Ingeniería en Computadores

CE-1102

Proyecto II – Text Finder

Documento de diseño

Profesor:
Isaac Ramírez Herrera

Estudiantes:

Jian Yong Zheng Wu #2023058713
Mauro Brenes Brenes #2023213314
Andrés Mauricio Alfaro Mayorga #2023396028
Alejandro Benavides Juárez #2021543778

Grupo 01

26 del 2024

Tabla de Contenidos

Introducción

Este documento describe el diseño y la implementación de "Text Finder", una aplicación en Java para buscar texto en documentos. Utiliza estructuras de datos como listas enlazadas y árboles, así como algoritmos de ordenamiento. La aplicación permite gestionar una biblioteca de documentos, indexarlos para búsqueda eficiente y realizar búsquedas precisas por texto. Se destaca la capacidad de buscar frases exactas y abrir documentos en la posición de las ocurrencias. El proyecto busca ofrecer una solución integral y amigable para la gestión y búsqueda de información en diversos formatos de archivo.

Descripción de problema

Se requiere un programa que busque palabras o frases dentro de una biblioteca de documentos proporcionada por el usuario. El programa solo maneja archivos con

extensiones mencionadas en las especificaciones (.pdf, .txt y .docx), cualquier otro tipo de archivo no será válido. El usuario puede ordenar los resultados, para lo que se ofrecen tres tipos de ordenamiento: nombre de archivo, fecha de creación y tamaño del archivo. La búsqueda de las palabras o frases debe ser precisa de forma que solo muestre documentos con la palabra exacta, tal como se escribió. Finalmente, el usuario puede abrir el documento en la posición donde aparecen las ocurrencias desde la aplicación.

Descripción de la solución propuesta

La solución se implementará en Java, utilizando `FileInputStream` la biblioteca `java.io` para leer el contenido de los archivos de texto. Se creará un método que recibirá los bytes leídos por el `FileInputStream`, los transformará en un `String` para separar las palabras del contenido del archivo utilizando los delimitadores proporcionados por el `Scanner` de la biblioteca `java.util`, y así guardar su índice, es decir, su posición dentro del archivo. De esta manera, las palabras encontradas junto con sus ocurrencias se agregarán directamente a un árbol AVL, minimizando las búsquedas posteriores. Para evitar repeticiones, se mantendrá una lista temporal de palabras ya encontradas, donde las repeticiones se agregarán a sus ocurrencias existentes y las nuevas palabras se añadirán como nuevos nodos al árbol.

Además, se guardarán los archivos en la biblioteca de archivos de la aplicación utilizando una clase `Documento` que contenga todos los datos importantes de este. Cuando el usuario seleccione archivos de texto en el File Explorer, solo se permitirán directorios o archivos individuales con las extensiones permitidas. Si se selecciona un directorio, solo se agregarán los archivos con extensiones válidas. Si el directorio está vacío, se enviará un mensaje de error al usuario. Estas verificaciones se realizarán mediante la clase `File` de `java.io`.

Para la búsqueda de palabras, se tomará la entrada del usuario, ya sea una palabra o una frase. En el caso de las frases, se buscará la primera palabra de la frase y por cada una de sus ocurrencias abrirá el archivo donde se encuentra para obtener las palabras que

le siguen y así compararlas con el resto de la frase. Para realizar la comparación de palabras se utilizará el método `equals` de la clase `String` de Java. Para mostrar la palabra resaltada en color junto con su contexto, se volverá a abrir el documento para obtener el texto circundante. Luego, este resultado se mostrará en la interfaz de usuario.

Para abrir el documento en la posición de la palabra buscada, se utilizará una interfaz gráfica de usuario basada en JavaFX. Se mostrará el contenido del documento en un área de texto donde se resaltará la palabra buscada. El usuario podrá navegar por el texto y ver el contexto alrededor de la palabra resaltada

Decisiones de diseño

Alternativas consideradas

Durante el proceso de diseño del proyecto Text Finder, se tomaron decisiones clave en cuanto a la lectura de archivos, la estructura de datos para almacenar palabras y ocurrencias, y el método de búsqueda de palabras o frases. Algunas de las alternativas consideradas incluyeron:

- Lectura de archivos: se contemplaron dos enfoques principales, el uso de las clases FileReader y BufferedReader de Java, que ofrecen métodos específicos para la lectura de texto de manera orientada a caracteres; y la clase FileInputStream, centrada en la lectura de bytes de archivos, lo cual es esencial para el manejo de archivos binarios como .docx y .pdf.
- Estructura de datos para almacenar palabras y ocurrencias: se evaluaron varias opciones. La primera fue el HashMap, que proporciona un mapeo eficiente de clave-valor, aunque no garantiza un orden específico de las palabras. Otra opción fue el TreeSet, que almacena elementos en orden ascendente, facilitando búsquedas eficientes, pero podría no ser la elección óptima para manejar múltiples ocurrencias de la misma palabra. Por último, se consideró el AVL Tree, que ofrece un árbol binario

de búsqueda balanceado, garantizando un tiempo de búsqueda logarítmico y una estructura ordenada.

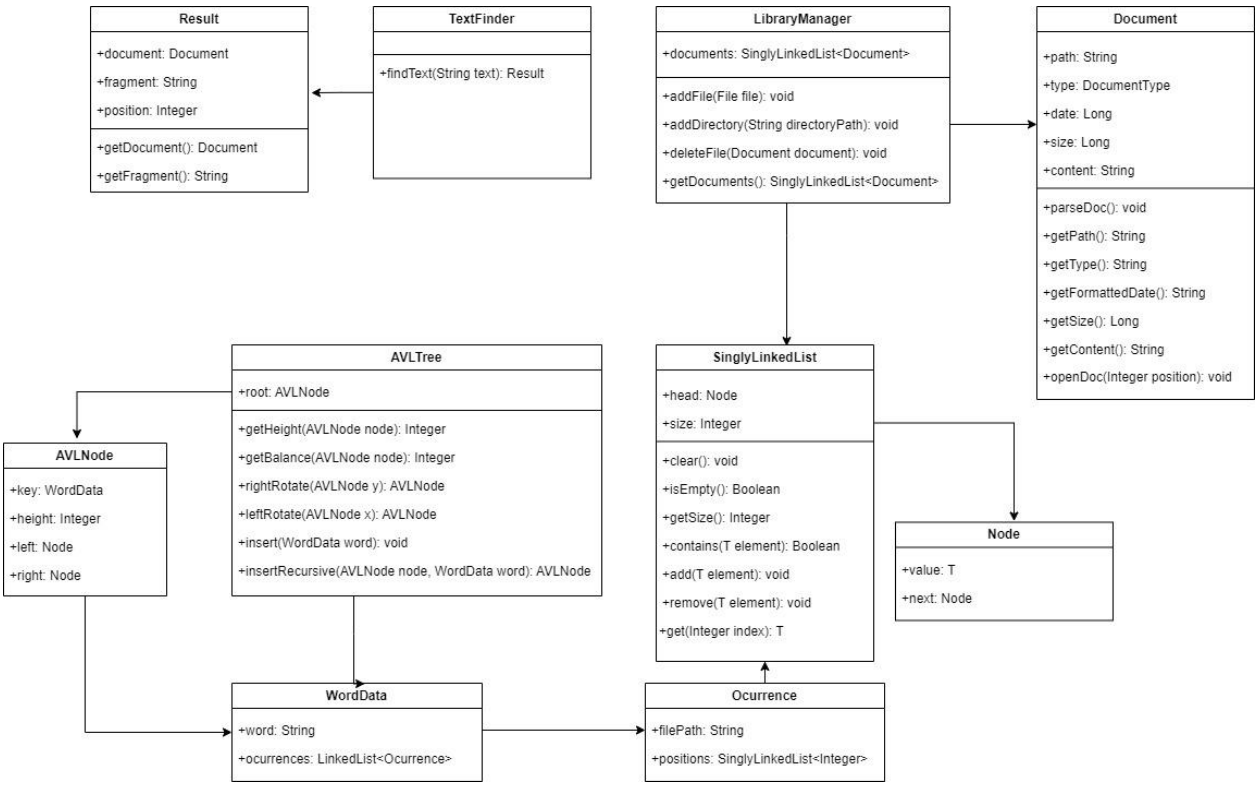
- Método de búsqueda de palabras o frases: se evaluaron dos enfoques principales, la búsqueda secuencial y el uso de expresiones regulares. La búsqueda secuencial consiste en recorrer cada palabra del documento para encontrar coincidencias, pero puede resultar ineficiente para conjuntos de datos extensos. Por otro lado, las expresiones regulares permiten definir patrones de búsqueda para encontrar palabras o frases específicas en el texto del documento, ofreciendo una alternativa más precisa y flexible.

Alternativas seleccionadas y razones de selección

Después de evaluar cuidadosamente todas las alternativas, se tomaron las siguientes decisiones de diseño:

- Lectura de archivos: se seleccionó FileInputStream debido a su capacidad para manejar archivos binarios y de texto por igual, lo que garantiza una lectura eficiente y flexible del contenido de los archivos. Esto es crucial para procesar diferentes tipos de archivos (.txt, .docx, .pdf) en la aplicación.
- Estructura de datos para almacenar palabras y ocurrencias: se seleccionó un árbol AVL debido a su capacidad para mantener un orden específico de las palabras, permitir búsquedas eficientes y manejar múltiples ocurrencias de la misma palabra. Esto facilita la ordenación de resultados y la búsqueda rápida de palabras en la biblioteca de documentos.
- Método de búsqueda de palabras o frases: se optó por un enfoque de búsqueda secuencial para la búsqueda de palabras y frases. Aunque puede ser menos eficiente que las expresiones regulares, garantiza resultados precisos al buscar frases exactas y simplifica la lógica de búsqueda.

Diagrama de clases



Problemas encontrados

Un desafió ha sido el proceso de indexar grandes volúmenes de texto ya que puede volverse ineficiente si no se implementan adecuadamente las estructuras de datos y algoritmos necesarios. Por ejemplo, si la aplicación no optimiza el tiempo de búsqueda en el árbol AVL que contiene las palabras indexadas, podría experimentar retrasos significativos al realizar búsquedas en la biblioteca de documentos.

Otro ha sido Implementar una búsqueda precisa de frases, ya que implica identificar y comparar secuencias de palabras completas en lugar de palabras individuales. Si el algoritmo de búsqueda de frases no se diseña adecuadamente, produce resultados inexactos o mostrar documentos que contienen palabras individuales, pero no la frase completa.

Un desafió adicional que no es un problema técnico, es el diseño de una interfaz de usuario intuitiva y fácil de usar es crucial para la adopción y satisfacción del usuario. Si la

interfaz de usuario no está bien diseñada o es difícil de navegar, los usuarios podrían tener dificultades para utilizar la aplicación de manera efectiva, independientemente de su funcionalidad subyacente. Se tuvo que reorganizar la interfaz ya que al principio era un poco confusa.

Preguntas abiertas

1. ¿Cuáles son las principales necesidades y expectativas que el proyecto TextFinder debe abordar?
2. ¿Qué roles se asignan a cada miembro del equipo y cuáles serán sus responsabilidades específicas?
3. ¿Cómo se controla los errores y excepciones que puedan surgir a la hora de realizar el código?
4. ¿Cómo se asegurará que la interfaz sea atractiva y accesible para los usuarios?
5. ¿Qué estrategias está considerando el equipo para mejorar la eficiencia en el proceso de búsqueda de frases dentro de la aplicación?
6. ¿Cómo se puede optimizar el rendimiento del TextFinder para garantizar una carga rápida de contenido y una experiencia fluida?
7. ¿Existe alguna herramienta que el proyecto no contenga, la cual lo hace menos eficiente que los TextFinders del mercado?
8. ¿Qué medidas se toman para optimizar el tiempo de búsqueda en la estructura de datos del árbol AVL con las palabras indexadas, para evitar retrasos significativos en las operaciones?
9. ¿Qué enfoque se está siguiendo para realizar pruebas y validación del software, incluyendo pruebas de unidad, pruebas de integración y pruebas de aceptación del usuario?
10. ¿Cómo se está diseñando la aplicación para garantizar su escalabilidad en términos de capacidad para manejar grandes volúmenes de documentos?