Informe de Complejidad de Algoritmos: **count_word**

Sebastian Samaniego

September 2023

Estructura de Datos

1 Introducción

En este informe, analizaremos la complejidad de tiempo y espacio de dos algoritmos diferentes. El algoritmo **"documents_containing"** crea un índice invertido a partir de una lista de documentos, mientras que el algoritmo **"most_repeated_words"** cuenta la frecuencia de palabras en una lista de documentos y genera un ranking de las palabras más repetidas. A continuación, se presentan los análisis de ambos algoritmos.

2 Análisis algoritmo documents_containing

2.1 Descripción

El algoritmo tiene como objetivo crear un índice invertido a partir de una lista de documentos. El índice invertido es una estructura de datos que asocia palabras con los documentos en los que aparecen. El algoritmo consta de dos funciones principales: `create_inverted_index` y `search_documents_with_words`.

2.1.1 `create_inverted_index`

La función `create_inverted_index` recorre todos los documentos y crea un índice invertido de palabras. Para cada palabra en cada documento, agrega el índice del documento a la lista de documentos en los que aparece esa palabra.

2.1.2 `search_documents_with_words`

La función `search_documents_with_words` simplemente copia el índice invertido en un nuevo diccionario para buscar documentos que contengan cada palabra.

2.2 Complejidad de Tiempo

La complejidad de tiempo del algoritmo se puede expresar como $O(n \cdot m)$ para la creación del índice invertido y $O(k)$ para buscar documentos con palabras específicas, donde n es el número de documentos, m es el número total de palabras en todos los documentos y k es el número de palabras únicas en los documentos.

2.3 Complejidad de Espacio

La complejidad de espacio está relacionada con la cantidad total de palabras únicas en todos los documentos y el número de documentos en los que aparece cada palabra.

3 Análisis algoritmo most_repeated_words

3.1 Descripción

El algoritmo tiene como objetivo contar la frecuencia de palabras en una lista de documentos y generar un ranking de las palabras más repetidas en todos los documentos. El algoritmo consta de las siguientes partes: `contar_palabras` y `ranking_palabras`.

3.1.1 `contar_palabras`

La función `contar_palabras` recorre cada palabra en un documento y actualiza la frecuencia en un diccionario.

3.1.2 `ranking_palabras`

La función `ranking_palabras` utiliza la memoización para evitar recalcular las frecuencias de palabras en los documentos. Luego, itera a través de los documentos, actualiza la frecuencia total de cada palabra y genera un ranking de palabras más repetidas.

3.2 Complejidad de Tiempo

La complejidad de tiempo del algoritmo se puede expresar como $O(m \cdot n + k \log k)$, donde m es el número de documentos, n es el número promedio de palabras en un documento y k es el número total de palabras únicas en todos los documentos.

3.3 Complejidad de Espacio

La complejidad de espacio está relacionada principalmente con el número total de palabras únicas en todos los documentos, es decir, $O(k)$.

4 Conclusiones

Ambos algoritmos son eficientes para sus respectivas tareas. El algoritmo **"documents_containing"** es adecuado para crear un índice invertido a partir de una lista de documentos, mientras que el algoritmo **"most_repeated_words"** es útil para contar la frecuencia de palabras y generar un ranking. Sus complejidades de tiempo y espacio dependen de la naturaleza de los datos y las operaciones realizadas. En general, estos algoritmos son adecuados para conjuntos de datos de tamaño moderado y tienen un buen rendimiento en términos de tiempo y espacio.

5 Anexo

Repositorio: https://github.com/SebastianSamanieg/count_word