Taller 3 y 4 ”Count Word”

Daniel Osorio Valencia - 614222007

I. Introducción

En este taller se busca la implementación de dos algoritmos que permitan encontrar, contar y mostrar que palabras se repiten con mas frecuencia. Por otro lado, el otro algoritmo deberá devolver una nueva lista con los documentos relacionados a una palabra que se le ongrese como parámetro.

II. Código

A. Algoritmo contador de palabras

```python
from collections import Counter
import time

1 usage
def archivo(name_archivo):
    with open(name_archivo, 'r', encoding='utf-8') as archivo:
        texto = archivo.read()
    return texto

name_archivo = "documento.txt"
texto = archivo(name_archivo)

1 usage
def cont_palabras_repetidas(texto, memo={}):
    if texto in memo:
        return memo[texto]

    cont = contar_palabras(texto)

    palabras4caracteres = [palabra for palabra in cont.keys() if len(palabra) > 4]
```

```python
    contador_filtrado = {palabra: cont[palabra] for palabra in palabras4caracteres}

    if not contador_filtrado:
        return []

    palabrasReincidentes = sorted(contador_filtrado.items(), key=lambda x: x[1], reverse=True)

    memo[texto] = palabrasReincidentes
    return palabrasReincidentes

1 usage
def contar_palabras(texto):
    palabras = texto.split()
    contador = Counter(palabras)
    return contador
```

```python
1 usage
def imprimir_reincidentes(podio):
    print("Palabras más repetidas:")
    for i, (palabra, frecuencia) in enumerate(podio, 1):
        print(f"{i}. {palabra}: {frecuencia} veces")

if texto:
    start_time = time.time()  # Captura el tiempo de inicio
    palabrasReincidentes = cont_palabras_repetidas(texto)
    end_time = time.time()  # Captura el tiempo de finalización
    if palabrasReincidentes:
        podio = palabrasReincidentes[:5]
        imprimir_reincidentes(podio)
    else:
        print("No se encontraron palabras repetidas.")
    elapsed_time = end_time - start_time  # Calcula el tiempo transcurrido
    print(f"Tiempo de ejecución: {elapsed_time} segundos")
else:
    print(f"No se pudo encontrar el archivo '{name_archivo}'")
```

el propósito de este algoritmo es contar y mostrar las palabras más repetidas en un texto, utilizando una biblioteca estándar de Python de tipo modular. También se implementa una memoria caché para mejorar la eficiencia y finalmente se muestra el resultado y el tiempo de ejecución.

Resultado

```
C:\Users\dosor\AppData\Loc
Palabras más repetidas:
1. código: 14 veces
2. aplicaciones: 12 veces
3. datos: 11 veces
4. programación: 9 veces
5. desarrollo: 9 veces
```

Figure 1. Output

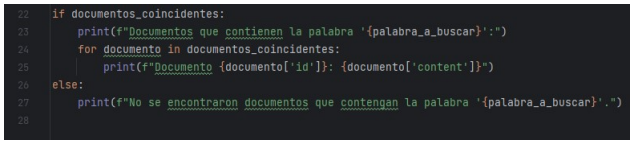
```
C:\Users\dosor\AppData\Local\Microsoft\WindowsApps\python3.11.exe C:\Users\dosor\Downloads\taller3\taller3\taller3punto1.py
Palabras más repetidas:
1. código: 14 veces
2. aplicaciones: 12 veces
3. datos: 11 veces
4. programación: 9 veces
5. desarrollo: 9 veces
Tiempo de ejecución: 0.000244140625 segundos

Process finished with exit code 0
```

Figure 2. Output

B. Algoritmo de Búsqueda de Repetición de Palabras

```python
def cargar_documentos(name_archivo):
    with open(name_archivo, 'r', encoding='utf-8') as archivo:
        lineas = archivo.readlines()
    documentos = [{"id": i, "content": linea.strip()} for i, linea in enumerate(lineas, 1)]
    return documentos

1 usage
def buscar_documentos_por_palabra(palabra, documentos):
    documentos_coincidentes = []

    for documento in documentos:
        if palabra.lower() in documento["content"].lower():
            documentos_coincidentes.append(documento)

    return documentos_coincidentes

name_archivo = "documento.txt"
documentos = cargar_documentos(name_archivo)

palabra_a_buscar = "buscar"
documentos_coincidentes = buscar_documentos_por_palabra(palabra_a_buscar, documentos)
```

Figure 3. Search-Like

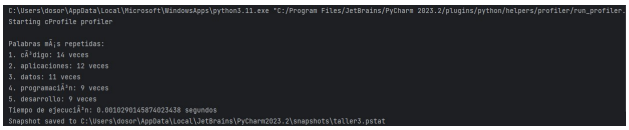


```
if documentos_coincidentes:
    print(f"Documentos que contienen la palabra '{palabra_a_buscar}':")
    for documento in documentos_coincidentes:
        print(f"Documento {documento['id']}: {documento['content']}")
else:
    print(f"No se encontraron documentos que contengan la palabra '{palabra_a_buscar}'.")

```

Figure 4. Search-Like

El propósito de este algoritmo es encontrar documentos que contengan una palabra específica en un archivo de texto, cargando el contenido del archivo en una estructura de datos y proporcionando una función de búsqueda simple.

C. Complejidad Espacio-Tiempo



```
C:\Users\dosor\AppData\Local\Microsoft\WindowsApps\python3.11.exe "C:/Program Files/JetBrains/PyCharm 2023.2/plugins/python/helpers/profiler/run_profiler.
Starting cProfile profiler

Palabras mÃ¡s repetidas:
1. cÃ³digo: 14 veces
2. aplicaciones: 12 veces
3. datos: 11 veces
4. programaciÃ³n: 9 veces
5. desarrollo: 9 veces
Tiempo de ejecuciÃ³n: 0.0010290145874023438 segundos
Snapshot saved to C:\Users\dosor\AppData\Local\JetBrains\PyCharm2023.2\snapshots\taller3.pstat
```

Figure 5. Complejidad Temporal

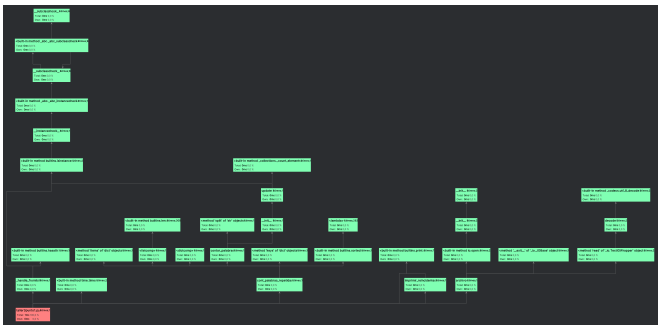


Figure 6. Complejidad Espacial.

D. Conclusiones

El primer código es útil para análisis de texto y procesamiento de datos, proporcionando estadísticas detalladas sobre palabras repetidas. El segundo código es adecuado para tareas de búsqueda y recuperación de información en documentos de texto, siendo más simple y específico. La elección entre ambos depende de la tarea que necesites realizar: análisis de texto detallado o búsqueda de palabras en documentos.