King Saud University
College of Computer and Information Sciences
Department of Information Technology

CSC212: Data Structure
1st Semester 1446 H

A Simple Search Engine

Section 666510

NAME	ID
Fajer Mohammed Alamro	*444200800*
Aljawharah Alsubaie	*444201140*
Mashael Alammar	*444200786*

Supervised By: Dr.Rehab Alahmadi

Contents

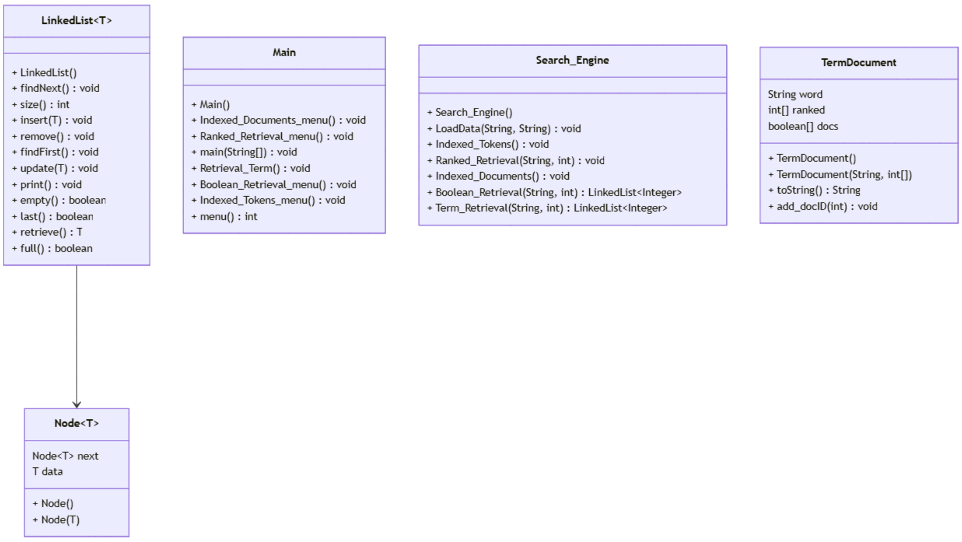
Introduction:

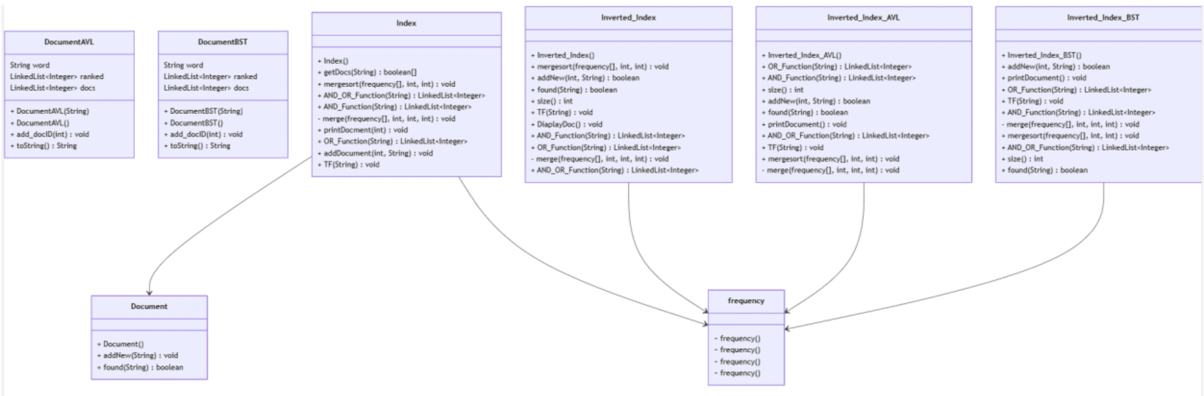
This project's search engine is designed to make document indexing, retrieval, and ranking straightforward and efficient. It uses lists for basic indexing, inverted indices to link terms with the documents they appear in, and Binary Search Trees (BSTs) to speed up the search process. The engine handles Boolean queries like AND and OR, allowing users to refine their searches, and ranks results based on how often the query terms appear in each document. By cleaning up the text—removing unnecessary words and standardizing the format—it creates a more accurate and reliable index. This system highlights the importance of using data structures and algorithms to build a search engine that is not only functional but also user-friendly and efficient.

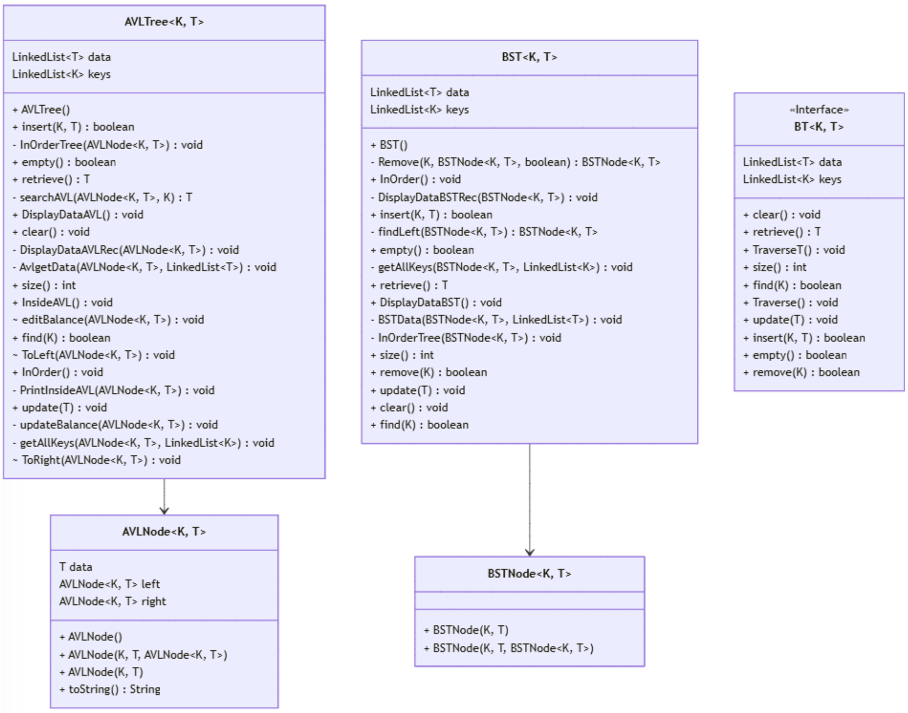
Overview:

In our project, we utilized AVL trees, BSTs, and linked lists to efficiently manage data and create an inverted index that maps terms to the documents where they appear. We implemented Boolean query processing to handle operations like AND and OR by intersecting or merging document lists, with AND queries given priority in combined operations. Additionally, the system calculates term frequencies to rank documents based on relevance, using merge sort to organize results efficiently. We also calculated the unique word with stop words. This approach ensures fast, reliable, and scalable data retrieval, even for complex queries.

UML:


LinkedList<T>
+ LinkedList()
+ findNext() : void
+ size() : int
+ insert(T) : void
+ remove() : void
+ findFirst() : void
+ update(T) : void
+ print() : void
+ empty() : boolean
+ last() : boolean
+ retrieve() : T
+ full() : boolean
Main
+ Main()
+ Indexed_Documents_menu() : void
+ Ranked_Retrieval_menu() : void
+ main(String[]) : void
+ Retrieval_Term() : void
+ Boolean_Retrieval_menu() : void
+ Indexed_Tokens_menu() : void
+ menu() : int
Search_Engine
+ Search_Engine()
+ LoadData(String, String) : void
+ Indexed_Tokens() : void
+ Ranked_Retrieval(String, int) : void
+ Indexed_Documents() : void
+ Boolean_Retrieval(String, int) : LinkedList<Integer>
+ Term_Retrieval(String, int) : LinkedList<Integer>
TermDocument
String word
int[] ranked
boolean[] docs
+ TermDocument()
+ TermDocument(String, int[])
+ toString() : String
+ add_docID(int) : void
Node<T>
Node<T> next
T data
+ Node()
+ Node(T)


DocumentAVL
String word
LinkedList<Integer> ranked
LinkedList<Integer> docs
+ DocumentAVL(String)
+ DocumentAVL()
+ add_docID(int) : void
+ toString() : String
DocumentBST
String word
LinkedList<Integer> ranked
LinkedList<Integer> docs
+ DocumentBST(String)
+ DocumentBST()
+ add_docID(int) : void
+ toString() : String
Index
+ Index()
+ getDocs(String) : boolean[]
+ mergesort(frequency[], int, int) : void
+ AND_OR_Function(String) : LinkedList<Integer>
+ AND_Function(String) : LinkedList<Integer>
- merge(frequency[], int, int, int) : void
+ printDocment(int) : void
+ OR_Function(String) : LinkedList<Integer>
+ addDocument(int, String) : void
+ TF(String) : void
Inverted_Index
+ Inverted_Index()
+ mergesort(frequency[], int, int) : void
+ addNew(int, String) : boolean
+ found(String) : boolean
+ size() : int
+ TF(String) : void
+ DiaplayDoc() : void
+ AND_Function(String) : LinkedList<Integer>
+ OR_Function(String) : LinkedList<Integer>
- merge(frequency[], int, int, int) : void
+ AND_OR_Function(String) : LinkedList<Integer>
Inverted_Index_AVL
+ Inverted_Index_AVL()
+ OR_Function(String) : LinkedList<Integer>
+ AND_Function(String) : LinkedList<Integer>
+ size() : int
+ addNew(int, String) : boolean
+ found(String) : boolean
+ printDocument() : void
+ AND_OR_Function(String) : LinkedList<Integer>
+ TF(String) : void
+ mergesort(frequency[], int, int) : void
- merge(frequency[], int, int, int) : void
Inverted_Index_BST
+ Inverted_Index_BST()
+ addNew(int, String) : boolean
+ printDocument() : void
+ OR_Function(String) : LinkedList<Integer>
+ TF(String) : void
+ AND_Function(String) : LinkedList<Integer>
- merge(frequency[], int, int, int) : void
+ mergesort(frequency[], int, int) : void
+ AND_OR_Function(String) : LinkedList<Integer>
+ size() : int
+ found(String) : boolean
Document
+ Document()
+ addNew(String) : void
+ found(String) : boolean
frequency
~ frequency()
~ frequency()
~ frequency()
~ frequency()


AVLTree<K, T>
LinkedList<T> data
LinkedList<K> keys
+ AVLTree()
+ insert(K, T) : boolean
- InOrderTree(AVLNode<K, T>) : void
+ empty() : boolean
+ retrieve() : T
- searchAVL(AVLNode<K, T>, K) : T
+ DisplayDataAVL() : void
+ clear() : void
- DisplayDataAVLRec(AVLNode<K, T>) : void
- AvlgetData(AVLNode<K, T>, LinkedList<T>) : void
+ size() : int
+ InsideAVL() : void
~ editBalance(AVLNode<K, T>) : void
+ find(K) : boolean
~ ToLeft(AVLNode<K, T>) : void
+ InOrder() : void
- PrintInsideAVL(AVLNode<K, T>) : void
+ update(T) : void
- updateBalance(AVLNode<K, T>) : void
- getAllKeys(AVLNode<K, T>, LinkedList<K>) : void
~ ToRight(AVLNode<K, T>) : void
BST<K, T>
LinkedList<T> data
LinkedList<K> keys
+ BST()
- Remove(K, BSTNode<K, T>, boolean) : BSTNode<K, T>
+ InOrder() : void
- DisplayDataBSTRec(BSTNode<K, T>) : void
+ insert(K, T) : boolean
- findLeft(BSTNode<K, T>) : BSTNode<K, T>
+ empty() : boolean
- getAllKeys(BSTNode<K, T>, LinkedList<K>) : void
+ retrieve() : T
+ DisplayDataBST() : void
- BSTData(BSTNode<K, T>, LinkedList<T>) : void
- InOrderTree(BSTNode<K, T>) : void
+ size() : int
+ remove(K) : boolean
+ update(T) : void
+ clear() : void
+ find(K) : boolean
«Interface»
BT<K, T>
LinkedList<T> data
LinkedList<K> keys
+ clear() : void
+ retrieve() : T
+ TraverseT() : void
+ size() : int
+ find(K) : boolean
+ Traverse() : void
+ update(T) : void
+ insert(K, T) : boolean
+ empty() : boolean
+ remove(K) : boolean
AVLNode<K, T>
T data
AVLNode<K, T> left
AVLNode<K, T> right
+ AVLNode()
+ AVLNode(K, T, AVLNode<K, T>)
+ AVLNode(K, T)
+ toString() : String
BSTNode<K, T>
+ BSTNode(K, T)
+ BSTNode(K, T, BSTNode<K, T>)

Performance analysis (PA): (Retrieval)

- Index Class

- for loop for comparison in search Egin class:

Statement	S/E	Freq	Total
boolean [] document1 = index.getDocs(str); **Its explain in getDocs method**	1	n	n
for (int i = 0 ; i < 50 ; i++)	1	51	51
if (document1[i] == true)	1	50	50
document.insert(i); **Its explain in insert method**	1	50	50

Big(O): O (n)

- getDocs method:

Statement	S/E	Freq	Total
public boolean [] getDocs (String str) {	1	-	-
boolean [] result = new boolean [50];	1	1	1
for (int i = 0 ; i < result.length ; i++)	1	51	51
result[i] = false;	1	50	50
for (int i = 0 ; i < result.length ; i++)	1	51	51
if (Docs[i].found(str))	1	50n	50n
result[i] = true;	1	50	50
return result; }	1	1	1

Big(O): O (n)

- insert method in linked list class:

Statement	S/E	Freq	Total
public void insert (T value) {	0	-	1
Node<T> tmp;	1	1	1
if (empty()) {	1	1	1
current = head = new Node<T> (value);	1	1	1
} else {	1	1	1
tmp = current.next;	1	1	1
current.next = new Node<T> (value);	1	1	1
current = current.next; current.next = tmp;	1	2	2
} size++ ;}	1	1	1

- **Big(O): O (1)**

- Inverted index Class

- If statement in in search Egin class:

Statement	S/E	Freq	Total
if (invIndex.found(str)) { **Its explain in found method**	1	n	n
boolean [] document1 = invIndex.invertedindex.retrieve().getDocs(); **getDocs already explained in Index class**	1	n	n
for (int i = 0 ; i < 50 ; i++)	1	51	51
if (document1[i] == true)	1	50	50
document.insert(i); **} Its already explained in Index class**	1	50	50

Big(O): O (n)

- found method:

Statement	S/E	Freq	Total
public boolean found(String word) {	0	-	-
if (invertedindex.empty())	1	1	1
invertedindex.findFirst();	1	1	1
for (int i = 0 ; i < invertedindex.size ; i++) {	1	n+1	n+1
if (invertedindex.retrieve().word.compareTo(word) == 0)	1	n	n
return true;	1	1	1
invertedindex.findNext(); }	1	n	n
return false; }	1	1	1

Big(O): O (n)

- Inverted index (using BST)

- If statement in in search Egin class:

Statement	S/E	Freq	Total
if (invIndexBST.found(str)) **Its explain in find method**	1	Log n	Log n
Document = invIndexBST.invertedindexBST.retrieve().getDocs();	1	1	1

Big(O): O (log n)

-Find method:

Statement	S/E	Freq	Total
public boolean find(K key) {	0	-	-
BSTNode<K,T> Indicators = root;	1	1	1
if(empty())	1	1	1
return false;	1	1	1
while(Indicators != null) {	1	n+1	n+1
if(Indicators.key.compareTo(key) == 0) {	1	n	n
current = Indicators;	1	n	n
return true; }	1	1	1
} else if(key.compareTo(Indicators.key) < 0)	1	n	n
Indicators = Indicators.left;	1	1	1
else Indicators = Indicators.right;}	1	1	1
return false; }	1	1	1

Big(O): O (n) But In Average Case It will be O(log n)!!

Comparison:

The Index and Inverted Index Classes have a search complexity of O(n), relying on linear scans. The Inverted Index (BST) excels with O (log n) in best case, though it can degrade to O(n) if worst case. For smaller datasets, linear approaches suffice, but the BST offers better scalability for larger datasets.

Performance analysis (PA): (Boolean Retrieval)

Metric	Index	Inverted index	Inverted index (BST)
Data Structure	List of List	List of List	Binary Search Tree
Big(O) (Worst case scenario)	$O(n^3)$	$O(n^2)$	O (n)
Big(O) (Best case scenario)	O(n)	O(n)	O (log n)

The **Binary Search Tree (BST)** outperforms the **List of Lists** in inverted indexing due to its efficient **O (log n)** search time, compared to the **O(n)** time for the List of Lists. In the worst case, a List of Lists can have **$O(n^3)$** complexity, while a BST operates in **$O(n^2)$**. The BST's ability to keep data sorted and use binary search allows it to handle larger datasets more efficiently, making it a faster and more scalable choice for querying documents. This efficiency is particularly noticeable in **AND** and **OR** queries, where the BST minimizes time complexity.

GitHup Link:

https://github.com/FajerAlamro/Data-Stucture-Project/tree/main