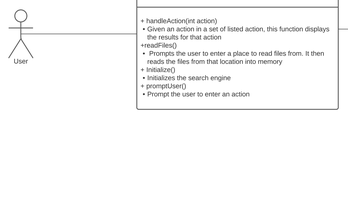
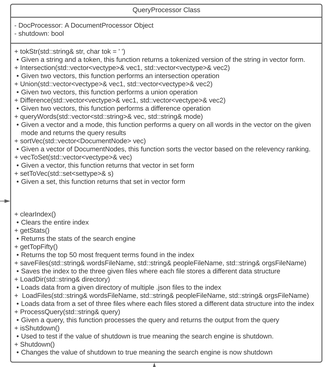
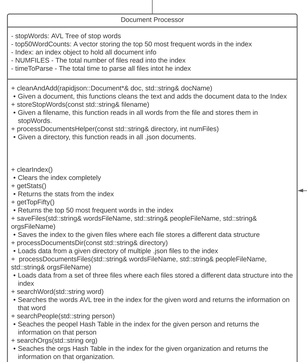
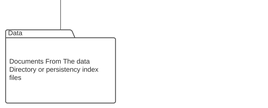
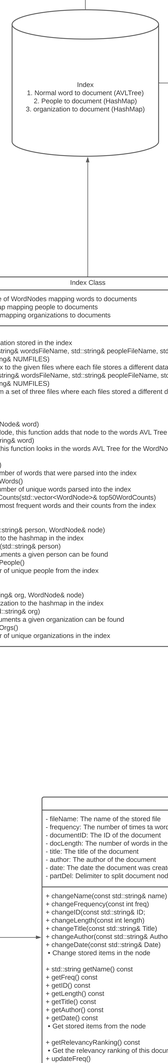

User

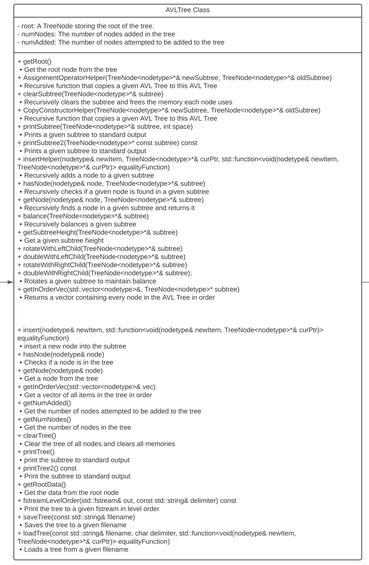
UserInterface Class
- QP: A Query Processor Used to handle queries entered by the user
- indexClear: True if the index is clear, false otherwise
+ handleAction(int action)
• Given an action in a set of listed action, this function displays the results for that action
+readFiles()
• Prompts the user to enter a place to read files from. It then reads the files from that location into memory
+ Initialize()
• Initializes the search engine
+ promptUser()
• Prompt the user to enter an action

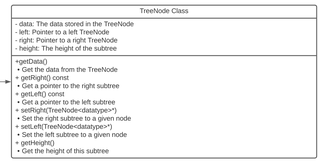

QueryProcessor Class
- DocProcessor: A DocumentProcessor Object
- shutdown: bool
+ tokStr(std::string& str, char tok = ' ')
• Given a string and a token, this function returns a tokenized version of the string in vector form.
+ Intersection(std::vector<vectype>& vec1, std::vector<vectype>& vec2)
• Given two vectors, this function performs an intersection operation
+ Union(std::vector<vectype>& vec1, std::vector<vectype>& vec2)
• Given two vectors, this function performs a union operation
+ Difference(std::vector<vectype>& vec1, std::vector<vectype>& vec2)
• Given two vectors, this function performs a difference operation
+ queryWords(std::vector<std::string>& vec, std::string& mode)
• Given a vector and a mode, this function performs a query on all words in the vector on the given mode and returns the query results
+ sortVec(std::vector<DocumentNode> vec)
• Given a vector of DocumentNodes, this function sorts the vector based on the relevency ranking.
+ vecToSet(std::vector<vectype>& vec)
• Given a vector, this function returns that vector in set form
+ setToVec(std::set<settype>& s)
• Given a set, this function returns that set in vector form
+ clearIndex()
• Clears the entire index
+ getStats()
• Returns the stats of the search engine
+ getTopFifty()
• Returns the top 50 most frequent terms found in the index
+ saveFiles(std::string& wordsFileName, std::string& peopleFileName, std::string& orgsFileName)
• Saves the index to the three given files where each file stores a different data structure
+ LoadDir(std::string& directory)
• Loads data from a given directory of multiple .json files to the index
+ LoadFiles(std::string& wordsFileName, std::string& peopleFileName, std::string& orgsFileName)
• Loads data from a set of three files where each files stored a different data structure into the index
+ ProcessQuery(std::string& query)
• Given a query, this function processes the query and returns the output from the query
+ isShutdown()
• Used to test if the value of shutdown is true meaning the search engine is shutdown.
+ Shutdown()
• Changes the value of shutdown to true meaning the search engine is now shutdown

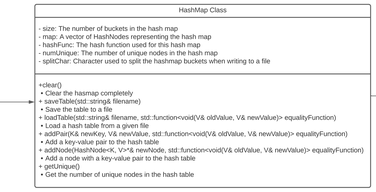

Document Processor
- stopWords: AVL Tree of stop words
- top50WordCounts: A vector storing the top 50 most frequent words in the index
- Index: an index object to hold all document info
- NUMFILES - The total number of files read into the index
- timeToParse - The total time to parse all files intot he index
+ cleanAndAdd(rapidjson::Document*& doc, std::string& docName)
• Given a document, this functions cleans the text and adds the document data to he Index
+ storeStopWords(const std::string& filename)
• Given a filename, this function reads in all words from the file and stores them in stopWords.
+ processDocumentsHelper(const std::string& directory, int numFiles)
• Given a directory, this function reads in all .json documents.
+ clearIndex()
• Clears the index completely
+ getStats()
• Returns the stats from the index
+ getTopFifty()
• Returns the top 50 most frequent words in the index
+ saveFiles(std::string& wordsFileName, std::string& peopleFileName, std::string& orgsFileName)
• Saves the index to the given files where each file stores a different data structure
+ processDocumentsDir(const std::string& directory)
• Loads data from a given directory of multiple .json files to the index
+ processDocumentsFiles(std::string& wordsFileName, std::string& peopleFileName, std::string& orgsFileName)
• Loads data from a set of three files where each files stored a different data structure into the index
+ searchWord(std::string word)
• Searches the words AVL tree in the index for the given word and returns the information on that word
+ searchPeople(std::string person)
• Seaches the peopel Hash Table in the index for the given person and returns the information on that person
+ searchOrgs(std::string org)
• Seaches the orgs Hash Table in the index for the given organization and returns the information on that organization.

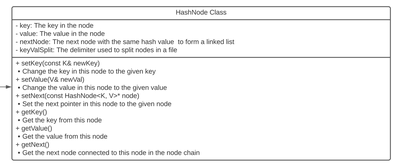

Data
Documents From The data Directory or persistency index files

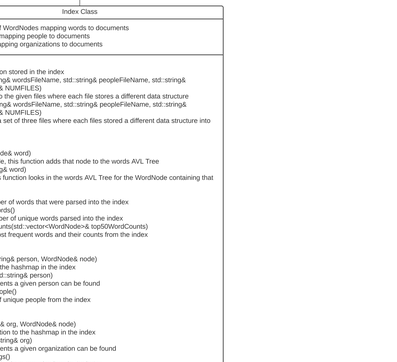

Index
1. Normal word to document (AVLTree)
2. People to document (HashMap)
3. organization to document (HashMap)

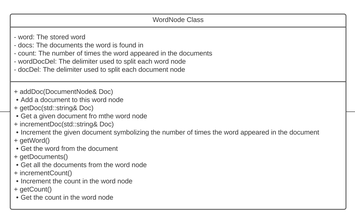

AVLTree Class
- root: A TreeNode storing the root of the tree.
- numNodes: The number of nodes added in the tree
- numAdded: The number of nodes attempted to be added to the tree
+ getRoot()
• Get the root node from the tree
+ AssignmentOperatorHelper(TreeNode<nodetype>*& newSubtree, TreeNode<nodetype>*& oldSubtree)
• Recursive function that copies a given AVL Tree to this AVL Tree
+ clearSubtree(TreeNode<nodetype>*& subtree)
• Recursively clears the subtree and frees the memory each node uses
+ CopyConstructorHelper(TreeNode<nodetype>*& newSubtree, TreeNode<nodetype>*& oldSubtree)
• Recursive function that copies a given AVL Tree to this AVL Tree
+ printSubtree(TreeNode<nodetype>*& subtree, int space)
• Prints a given subtree to standard output
+ printSubtree2(TreeNode<nodetype>* const subtree) const
• Prints a given subtree to standard output
+ insertHelper(nodetype& newItem, TreeNode<nodetype>*& curPtr, std::function<void(nodetype& newItem, TreeNode<nodetype>*& curPtr)> equalityFunction)
• Recursively adds a node to a given subtree
+ hasNode(nodetype& node, TreeNode<nodetype>*& subtree)
• Recursively checks if a given node is found in a given subtree
+ getNode(nodetype& node, TreeNode<nodetype>*& subtree)
• Recursively finds a node in a given subtree and returns it
+ balance(TreeNode<nodetype>*& subtree)
• Recursively balances a given subtree
+ getSubtreeHeight(TreeNode<nodetype>*& subtree)
• Get a given subtree height
+ rotateWithLeftChild(TreeNode<nodetype>*& subtree)
+ doubleWithLeftChild(TreeNode<nodetype>*& subtree)
+ rotateWithRightChild(TreeNode<nodetype>*& subtree)
+ doubleWithRightChild(TreeNode<nodetype>*& subtree);
• Rotates a given subtree to maintain balance
+ getInOrderVec(std::vector<nodetype>&, TreeNode<nodetype>* subtree)
• Returns a vector containing every node in the AVL Tree in order
+ insert(nodetype& newItem, std::function<void(nodetype& newItem, TreeNode<nodetype>*& curPtr)> equalityFunction)
• insert a new node into the subtree
+ hasNode(nodetype& node)
• Checks if a node is in the tree
+ getNode(nodetype& node)
• Get a node from the tree
+ getInOrderVec(std::vector<nodetype>& vec)
• Get a vector of all items in the tree in order
+ getNumAdded()
• Get the number of nodes attempted to be added to the tree
+ getNumNodes()
• Get the number of nodes in the tree
+ clearTree()
• Clear the tree of all nodes and clears all memories
+ printTree()
• print the subtree to standard output
+ printTree2() const
• Print the subtree to standard output
+ getRootData()
• Get the data from the root node
+ fstreamLevelOrder(std::fstream& out, const std::string& delimiter) const
• Print the tree to a given fstream in level order
+ saveTree(const std::string& filename)
• Saves the tree to a given filename
+ loadTree(const std::string& filename, char delimiter, std::function<void(nodetype& newItem, TreeNode<nodetype>*& curPtr)> equalityFunction)
• Loads a tree from a given filename

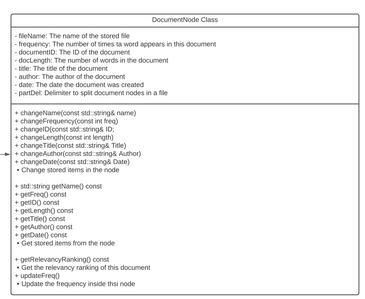

TreeNode Class
- data: The data stored in the TreeNode
- left: Pointer to a left TreeNode
- right: Pointer to a right TreeNode
- height: The height of the subtree
+getData()
• Get the data from the TreeNode
+ getRight() const
• Get a pointer to the right subtree
+ getLeft() const
• Get a pointer to the left subtree
+ setRight(TreeNode<datatype>*)
• Set the right subtree to a given node
+ setLeft(TreeNode<datatype>*)
• Set the left subtree to a given node
+ getHeight()
• Get the height of this subtree


HashMap Class
- size: The number of buckets in the hash map
- map: A vector of HashNodes representing the hash map
- hashFunc: The hash function used for this hash map
- numUnique: The number of unique nodes in the hash map
- splitChar: Character used to split the hashmap buckets when writing to a file
+clear()
• Clear the hasmap completely
+ saveTable(std::string& filename)
• Save the table to a file
+ loadTable(std::string& filename, std::function<void(V& oldValue, V& newValue)> equalityFunction)
• Load a hash table from a given file
+ addPair(K& newKey, V& newValue, std::function<void(V& oldValue, V& newValue)> equalityFunction)
• Add a key-value pair to the hash table
+ addNode(HashNode<K, V>*& newNode, std::function<void(V& oldValue, V& newValue)> equalityFunction)
• Add a node with a key-value pair to the hash table
+ getLinique()
• Get the number of unique nodes in the hash table


HashNode Class
- key: The key in the node
- value: The value in the node
- nextNode: The next node with the same hash value to form a linked list
- keyValSplit: The delimiter used to split nodes in a file
+ setKey(const K& newKey)
• Change the key in this node to the given key
+ setValue(V& newVal)
• Change the value in this node to the given value
+ setNext(const HashNode<K, V>* node)
• Set the next pointer in this node to the given node
+ getKey()
• Get the key from this node
+ getValue()
• Get the value from this node
+ getNext()
• Get the next node connected to this node in the node chain


Index Class
- words: AVL Tree of WordNodes mapping words to documents
- people: Hashmap mapping people to documents
- orgs: Hashmap mapping organizations to documents
+ clearIndex()
• Clear all information stored in the index
+ saveFiles(std::string& wordsFileName, std::string& peopleFileName, std::string& orgsFileName, long& NUMFILES)
• Saves the index to the given files where each file stores a different data structure
+ LoadFiles(std::string& wordsFileName, std::string& peopleFileName, std::string& orgsFileName, long& NUMFILES)
• Loads data from a set of three files where each files stored a different data structure into the index
+ addWord(WordNode& word)
• Given a WordNode, this function adds that node to the words AVL Tree
+ getWord(std::string& word)
• Given a word, this function looks in the words AVL Tree for the WordNode containing that word.
+ getNumWords()
• Get the toal number of words that were parsed into the index
+ getNumUniqueWords()
• Get the total number of unique words parsed into the index
+ getTop50WordCounts(std::vector<WordNode>& top50WordCounts)
• Get the top 50 most frequent words and their counts from the index
+ addPerson(std::string& person, WordNode& node)
• Adds a person to the hashmap in the index
+ getPeopleDocs(std::string& person)
• Get all the documents a given person can be found
+ getNumUniquePeople()
• Get the number of unique people from the index
+ addOrg(std::string& org, WordNode& node)
• Adds an organization to the hashmap in the index
+ getOrgDocs(std::string& org)
• Get all the documents a given organization can be found
+ getNumUniqueOrgs()
• Get the number of unique organizations in the index


WordNode Class
- word: The stored word
- docs: The documents the word is found in
- count. The number of times the word appeared in the documents
- wordDocDel: The delimiter used to split each word node
- docDel: The delimiter used to split each document node
+ addDoc(DocumentNode& Doc)
• Add a document to this word node
+ getDoc(std::string& Doc)
• Get a given document fro mthe word node
+ incrementDoc(std::string& Doc)
• Increment the given document symbolizing the number of times the word appeared in the document
+ getWord()
• Get the word from the document
+ getDocuments()
• Get all the documents from the word node
+ incrementCount()
• Increment the count in the word node
+ getCount()
• Get the count in the word node


DocumentNode Class
- fileName: The name of the stored file
- frequency: The number of times ta word appears in this document
- documentID: The ID of the document
- docLength: The number of words in the document
- title: The title of the document
- author: The author of the document
- date: The date the document was created
- partDel: Delimiter to split document nodes in a file
+ changeName(const std::string& name)
+ changeFrequency(const int freq)
+ changeID(const std::string& ID;
+ changeLength(const int length)
+ changeTitle(const std::string& Title)
+ changeAuthor(const std::string& Author)
+ changeDate(const std::string& Date)
• Change stored items in the node
+ std::string getName() const
+ getFreq() const
+ getID() const
+ getLength() const
+ getTitle() const
+ getAuthor() const
+ getDate() const
• Get stored items from the node
+ getRelevancyRanking() const
• Get the relevancy ranking of this document
+ updateFreq()
• Update the frequency inside thsi node