A PROJECT REPORT ON

“DESKTOP SEARCH ENGINE”

SUBMITTED BY

NAME : Nazlım Serra DİLAVER - Meryem KILIÇ

ID : 2121251017 - 2121251018

COURSE : Data Structures

DESIGN

This system employs a Binary Search Tree (BST) where each node not only contains the word but also a linked list. This linked list details every file the word appears in, along with the frequency of its appearances.

Binary Search Tree (BST) Structure

- The **NazlimSerraDilaverMeryemKilic_BinarySearchTree** class forms the core of the search engine, managing words as keys within a binary search tree. Each node (**NazlimSerraDilaverMeryemKilic_NodeBST**) in this tree corresponds to a unique word and is linked to a linked list that records the documents where the word appears, along with the frequency of its occurrence.
- **Purpose and Utilization:** The binary search tree allows efficient searching, insertion, and deletion of words. The structure is chosen for its ability to maintain a balanced form, thereby ensuring that operations like search and insert operate in O(log n) time complexity on average. The linked list at each node ensures that multiple occurrences of the word across different documents can be efficiently recorded and accessed without duplicating the nodes for the same word.

Linked List Structure

- The **NazlimSerraDilaverMeryemKilic_LinkedList** and **NazlimSerraDilaverMeryemKilic_NodeLL** classes provide a mechanism to store and track the occurrences and frequency of each word in the selected documents. Each node in the linked list corresponds to a document in which the word appears.
- **Purpose and Utilization:** Using a linked list allows dynamically adding entries as new occurrences are found during the file processing phase. This is especially useful when the exact number of documents containing the word is unknown or when the frequency of words needs to be updated. The linked list supports efficient sequential accesses, which is ideal for this application since updates (incrementing counts, adding new files) follow the sequence of data processing.

Node Structure

- Nodes in the binary search tree (**NazlimSerraDilaverMeryemKilic_NodeBST**) are designed to hold a word and a reference to a linked list that stores information about the documents the word appears in. Each node also contains links to its left and right child nodes, facilitating the binary tree structure.
- **Purpose and Utilization:** The node structure supports the binary search tree's requirement of sorted data, which speeds up the search operations. The linked list within each node eliminates the need for multiple entries of the same word in the tree, optimizing memory usage and retrieval speed.

Desktop Search Engine

Purpose: The goal of this project is to develop a "mini desktop search engine" written in the Java programming language, which searches through a collection of documents for specified keywords and returns a list of the documents where these keywords are found.

Structure and Rules:

- **Document Processing:** Documents to be processed are given as input, and their contents are stored in a binary search tree. The frequency of each word is calculated.

- **Search Engine Functionality:** The program searches for a word given by the user and outputs the frequency of that word in each file.
- **Data Structures:** All operations are performed using binary search trees and linked lists. Arrays or other data structures are not used in favor of these structures.
- **User Interface:** The program should have a designed user interface (GUI) that allows users to select documents and perform searches.

Implementation Details:

- The program reads all the given documents and builds a binary search tree.
- It enables searching the binary search tree for a given word.
- The binary search tree must provide post-order, pre-order, and in-order traversal of the file obtained at the end.

Classes and Methods

NazlimSerraDilaverMeryemKilic_MainFrame

Purpose: Acts as the main window of the application, managing different panels through a CardLayout.

Methods:

- NazlimSerraDilaverMeryemKilic_MainFrame(): Constructor that initializes the main frame, sets up the CardLayout, and adds the main panel.
- setPage(String panelName): Switches to the specified panel using the CardLayout and triggers the onPageSet() method for the panel if it implements NazlimSerraDilaverMeryemKilic_IPanel.

NazlimSerraDilaverMeryemKilic_MainPanel

Purpose: The primary user interface panel for interaction, handling file selection, and displaying search results.

Methods:

- loadIgnoreList(): Prompts the user to select a file containing words to be ignored, loads these words into a set.
- addFilesToJList(): Allows the user to select HTML files, adding them to a list displayed in the GUI.
- getSelectedFiles(): Retrieves the list of files selected by the user from the GUI list.
- processWords(): Processes the selected files, parses the text, and inserts words into the binary search tree, excluding those in the ignore list.
- updateList(String order): Updates the GUI list to display words in the specified traversal order of the binary search tree.
- performSearch(String query): Searches for a user-specified query in the binary search tree and displays the results in the GUI.

NazlimSerraDilaverMeryemKilic_IPanel

Purpose: Provides a method to ensure that any panel implementing this interface will perform specific actions when it is set to visible.

Methods:

- onPageSet(): A method that is called when the panel becomes visible in the main frame, used to initiate any necessary updates or actions.

NodeBST:

Purpose: Represents a node in the Binary Search Tree.

Attributes:

- key: A string that holds the word.
- head: The head node of a linked list that stores file names and counts.
- left, right: Pointers to the left and right child nodes in the BST.

Constructor: Initializes a new BST node with a specified key and creates the first linked list node with a provided filename.

NodeLL:

Purpose: Represents a node in the Linked List associated with each BST node.

Attributes:

- fileName: The name of the file where the word (key from BST) is found.
- count: The frequency of the word in the specified file.
- next: A pointer to the next node in the linked list.

Constructor: Initializes a node with a filename, setting count to 1, and next to null.

BinarySearchTree:

Purpose: Manages the entire structure of the BST.

Attributes:

- root: The root node of the BST.

Methods:

- insert: Public method to insert a word and filename into the BST.
- insertRec: Helper method that recursively finds the correct spot in the BST and updates the linked list.
- printInOrder, printPreOrder, printPostOrder: Methods to return strings representing the respective tree traversal outputs.
- search: Public method to find and display occurrences and counts of a word.
- searchRec: Helper method for recursive search in the BST.

Complex Rules:

- **Data Structure Utilization:**
    - The system uses a binary search tree (BST) to manage words and their occurrences across documents, with linked lists at each BST node to store the frequency and

document details for each word. This ensures efficient management of data without redundancy.

- **Input Processing:**
    - Text is extracted from HTML files, with specific stopwords ignored, which are defined by the user and loaded from a file. Words are normalized (converted to lowercase and stripped of punctuation) before being indexed to ensure uniformity in search queries.
- **Search Functionality:**
    - Searches are case-insensitive and ignore punctuation, enhancing the user's ability to find words despite textual variations. The search results display both the occurrence and frequency of words in the documents.
- **Traversal and Output:**
    - The system offers in-order, pre-order, and post-order traversal options for displaying search results, providing different views of the data structure for educational purposes or specific user needs.
- **GUI Dynamics:**
    - The interface allows users to interactively manage files and perform operations such as loading, searching, and displaying results. Transitions between different panels are managed using CardLayout, with specific actions initiated when panels are made visible.
- **Feedback and Error Handling:**
    - Users receive immediate feedback for their actions, including error messages for failed operations, ensuring a responsive and user-friendly experience.
- **Performance Considerations:**
    - While the BST provides efficient search capabilities, the potential for becoming unbalanced is noted, which could affect performance. Future enhancements might consider implementing tree balancing techniques.

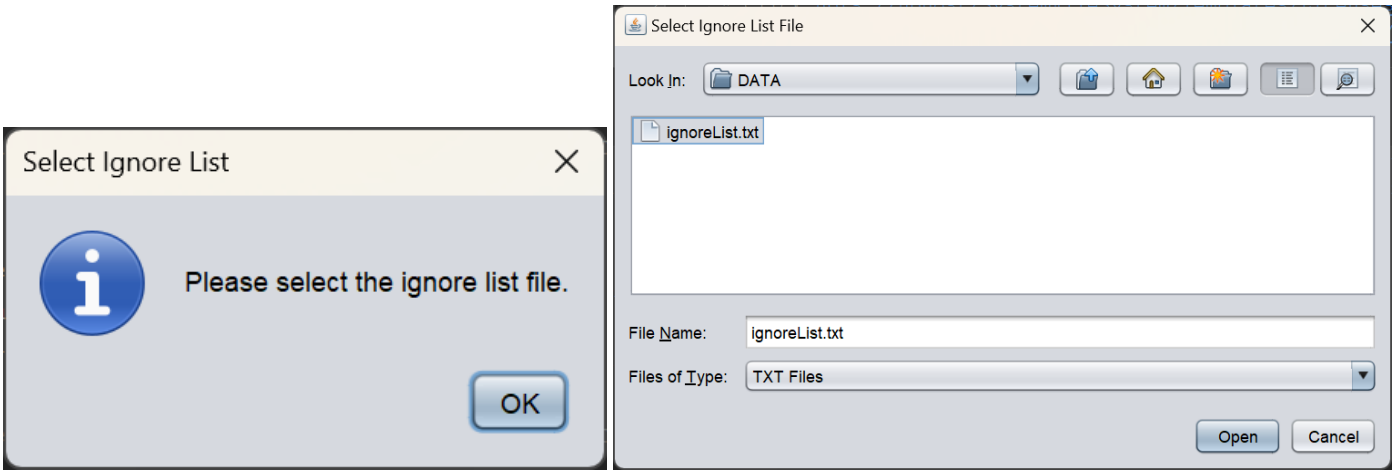
IMPLEMENTATION DETAILS

Programming Language: Java

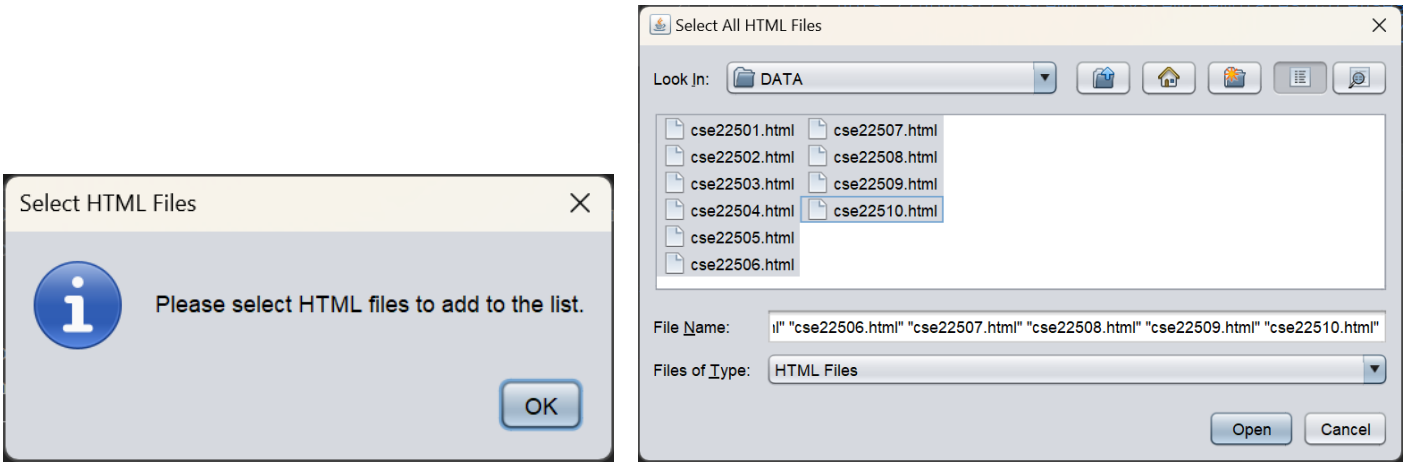
IDE: NetBeans

Libraries: Java Swing

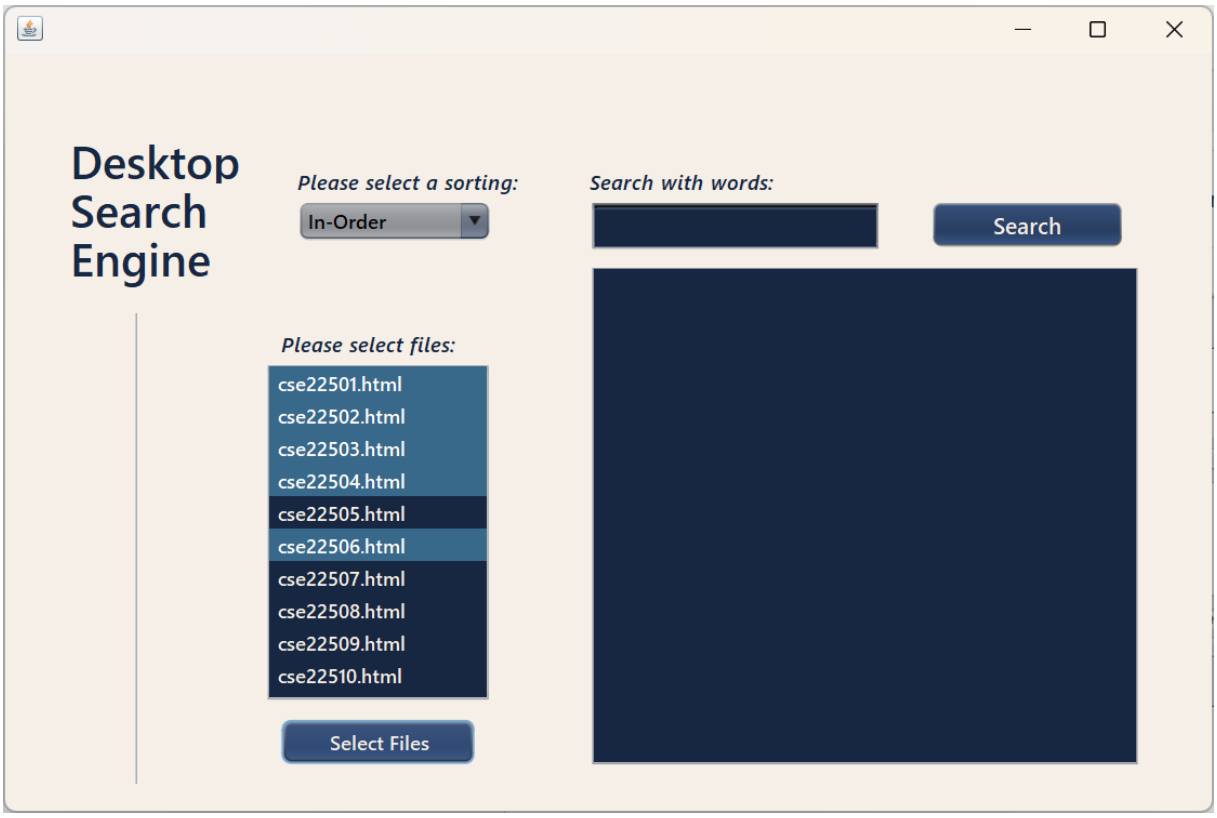
AN EXEMPLARY SCENARIO



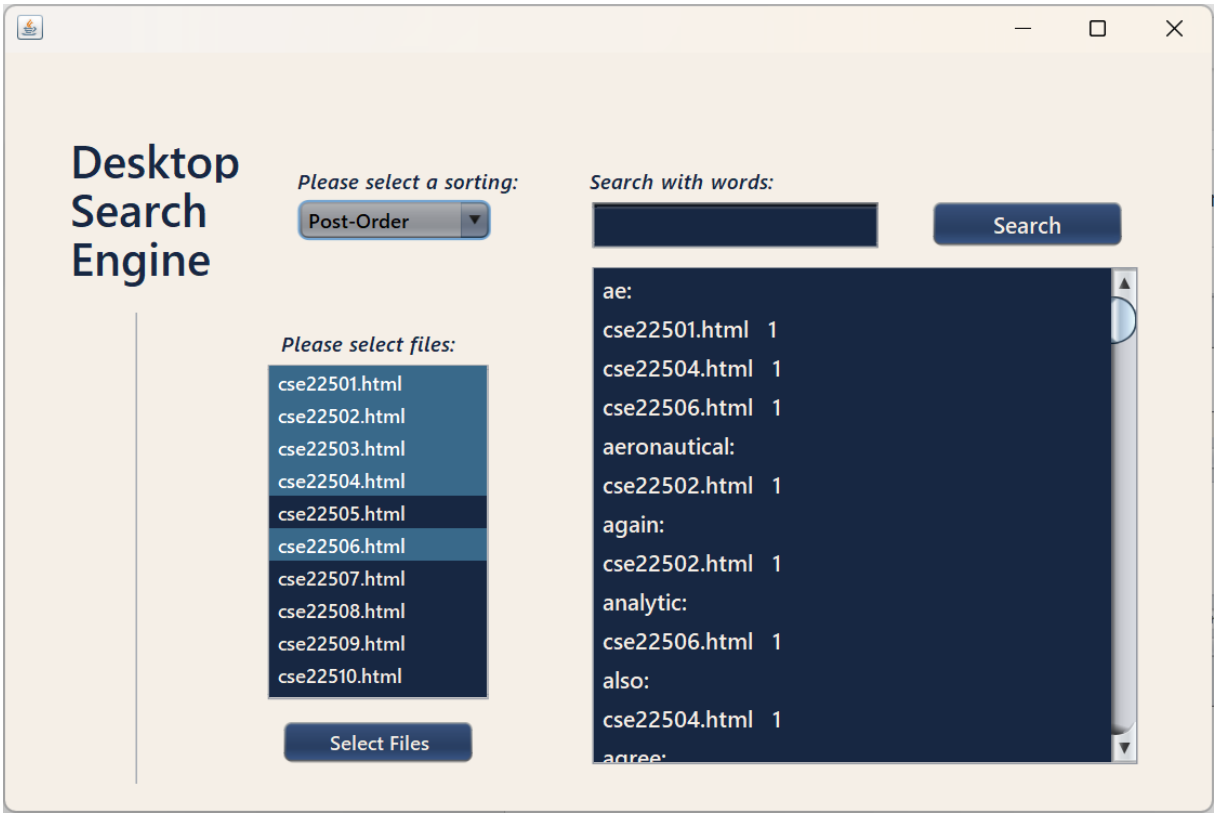
File Chooser 1: Select Ignore List txt



File Chooser 2: Select All HTML Files



Panel 1: Select HTML files and click "Select Files" button



Panel 2: Select a sorting type

Panel 3: Search with word and click "Search" button