Documentación Temporal Spatial

Daniel F. Gonzalez

I. Introducción

En este tercer taller desarrollamos 2 algoritmos simples, en los que hicimos uso de los conceptos desarrollados en clase sobre maneras de programar distintos algoritmos, el primero se enfoca en encontrar las palabras más repetidas en un conjunto de documentos y el segundo permite buscar documentos que tengan una palabra específica, a continuación describiré el proceso cuál de ambos algoritmos de una manera sencilla

II. Código

A. Algoritmo contador de palabras

```python
def traer_archivo(nombre_archivo):
    with open(nombre_archivo, 'r', encoding='utf-8') as archivo:
        texto = archivo.read()
    return texto

nombre_archivo = "documento.txt"
texto = traer_archivo(nombre_archivo)
```

Figure 1.

En la primera sección del código lo que hice fue crear una función que traiga el archivo TXT y lo guarde cómo archivo de texto, esto para que el código se vea más limpio y corto

```python
def palabras_mas_repetidas(texto, memo={}):
    if texto in memo:
        return memo[texto]

    contador = contar_palabras(texto)

    palabras5caracteres = [palabra for palabra in contador.keys() if len(palabra) > 5]

    contador_filtrado = {palabra: contador[palabra] for palabra in palabras5caracteres}

    if not contador_filtrado:
        return []

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

    memo[texto] = palabrasRepetidas
    return palabrasRepetidas
```

Figure 2.

Esta es la parte principal del codigo, aqui hacemos uso de memorization preguntando en primera instancia si la palabra esta en el diccionario memo, tuve que hacer una excepcion en el codigo para que palabras muy cortas como los conectores no sean tomados en la ejecucion, despues creamos un nuevo diccionario que almacene las palabras mas grandes de 5 caracteres y su repeticion a modo de tuplas y por ultimo organizamos el diccionario de tal manera que se vea de mayor a menor

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

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

Figure 3.

Por ultimo tenemos la funcion contador que se usa en el metodo anterior y para finzalizar tenemos la impresion de las 5 palabras mas repetidas

```python
if texto:
    start_time = time.time()
    palabrasRepetidas = palabras_mas_repetidas(texto)
    end_time = time.time()

    if palabrasRepetidas:
        podio = palabrasRepetidas[:5]
        imprimir_palabras_repetidas(podio)
    else:
        print("No se encontraron palabras repetidas en el texto.")
else:
    print(f"No se pudo encontrar el archivo '{nombre_archivo}'. Verifica que el archivo exista y esté en la ubic

execution_time = end_time - start_time
print(f"Tiempo de ejecución: {execution_time:.6f} segundos")
```

Figure 4.

Esto es la parte logica que maneja las ecepciones y la impresion del tiempo de ejecucion del algoritmo. Resultado

```
Palabras mÃ¡s repetidas:
1. cÃ³digo: 14 veces
2. aplicaciones: 12 veces
3. programaciÃ³n: 9 veces
4. desarrollo: 9 veces
5. software: 7 veces
Tiempo de ejecuciÃ³n: 0.001000 segundos
Snapshot saved to C:\Users\elpro\AppData\Local\JetBrains\PyCharm2023.2\snapshots\taller3.pstat
```

Figure 5.

Este algoritmo tiene un tiempo de ejecucion de 0.001 segundos y nos imprime las palabras mas repetidas.

1. código: 14 veces
2. aplicaciones: 12 veces
3. programación: 9 veces
4. desarrollo: 9 veces
5. software: 7 veces

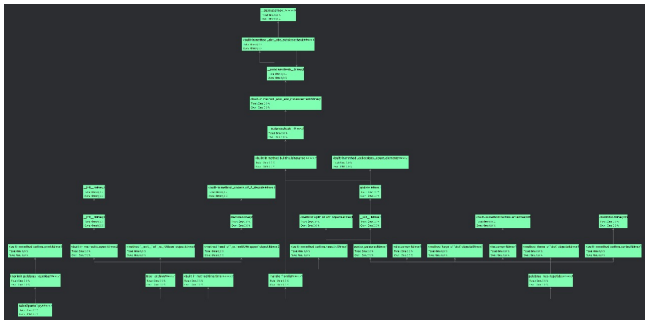

Figure 6. Diagrama de Ejecucion.

B. Algoritmo de Búsqueda de Repetición de Palabras

```python
def traer_archivo(nombre_archivo):
    with open(nombre_archivo, 'r', encoding='utf-8') as archivo:
        texto = archivo.read()
    return texto

1 usage
def buscar_en_texto(palabras, texto):
    resultados = {}

    for palabra in palabras:
        resultados[palabra] = []
        for i, linea in enumerate(texto.split('\n')):
            if palabra in linea:
                resultados[palabra].append(i + 1)
    return resultados
```

Figure 7.

En la primera sección del código lo que hicie fue crear una función que traiga el archivo TXT y lo guarde cómo archivo de texto, esto para que el código se vea más limpio y corto. Luego, cree una funcion que recorra posicion por posicion y revise si la palabra que estamos evaluando es igual a la palabras que se definieron mas abajo en el codigo.

```python
nombre_archivo = "documento.txt"
texto = traer_archivo(nombre_archivo)
palabras_buscadas = ["código", "aplicaciones", "programación", "desarrollo", "software"]

inicio_tiempo = time.time()
resultados = buscar_en_texto(palabras_buscadas, texto)
fin_tiempo = time.time()
tiempo_ejecucion = fin_tiempo - inicio_tiempo

for palabra, documentos in resultados.items():
    print(f'"{palabra}": {documentos}')

print(f"Tiempo de ejecución: {tiempo_ejecucion} segundos")
```

Figure 8.

Aqui esta la parte logica que imprime el tiempo de ejecucion, la lista de palabras a buscar y la manera de imprimir los resultados.

```
C:\Users\elpro\AppData\Local\Programs\Python\Python39\python.exe "C:/Program Files/JetBrains/PyCh
Starting cProfile profiler

"cÃ³digo": [11, 14, 15, 16, 17, 19, 21, 23, 25, 27, 32, 49, 56, 63]
"aplicaciones": [4, 6, 20, 40, 44, 45, 47, 52, 59, 60, 64, 66]
"programaciÃ³n": [1, 5, 7, 17, 18, 26, 29, 38, 46]
"desarrollo": [3, 6, 12, 24, 27, 34, 36, 53, 69]
"software": [3, 12, 13, 27, 30, 54, 62]
Tiempo de ejecuciÃ³n: 0.0010011196136474 61 segundos
Snapshot saved to C:\Users\elpro\AppData\Local\JetBrains\PyCharm2023.2\snapshots\punto 21.pstat
```

Figure 9. a

El codigo tiene una ejecucion de 0.00100111 segundos, para 5 palabras a buscar en un texto corto.

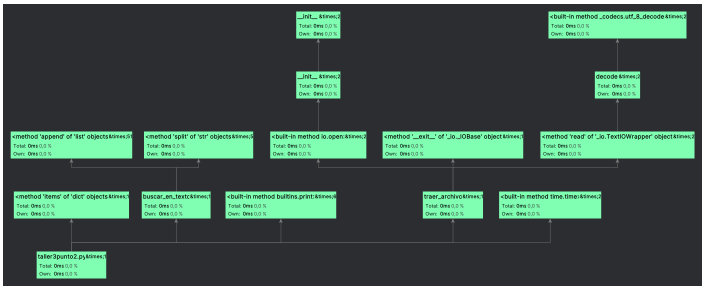

Figure 10.

III. Conclusiones

Las conclusiones de nuestro ejercicio es que podemos implementar la técnica de memorization en distintas clases de algoritmos donde en la mayoría de casos tengamos que recorrer una lista o recorrer algo en específico en busca de aumentar un contador o buscar una palabra en un determinado texto. También podemos concluir que el tiempo de ejecución de estos 2 ejercicios es bastante corto usando esa técnica.