Building A Search Engine For Querying Academic Specialists

By Sahil Mutneja
under the mentorship of Dr. Dileep A.D.

Problem Statement

- Search Engine that provides the user with a list of experts when queried against a specialization
    - The experts will be from the premier institutes spanning across India
- The list will be such that the most relevant of the result will be towards the top.

Final Outcome

- The result will comprise of the links of web pages of the specialist along with the departments they belong in the respective institute.
- The results will be shown using a web application where user will enter the specialization and for each query entered two sets, one for the faculty web pages and other for the concerned department pages will be displayed.

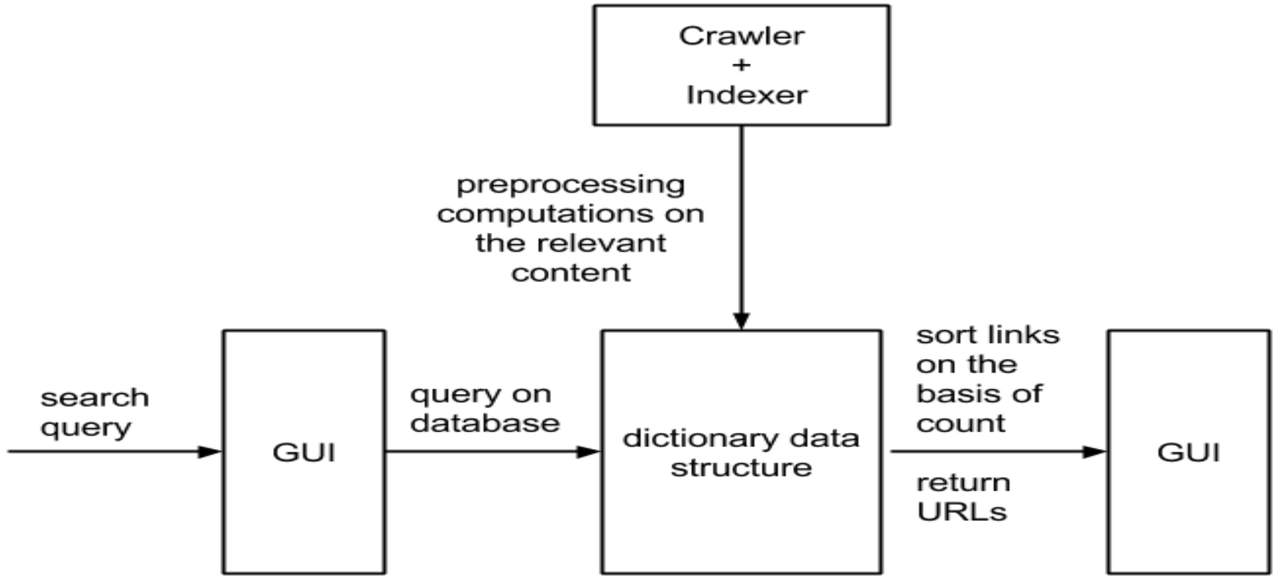
Description of the search engine

Decomposing the Problem Statement gives us three major subparts:

- Web Crawling
  - Browses and saves all the relevant web pages that can be found from the seed page we hard coded in our program.
- Indexing
  - The saved content gets decomposed and gets saved in the data structure based on the words occurrence
- Ranking and Searching
  - The pages will be ranked based on various parameters based on the domain of search engine.
  - The searched query will be decomposed and then searching and ranking of the words will give us the final result.

Search engine for querying academic experts : Working

- In order to make the picture more clear we will analyse the search engine in the way user interacts with the system.
- Figure below shows the various steps which covers both the front end, backend and the pre computations required.


Crawler
+
Indexer
preprocessing computations on the relevant content
search query
GUI
query on database
dictionary data structure
sort links on the basis of count
return URLs
GUI

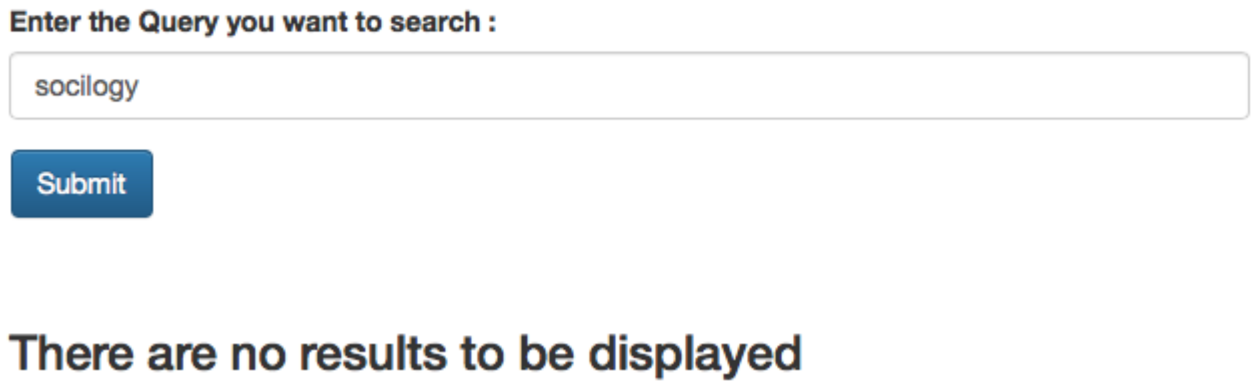
From the query to the retrieval of final results

Given below are the various stages search engine undergoes :

- User enters the query in the text box provided by the interface. Query will be of the form of specialization namely machine learning, pattern recognition, computer vision and so on.
- The searched query will be looked into the database in our case which is known as index data structure.
    - This index data structure is formed as a result of preprocessing done on a set of documents. The operation that are carried out are
        - Crawling
        - Indexing
- Finally, the links from the database gets returned to the search engine script which further sorts them and hence rank them on the basis of frequency of keywords present in the query
- Once the sorting is done, sorted list is finally shown to the user

Step 1 : User enters search query

- The first step of any information retrieval process is where user enters the query he wants to get results of.
- User will be searching for specialization and will enter the same in the text box provided by the web application.
- The web page is loaded from the python script which will be doing all the computations. HTML and CSS of the page is embedded in the script itself which is present on the server.



There are no results to be displayed

Step 2 : Lookup operation

- Once the script has the searched query, next step is to lookup the same in the database which is precomputed before the web application even loaded at the client side.
- The entire database in the form of dictionary data structure is stored on the server.
- This data gets accessed by the script according to the keywords present in the query.

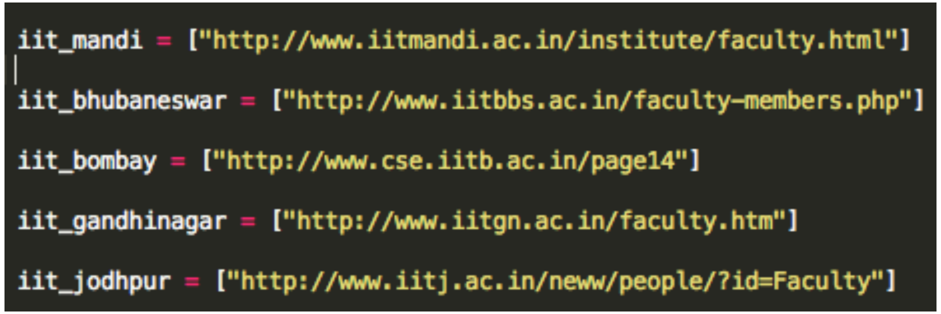
Now we know that there is some database from which all the query gets answered from. Let’s look at what exactly is this database?

In order to know the concept of this database lets look at the components which are behind the formation of this entire data in an organised manner.

Formation of Database

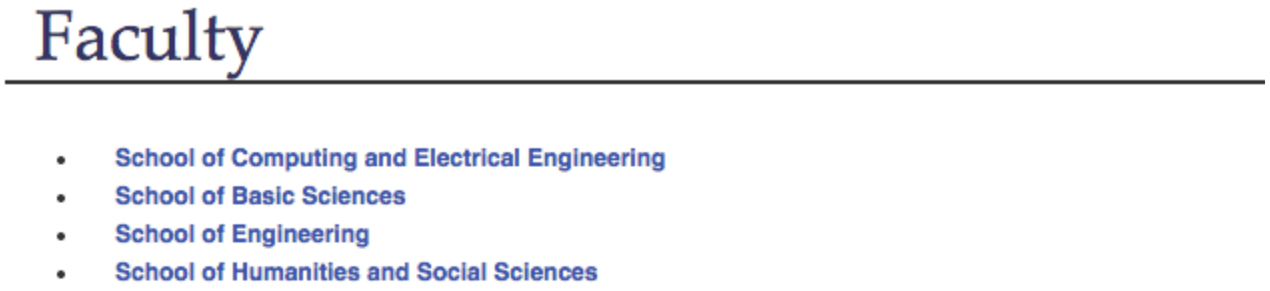
This part deals with the data we will be saving related to the content we are interested in. Two things we will be dealing with is crawling and indexing.

- Crawling
  - Its work is to recursively visit websites starting from the seed pages hardcoded into the system
  - The institutes we will be crawling are IIT Mandi, IIT Roorkee, IIT Bhubaneswar, IIT Gandhinagar, IIT Patna and IIT Jodhpur.
  - Working :
    - Initially we will be having a list of seed pages
    - For each of the seed page we will start exploring all the links that can be reached from that till the point all the URLs are exhausted
    - The same process will be repeated for all the seed pages and at the same time the document will be sent to indexer part so as to keep a record of this document in the database.



```
iit_mandi = ["http://www.iitmandi.ac.in/institute/faculty.html"]

iit_bhubaneswar = ["http://www.iitbbs.ac.in/faculty-members.php"]

iit_bombay = ["http://www.cse.iitb.ac.in/page14"]

iit_gandhinagar = ["http://www.iitgn.ac.in/faculty.htm"]

iit_jodhpur = ["http://www.iitj.ac.in/neww/people/?id=Faculty"]
```

Some of the seed pages that are given as input to the crawler



Faculty

- School of Computing and Electrical Engineering
- School of Basic Sciences
- School of Engineering
- School of Humanities and Social Sciences

Seed page for IIT Mandi

Indexing the Crawled Content

At the time we are crawling web pages comes the part of Indexer as well. Each of the link that is classified to be relevant by the crawler is passed to the indexer for saving purpose. Mentioned below are the steps :

- ❖ Indexer will be provided with the URL of the document
- ❖ In order to analyse the HTML content of the page **URLLib** is used.
  - ➢ Opens the link, checks for the content and return to the user the content of the page if found else returns the error code along with the message.
- ❖ To extract the content of the page, it is essential to format it on the basis of tags. **BeautifulSoup** module is used to achieve the same.
  - ➢ It is mostly used for parsing HTML and XML pages. It creates a parse tree for parsed pages that can be used to extract the data from HTML.
  - ➢ Helps in extracting all the anchor tags by utilising this property.

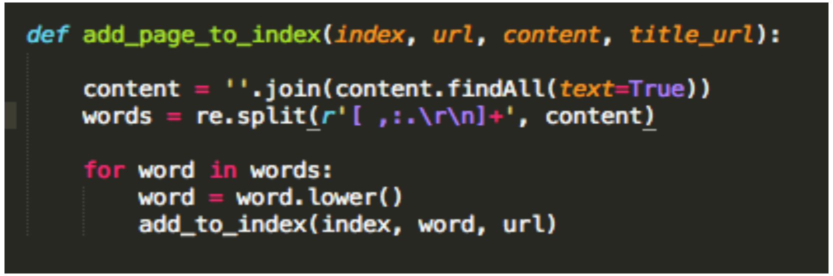
Formation of Database

After the content of the page is in formatted manner, we send it for indexing purpose. In this stage we use a dictionary data structure whose structure is like this
dictionary :: { key : value }

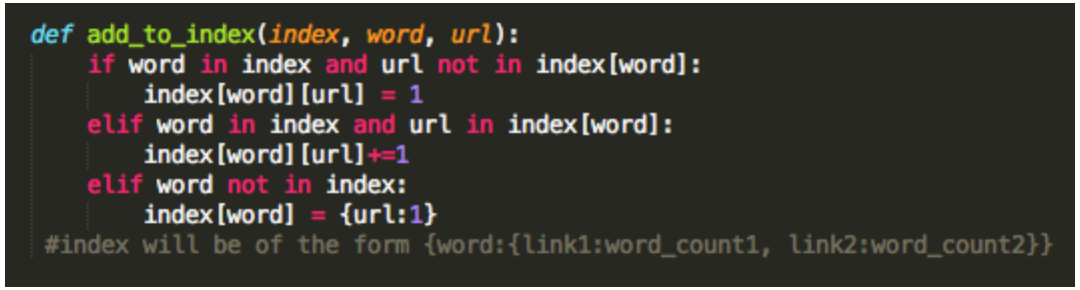
database :: { keyword1 : {url1 : 7, url2 : 9}, keyword2 : {url5 : 13} }

- ❖ Here keyword is a term that is encountered in the page
- ❖ URL is link of the document where the keyword is seen.
- ❖ Count in front of the URL is the number of times the keyword is found in the document. For every word seen in the text of the document three possible cases arises :
  - ➢ Keyword present in the database but URL not present
  - ➢ Keyword present in the database and URL is also present
  - ➢ Keyword not present in the index

- Given below is the corresponding code for the keyword addition in the database.
- *add_to_index* function is called not on the entire HTML of the page but on a modified document with all the HTML tags removed. Given below is the code snippet.



```
def add_page_to_index(index, url, content, title_url):

    content = ''.join(content.findAll(text=True))
    words = re.split(r'[ ,:.\r\n]+', content)

    for word in words:
        word = word.lower()
        add_to_index(index, word, url)
```



```
def add_to_index(index, word, url):
    if word in index and url not in index[word]:
        index[word][url] = 1
    elif word in index and url in index[word]:
        index[word][url]+=1
    elif word not in index:
        index[word] = {url:1}
 #index will be of the form {word:{link1:word_count1, link2:word_count2}}
```

Optimization of lookup operation

- The most important thing every search engine should have is the ability to quickly retrieve results when a query is entered.
- The important factor is to decide which data structure to use which has most optimized lookup time.
- The data structure we will be using for storing the database is dictionary. Data in dictionary is stored as key value pair.
- The average time complexity for lookup operation is *O(1)* which being a constant serves good for the purpose of our search engine.
- Dictionary being a hashmap, its worst case lookup time is *O(n)* where n are the number of elements present in the dictionary.
    - This worst case is due to very bad hash function, resulting in a lot of collisions.
    - Being a implementation in python, it’s very unlikely that every keyword will have the same hash and is added to the same chain.

Working of dictionary

- Python dictionaries are implemented as **hash tables**
- Hash tables allow hash collisions i.e. even if two keys have the same hash value, proper strategy is in place to retrieve the key and value pairs.
- It uses **open addressing** to resolve hash collisions. In this method it is resolved by **probing.**
  - Probing is basically searching through alternate locations in the array until the target record is found or unused array slot is found.
- Hash table is just a contiguous block of memory similar to array. Due to this a constant time for lookup operation is possible.
- While adding entries *(key, value)* pair to the hash table, we start with slot i based on hash of the key. ***(i = hash(key)).*** Cases that arises are :
  - slot empty => (*hash, key, value)* added to the slot
  - slot occupied
    - If the hash and value matches, pair already exists
    - Starts probing till it reaches a empty slot or it encounters a filled slot with same set of values

Ranking mechanism

Till now we have reached a point where if a user enters a query, results are given to the search engine script in the form of a dictionary where key is URL and value is the cumulative count of occurrence of search query in the URL.

The pages that we have taken into consideration are not linked the way normal pages are linked. The web pages are completed restricted to the content of the concerned faculty members only. So the conventional ranking techniques are not applicable here

Now in order to rank the URLs we will use the concept of inverted index. This is why we choose the format of the data structure the way it is.

Inverted index keeps a record of keyword, their point of occurrence and their count in respected document of presence.

Inverted Index

Lets take an example in order to describe the working of inverted index :

Document 1: Web Search and Information Retrieval

Document 2: Search Engine Ranking

Document 3: Web Search Course

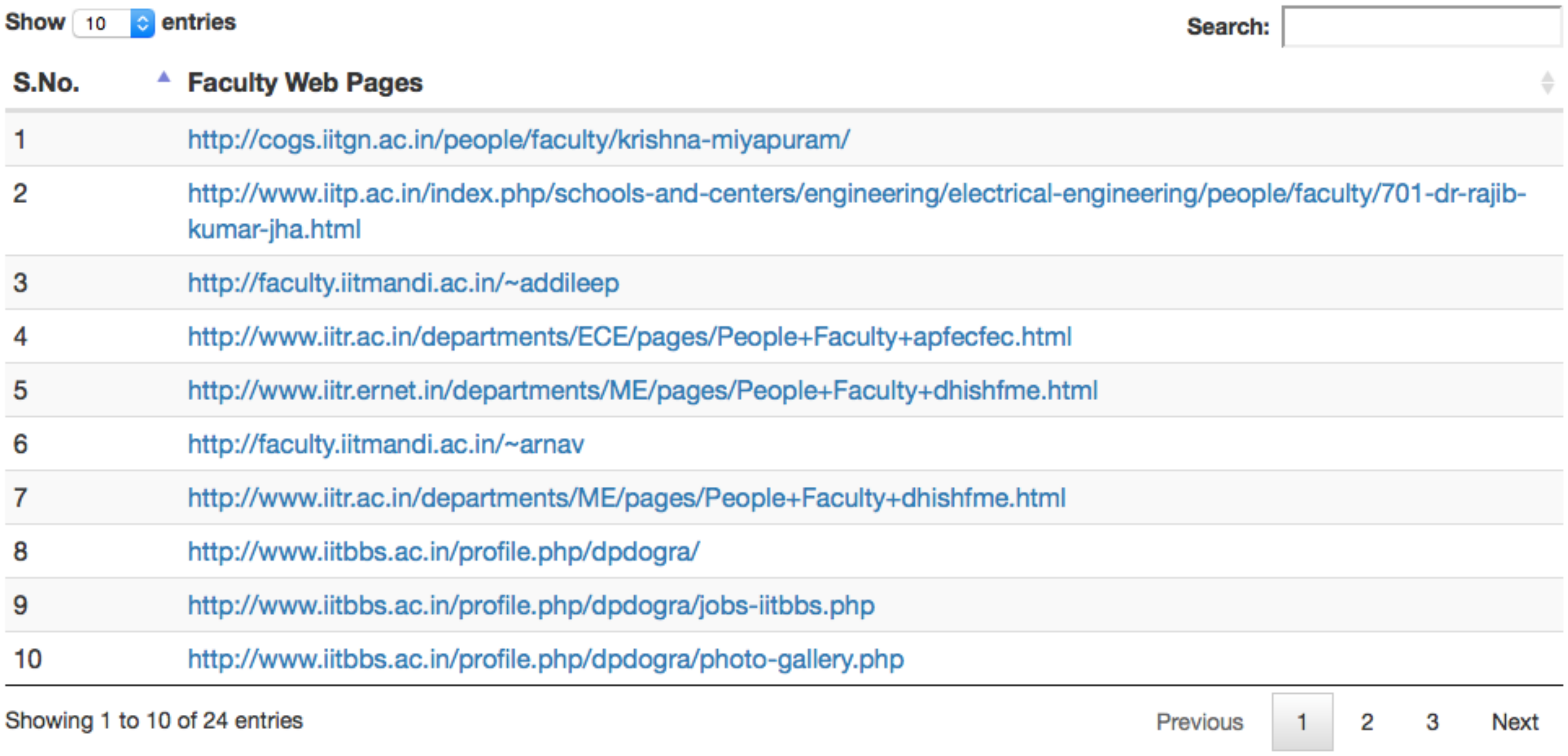
Then the postings list of the term ‘web’ would be the list [1, 3]. It means the term ’web’ appears in documents with IDs 1 and 3.

Similarly the postings list of the term ‘search’ would be [1, 2, 3], and for the term ‘course’ the postings list would be [3].

*Posting list is basically the list of documents where the term appears in.

Final retrieved results

- After completing the processing part, the final part is to display contents to the user.
- Till now, search engine script has obtained all the relevant links and sorted them to get the final order of display.
- Before the display part is done, based on some unique parameters of the URL links are categorised into two sets :
    - First set deals with the pages of faculty members. These are the official pages hosted on institute server.
    - Second set consists of web pages of departments the specialization is related to. In addition to the departments, relevant web pages will also be presented in the final list.
- ***Issue*** :: How to combine python script with the HTML of the web page ?
    - CGI when implemented on the web server provides an interface between the web server and the script that will generate dynamic content.
    - These scripts are usually written in a scripting language such as perl and python.



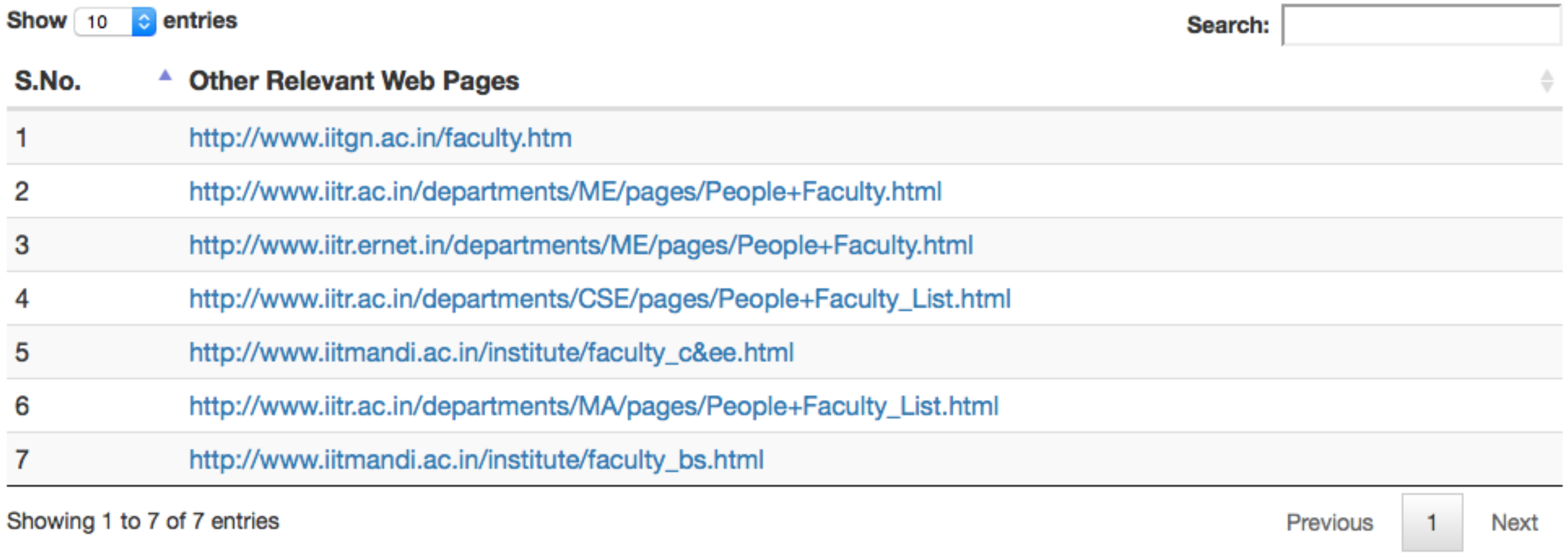
Show 10 entries

Search:

S.No.	Faculty Web Pages
1	http://cogs.iitgn.ac.in/people/faculty/krishna-miyapuram/
2	http://www.iitp.ac.in/index.php/schools-and-centers/engineering/electrical-engineering/people/faculty/701-dr-rajib-kumar-jha.html
3	http://faculty.iitmandi.ac.in/~addileep
4	http://www.iitr.ac.in/departments/ECE/pages/People+Faculty+apfecfec.html
5	http://www.iitr.ernet.in/departments/ME/pages/People+Faculty+dhishfme.html
6	http://faculty.iitmandi.ac.in/~arnav
7	http://www.iitr.ac.in/departments/ME/pages/People+Faculty+dhishfme.html
8	http://www.iitbbs.ac.in/profile.php/dpdogra/
9	http://www.iitbbs.ac.in/profile.php/dpdogra/jobs-iitbbs.php
10	http://www.iitbbs.ac.in/profile.php/dpdogra/photo-gallery.php

Showing 1 to 10 of 24 entries

Previous 1 2 3 Next



Show 10 entries

Search:

S.No.	Other Relevant Web Pages
1	http://www.iitgn.ac.in/faculty.htm
2	http://www.iitr.ac.in/departments/ME/pages/People+Faculty.html
3	http://www.iitr.ernet.in/departments/ME/pages/People+Faculty.html
4	http://www.iitr.ac.in/departments/CSE/pages/People+Faculty_List.html
5	http://www.iitmandi.ac.in/institute/faculty_c&ee.html
6	http://www.iitr.ac.in/departments/MA/pages/People+Faculty_List.html
7	http://www.iitmandi.ac.in/institute/faculty_bs.html

Showing 1 to 7 of 7 entries

Conclusion and future work

Being a prototype, there are things that needs to be added to the project so as to use it on a much larger level. Some of the work that could be done to greatly enhance its usability are ::

- ❖ Ranking Techniques
  - ➢ Currently inverted index is used but once the domain of the project increases conventional ranking techniques namely pagerank and hubs and authorities can be brought to use.
- ❖ Domain of Crawler
  - ➢ As mentioned earlier there are 6 IIT's that are getting crawled.
  - ➢ More IIT's can be taken into consideration by the crawler so as to diversify the search results. Adding to it, other works namely, pdf, doc, jpg can also be taken into consideration
  - ➢ As these documents increases complexity of the crawler, the same has been opted out from the prototype.

- GUI enhancement :: Many more features can be included in the front end of the application. Some of those are mentioned below :
  - Auto query completion. While entering the query suggestions can be showed to the user.
  - Mistakes in the query can be rectified and results from related real query can be brought to the user.
  - Related search terms can be changed in order to dynamically create the links.

The project has been uploaded on Github(open source network) and on public server http://sahilmutneja.com/cgi-bin/search_engine.py so that anyone can access the search engine.

Users can contribute to the open source project by adding their code to the central repository on github and can also suggest changes to the same by following the link http://sahilmutneja.com/contact.php

Point of discussion

Q. Why choose hash tables over data structures like B-Tree, RBT, AVL and other ?

Ans. Dominating factor is lookup time. Its *O(1) vs O(logn)* where n are the number of items. Constant vs logarithmic

Q. Scope of semantic based search engine concept into this prototype ?

Ans. It consider various points including context of search, location, intent, variation of words, synonyms and others. As the domain of project is restricted, there is no much use for this in the prototype.

Q. Why not using pagerank ranking algorithm ?

Ans. Pages are not related to each other or technically the web pages doesn’t form a well connected graph between various components.

Q. Why not include pdf ?

Ans. In order to crawl documents like these, crawler needs to be designed in such a manner that it loads the document and extracts all the content. Changes in crawler should decode the content of the documents which are directly present on the server.