Análisis de Algoritmos y Estructuras para Datos Masivos 2023

Unidad 8 - Proyecto integrador

David Aarón Ramírez Olmeda

Introducción a los índices invertidos:

Los índices invertidos son estructuras de datos utilizadas para mejorar la eficiencia en la búsqueda de información en grandes conjuntos de documentos. A diferencia de los índices tradicionales, que mapean términos a documentos, los índices invertidos mapean términos a las ubicaciones en las que aparecen en los documentos.

Un índice invertido consta de dos componentes principales: el vocabulario y los postings. El vocabulario es una lista de términos únicos en los documentos, y los postings son las listas de ubicaciones donde aparece cada término. Estos índices son ampliamente utilizados en motores de búsqueda y sistemas de recuperación de información para acelerar el proceso de búsqueda y recuperación de documentos relevantes.

Resumen:

Se presenta la construcción de un índice invertido a partir de una colección de documentos. Se diseñó un algoritmo de intersección binaria para resolver consultas conjuntivas utilizando el índice invertido.

1. Construye un índice invertido a partir de una de las colecciones provistas.

In [1]:
```
import os
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from collections import defaultdict
```

In [2]:
```python
def construir_indice_invertido(collection_folder, files):
    inverted_index = defaultdict(list)

    # Procesamiento de texto y construcción del índice invertido
    for file in files:
        file_path = os.path.join(collection_folder, file)
        with open(file_path, 'r') as f:
            document = f.read()
            tokens = word_tokenize(document.lower())
            tokens = [token for token in tokens if token.isalpha()]
            tokens = [token for token in tokens if token not in stopwords
            for position, token in enumerate(tokens):
                inverted_index[token].append((file, position))

    return inverted_index
```

2. Diseña un algoritmo para resolver consultas conjuntivas (terminoA & termino B & ...) utilizando el algoritmo de intersección binary search

In [3]:
```python
def consulta(query_terms, inverted_index):
    postings_lists = [inverted_index[term] for term in query_terms]
    postings_lists.sort(key=len)
    comparisons = defaultdict(int)

    # Algoritmo de intersección binaria
    result = []
    for i in range(len(postings_lists[0])):
        current_doc = postings_lists[0][i][0]
        for j in range(1, len(postings_lists)):
            postings = postings_lists[j]
            low = 0
            high = len(postings) - 1
            while low <= high:
                mid = (low + high) // 2
                if postings[mid][0] < current_doc:
                    low = mid + 1
                elif postings[mid][0] > current_doc:
                    high = mid - 1
                else:
                    comparisons[current_doc] += 1
                    break
            else:
                break
        else:
            result.append(current_doc)

    return result, comparisons
```

El algoritmo de intersección binaria es eficiente porque aprovecha la propiedad de las listas ordenadas y reduce la cantidad de comparaciones necesarias para encontrar la intersección. Al realizar búsquedas binarias en las listas, se descartan rápidamente las partes que no contienen elementos comunes, reduciendo el tiempo de ejecución en comparación con una búsqueda lineal.

3. Construya el índice invertido.

In [4]:
```
collection_folder = 'datasets'
files = ['politicos.json',
         'pcovid2020.json',
         'quejas2020.json',
         'etiquetadofrontal2020.json']

inverted_index = construir_indice_invertido(collection_folder, files)
```

4. Realizar consultas.

In [6]:
```
# Consulta de términos
query_terms = ['sol', 'playa']
result, comparisons = consulta(query_terms, inverted_index)

for doc in result:
    print(f'Documento: {doc}, Comparaciones: {comparisons[doc]}')
```

```
Documento: politicos.json, Comparaciones: 5
Documento: politicos.json, Comparaciones: 5
Documento: politicos.json, Comparaciones: 5
Documento: politicos.json, Comparaciones: 5
Documento: politicos.json, Comparaciones: 5
Documento: quejas2020.json, Comparaciones: 1
```

5. Realiza una revisión de los algoritmos vistos en el curso, unidad por unidad, repasa los experimentos y comparaciones en cada tema. Discuta los resultados de cada unidad.

...

Explicación de cada unidad y discusión de los resultados experimentales:

...

Conclusiones:

El algoritmo de intersección binaria se mostró como una técnica efectiva para encontrar documentos que contienen todos los términos de una consulta conjuntiva. Su capacidad para realizar búsquedas eficientes en listas ordenadas y reducir la cantidad de comparaciones necesarias mejoró el rendimiento y la eficiencia del proceso de búsqueda.

Referencias:

- Sadit, T. (2021). InvertedFiles.jl. GitHub repository. Retrieved from: https://github.com/sadit/InvertedFiles.jl (https://github.com/sadit/InvertedFiles.jl)
- Sadit, T. (2021). TextSearch.jl. GitHub repository. Retrieved from: https://github.com/sadit/TextSearch.jl (https://github.com/sadit/TextSearch.jl)
- Zobel, J., & Moffat, A. (2014). Inverted files. Foundations and Trends® in Information Retrieval, 8(4), 263-369.

In [ ]: