# Algoritmo de Merge sort

1<sup>st</sup> Juan Esteban Pinzon Hugo Alejandro Latorre José David Ramírez Beltrán Amir Rodriguez Mejia Fundación Universitaria Konrad Lorenz Bogotá, Colombia

Abstract—Este trabajo demuestra la eficacia del algoritmo de Merge Sort en la clasificación de documentos generados aleatoriamente. El algoritmo se divide en dos etapas: la división de documentos en subconjuntos y su fusión ordenada. Esto ofrece una solución eficiente para la organización de información en aplicaciones de procesamiento de texto y recuperación de datos. El estudio subraya el potencial del algoritmo en la mejora de la eficiencia de acceso a datos relevantes en grandes conjuntos de texto.

Index Terms—Optimización, recursos, lógica, complejidad, análisis

#### I. Introducción

En el vasto panorama de la información digital, la capacidad de clasificar y organizar documentos de manera eficiente es esencial para una variedad de aplicaciones, desde motores de búsqueda hasta análisis de texto a gran escala. Este trabajo se adentra en la aplicación del algoritmo de Merge Sort en el contexto de la clasificación de documentos generados aleatoriamente. Merge Sort, conocido por su eficacia en la ordenación, ofrece un enfoque prometedor para mejorar la accesibilidad y la capacidad de análisis de datos relevantes en conjuntos de texto de gran tamaño. A través de este estudio, buscamos destacar la utilidad de los algoritmos de clasificación en el procesamiento de datos textuales en la era digital.

La exploración de la implementación de Merge Sort en la clasificación de documentos proporciona una visión más clara de cómo estas técnicas pueden contribuir a la gestión eficaz de información textual en entornos de procesamiento de lenguaje natural y recuperación de datos. En el próximo análisis, examinaremos en detalle el funcionamiento del algoritmo y su aplicación práctica en el procesamiento de grandes volúmenes de datos de texto, subrayando su potencial para optimizar la organización y el análisis de documentos en un mundo cada vez más digital.

#### II. ANÁLISIS DEL CÓDIGO

## A. Función para Contar Ocurrencias de Palabras

La función contar\_ocurrencias (palabra, diccionario) toma una palabra y un diccionario como entrada. Si la palabra no se encuentra en el diccionario, se devuelve 0, lo que significa que no hay ocurrencias de esa

palabra. En caso de que la palabra esté en el diccionario, la función cuenta las ocurrencias de la palabra en los documentos y devuelve el recuento total.

#### B. Función para Clasificar Palabras por Frecuencia

La función clasificar\_palabras (diccionario) realiza la clasificación de palabras en el diccionario en función de su frecuencia de ocurrencia. Utiliza la función contar\_ocurrencias para determinar la frecuencia de cada palabra y luego ordena las palabras en orden descendente de acuerdo con su frecuencia. Luego, imprime las palabras clasificadas junto con la cantidad de veces que aparecen en los documentos.

#### C. Cargar el Diccionario con Información

Este fragmento de código carga el diccionario con información a partir de una lista de documentos llamada my\_documents. Itera a través de los documentos y divide cada documento en palabras. Luego, agrega la información al diccionario, asignando a cada palabra una lista de índices de documentos en los que aparece. Si la palabra no existe en el diccionario, se crea una entrada para ella.

### D. Función para Buscar una Palabra en el Diccionario

La función buscar (diccionario, palabra\_a\_buscar) permite buscar una palabra específica en el diccionario. Si la palabra se encuentra en el diccionario, se devuelve una lista de índices de documentos en los que aparece. Si la palabra no se encuentra en el diccionario, se devuelve una lista vacía.

# E. Buscar una Palabra Específica

En este fragmento, se establece la palabra específica que se desea buscar, que en este caso es "Python". Luego, se utiliza la función buscar para encontrar los documentos que contienen la palabra "Python" y se muestran en la salida estándar.

Estos elementos del código trabajan juntos para contar, clasificar y buscar palabras en una colección de documentos, lo que puede ser útil en tareas de procesamiento de lenguaje natural y análisis de texto.

```
△ Untitled1.ipynb ☆

       Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda Se han gu
       0
           import random
            def merge_sort(documento):
{x}
izquierda = merge_sort(documento[:medio])
               derecha = merge_sort(documento[medio:])
               return merge(izquierda, derecha)
            def merge(izquierda, derecha):
               while izquierda and derecha:
                   if izquierda[0] < derecha[0]:
                       resultado.append(izquierda.pop(0))
                       resultado.append(derecha.pop(0))
return resultado
```

Fig. 1. Importaciones y creación de biblioteca para los datos.

```
def generar documentos():
   for i in range(500):
        documento = {
            "texto": random.choice(
                 "El sol brilla en el cielo.".
    "Los pájaros cantan en el parque.",
    "Aprender programación es divertido.",
    "Me encanta la comida italiana.",
    "Caminar en la playa es relajante.",
    "Los libros son una fuente de conocimiento.",
    "La amistad es un tesoro invaluable."
    "Disfruta cada momento de la vida.".
    "La familia es lo más importante.".
    "La naturaleza es asombrosa.",
    "Nunca deies de aprender.".
    "La paciencia es una virtud.",
    "La perseverancia lleva al éxito.",
    "Haz lo que amas y no trabajarás un día en tu vida.",
"La belleza está en los ojos del espectador.",
```

Fig. 2. Se realiza el ciclo para que pase por cada una de las palabras.

#### III. CONCLUSIONES

 el código presentado es una herramienta versátil y eficiente para el procesamiento de grandes conjuntos de datos de texto. Sus funciones permiten contar las ocurrencias de palabras, clasificarlas en función de su frecuencia y buscar palabras específicas en una colección de documentos. La capacidad de cargar datos de manera eficiente en un diccionario y llevar a cabo búsquedas lo convierte en una herramienta valiosa en campos como

```
"La paciencia es una virtud valiosa.",

"La familia es un refugio seguro.",

"La confianza se gana con acciones.",

"La lealtad es un tesoro.",

"La ética guía nuestras decisiones.",

"La confianza es la base de toda relación.",

"La lealtad es una promesa sincera.",

"La ética guía nuestras acciones.",

"La libertad es un derecho sagrado.",

"La perseverancia conduce al éxito.",

"La generosidad abre puertas.",

"El respeto mutuo es esencial.",

"La gratitud nos llena de alegría.",

"La amistad es un tesoro inestim"

| )

| )

# Devuelve la frase

yield documento["texto"]

# Crea un array con las frases generadas

frases = list(generar_documentos())

# Imprime el array

print(frases)
```

Fig. 3. Clasificación de palabras.

```
## Imprime las frases verticalmente for frase in frases: print(frase)

### Imprime las frases: print(frase)

### It sol brills en el cielo.

### It sol brills en el cielo praque.

### It sol brills en el cielo praque.

### It sol brills en el cielo.

#### It so
```

Fig. 4. Palabras más repetidas.

el procesamiento del lenguaje natural y la recuperación de información.

Este código proporciona una base sólida para proyectos de investigación y desarrollo en el campo del procesamiento de lenguaje natural y la recuperación de información, lo que puede resultar en una comprensión más profunda y la extracción de información valiosa de grandes volúmenes de datos de texto. Su flexibilidad .