

MINI DISKTOP SEARCH ENGINE PROJECT

MINI DISKTOP SEARCH ENGINE PROGRAM USING JAVA (BINARY SEARCH TREE – SINGLE LINKLIST)

AYMAN SAEID

1. Introduction

- In this report, we present the design and implementation details of our Mini Desktop Search Engine project developed in Java. The objective of this project is to create a program that can search through a collection of documents for specified keywords and return the list of documents where those keywords are found.

2. Features

- Ability to search through a collection of documents.
- Utilization of a binary search tree to store words and their frequencies.
- Implementation of a GUI for user interaction.
- Support for post-order, pre-order, and in-order traversal of the binary search tree.
- Proper handling of capitalization and punctuation in input text.

3. Implementation Details

- **3.1 Data Structures Used :**

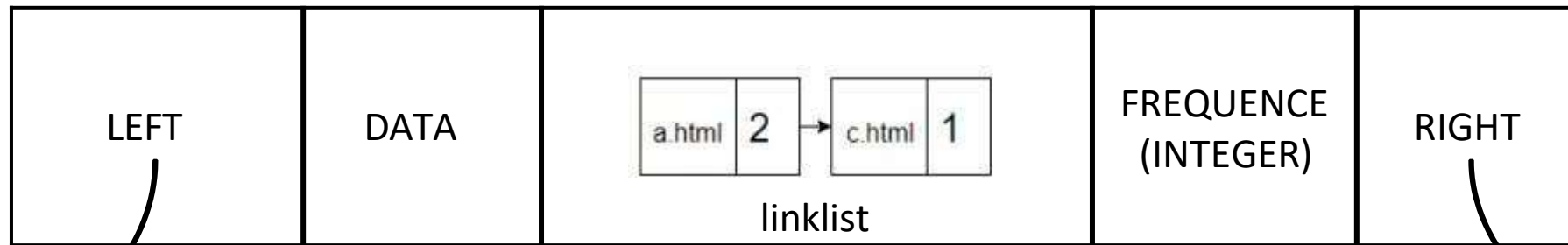
- A) **Binary Search Tree (BST):** Implemented to store words and their frequencies.
- B) **Linked List:** Utilized within the BST node to store the filenames where each word is found.

- **3.2 Algorithm :**

- 1 Read each file selected by the user.
- 2 Split each line into words, ignoring common short words, HTML tags, and punctuation.
- 3 Insert each word into the binary search tree, updating the frequency and linking the filename where it occurs.
- 4 Provide functionality to search for a word in the binary search tree and display its frequency in each file.
- 5 Implement traversals (in-order, pre-order, post-order) to display the contents of the binary search tree.

Binary Search Tree Node

Node



```
package NEW;

import java.util.LinkedList;

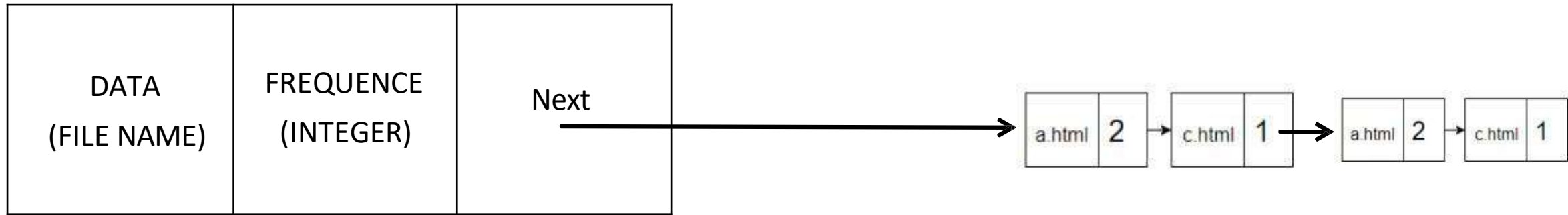
public class AymanSaeidNode4BST<T>{

    AymanSaeidNode4BST right;
    AymanSaeidNode4BST left;
    T data;
    int frequency = 1;
    AymanSaeidLinkedList<String> linkedList = new AymanSaeidLinkedList<>();

    public AymanSaeidNode4BST(T data) {
        this.data = data;
    }

}
```

Linklist Node



```
public class AymanSaeidNode4Linklist<T> {  
    T data;  
    AymanSaeidNode4Linklist<T> next;  
    int frequency = 1;  
  
    public AymanSaeidNode4Linklist(T data) {  
        this.data = data;  
        this.next = null;  
    }  
}
```

Linklist Class And functions

- The Linklist (AymanSaeidLinkList) class represents a linked list implementation used within the Mini Desktop Search Engine project. This class is responsible for managing a linked list of nodes, where each node contains data and a reference to the next node in the list.
- Instance Variables:

head: Represents the first node in the linked list.

Linklist Class Functions

- **add_first(T data):**

This method adds a new node with the given data at the beginning of the linked list.

It first checks if the data already exists in the list by calling the findNode method.

If the data already exists, it increments the frequency of the existing node.

If the data doesn't exist, it creates a new node and adds it at the beginning of the list.

```
package NEW;

import NEW.AymanSaeidNode4Linklist;

public class AymanSaeidLinkList<T extends Comparable<T>> {

    AymanSaeidNode4Linklist<T> head;

    void add_first(T data) {
        AymanSaeidNode4Linklist<T> existing = findNode(data);

        if (existing != null) {
            // if the data already exists, increment its frequency
            existing.frequency++;
            return;
        }

        // data does not exist, add a new node at the beginning
        AymanSaeidNode4Linklist<T> newAymanSaeidNode4Linkliste = new AymanSaeidNode4Linklist<>(data);

        // set the next of the new node to point to the current head
        newAymanSaeidNode4Linkliste.next = head;

        // update the head to point to the new node
        head = newAymanSaeidNode4Linkliste;
    }
}
```


- **findNode(T data):**

This method searches for a node with the given data in the linked list.

It iterates through the list starting from the head node and compares the data of each node with the given data.

If a node with the same data is found, it returns the reference to that node.

If the data is not found, it returns null.

```
private AymanSaeidNode4Linklist<T> findNode(T data) {  
    AymanSaeidNode4Linklist<T> temp = head;  
    while (temp != null) {  
        if (temp.data.equals(data)) {  
            return temp; // NODE found  
        }  
        temp = temp.next;  
    }  
    return null; // NODE not found  
}
```

- **contains(T data):**

This method checks if the linked list contains a node with the given data.

It iterates through the list similar to the findNode method and returns true if the data is found.

If the data is not found, it returns false.

```
boolean contains(T data) {  
    AymanSaeidNode4Linklist<T> temp = head;  
    while (temp != null) {  
        if (temp.data.equals(data)) {  
            return true; // Data found in the list  
        }  
        temp = temp.next;  
    }  
    return false; // Data not found in the list  
}
```

Usage:

This linked list implementation is utilized within the project to manage filenames associated with words stored in the binary search tree. It enables efficient storage and retrieval of filenames, contributing to the functionality of the Mini Desktop Search Engine.

4. GUI Design

- We designed a user-friendly GUI using Java Swing components. The GUI includes options to choose files, specify an ignore list, search for words, functions for binray search tree and display traversal results.

5. Functions (JFrame class)

- **ChooseFilesActionPerformed(Action Event evt):**

This function is triggered when the user clicks the "CHOOSE THE FILES" button.

It opens a file chooser dialog allowing the user to select one or more files.

For each selected file, it reads the content line by line, splits each line into words, and inserts each word along with the filename into the binary search tree using the insert method.

```
private void ChooseFilesActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    JFileChooser fileChooser = new JFileChooser();  
    fileChooser.setMultiSelectionEnabled(true);  
    fileChooser.showOpenDialog(null);  
    File[] files = fileChooser.getSelectedFiles(); // save the files in array  
    for (File file : files) {  
        if (file != null) {  
            try {  
                String fileName = file.getName();  
                // open the file for reading  
                BufferedReader reader = new BufferedReader(new FileReader(file));  
  
                // read each line from the file  
                String line;  
                while ((line = reader.readLine()) != null) {  
                    // split the line into words using whitespace as delimiter  
                    String[] words = line.split("\\s+");  
  
                    // add each word to the ArrayList  
                    for (String word : words) {  
                        // insert the word and file name into the binray search tree  
                        insert((T) word, fileName);  
                        System.out.println(fileName);  
                    }  
                }  
                reader.close();  
            } catch (IOException e) {  
                e.printStackTrace();  
            }  
        }  
    }  
}
```

- **IgonreFileActionPerformed(ActionEvent evt):**

This function is triggered when the user clicks the "CHOOSE THE IGNORE LIST" button.

It opens a file chooser dialog allowing the user to select a file containing a list of words to ignore.

It reads the words from the selected file and adds them to the ignore ArrayList.

```
private void IgonreFileActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    JFileChooser fileChooser = new JFileChooser();  
    fileChooser.showOpenDialog(null);  
    File file = fileChooser.getSelectedFile();  
    if (file != null) {  
        try {  
            // open the file for reading  
            BufferedReader reader = new BufferedReader(new FileReader(file));  
  
            // read each line from the file  
            String line;  
            while ((line = reader.readLine()) != null) {  
                // split the line into words using whitespace as delimiter  
                String[] words = line.split("\\s+");  
  
                // add each word to the array list  
                for (String word : words) {  
                    // add the word into the ignore list  
                    ignore.add(word);  
                }  
            }  
            reader.close();  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
        // printing the words in the ArrayList  
        for (String word : ignore) {  
            text.append(word + "\n");  
        }  
    }  
}
```

```

void postfix(AymanSaeidNode4BST<T> AymanSaeidNode4BST) { // [ R , L , N ]
    if (AymanSaeidNode4BST != null) {
        postfix(AymanSaeidNode4BST.right);
        postfix(AymanSaeidNode4BST.left);
        output.append(AymanSaeidNode4BST.data + " , " + AymanSaeidNode4BST.frequency + "\n");
    }
}

```

- **postOrderBottunActionPerformed(ActionEvent evt):**

This function is triggered when the user clicks the "POST-ORDER" button.

It initiates a post-order traversal of the binary search tree by calling the postfix method, appending the traversal result to the output text area.

```

private void postOrderBottunActionPerformed(java.awt.event.ActionEvent evt) {
    output.append("postfix : \n \n");
    postfix(root);
    output.append("\n");
}

```

```

void inorder(AymanSaeidNode4BST<T> AymanSaeidNode4BST) { // [ L , N , R ]
    if (AymanSaeidNode4BST != null) {
        inorder(AymanSaeidNode4BST.left);
        output.append(AymanSaeidNode4BST.data + " , " + AymanSaeidNode4BST.frequency + "\n");
        inorder(AymanSaeidNode4BST.right);
    }
}

```

- **inOrderBottunActionPerformed(ActionEvent evt):**

This function is triggered when the user clicks the "IN-ORDER" button.

It initiates an in-order traversal of the binary search tree by calling the inorder method, appending the traversal result to the output text area.

```

private void inOrderBottunActionPerformed(java.awt.event.ActionEvent evt) {
    output.append("inorder : \n \n");
    inorder(root);
    output.append("\n");
}

```



```

void preorder (AymanSaeidNode4BST<T> AymanSaeidNode4BST) { // [ N , L , R ]
    if (AymanSaeidNode4BST != null) {
        output.append(AymanSaeidNode4BST.data + " , " + AymanSaeidNode4BST.frequency + "\n");
        preorder (AymanSaeidNode4BST.left);
        preorder (AymanSaeidNode4BST.right);
    }
}

```

- **preOrderBottunActionPerformed(ActionEvent evt):**

This function is triggered when the user clicks the "PRE-ORDER" button.

It initiates a pre-order traversal of the binary search tree by calling the preorder method, appending the traversal result to the output text area.

```

private void preOrderBottunActionPerformed(java.awt.event.ActionEvent evt) {
    output.append("preorder : \n \n");
    preorder(root);
    output.append("\n");
}

```


insert(T data, String fileName):

- This method inserts a new node into the binary search tree.
- It takes two parameters: data, representing the data to be inserted, and fileName, representing the filename associated with the data.
- It first checks if the data is in the ignore list. If it is, the method returns without inserting the node.
- If the data is not in the ignore list, it creates a new node with the provided data.
- It then checks if the fileName is already in the linked list associated with the node. If not, it adds the fileName to the linked list.
- If the tree is empty (i.e., root is null), it sets the new node as the root.
- If the tree is not empty, it traverses the tree to find the appropriate position to insert the new node based on the comparison of data with the data in each node.
 - If data is less than the data in the current node, it moves to the left child.
 - If data is greater than the data in the current node, it moves to the right child.
 - If data is equal to the data in the current node, it increments the frequency of the node and adds the fileName to the linked list.

The method ensures that the binary search tree maintains its property of being ordered by data values.

```

void insert(T data, String fileName) {
    if (ignore.contains(data)) {
        return;
    }
    AymanSaeidNode4BST<T> newnode = new AymanSaeidNode4BST<>(data);
    if (!newnode.linkedList.contains(fileName)) {
        newnode.linkedList.add_first(fileName);
    }
    if (root == null) { // If the root of the tree is null, meaning the tree is empty, it sets the root to be the new node.
        root = newnode;
    } else {
        AymanSaeidNode4BST<T> temp = root;
        while (temp != null) { // If newData is less than the data in the current node, move to the left child. newnode.data - temp.data
            if (newnode.data.compareTo(temp.data) < 0) {
                if (temp.left == null) {
                    temp.left = newnode;
                    return;
                }
                temp = temp.left;
            } else if (newnode.data.compareTo(temp.data) > 0) { // If newData is greater than the data in the current node, move to the
                if (temp.right == null) {
                    temp.right = newnode;
                    return;
                }
                temp = temp.right;
            } else if (newnode.data.compareTo(temp.data) == 0) { // newnode.data - temp.data = 0 thats mean they are equal .
                // System.out.println("data already exists" + " " + data);
                temp.frequency++;
                temp.linkedList.add_first(fileName); // Add filename to the linked list

                return;
            }
        }
    }
}

```

- **SearchForWordActionPerformed(ActionEvent evt):**

This function is triggered when the user clicks the "SEARCH" button.

It retrieves the word entered by the user from the search text field.

It searches for the word in the binary search tree and displays the result, including the frequency of the word and the filenames in which it is found, in the found text area.

```
private void SearchForWordActionPerformed(java.awt.event.ActionEvent evt) {  
  
    String searchText = searchhh.getText();  
    AymanSaeidNode4BST<T> current = root;  
    while (current != null) {  
        int cmp = searchText.compareTo((String) current.data);  
        if (cmp < 0) {  
            current = current.left;  
        } else if (cmp > 0) {  
            current = current.right;  
        } else {  
            found.append(searchText + " : Found " + current.frequency + " times | found in : "); // key found, return the key  
            AymanSaeidNode4Linklist<T> temp = (AymanSaeidNode4Linklist<T>) current.linkedList.head; //retrieves the head node of a linked list  
            while (temp != null) {  
                found.append(temp.data + " -> " + temp.frequency + " | ");  
                temp = temp.next;  
            }  
            found.append("\n");  
            return;  
        }  
    }  
    found.setText(searchText + " not found");  
}
```

- These functions are crucial for the functionality of the Mini Desktop Search Engine GUI.
- They handle various user interactions such as file selection, traversal initiation, and word searching.
- The functions ensure efficient handling and processing of user requests, contributing to a smooth user experience.

5. Code Overview

- `AymanSaeidNode4Linklist`: Represents a node for the linked list.
- `AymanSaeidLinkList`: Implements a linked list for storing filenames.
- `AymanSaeidNode4BST`: Represents a node for the binary search tree.
- `JFrame`: Implements the GUI for user interaction.

6. Results

DESKTOP SEARCH ENGINE

Traversal :

IGNORE LIST :

CHOOSE THE FILES

CHOOSE THE IGNORE LIST

PRE-ORDER

IN-ORDER

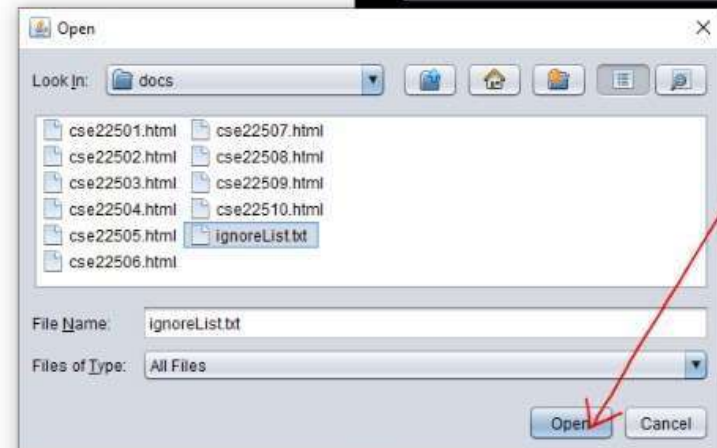
POST-ORDER

SEARCH

Traversal :**IGNORE LIST :**

CHOOSE THE FILES


CHOOSE THE IGNORE LIST



POST-ORDER

SEARCH

Traversal :


CHOOSE THE FILES

CHOOSE THE IGNORE LIST

PRE-ORDER

IN-ORDER

POST-ORDER

IGNORE LIST :

wouldn't
yet
you
you'd
you'll
your
you're
yours
you've
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

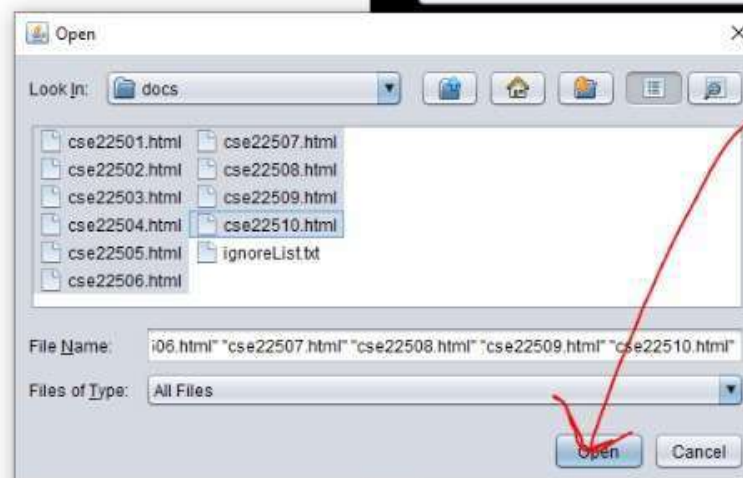
SEARCH

Traversal :

IGNORE LIST :

CHOOSE THE FILES

CHOOSE THE IGNORE LIST



POST-ORDER

wouldn't
yet
you
you'd
you'll
your
you're
yours
you've

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

SEARCH

Traversal :



CHOOSE THE FILES

CHOOSE THE IGNORE LIST

PRE-ORDER

IN-ORDER

POST-ORDER

IGNORE LIST :

wouldn't
yet
you
you'd
you'll
your
you're
yours
you've
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

Traversal :

inorder :

additional , 1
ae , 3
aerodynamic , 1
aerodynamics , 1
aeronautical , 1
after , 3
again , 1
agree , 1
agreement , 2
air , 3
along , 1
also , 1
although , 1
amount , 1
analysis , 1
analytic , 2
angle , 1
angles , 1
appeared , 1
appears , 1
appreciably , 1
approximate , 1
approximately , 1
approximation , 1
arises , 1
attack , 1

CHOOSE THE FILES

CHOOSE THE IGNORE LIST

PRE-ORDER

IN-ORDER

POST-ORDER

IGNORE LIST :

wouldn't
yet
you
you'd
you'll
your
you're
yours
you've
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

SEARCH

Traversal :

inorder :

additional , 1
ae , 3
aerodynamic , 1
aerodynamics , 1
aeronautical , 1
after , 3
again , 1
agree , 1
agreement , 2
air , 3
along , 1
also , 1
although , 1
amount , 1
analysis , 1
analytic , 2
angle , 1
angles , 1
appeared , 1
appears , 1
appreciably , 1
approximate , 1
approximately , 1
approximation , 1
arises , 1
attack , 1

IGNORE LIST :

wouldn't
yet
you
you'd
you'll
your
you're
yours
you've

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

CHOOSE THE FILES

CHOOSE THE IGNORE LIST

PRE-ORDER

IN-ORDER

POST-ORDER

ae

SEARCH

ae : Found 3 times | found in : cse22506.html -> 1 | cse22504.html -> 1 | cse22501.html -> 1 |

7. Conclusion

In conclusion, our Mini Desktop Search Engine project successfully achieves the objectives outlined. The program effectively searches through documents, stores word frequencies using a binary search tree, and provides a user-friendly GUI for interaction. Further improvements could include optimization for larger document collections and additional search functionalities.