmo de Thompson.
- Correcciones para la interfaz gráfica.
- Union del código con la interfaz gráfica.

A partir de nuestra segunda reunión con el equipo de desarrollo, comenzamos con los Daily Scrum, aprendiendo a ser autoorganizados, multifuncionales y cooperativos, donde en cada reunión aprendíamos un poco sobre esta metodología y a madurar en sí, bajo los tres pilares empíricos de Scrum que son la transparencia, la inspección y la adaptación.

Primer Sprint

La primera tarea sobre la exploración del algoritmo de Thompson, constó primero en comprender las instrucciones para diseñar el algoritmo de Thompson definido por todos como:
Un método utilizado para convertir una expresión regular en un Autómata Finito No Determinista (AFN). Este algoritmo es un paso fundamental en la construcción de analizadores léxicos y en la implementación de motores de búsqueda de patrones, entre otros sistemas relacionados con el procesamiento de lenguaje y la búsqueda de cadenas de texto. A continuación mencionaremos los pasos del algoritmo:

Definición de estructuras básicas:
Se definen tres tipos de nodos: nodo inicial, nodo final y nodo de transición.
Se define el alfabeto de entrada (el conjunto de símbolos que se pueden reconocer en la expresión regular).

Construcción del AFN para símbolos individuales:
Para cada símbolo en la expresión regular, se crea un AFN básico.
El AFN básico consta de dos nodos (inicial y final) y una transición entre ellos etiquetada con el símbolo.

Unión de AFN (Operador '|'):
Para expresiones regulares que utilizan el operador '|' (unión), se crean dos AFN básicos y se conecta un nuevo nodo inicial a los nodos iniciales de los AFN originales mediante transiciones nulas (ε), y se conecta un nuevo nodo final a los nodos finales de los AFN originales también mediante transiciones nulas (ε).

Clausura de Kleene (Operador '*'):
Para expresiones regulares que utilizan el operador '*' (cero o más repeticiones), se toma el AFN original y se agrega un nuevo nodo inicial y un nuevo nodo final. Luego, se conecta el nuevo nodo inicial al nodo inicial del AFN original y se conecta el nodo final del AFN original al nuevo nodo final, además de agregar transiciones nulas (ε) entre el nuevo nodo inicial y el nuevo nodo final.

Concatenación de AFN:
Para concatenar dos expresiones regulares, se toma el AFN del primer término y se conecta el nodo final al nodo inicial del segundo término mediante una transición nula (ε).

Clausura Positiva (Operador '+'):
Para expresiones regulares que utilizan el operador '+' (una o más repeticiones), se toma el AFN original, se agrega un nuevo nodo inicial y un nuevo nodo final y se conecta el nuevo nodo inicial al nodo inicial del AFN original. Luego, se conecta el nodo final del AFN original al nuevo nodo final, y se agrega una transición nula (ε) entre el nodo final del AFN original y el nuevo nodo inicial.

AFN resultante:
El resultado final será un AFN que representa la expresión regular completa.

Sin embargo para agilizar el proceso en este mismo Sprint se estudió los AFN y expresiones regulares llegando a lo siguiente:
Un **AFN (Autómata Finito No Determinista)** es un modelo teórico en la teoría de la computación que representa un tipo de autómata con estados y transiciones que puede aceptar o rechazar cadenas de caracteres siguiendo ciertas reglas, permitiendo transiciones no deterministas.

Una **expresión regular** es una secuencia de caracteres que define un patrón de búsqueda en cadenas de texto. Se utiliza en la construcción de analizadores léxicos en la materia de compiladores para identificar y reconocer tokens o lexemas en un programa fuente, como palabras clave, identificadores, números, y otros elementos del lenguaje de programación. Las expresiones regulares son una herramienta poderosa para realizar búsquedas y análisis sintáctico en el proceso de compilación.

Concluido esto llegamos ahora nuestra revisión y retrospectiva del Sprint, en este caso, salió muy bien esta etapa de investigación ya que concordamos en la mayoría de los aspectos las definiciones, así como tuvimos una facilidad para llegar a conceptos generales a través de lluvias de ideas completando en conjunto las definiciones y entendiendo el tema.

Si bien esta parte puede ser lo que mejor se planeó, conllevo cerca de dos días por falta de disponibilidad en los tiempos del equipo SCRUM, lo que más tarde también repercutió con los cronogramas propuestos.

Segundo Sprint

Completando el entendimiento de los temas y la capacitación para la mayoría de los miembros del equipo, tuvimos nuestro Daily SCRUM donde elegimos el mejor lenguaje para la realización del algoritmo de Thompson e interfaces gráficas y fue Python, usando el paradigma de Objetos o POO, para resolver el problema. Ya que la mayoría del equipo tenía un mejor dominio en este lenguaje que en otros comparados como Java o C++.

Como parte de la dirección de nuestro SCRUM Master pasamos con el siguiente backlog para el sprint, repitiendo otra vez el ciclo de trabajo. En este ciclo se prioriza completar la lectura del archivo del cual se tomarían las expresiones regulares, pero también se comenzó con el desarrollo de los módulos para la interfaz, respecto a la primera tarea se buscaron los requisitos correspondientes para hacer el desglose de las tareas:

- Comenzar a leer archivos .txt y guardar las líneas con la información del alfabeto y expresión regular en diferentes variables.
- Crear el algoritmo para encontrar recurrencias y eliminar palabras extras del archivo que no se procesarán, así como eliminar repeticiones de los símbolos del alfabeto.
- Una vez encontrado un identificador como “letras” o “dígitos” sustituirlo por sus correspondientes, para letras todas las del alfabeto incluyendo mayúsculas y minúsculas y paras los dígitos los números del 0 al 9.

Sin embargo como estrategia del SCRUM master se dividió el trabajo en dos partes, dos de nuestros compañeros Jorge Elorza y Ramirez Fernando comenzaron con la lectura de los archivos, ambas partes resumiendo sus palabras consiguieron lo siguiente:

- Se consiguió el archivo que leyera las expresiones del .txt.

- Se usaron expresiones regulares de una librería de python para obtener las expresiones y el alfabeto o alfabetos del archivo.
- Se completó la traducción de los identificadores en símbolos para los alfabetos.

prueba_lectura.py > ...

```python
import re

# Función para generar el alfabeto a partir de las líneas del archivo
def generar_alfabeto(lineas):
    alfabeto = set()
    for linea in lineas:
        # Buscamos las líneas que comienzan con "alfabetoX:"
        match = re.match(r'^alfabeto(\d+): (.+)$', linea)
        if match:
            numero_alfabeto = match.group(1)
            contenido = match.group(2)
            # Si contiene "letras", agregamos todas las letras del alfabeto
            if contenido == 'letras':
                alfabeto.update(set(chr(i) for i in range(65, 91)))
                alfabeto.update(set(chr(i) for i in range(97, 123)))
            # Si contiene "digitos", agregamos los dígitos del 0 al 9
            elif contenido == 'digitos':
                alfabeto.update(set(str(i) for i in range(10)))
            # De lo contrario, agregamos las letras únicas
            else:
                alfabeto.update(set(contenido))
    return sorted(alfabeto)

# Función para obtener la expresión regular
def obtener_expresion(lineas):
    for linea in lineas:
        if linea.startswith("expresion:"):
            return re.sub(r'^expresion: (.+)$', r'\1', linea)

# Leer el archivo de entrada
archivo_entrada = 'expresiones.txt'
with open(archivo_entrada, 'r') as f:
    lineas = f.readlines()

# Generar el alfabeto
alfabeto = generar_alfabeto(lineas)

# Eliminar repeticiones en el alfabeto
alfabeto = sorted(set(alfabeto))

# Obtener la expresión regular
expresion_regular = obtener_expresion(lineas)

# Imprimir el resultado
print("Alfabeto:", ''.join(alfabeto))
print("Expresión regular:", expresion_regular)
```

Código inicial usado para la realización de lectura del archivo

Ahora haciendo un repaso de lo logrado en esta parte, se terminó en tiempo y forma pues fue un desarrollo simple y la lógica implementada así como la ayuda de las funciones en python para el desarrollo de expresiones regulares y funciones de conjuntos y funciones para listas, facilitaron el código. El inconveniente más grave en este punto fue el de no saber muy bien cómo sería la trabajada la salida para más adelante si como cadena o lista pero no supuso un problema grave.

Diseño de la interfaz:

Respecto a la otra parte el resto de nuestros compañeros restantes se dedicaron a realizar los módulos para el funcionamiento de la interfaz mediante el tkinter:
Tkinter es una biblioteca de Python que se utiliza para crear interfaces gráficas de usuario (GUI, por sus siglas en inglés, Graphical User Interface). Tkinter es una parte estándar de la biblioteca estándar de Python y proporciona herramientas y widgets para la creación de ventanas, botones, etiquetas, cajas de texto, menús y otras interfaces gráficas.

Algunas características clave de Tkinter y por lo cual optamos por esto incluyen:

- **Facilidad de uso:** Tkinter es relativamente fácil de aprender y usar, lo que lo hace una excelente opción para desarrolladores que deseen crear aplicaciones con interfaces gráficas.

- **Multiplataforma:** Las aplicaciones creadas con Tkinter son multiplataforma, lo que significa que pueden ejecutarse en sistemas Windows, macOS y Linux sin modificaciones significativas.

- **Amplia gama de widgets:** Tkinter proporciona una variedad de widgets predefinidos, como botones, etiquetas, cajas de texto, listas, etc., que facilitan la creación de interfaces de usuario interactivas.

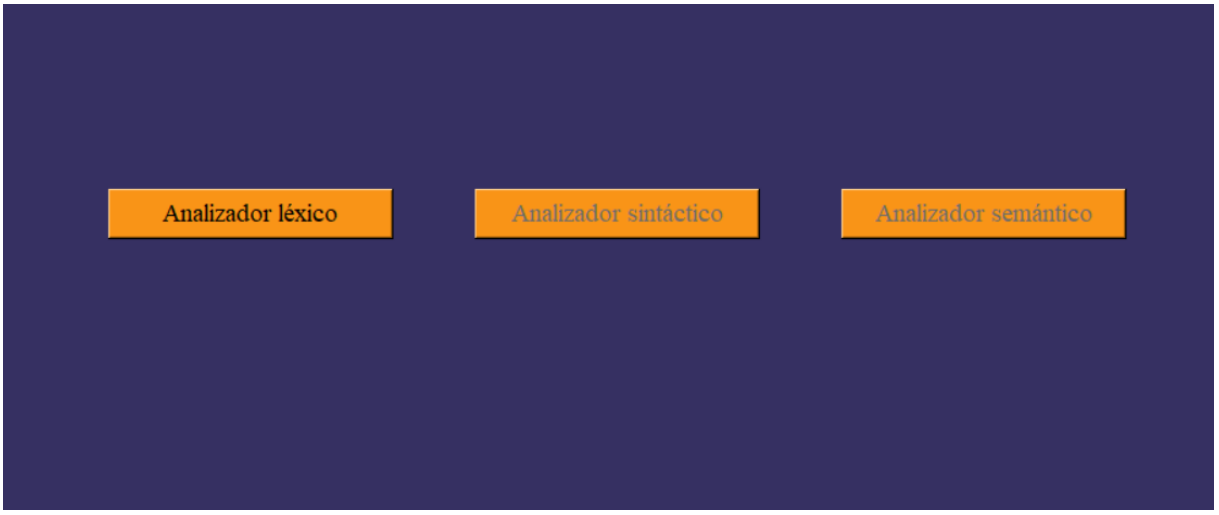
- **Personalización:** Puedes personalizar la apariencia de tus aplicaciones Tkinter utilizando colores, fuentes y estilos de widgets.

- **Eventos y manejo de eventos:** Tkinter permite definir y gestionar eventos, como hacer clic en un botón, escribir en una caja de texto, o mover el mouse, lo que permite que las aplicaciones respondan a la interacción del usuario.

- **Integración con otros módulos de Python:** Puedes integrar Tkinter con otros módulos de Python para crear aplicaciones más complejas y funcionales.

En esta etapa solo se concluyó con el diseño de la interfaz y las diferentes pantallas, ya para el siguiente Sprint se pensó más en su usabilidad y funcionamiento, por futuros problemas que acarreamos, si bien el avance pareciera no tan significativo esto es porque para esta fecha aproximadamente era sábado 14 de octubre.

Pantallas del ID:



Nuestra retrospectiva de estos días fue que el proyecto avanzaba a un ritmo lento y muchas funcionalidades realmente estaban algo estancadas ya que solo teníamos la parte más fácil que era la lectura de archivos. Si bien el objetivo era terminarlo alrededor del viernes o jueves, esto supuso un fallo en el sprint por no cumplir correctamente con el cronograma. Como plan de contingencia se buscó ahora participar todos en conjunto con el desarrollo del algoritmo de Thompson para lograr nuestra meta para el lanzamiento “deploy”.

Tercer Sprint

En esta fase conjunto al SCRUM master, se esperaba completar el uso de la interfaz para procesar cada parte del algoritmo de thompson y generar las tablas, también el de completar el algoritmo de thompson. Esperando que esto se completara el día miércoles 18 de octubre, esto en un lapso de 4 días.

Mediante una programación modular cada miembro tenía una tarea desde seguir con el diseño de la interfaz y probar, corregir y programar las sub-operaciones del algoritmo de thompson dentro del interfaz a cargo de los compañeros Alan Yael y Ángel de Jesus, donde consiguieron a grandes rasgos culminar lo necesario de la interfaz gráfica.

```
def create_labels(frame,canvas):
    k=0
    for i in range(100):
        label = tk.Label(frame, text=f"Label {i}",width=10)
       # label.grid(row=i, column=0, padx=10, pady=10)
        label.place(x=20,y=k)
        k+=35
        frame.update_idletasks()
    canvas.config(scrollregion=canvas.bbox("all"))

def show_scrollable_frame():
    new_window = tk.Toplevel(root)
    new_window.title("Scrollable Frame")

    canvas = tk.Canvas(new_window,bg="blue")
    canvas.place(x=0,y=500)

    scrollbar = ttk.Scrollbar(new_window, orient=tk.VERTICAL, command=canvas.yview)
    scrollbar.place(x=300,y=500,height=200)

    horizontal_scrollbar = ttk.Scrollbar(new_window, orient=tk.HORIZONTAL, command=canvas.xview)
    horizontal_scrollbar.set(0.0,0.9)
    horizontal_scrollbar.place(x=0,y=500,width=200)

    frame = tk.Frame(canvas)
    canvas.create_window((0, 0), window=frame, anchor=tk.NW)
    canvas.configure(yscrollcommand=scrollbar.set,xscrollcommand=horizontal_scrollbar.set)
    Button1=tk.Button(new_window,text="getLabel",command=lambda:create_labels(frame,canvas))
    Button1.place(x=30,y=50)
    frame.update_idletasks()
    canvas.config(scrollregion=canvas.bbox("all"))
```

Aunado a esta parte del código, trabajaron para modificar la impresión de pantalla de forma correcta en el archivo *Lexico.py* “printTbale” y “cleanTable”, el cual estaba unido a las funciones de salida del algoritmo de Thompson.
Mientras bajo la dirección del SCRUM master se definían las funciones y los métodos que crearán las sub-operaciones, a cargo de los miembros restantes, Frida Ximena, José Guadalupe, Jorge Elorza y Ramirez Fernando, esto en el archivo Procesamiento_REGEXP.py. Funciones que realizan:

CorrectNumerarion(List, adder):

- Propósito: Esta función recorre una lista enlazada de estados y ajusta sus identificadores numéricos sumando un valor específico (adder) a cada identificador. Esto es útil para mantener la numeración de los estados actualizada al realizar operaciones de combinación o modificación de autómatas.
- Tema: Mantenimiento y corrección de la numeración de estados.

SingleLetter(sym):

- Propósito: Esta función crea un autómata finito determinista (DFA) que reconoce una única letra (símbolo). El autómata tiene dos estados, uno inicial y otro final, y una transición entre ellos etiquetada con el símbolo proporcionado.
- Tema: Creación de un DFA para reconocer una letra específica.

Concatenation(L1, L2):

- Propósito: Esta función realiza la concatenación de dos autómatas representados como listas enlazadas. Asegura que el estado final del primer autómata no sea final y que las transiciones del segundo autómata se enumeran y copian correctamente al primero. Luego, se unen las listas.
- Tema: Operación de concatenación de autómatas.

Union(L1, L2):

- Propósito: Esta función realiza la unión (o alternancia) de dos autómatas representados como listas enlazadas. Ajusta la numeración de los estados del

segundo autómata y crea nuevos estados iniciales y finales para conectar los autómatas originales. Luego, se unen las listas.

- Tema: Operación de unión de autómatas.

Cerradura_de_Kleene(List):

- Propósito: Esta función aplica la cerradura de Kleene a un autómata representado como una lista enlazada. Agrega nuevos estados iniciales y finales, y las transiciones necesarias para permitir la repetición cero o más veces del autómata original.
- Tema: Aplicación de la cerradura de Kleene a un autómata.

Cerradura_Opcional(L):

- Propósito: Esta función aplica la cerradura opcional a un autómata representado como una lista enlazada. Agrega nuevos estados iniciales y finales, así como transiciones para permitir que el autómata original se omita o se repita una vez.
- Tema: Aplicación de la cerradura opcional a un autómata.

Cerradura_Positiva(L):

- Propósito: Esta función aplica la cerradura positiva a un autómata representado como una lista enlazada. Agrega nuevos estados iniciales y finales, así como transiciones para permitir que el autómata original se repita una o más veces.
- Tema: Aplicación de la cerradura positiva a un autómata.

También realizamos las clases pertinentes que representan componentes de autómatas finitos y una lista enlazada para construir y manipular autómatas. Definiendo propósito y funciones a usar, las cuales fueron implementadas con éxito

Clase Transition:

- Propósito: Representa una transición de un estado a otro en un autómata finito. La transición está etiquetada con un símbolo y apunta al siguiente estado.

- Funciones:
    1. __init__(self, sy, st): Constructor de la transición, que toma un símbolo sy y el estado siguiente st.
    2. getSymbol(self): Devuelve el símbolo de la transición.
    3. getState(self): Devuelve el estado al que la transición apunta.
    4. setSymbol(self, sy): Establece el símbolo de la transición.
    5. setState(self, st): Establece el estado al que la transición apunta.
    6. __str__(self): Devuelve una representación en cadena de la transición.

Clase State:

- Propósito: Representa un estado en un autómata finito. Puede tener múltiples transiciones salientes, ser un estado inicial o final, y tiene un identificador único.
- Funciones:
    1. __init__(self, i, transition, ini, fin): Constructor del estado, que toma un identificador i, una transición inicial transition, una bandera para indicar si es un estado inicial ini, y una bandera para indicar si es un estado final fin.
    2. getIniState(self): Devuelve la bandera que indica si es un estado inicial.
    3. getFinalState(self): Devuelve la bandera que indica si es un estado final.
    4. getTransitions(self): Devuelve una lista de transiciones salientes desde el estado.
    5. getId(self): Devuelve el identificador del estado.
    6. setIniState(self, i): Establece la bandera de estado inicial.
    7. setFinalState(self, f): Establece la bandera de estado final.
    8. setId(self, i): Establece el identificador del estado.
    9. addTransition(self, t): Agrega una transición saliente al estado.
    10. displayTransitions(self): Muestra las transiciones salientes del estado.
    11. __str__(self): Devuelve una representación en cadena del estado.
    12. Sort_Transitions(self): Ordena las transiciones del estado por el símbolo de transición.

Clase Node:

- Propósito: Define un nodo para la lista enlazada utilizada para almacenar estados de autómatas. Cada nodo contiene una referencia al estado.

- Funciones:
    1. __init__(self, s): Constructor del nodo que toma un estado s.

Clase LinkedList:

- Propósito: Define una lista enlazada que se utiliza para almacenar estados de autómatas y proporciona funciones para operar con ella.
- Funciones:
    1. __init__(self): Constructor de la lista enlazada.
    2. is_empty(self): Comprueba si la lista está vacía.
    3. append(self, state): Agrega un estado al final de la lista.
    4. prepend(self, state): Agrega un estado al principio de la lista.
    5. delete(self, state): Elimina un estado de la lista.
    6. getTail(self): Obtiene el último estado de la lista.
    7. getLastNode(self): Obtiene el último nodo de la lista.
    8. getNode(self, n): Obtiene el nodo en una posición específica de la lista.
    9. CountNodes(self): Cuenta la cantidad de nodos en la lista.
    10. concatenateWith(self, List): Concatena la lista actual con otra lista.
    11. display(self): Muestra los estados en la lista y sus transiciones.

Comenzando a hacer un análisis de lo logrado, ya se terminaría la interfaz gráfica, hubo un buen resultado para el diseño de la tabla de transiciones y se definieron los requisitos para solicitar de una manera particular la lectura del archivo, ahora por otro lado, nuestra retrospectiva de estos días fue que el proyecto no estaría listo para la fecha indicada, ya que en la realización de las funciones de expresiones regulares teníamos errores al procesar algunas cadenas más complejas, además con el afán de entregar funcionando al 100% el código para evitar darle demasiado mantenimiento a futuro o compostura, se siguió realizando su mejora, otros problemas también surgieron a causa de cómo venía del archivo de texto la expresión regular, ya que el programa para este punto no detectaba el orden de los operandos y cómo procesarlos. Ahora bien, aunque parecía que se habían realizado de mala manera las suboperaciones de Thompson, en realidad el problema fue el trabajar todo de manera aislada de los casos reales a probar. Siendo este el punto que mejoramos para realizar otro Sprint donde se uniría el proyecto y se compondrían los errores pertinentes.

Cuarto Sprint

Este último sprint, fue realizado desde el día miércoles 18 de octubre al viernes 20 de octubre, donde sí o sí terminaríamos todos los requerimientos y tendríamos culminados los módulos para unirlos.
Durante estos primeros dos días 18 y 19, se buscaron en conjunto en nuestro Daily SCRUM soluciones para el procesamiento de las expresiones regulares, nuestros compañeros Alan Yael y Ángel de Jesús se encargaron de pensar en una solución para poder leer correctamente el orden de las operaciones.
Para el código se ocupó de la conversión de una expresión regular infija en una expresión regular posfija (notación postfija) y la evaluación de esta expresión posfija para construir un autómata finito correspondiente a la expresión regular.

Función expresion_postfija(expresion):

- Propósito: Convierte una expresión regular en notación infija en una expresión regular en notación posfija utilizando el algoritmo de Shunting Yard.
- Funciones internas:
  1. precedencia(op): Retorna la precedencia de un operador dado (usado en el algoritmo Shunting Yard).
  2. a_postfija(expresion): Realiza la conversión de la expresión regular infija a posfija utilizando el algoritmo Shunting Yard.
- Tema: Conversión de expresiones regulares de notación infija a posfija.

Función evaluar_expresion_postfija(expresion_postfija):

- Propósito: Evalúa una expresión regular en notación posfija para construir un autómata finito correspondiente a la expresión regular.
- Función:
  1. Itera a través de los caracteres de la expresión posfija y apila estados y operaciones en una pila.
  2. Realiza las operaciones necesarias (Cerradura Opcional, Cerradura de Kleene, Cerradura Positiva, Concatenación y Unión) para construir el autómata finito correspondiente a la expresión regular.
- Tema: Evaluación de una expresión regular posfija y construcción de un autómata finito.

mientras que Ximena, José Guadalupe, Jorge Elorza y Ramirez Fernando,se dedicaban a corregir errores relacionados con los códigos y unir todo el programa, para esto nos guiamos de lo que propone la metodología SCRUM para Detección de errores:

- **Revisión continua del trabajo:** Durante la reunión diaria (Daily Scrum), los miembros del equipo compartimos el progreso y discutimos cualquier problema o error que haya surgido realizamos pruebas y analizamos los errores que las causaban haciendo un repaso largo y tardado de cada función mediante una depuración del código.

- **Revisión en la reunión de revisión:** En la reunión de revisión al final de cada sprint en este caso nuestro tercer sprint, se inspeccionó el trabajo completado y se identifican errores o deficiencias en el producto, siendo más que nada el caso de algunas sub-operaciones de Thompson.

- **Retroalimentación de los interesados:** Los interesados, proporcionamos una retroalimentación sobre el producto en desarrollo, lo que puede incluir la identificación de errores.

También gracias al conocimiento de algunos de los miembros se pudo resolver gran parte de los errores, propuestas en nuestro Daily Scrum, como el uso de GitHub Issues y GitLab, para registrar y dar seguimiento a los errores. Estas herramientas permiten asignar tareas, establecer prioridades y realizar un seguimiento del progreso.

Finalmente a cargo de uno de nuestros compañeros se integraron cada módulo por módulo al código principal, donde todos participamos, esto no tomó tanto tiempo, pero se requirió de todos los presentes para distinguir elementos importantes de cada archivo creado. Además se probaron varias expresiones regulares (***véase en anexos***)

Hablando sobre la última revisión y retrospectiva sobre este sprint y el proyecto, determinamos que hicimos un trabajo veloz para la corrección de errores gracias a

las herramientas propuestas ya mencionadas, además de que se demostró buena habilidad para la unificación del archivo final, además de que ahora si las pruebas realizadas se lograron procesar varios autómatas de diferentes complejidades. Sin embargo hablando de los fallos, más que nada el tiempo dedicado en las vidas diarias de cada integrante hicieron un poco difícil el cumplimiento de las agendas, además de que no se esperaba tener tantos errores y fallas en el programa de procesamiento de expresiones regulares, también otro punto a tratar fueron los requisitos, muchos surgieron internamente por una mala gestión y planteamiento del problema por parte de todos, lo que generó el retraso más adelante, ya que si bien una organización pareciera no ser importante o muy retrasada, muchos problemas creemos que se pudieron solventar con una mejor búsqueda de requisitos.

Conclusión:

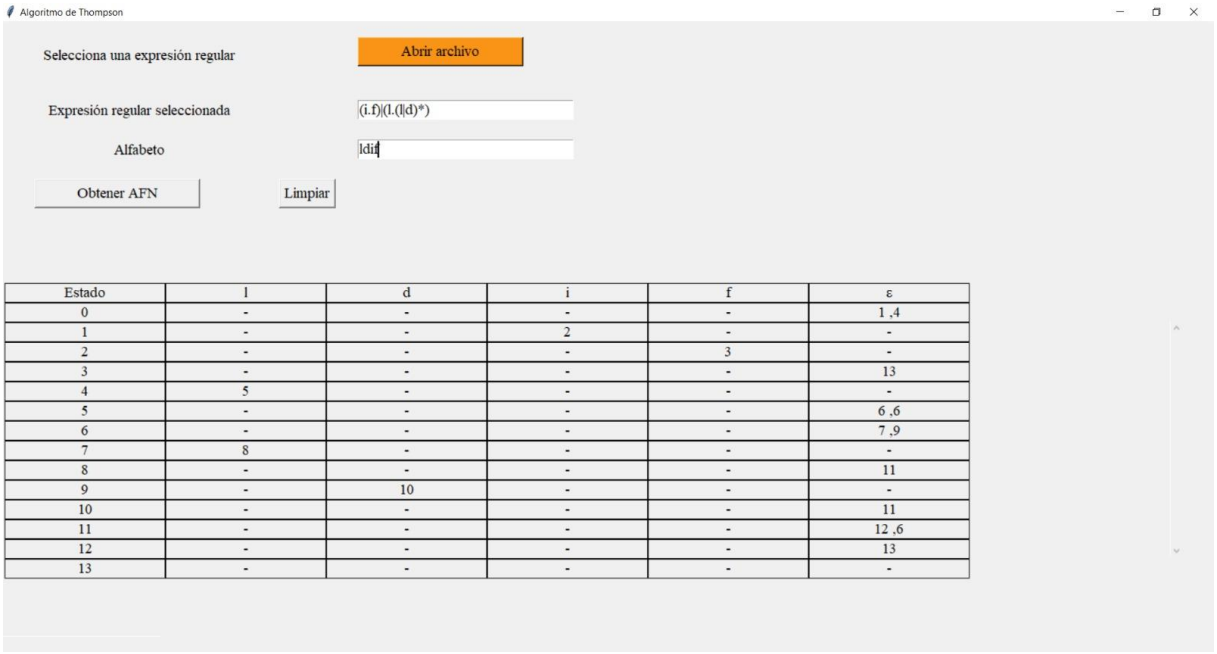
En conclusión podemos decir que el Algoritmo de Thompson usado para la construcción de AFN, son una herramienta útil para procesar expresiones regulares, la implementación del algoritmo de Thompson en un entorno de desarrollo integrado (IDE) de Python como parte de un proyecto gestionado bajo la metodología SCRUM demuestra la capacidad de combinar eficazmente la flexibilidad y agilidad de SCRUM con la destreza técnica para abordar problemas complejos. SCRUM proporciona un marco de trabajo que se adapta bien a proyectos de desarrollo de software al permitir una planificación iterativa, una comunicación constante y la capacidad de responder a cambios en los requisitos del cliente.

La elección de Python como lenguaje de programación ofrece ventajas adicionales, ya que es conocido por su sintaxis clara y su amplia biblioteca estándar, lo que facilitó el desarrollo de aplicaciones de manera rápida y eficiente. La implementación del algoritmo de Thompson, que se utiliza comúnmente en la construcción de compiladores y expresiones regulares, demuestra la versatilidad de Python para abordar problemas complejos de procesamiento de texto y análisis

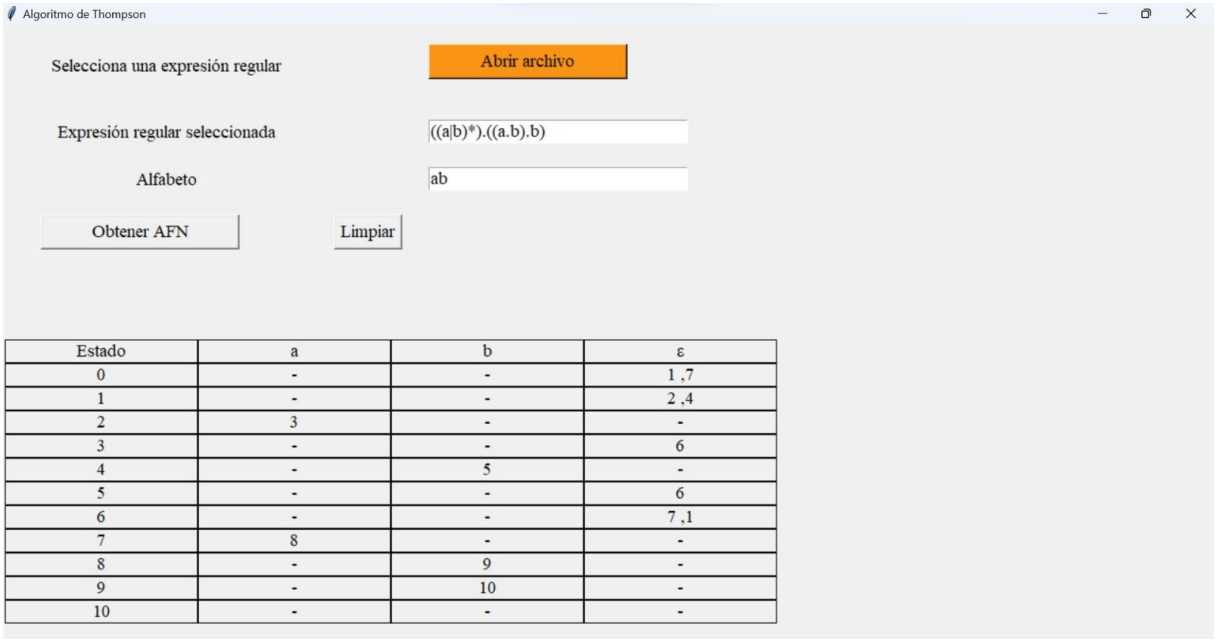
Anexos:

Pruebas para el procesamiento de las expresiones regulares leídas de los archivos “expresiones.txt” algunas reglas para estos archivos fueron:

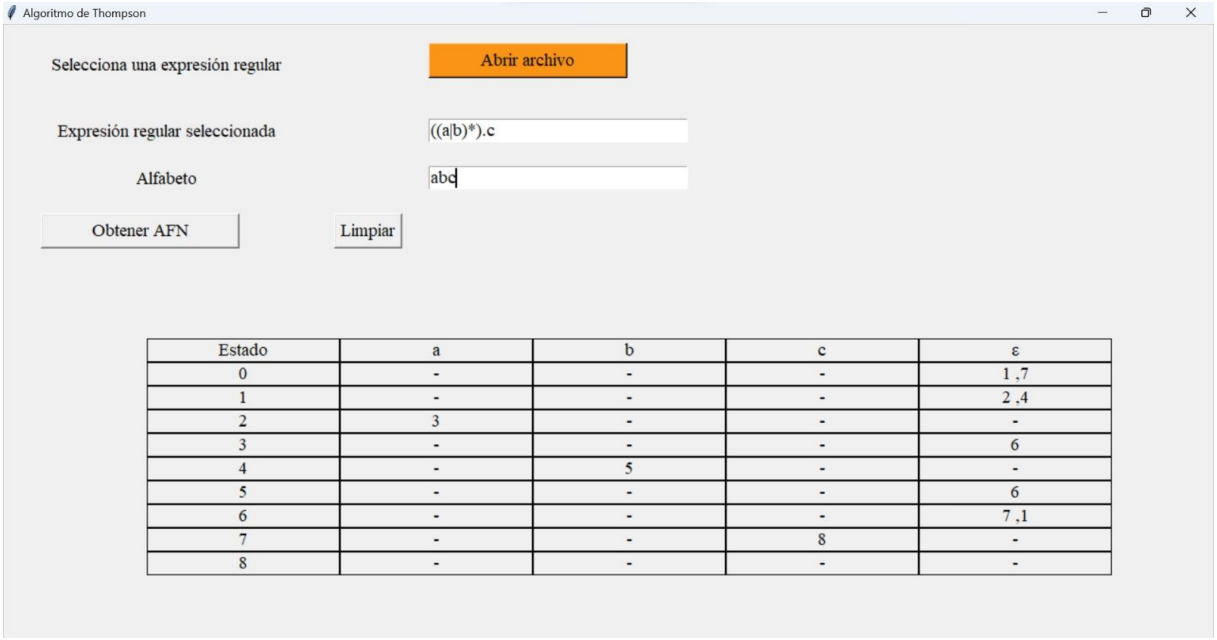
- Se requiere el uso de paréntesis para la clasificación de las operaciones.
- La concatenación debe darse con ayuda de un “.” esto para poder detectarla.
- El archivo debe estar definido en dos líneas la primera la expresión y la segunda el alfabeto



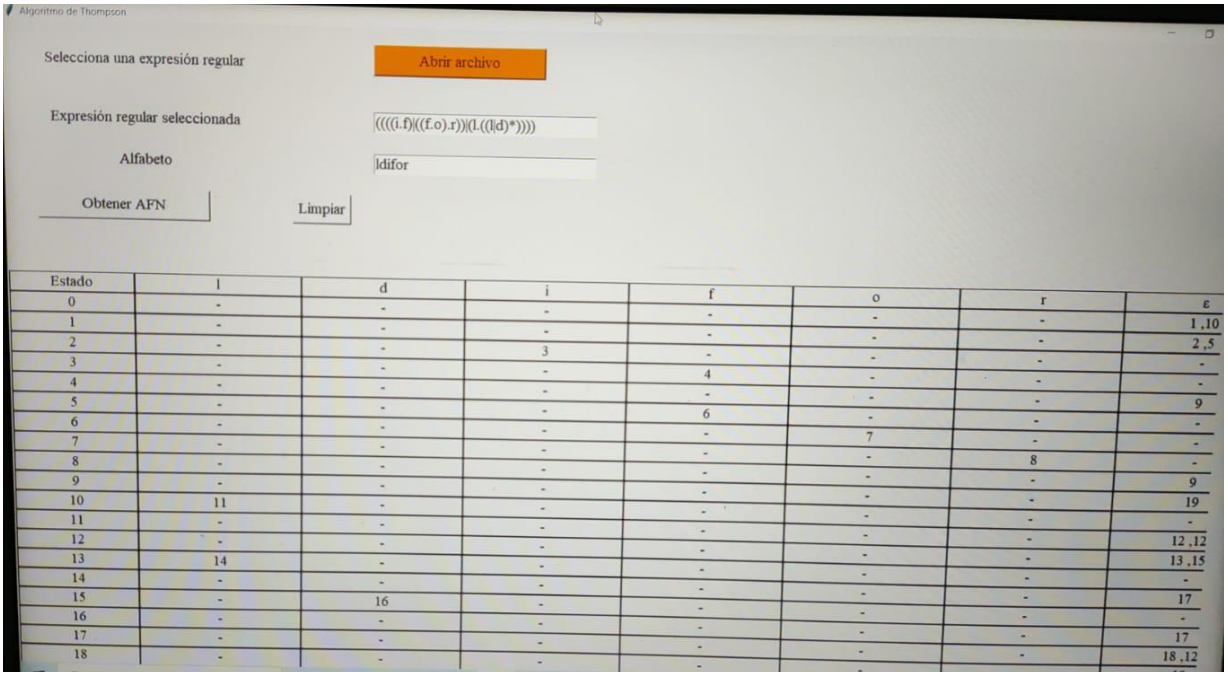
Estado	l	d	i	f	ε
0	-	-	-	-	1 ,4
1	-	-	2	-	-
2	-	-	-	3	-
3	-	-	-	-	13
4	5	-	-	-	-
5	-	-	-	-	6 ,6
6	-	-	-	-	7 ,9
7	8	-	-	-	-
8	-	-	-	-	11
9	-	10	-	-	-
10	-	-	-	-	11
11	-	-	-	-	12 ,6
12	-	-	-	-	13
13	-	-	-	-	-



Estado	a	b	ε
0	-	-	1 ,7
1	-	-	2 ,4
2	3	-	-
3	-	-	6
4	-	5	-
5	-	-	6
6	-	-	7 ,1
7	8	-	-
8	-	9	-
9	-	10	-
10	-	-	-



Estado	a	b	c	ε
0	-	-	-	1 ,7
1	-	-	-	2 ,4
2	3	-	-	-
3	-	-	-	6
4	-	5	-	-
5	-	-	-	6
6	-	-	-	7 ,1
7	-	-	8	-
8	-	-	-	-



Estado	l	d	i	f	o	r	ε
0	-	-	-	-	-	-	1 ,10
1	-	-	-	-	-	-	2 ,5
2	-	-	3	-	-	-	-
3	-	-	-	4	-	-	-
4	-	-	-	-	-	-	9
5	-	-	-	6	-	-	-
6	-	-	-	-	7	-	-
7	-	-	-	-	-	8	-
8	-	-	-	-	-	-	9
9	-	-	-	-	-	-	19
10	11	-	-	-	-	-	-
11	-	-	-	-	-	-	12 ,12
12	-	-	-	-	-	-	13 ,15
13	14	-	-	-	-	-	-
14	-	-	-	-	-	-	17
15	-	16	-	-	-	-	-
16	-	-	-	-	-	-	17
17	-	-	-	-	-	-	18 ,12
18	-	-	-	-	-	-	

Referencias:

Compiladores y Algoritmo de Thompson:

- Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2006). Compiladores: Principios, Técnicas y Herramientas (2ª ed.). Pearson.
- Aho, A. V., Sethi, R., & Ullman, J. D. (1986). Compilers: Principles, Techniques, and Tools. Addison-Wesley.
- Thompson, K. (1968). Regular expression search algorithm. Communications of the ACM, 11(6), 419-422.

Teoría de Autómatas:

- Sipser, M. (2012). Introduction to the Theory of Computation. Cengage Learning.

Tkinter:

- Roseman, J. (2005). Tkinter 8.4 reference: A GUI for Python. PythonWare.
- Moore, J. (2011). Python GUI Programming with Tkinter. CreateSpace Independent Publishing Platform.

Metodología SCRUM:

- Carlos Alberto Fernández y Fernández (2023). Notas de Desarrollo ágil de software - Scrum