**/\***

**Name: Kartheek Srinivas Maddipatla**

**CWID: 20012123**

**Final Project: Search Engine**

**\*/**

**Description:** The Search engine is written in Python and uses object-oriented programming. It can search, rank, and display data. It was developed using Visual Studio.

**System Requirements:** Visual studio

**Algorithms/libraries used**: Tire, Inverted File, TD-IDF, BeautifulSoap

Trie:

A Trie, also known as a prefix tree, is a tree-like data structure that is used to store a dynamic set or associative array where the keys are sequences (usually strings). It is an efficient data structure for searching, inserting, and deleting elements, especially when the keys are strings.

In a Trie, each node stores a single character and represents a prefix of a string. The root node represents the empty string, and the children of the root node represent the possible first characters of a string. The children of these nodes represent the possible second characters of a string, and so on. When a string is added to a Trie, it is inserted at a leaf node, and the path from the root node to the leaf node represents the string.

One of the main advantages of using a Trie is that it allows fast prefix searches. For example, if you want to search for all strings in a Trie that start with a particular prefix, you can simply start at the root node and traverse the tree until you reach the end of the prefix. This is much faster than searching a list or an array, where you would need to check every element to see if it starts with the prefix.

Tries are often used in spelling correction, word prediction, and text auto-completion systems, as well as in network routing, DNA sequence analysis, and many other applications where strings are used.

Ex: Apple, Apply, Approve, Appear

Inverted File:

Each word contains its appearances in documents and places as key value pairs (k, l), where k is a list of all the documents where the word appears and l is a list of all the positions at which those documents appear.

TF-IDF:

The algorithm used to rank the pages is this one. The term frequency (TF) measures how often a word appears overall relative to all other words. IDF, or inverse document frequency, measures how uncommon a word is across all texts.

BeautifulSoup:

To extract the data from the html file, use this library. The gorgeous soup is used to extract all the information from the file after first reading the data from a local server.

**Classes Description:**

Trienode: The class which has the nodes for trie. Every node in trie is the object of this database.

Trie: Contains method for insertion and searching the Trie tree

Stopwords: This class has a method to search using the trie class and adds all the stop words into the tree.

Document: This class inserts all the document data to the trie tree. It also has a method to search the tree using trie class

Scraping: This class uses beautifulsoup for extracting the data from html files. First files are loaded from local host and then scraped for the purpose

InitializeEngine: This classes initializes the search engine by calling functions from stopword and documents so all the data required for performing search queries is gathered

Ranking: This classes computes the TF-IDF for the every word which is passed to it.

Search: This class computes the final rank of every page by computing ranks for every word and calculating the final ranks

PresentationLayer: This layer gets all the ranks of the documents and prints a brief description of the page and the URL of the page

**Steps for running the program:**

* Open the file “ProjectCode” which is solution file for Microsoft visual studio
* Set the Path in the Microsoft visual studio to the Main folder “CS600 Project” in one of the Terminals and set the path to be CS600 Project/ProjectCode in the second terminal.
* Start the Server in 1st terminal using : python -m http.server
* Run the Main.py file in the second terminal using : python Main.py
* Type the query in console window
* Type exit if you want to close the program

**Control Flow:**

1. First the data is initialized i.e., The stopword and documents are loaded into Trie tree
2. Each stopword is inserted into the trie
3. For every document, data is scraped using the scraping class and inserted into Trie
4. When the Query is passed each word is striped from it and rank is formed
5. Ranks for all the words in query and final rank is computed
6. Description is scraped from all the URL which has rank more than zero
7. The description and URL is printed on the console based on their ranks.

**Input data:**

Input data is taken from Different websites such as W3 schools, Imbd (movie rating application) and geeks for geeks. It is of 10 web pages in which all the web pages are inter linked

**Output:**

Query: released

Output:

A screenshot of a computer

Description automatically generated with medium confidence

Query: is

Output:

A picture containing text, screenshot

Description automatically generated

Query: kartheek

Output:

A picture containing text, screenshot

Description automatically generated

Query: exit

Output:

A picture containing text, screenshot, multimedia software, software

Description automatically generated