**Ensayo sobre la Motivación y el Desarrollo de un Tokenizador Mínimo para Procesamiento de Lenguaje Natural**

**Introducción**

El procesamiento de lenguajes naturales (PLN) es una subdisciplina de la inteligencia artificial que se centra en la interacción entre computadoras y lenguajes humanos. El análisis léxico es una de las primeras etapas en el procesamiento de textos, donde se divide un texto en unidades más pequeñas llamadas lexemas o tokens. En este ensayo, se explora la motivación para construir un tokenizador mínimo, denominado MNLPTK (Minimal Natural Language Processing Tokenizer), diseñado para identificar y organizar palabras en textos en español.

**Motivación**

El procesamiento de lenguajes naturales se enfrenta a múltiples desafíos debido a la naturaleza ambigua y diversa de los lenguajes humanos. Entre estos desafíos se encuentran:

1. **Ambigüedades:** Las palabras pueden tener múltiples significados según el contexto.
2. **Expresiones literarias y modismos:** Variaciones regionales y contextuales que afectan la interpretación.
3. **Estructuras gramaticales complejas:** La variabilidad en la sintaxis y gramática de diferentes lenguajes.

Para abordar estos desafíos, los analizadores léxicos deben ser capaces de manejar estas situaciones utilizando herramientas específicas, como preprocesadores léxicos y tokenizadores.

**Objetivo del Trabajo Práctico**

El objetivo del trabajo práctico es construir un tokenizador mínimo (MNLPTK) que identifique lexemas en un texto de entrada en español, y los organice en una estructura de datos útil para el análisis sintáctico posterior. El tokenizador acepta un archivo de texto y devolver un diccionario organizado en tres columnas: TOKEN, PATRÓN, LEXEMAS y POSICIONES, en un archivo de formato JSON

**Descripción del Alcance del Trabajo Práctico**

El tokenizador maneja los siguientes tipos de tokens:

* **ARTÍCULO**
* **SUSTANTIVO**
* **VERBO**
* **ADJETIVO**
* **ADVERBIO**
* **OTROS**
* **ERROR\_LX** (para lexemas que no pueden ser clasificados)

El proceso incluye la lectura de un archivo de texto, la identificación de lexemas utilizando patrones predefinidos, y la actualización del diccionario de datos con la información de los tokens y sus posiciones. El programa también permite la reutilización y actualización del diccionario para nuevos textos, sin perder los datos anteriores.

**Decisiones Adoptadas**

El trabajo práctico es una implementación de un MNLPTK (Minimal Natural Language Processing Tokenizer), que será utilizado para crear un diccionario de datos a partir de un texto de entrada en formato .txt. El diccionario de datos estará creado utilizando un archivo en formato JSON. Además, se obtendrá un archivo de salida en formato .txt que será utilizado por el Analizador Sintáctico para la siguiente fase de compilación (no contemplada en este Trabajo Práctico)

Se tomaron varias decisiones:

1. Cada lexema, aunque pueda pertenecer a más de un token, pertenecerá a un solo. Dejaremos al Analizador Sintáctico la tarea de lidiar con la ambigüedad de un lexema dado pueda pertenecer a múltiples tokens y no lo contemplaremos para este Trabajo Práctico.
2. A la hora de tokenizar el texto para hallar sus componentes, esto se realizará por medio de una expresión regular que tomará los caracteres separadores y los espacios apropiadamente, además de tener en cuenta un caso especial donde puedan existir elementos numéricos en la entrada, como un valor 3.14, por ejemplo, lo tomará como un lexema “3.14” y no como 2 lexemas “3” y “14”
3. Se utilizó una función hash para obtener los valores de los lexemas cuando se analizara uno. En total, para cada uno de los tokens, se creó una tabla hash que se encontraba implementada en Python por medio de diccionarios. Y por cada lexema de cada token, implementamos otra tabla hash adicional para poder realizar una búsqueda rápida de los lexemas durante las lecturas de los archivos de entrada en formato .txt.
4. Se decidió usar patrones por extensión en lugar de patrones por compresión, debido a la dificultad de encontrar expresiones regulares para todos los tokens. Entonces como los patrones y los lexemas son lo mismo, se prescindió de la columna de patrones en el diccionario de datos.
5. Se usó un conjunto para mostrar las estadísticas de lexemas nuevos y lexemas hallados porque no era necesario tener ningún tipo de ordenación y solo necesitábamos obtener su tamaño para mostrar las estadísticas, además de que si se repiten lexemas este hecho no incrementa el número general de lexemas en el texto.
6. Utilizamos la función hash estándar de Python, que es SipHash, porque está optimizada para mensajes cortos, como los que encontramos en los lexemas, también cortos. Y porque los diccionarios, estructuras de Python, contiene por defecto el uso de estas funciones hash.
7. Los errores léxicos se almacenarán como parte del diccionario de datos por si se requiera para uso posterior en otra sección. El usuario los definirá y, además, que existan estos errores no supondrá un problema para el analizador léxico, pues simplemente tomará el siguiente valor durante el proceso de análisis léxico.
8. Para este trabajo no se aplicaron mejoras de ningún tipo, como las sugeridas en el enunciado del Trabajo Práctico.

**Metodología**

La implementación del tokenizador utilizará expresiones regulares por extensión por cada TOKEN definido, y ninguna expresión por compresión. Además, se utilizará la herramienta de CHATGPT para contribuir a la codificación.

**Código Fuente**

A continuación, se presenta un fragmento del código fuente en Python para el tokenizador:

import re

import json

from collections import defaultdict

# Estructura del diccionario de datos

data\_dict = {

'POSICIONES': defaultdict(lambda: defaultdict(list)),

'num\_files\_processed': 0, # Número de archivos procesados

'predefined\_lexemes\_used': False # Si se han utilizado los lexemas predefinidos

}

# Lexemas predefinidos para los tokens

predefined\_lexemes = {

'ARTICULO': ['el', 'la', 'los', 'las', 'un', 'una', 'unos', 'unas'],

'SUSTANTIVO': ['Nietzsche', 'temas', 'libro', 'mundo', 'persona'],

'VERBO': ['escribir', 'leer', 'ser', 'haber', 'ir', 'escribió', 'escriben'],

'ADJETIVO': ['grande', 'pequeño', 'bueno', 'malo', 'nuevo', 'viejo'],

'ADVERBIO': ['rápidamente', 'lentamente', 'bien', 'mal', 'cerca', 'lejos'],

'OTROS': ['sobre', 'con', 'sin', 'en', 'por', 'para', 'y', 'o', 'a', 'de'],

'ERROR\_LX': []

}

# Función para inicializar el diccionario de datos con lexemas predefinidos

def initialize\_with\_lexemes():

for token, lexemes in predefined\_lexemes.items():

for lexeme in lexemes:

data\_dict['POSICIONES'][token][lexeme.lower()] = []

data\_dict['predefined\_lexemes\_used'] = True

# Función para cargar el diccionario de datos existente

def load\_data\_dict(file\_path):

try:

with open(file\_path, 'r', encoding='utf-8') as file:

loaded\_dict = json.load(file)

data\_dict.update(loaded\_dict)

# Convertir la estructura del diccionario cargado a defaultdict

data\_dict['POSICIONES'] = defaultdict(

lambda: defaultdict(list), {

token: defaultdict(list, {

lexeme.lower(): positions

for lexeme, positions in lexemes.items()

})

for token, lexemes in data\_dict['POSICIONES'].items()

})

print("Diccionario de datos cargado exitosamente.")

return True

except FileNotFoundError:

print("Archivo del diccionario de datos no encontrado.")

return False

except KeyError as e:

print(f"Error de clave: {e}")

return False

# Función para pedir al usuario que asigne un token

def prompt\_for\_token(lexeme):

token\_options = {

1: 'ARTICULO',

2: 'SUSTANTIVO',

3: 'VERBO',

4: 'ADJETIVO',

5: 'ADVERBIO',

6: 'OTROS',

7: 'ERROR\_LX' # Añadir ERROR\_LX como opción

}

while True:

print(f"\nPor favor, asigne un token a este lexema: {lexeme}")

for number, token in token\_options.items():

print(f"{number} - {token}")

try:

choice = int(input("Ingrese el número correspondiente a su elección: "))

if choice in token\_options:

return token\_options[choice]

else:

print("Opción incorrecta, por favor ingrese un número válido.")

except ValueError:

print("Entrada no válida, por favor ingrese un número.")

# Función para leer y tokenizar el texto de entrada

def tokenize\_text(file\_path, entry\_number):

with open(file\_path, 'r', encoding='utf-8') as file:

text = file.read()

lexemes = re.split(r'\s+|(?<!\d)[.,;:!?](?!\d)', text)

found\_lexemes = set()

new\_lexemes = set()

posicion = 1

for lexeme in lexemes:

lexeme = lexeme.strip().lower() # Convertir lexema a minúsculas

if not lexeme:

continue

found\_lexemes.add(lexeme)

token\_found = False

for token in data\_dict['POSICIONES']:

if lexeme in data\_dict['POSICIONES'][token]:

data\_dict['POSICIONES'][token][lexeme].append(f'TXT{entry\_number}-{posicion}')

token\_found = True

break

if not token\_found:

new\_token = prompt\_for\_token(lexeme)

if new\_token == 'ERROR\_LX':

print(f"Lexema '{lexeme}' identificado como error léxico.")

if new\_token not in data\_dict['POSICIONES']:

data\_dict['POSICIONES'][new\_token] = defaultdict(list)

data\_dict['POSICIONES'][new\_token][lexeme].append(f'TXT{entry\_number}-{posicion}')

new\_lexemes.add(lexeme)

posicion += 1

return found\_lexemes, new\_lexemes

# Función para guardar el diccionario de datos en un archivo JSON

def save\_data\_dict(file\_path):

with open(file\_path, 'w', encoding='utf-8') as file:

json.dump(data\_dict, file, ensure\_ascii=False, indent=4)

print("Diccionario de datos guardado exitosamente.")

# Función para generar el archivo de salida para el analizador sintáctico

def generate\_output\_file(file\_path, entry\_number):

output\_data = {

token: {

lexeme: [pos for pos in positions if pos.startswith(f'TXT{entry\_number}-')]

for lexeme, positions in lexemes.items()

if any(pos.startswith(f'TXT{entry\_number}-') for pos in positions)

}

for token, lexemes in data\_dict['POSICIONES'].items()

}

with open(file\_path, 'w', encoding='utf-8') as file:

json.dump(output\_data, file, ensure\_ascii=False, indent=4)

print("Archivo de salida generado exitosamente.")

# Función para mostrar información estadística

def display\_statistics(found\_lexemes, new\_lexemes, prev\_lexemes\_count, new\_lexemes\_count):

total\_lexemes = len(found\_lexemes)

unprocessed\_lexemes = len(new\_lexemes)

processed\_lexemes = total\_lexemes - unprocessed\_lexemes

print("\n----------------------------------------------------")

print(f"Total de lexemas en el texto actual: {total\_lexemes}")

print(f"Lexemas procesados: {processed\_lexemes} ({(processed\_lexemes / total\_lexemes) \* 100:.2f}%)")

print(f"Lexemas no procesados: {unprocessed\_lexemes} ({(unprocessed\_lexemes / total\_lexemes) \* 100:.2f}%)")

print("----------------------------------------------------")

for token in data\_dict['POSICIONES']:

print(f"\n{token}:")

print(f" Lexemas antes del procesamiento: {prev\_lexemes\_count.get(token, 0)}")

print(f" Lexemas añadidos en este archivo: {new\_lexemes\_count.get(token, 0)}")

print(f" Total de lexemas: {len(data\_dict['POSICIONES'][token])}")

# Función principal para ejecutar el tokenizador

def main():

data\_dict\_file = 'data\_dict.json'

if load\_data\_dict(data\_dict\_file):

print("Diccionario de datos cargado exitosamente.")

else:

print("No se encontró el diccionario de datos existente. Comenzando con uno nuevo.")

if data\_dict['num\_files\_processed'] == 0 and not data\_dict['predefined\_lexemes\_used']:

use\_predefined\_patterns = input("¿Desea usar lexemas predefinidos? (sí/no): ").strip().lower() == 'sí'

if use\_predefined\_patterns:

initialize\_with\_lexemes()

print("Diccionario de datos inicializado con lexemas predefinidos.")

else:

print("Los lexemas predefinidos ya han sido utilizados o no es la primera iteración. Omitiendo inicialización.")

input\_file = input("Ingrese la ruta del archivo de entrada: ").strip() # Obtener la ruta del archivo de entrada del usuario

entry\_number = data\_dict['num\_files\_processed'] + 1 # Incrementar el número de entrada basado en los archivos procesados

output\_file = f'output{entry\_number}.txt' # Nombre del archivo de salida basado en el número de entrada

# Contar la cantidad de lexemas antes del procesamiento

prev\_lexemes\_count = {token: len(lexemes) for token, lexemes in data\_dict['POSICIONES'].items()}

found\_lexemes, new\_lexemes = tokenize\_text(input\_file, entry\_number)

# Contar la cantidad de lexemas después del procesamiento

new\_lexemes\_count = {token: len(data\_dict['POSICIONES'][token]) - prev\_lexemes\_count.get(token, 0) for token in data\_dict['POSICIONES']}

save\_data\_dict(data\_dict\_file)

generate\_output\_file(output\_file, entry\_number)

display\_statistics(found\_lexemes, new\_lexemes, prev\_lexemes\_count, new\_lexemes\_count)

data\_dict['num\_files\_processed'] = entry\_number # Actualizar el número de archivos procesados

save\_data\_dict(data\_dict\_file) # Guardar el diccionario de datos actualizado

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Resultados y Observaciones**

El tokenizador desarrollado fue probado con un texto de ejemplo en español. El proceso de tokenización identificó correctamente la mayoría de los lexemas y los clasificó en los tokens correspondientes. Se presentaron algunos lexemas que no pudieron ser clasificados automáticamente y requirieron intervención del usuario.

**Ejemplo de Resultado**

El diccionario de datos resultante mostró una estructura organizada de los lexemas y sus posiciones en el texto. A continuación, se muestra el resultado del diccionario para la entrada “Nietzsche escribió sobre temas varios y asombrosos despampanantes”, con un diccionario de datos pre-cargado:

{

"POSICIONES": {

"ARTICULO": {

"el": [],

"la": [],

"los": [],

"las": [],

"un": [],

"una": [],

"unos": [],

"unas": []

},

"SUSTANTIVO": {

"nietzsche": [

"TXT1-1"

],

"temas": [

"TXT1-4"

],

"libro": [],

"mundo": [],

"persona": []

},

"VERBO": {

"escribir": [],

"leer": [],

"ser": [],

"haber": [],

"ir": [],

"escribió": [

"TXT1-2"

],

"escriben": []

},

"ADJETIVO": {

"grande": [],

"pequeño": [],

"bueno": [],

"malo": [],

"nuevo": [],

"viejo": [],

"asombrosos": [

"TXT1-7"

],

"despampanantes": [

"TXT1-8"

]

},

"ADVERBIO": {

"rápidamente": [],

"lentamente": [],

"bien": [],

"mal": [],

"cerca": [],

"lejos": [],

"varios": [

"TXT1-5"

]

},

"OTROS": {

"sobre": [

"TXT1-3"

],

"con": [],

"sin": [],

"en": [],

"por": [],

"para": [],

"y": [

"TXT1-6"

],

"o": [],

"a": [],

"de": []

}

},

"num\_files\_processed": 1,

"predefined\_lexemes\_used": true

}

Mientras que el archivo de salidad output1.txt luce así: {

"ARTICULO": {},

"SUSTANTIVO": {

"nietzsche": [

"TXT1-1"

],

"temas": [

"TXT1-4"

]

},

"VERBO": {

"escribió": [

"TXT1-2"

]

},

"ADJETIVO": {

"asombrosos": [

"TXT1-7"

],

"despampanantes": [

"TXT1-8"

]

},

"ADVERBIO": {

"varios": [

"TXT1-5"

]

},

"OTROS": {

"sobre": [

"TXT1-3"

],

"y": [

"TXT1-6"

]

}

}

**Conclusiones**

El desarrollo de un tokenizador mínimo para el procesamiento de lenguaje natural es una tarea fundamental en la implementación de sistemas de PLN. A través de este trabajo práctico, se ha demostrado la importancia de manejar las ambigüedades y variaciones en los lenguajes humanos, y cómo herramientas como los tokenizadores pueden simplificar el análisis sintáctico posterior. Además, se ha resaltado la necesidad de actualizar y mejorar continuamente el diccionario de datos para adaptarse a nuevos textos y contextos.

La experiencia obtenida en este proyecto proporciona una comprensión más profunda del impacto del análisis léxico en el procesamiento de lenguajes y establece una base sólida para futuros desarrollos en esta área.