Evaluación unidad III

DETECCIÓN DE PLAGIO

Descripción y objetivos

Detectar similitudes entre documentos de texto, simulando y analizando casos de plagio mediante el uso de **n-gramas, **funciones hash personalizadas** y **visualización de grafos**.**

Mostrar de forma clara cuándo dos documentos son similares, y visualizar esas relaciones para facilitar la detección de plagios potenciales.

Requisitos para ejecutar el detector

Contar con python 3.7 en adelante

Instalar librerías requeridas:

pip install networkx matplotlib

o

- **pip install networkx**
- **pip install matplotlib**
- **pip install mmh3**

MAIN

```python
import os
import time
from src.preprocessing import preprocess_documents
from src.similarity import calculate_jaccard_similarity
from src.sorting import merge_sort
from src.visualization import generate_similarity_graph
from document_generator import generate_documents
```

```python
def main():
    start_time = time.time()

    # Paso 1: Generar documentos sintéticos
    print("Generando 100 documentos...")
    generate_documents(num_docs=100)

    # Paso 2: Cargar documentos (con manejo de codificación)
    docs_dir = 'documentos'
    documents = {}
    try:
        for filename in os.listdir(docs_dir):
            with open(os.path.join(docs_dir, filename), 'r',
                      encoding='utf-8',
                      errors='replace') as f:  # Ignorar bytes inválidos
                documents[filename] = f.read().strip()  # Eliminar espacios extras
    except Exception as e:
        print(f"Error al cargar documentos: {e}")
        return
```

```python
    # Paso 3: Preprocesamiento con hashing y Bloom Filters
    print("\nPreprocesando documentos...")
    processed_docs = preprocess_documents(documents, n=3)

    # Paso 4: Mostrar información de hashes y Bloom Filters
    print("\n=== HASHES Y BLOOM FILTERS ===")
    for doc_name, data in processed_docs.items():
        print(f"\nDocumento: {doc_name}")
        print(f"Cantidad de hashes: {len(data['hashes'])}")
        print(f"Bits activos en Bloom Filter: {sum(data['bloom'].bit_array)}")  # Conteo de bits True

    # Paso 5: Calcular similitudes entre todos los pares
    print("\nCalculando similitudes...")
    similarities = []
    for doc1 in processed_docs:
        for doc2 in processed_docs:
            if doc1 < doc2:  # Evitar duplicados
                sim = calculate_jaccard_similarity(
                    processed_docs[doc1]['hashes'],
                    processed_docs[doc2]['hashes']
                )
                similarities.append((doc1, doc2, sim))
```

```python
    # Paso 6: Ordenar con Merge Sort personalizado
    print("\nOrdenando resultados...")
    sorted_sims = merge_sort(similarities, key=lambda x: -x[2])

    # Paso 7: Mostrar top 5
    print("\nTop 5 pares más similares:")
    for pair in sorted_sims[:5]:
        print(f"- {pair[0]} y {pair[1]}: {pair[2]:.2f}")

    # Paso 8: Generar visualización
    print("\nGenerando gráfico de similitudes...")
    timestamp = time.strftime("%Y%m%d_%H%M%S")
    sorted_sims = merge_sort(similarities, key=lambda x: -x[2])
    generate_similarity_graph(sorted_sims, filename=f"similitudes_{timestamp}.png")

    print(f"\nTiempo total: {time.time() - start_time:.2f} segundos")

if __name__ == "__main__":
    main()
```

GENERADOR DE DOCUMENTOS

```python
# document_generator.py
import os
import random
```

```python
subjects = ["Estudiante", "Profesor", "Investigador", "Empresa", "Gobierno"]
verbs = ["analizó", "desarrolló", "implementó", "investigó", "presentó"]
objects = ["algoritmo", "modelo", "sistema", "solución", "proyecto"]
adjectives = ["innovador", "eficiente", "complejo", "sostenible", "tecnológico"]
adverbs = ["rápidamente", "eficazmente", "precisamente", "exitosamente"]

def generate_sentence():
    # Genera frases con estructuras variadas
    structure = random.choice([
        f"{random.choice(subjects)} {random.choice(verbs)} un {random.choice(adjectives)} {random.choice(objects)} {random.choice(adverbs)}",
        f"{random.choice(adverbs)} {random.choice(subjects)} {random.choice(verbs)} un {random.choice(objects)} {random.choice(adjectives)}",
        f"{random.choice(subjects)} {random.choice(verbs)} un {random.choice(objects)} {random.choice(adjectives)} {random.choice(adverbs)}"
    ])

    # Introduce variabilidad:
    words = structure.split()
    if random.random() < 0.3:
        words.pop(random.randint(0, len(words)-1))  # Elimina una palabra al azar
    return ' '.join(words)
```

```python
def generate_documents(num_docs=100):
    if not os.path.exists("documentos"):
        os.makedirs("documentos")

    # Eliminar documentos anteriores
    for f in os.listdir("documentos"):
        os.remove(os.path.join("documentos", f))

    # Generar documentos con niveles de plagio aleatorios
    base_sentences = [generate_sentence() for _ in range(200)]  # Base de frases
```

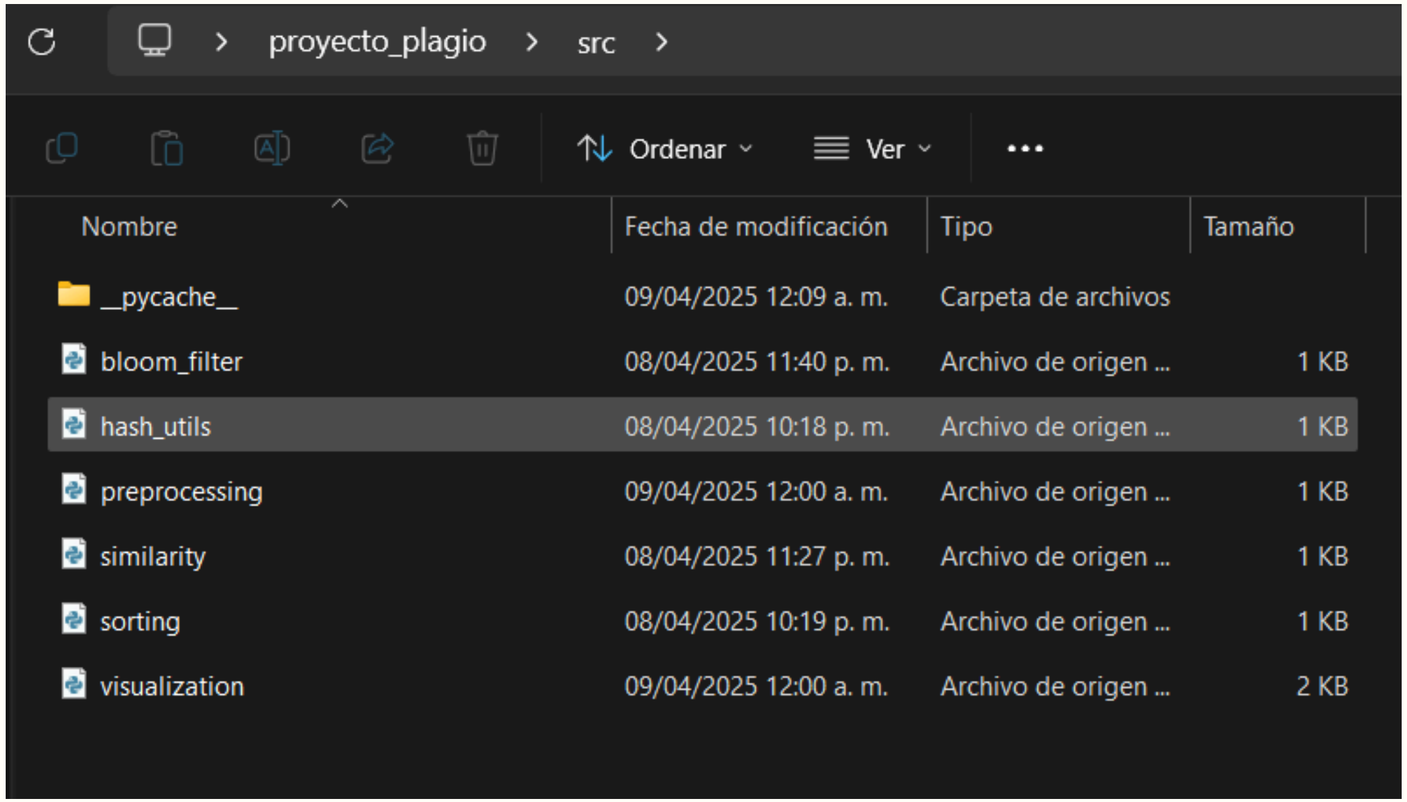
Los documentos generados son almacenados en la carpeta recién generada “documentos”.

GENERADOR DE DOCUMENTOS

```python
for i in range(1, num_docs+1):
    plagiarism_level = random.randint(0, 100)  # Porcentaje de plagio

    # Mezclar contenido original y plagiado
    if i == 1:
        # Documento base (100% original)
        sentences = random.sample(base_sentences, 100)
    else:
        # Documentos derivados
        plagiarized = random.sample(base_sentences, int(100 * (plagiarism_level / 100)))
        original = [generate_sentence() for _ in range(100 - len(plagiarized))]
        sentences = plagiarized + original
        random.shuffle(sentences)  # Mezclar para simular mejor el plagio

    # Guardar documento
    with open(f"documentos/documento_{i}.txt", "w", encoding='utf-8') as f:
        f.write(f"Porcentaje de plagio: {plagiarism_level}%\n")
        for sentence in sentences:
            f.write(sentence + "\n")

    print(f"Generado: documento_{i}.txt ({plagiarism_level}% de plagio)")
```

Detector de plagio

En la carpeta src se encuentran las partes del detector:



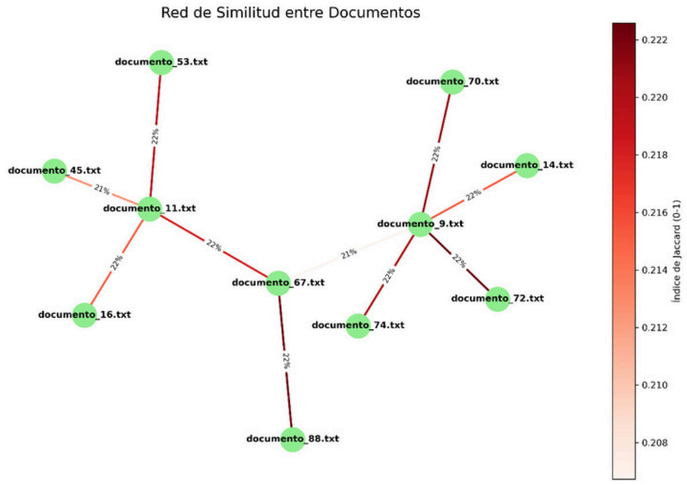
Resultados

Los resultados se pueden observar de la siguiente manera:

Top 10 documentos más similares en la consola:

```
Top 10 documentos más similares:

doc_100%_149.txt <-> doc_100%_150.txt | Similitud: 100.00% | 🟢 100%
doc_100%_148.txt <-> doc_100%_150.txt | Similitud: 100.00% | 🟢 100%
doc_100%_148.txt <-> doc_100%_149.txt | Similitud: 100.00% | 🟢 100%
doc_100%_147.txt <-> doc_100%_150.txt | Similitud: 100.00% | 🟢 100%
doc_100%_147.txt <-> doc_100%_149.txt | Similitud: 100.00% | 🟢 100%
doc_100%_147.txt <-> doc_100%_148.txt | Similitud: 100.00% | 🟢 100%
doc_100%_146.txt <-> doc_100%_150.txt | Similitud: 100.00% | 🟢 100%
doc_100%_146.txt <-> doc_100%_149.txt | Similitud: 100.00% | 🟢 100%
doc_100%_146.txt <-> doc_100%_148.txt | Similitud: 100.00% | 🟢 100%
doc_100%_146.txt <-> doc_100%_147.txt | Similitud: 100.00% | 🟢 100%

Grafo guardado como: resultados/grafo_20250409_015416.png
```

Grafo guardado en la carpeta “resultados” generada en main:



Muchas

GRACIAS