Data Structures and Algorithms Lab

9. Sorting and Searching

Subject Code: 19ECSP201 **Lab No:** 9 **Semester:** III

Date: 09 Oct 2019 **Batch:** C1

Question: Search Engine Simulation

Objective: Usage of searching, sorting and appropriate data structures in implementing a mock-search engine

---

Note: A search engine is more than what you see here. It's a mini version to get a mock feel of it. Let's get a quick view.

A search engine builds an index of all the web pages using a program called a 'web crawler.' This automatically browses the web and stores information about the pages it visits. Every time a web crawler visits a webpage, it makes a copy of it and adds its URL to an index. Once this is done, the web crawler follows all the links on the page, repeating the process of copying, indexing and then following the links (uses a BFS). It keeps doing this, building up a huge index of many webpages as it goes. Do you see that this can get cyclic and go on forever? Do you see what data structure can possibly be picked to visit and parse the links? Think of it, though we don't need a solution for that, for now.

The information that the web crawler puts together is then used by search engines. It becomes the search engine's index. Every webpage recommended by a search engine has been visited by a web crawler.

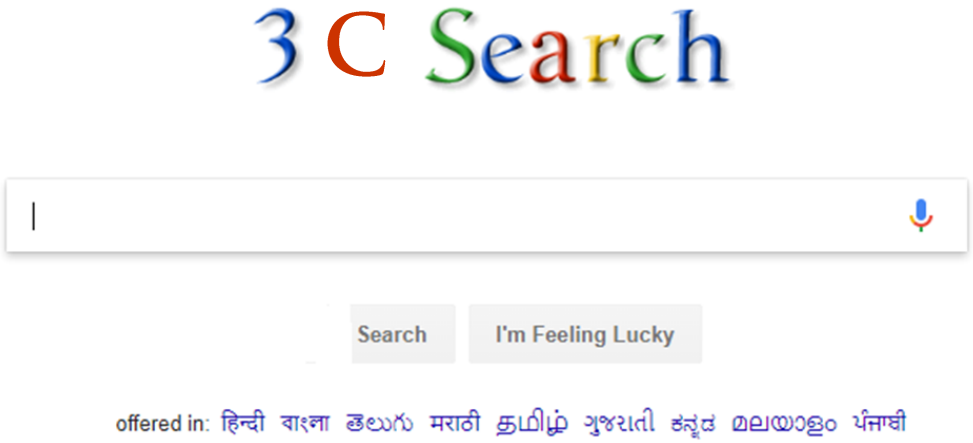
Search engines sort results to show you the ones they think are the most useful. PageRank is the best-known algorithm that is used to improve web search results. This algorithm is used by Google search engine. In simple terms, PageRank is a popularity contest. The more links that point to a webpage, the more useful it will seem. This means it will appear higher up in the results. Today, the algorithm has more than 100 parameters to evaluate a webpage. The 'Humming Bird' version from 2013 is known to be the stable version to date. It has parameters that rank on semantic contents too.

There are many search engines to choose from. Different search engines use different algorithms. This means that some sites will give their results in a different order, or they may even show completely different results altogether.

When it comes to the search engine, we all know who leads the competition. Though surrounded by many competitors, there is a clear winner in the race.

No, we don't want to be a competitor yet! But let's implement one. Let's Implement a 3C Search Engine.

It's an III semester C division search engine (C1, to be specific).



Let us say you had a crawler that has gathered all the news information. Now you want to build a search engine on top of that. A user will give a keyword, and you will display the news for his interest.

Following is the process you will follow:

Step 00: You have already been given a file named – **SearchIndex.txt** which has few data sets. Each data set has News Feeds and a Priority number. Populate the file with more data sets if you want. For the crawled data, the priority is set from 1 to 10. If you want to add more data to file, add it.

Step 01: Load all the data from the file into an appropriate data structure (You are here working from secondary storage to primary storage). Again, this part is already done for you. If you want to change the data structure, you can.

Step 02: Collect a search string from the user. The search string will be of one or two words. If more than two words are entered, display an appropriate message and re-prompt to enter a new input.

Step 03: Make a search based on user entered query string, get all the matching news feed and load them to an appropriate data structure. Decide an appropriate strategy in case of two word input.

Step 04: Sort the results based on the priority. The highest priority news, come first. Same priority news can be presented in any order.

Step 05: Print the results to the user one by one, until there are no more results or the user terminates. If no results, print appropriate message. The news has to be displayed to the user one by one (Eg: press 1 to read next news).

Step 06: Analyze the efficiency of your engine. Can you make it better? If so, optimize the necessary operations.

**** Happy Coding ****