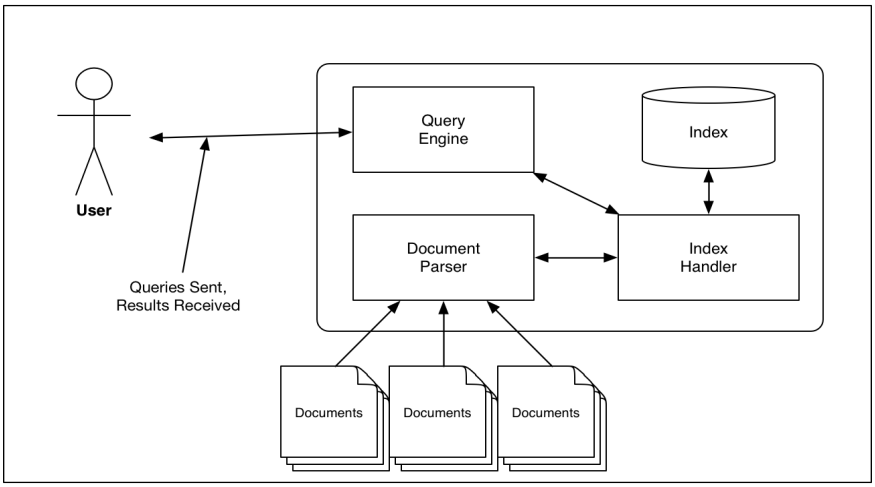
Final Design for Search Engine

Architecture

The four major components' of a typical search engine are the :

1. Document parser/processor,
2. Query processor,
3. Search processor, and
4. Ranking processor.

The image below provides a general overview of the search engine's architecture.



Core Data Structure

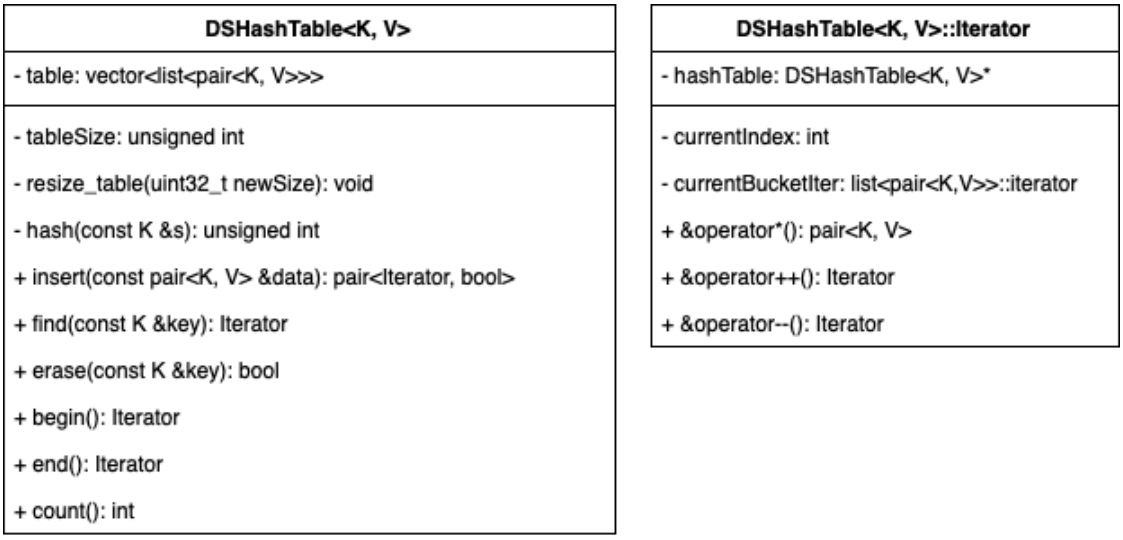
DSAvlTree for keyword index

DSAvlTree<T>
- root: DSAvlNode<T>*
+ insert(const T &x): void + search(T data): DSAvlNode<T>* + count(): int + iterator(): Iterator + serialize(ostream &os): void + deserialize(istream &is): void - rotateWithLeftChild(DSAvlNode<T>* &k2) void - rotateWithRightChild(DSAvlNode<T>* &k1) void - doubleWithRightChild(DSAvlNode<T>* &k3) void - doubleWithLeftChild(DSAvlNode<T>* &k3) void

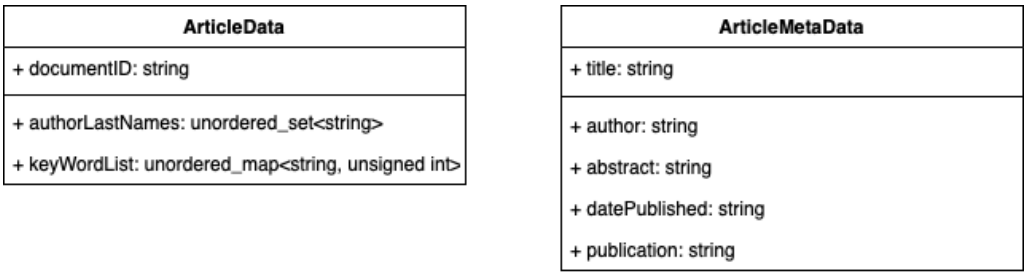
DSAvlNode<T>
+ left: DSAvlNode<T>*
+ right: DSAvlNode<T>* + height: int + element: T

DSAvlTree<T>::Iterator
- tree: DSAvlTree<T>*
- stack: stack<DSAvlNode<T>*> + hasNext(): bool + next(): DSAvlNode<T>*

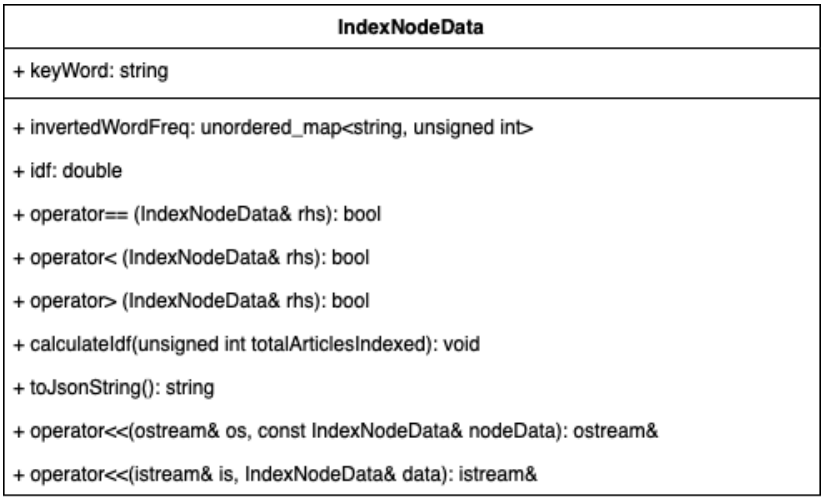
DSHashTable for author index



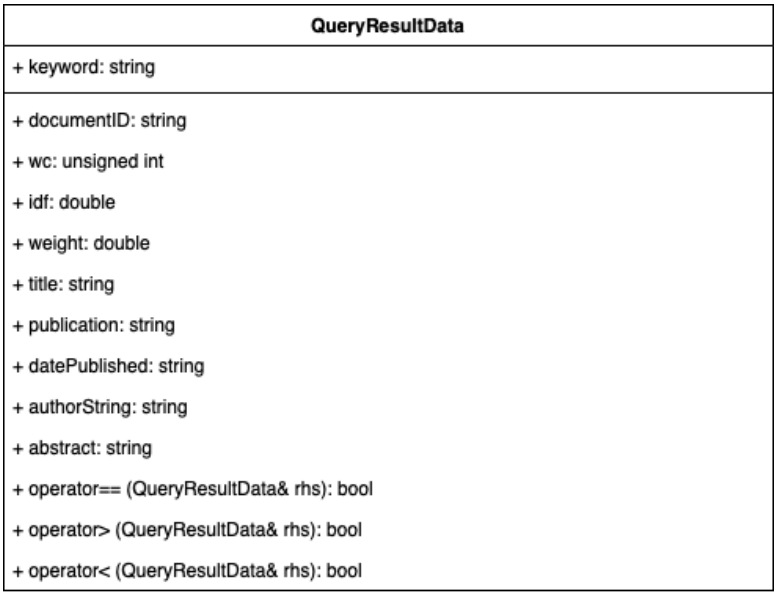
ArticleData to store the parsed article data



IndexNodeData to store the data in a node in the AVL tree



QueryResultData to store the search result from the keyword index, to implement the ranking algorithm, and store data points to display on the UI



DocumentParser to parse the articles in the corpus and load the inverted file index to the AVL tree and author to the HashTable

DocumentParser
+ DocumentParser(string &corpusPath, string &stopwordPath, unordered_map<string, ArticleMetaData> &metaDataMap)
+ corpusPath: string + stopwordPath: string + metaDataMap: unordered_map<string, ArticleMetaData> + parse(DSAvlTree<IndexNodeData>: *keywordIndex, DSHashTable<string, unordered_set<string>> *authorIndex): void + loadStopWords(string &filePath): unordered_set<string> + addArticleToKeywordIndex(DSAvlTree<IndexNodeData> *avlTree, ArticleData &articleData): void + addAuthorsToHashTable(DSHashTable<string, unordered_set<string>> *authorIndex, ArticleData &input): void

---

IndexHandler to store the keyword index, author index, and the metadata and handles all the index-related operations

IndexHandler
+ IndexHandler (const string &corpusPath) - authorIndex: DSHashTable<string, unordered_set<string>>* - keyWordIndex: DSAvlTree<IndexNodeData>* + metaDataMap: unordered_map<string, ArticleMetaData>
+ loadMetaData(string &corpusPath): unordered_map<string, ArticleMetaData> + createIndex(): bool + persistIndices(): bool + restoreIndices(string &dir = ""): bool + searchByKeyword(string &keyWord): IndexNodeData* + searchByAuthor(string &author): unordered_set<string> + clearIndex(): void + isIndexEmpty(): bool + persistKeywordIndex(): bool + restoreKeywordIndex(): bool + persistAuthorIndex(): bool + restoreAuthorIndex(): bool + persistStats(): bool + restoreStats(): bool + totalArticlesIndexed: unsigned int + avgWordsIndexedPerArticle: unsigned int + totalWordsIndexed: unsigned int + totalUniqueAuthors: int - maxFilesLoaded: int - persistentDir: string - stopWordFile: string - keyWordIndexFile: string - authorIndexFile: string - statsFile: string

QueryProcessor to parse the user input and invoke IndexHandler

QueryProcessor
+ QueryProessor(IndexHandler *indexHandler) - indexHandler: IndexHandler*
+ clearIndex(): void + isIndexEmpty(): bool + createIndex(): bool + loadIndices(string &dir): bool + parseQueryString(string &queryString): vector<string>* + preprocess(string &input, bool shouldStem): void + search(string logicOp, vector<string> searchWords, vector<string> excludedWords, vector<string> authors): set<QueryResultData> + searchKeywordindex(vector<string> &searchWords, vector<IndexNodeData*> &searchResults, vector<set<string>> &documentIDSets): void + searchAuthorindex(vector<string> &authors, set<string> &documentIDSet): void + getTop50OriginalWords(): vector<string> + getTotalUniqueAuthors(): int + getTotalWordsIndexed(): int + getAvgWordsIndexedPerArticle(): int + getTotalArticlesIndexed(): int

SearchEngineUI to display user interface and handle the user interactions