

BUILDING A SEARCH ENGINE FOR QUERYING ACADEMIC SPECIALISTS

*MTP Report submitted to
Indian Institute of Technology Mandi
for the award of the degree*

of

B. Tech

by

SAHIL MUTNEJA

under the guidance of

Dr. Dileep A. D.



SCHOOL OF COMPUTING AND ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY MANDI

JUNE 2015

CERTIFICATE OF APPROVAL

Certified that the thesis entitled **BUILDING A SEARCH ENGINE FOR QUERYING ACADEMIC SPECIALISTS**, submitted by **SAHIL MUTNEJA**, to the Indian Institute of Technology Mandi, for the award of the degree of **B. Tech** has been accepted after examination held today.

Date :

Mandi, 175001

Faculty Advisor

CERTIFICATE

This is to certify that the thesis titled **BUILDING A SEARCH ENGINE FOR QUERY-ING ACADEMIC SPECIALISTS**, submitted by **SAHIL MUTNEJA**, to the Indian Institute of Technology, Mandi, is a record of bonafide work under my (our) supervision and is worthy of consideration for the award of the degree of **B. Tech** of the Institute.

Date :

Mandi, 175001

Supervisor

DECLARATION BY THE STUDENT

This is to certify that the thesis titled **BUILDING A SEARCH ENGINE FOR QUERY-ING ACADEMIC SPECIALISTS**, submitted by me to the Indian Institute of Technology Mandi for the award of the degree of **B. Tech** is a bonafide record of work carried out by me under the supervision of **Dr. Dileep A. D.**. The contents of this MTP, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Date :

Mandi, 175001

Sahil Mutneja

Acknowledgments

First and foremost, I would like to thank my mentor **Dr. Dileep A.D.** Without his assistance and dedicated involvement in every step throughout the process, this paper would have never been accomplished. I would like to thank you very much for your support and understanding over the past year during the course of the project. His technical and editorial advice was essential to the completion of this dissertation and has taught me innumerable lessons and insights on the workings of academic project in general.

I would also like to show gratitude to my committee supervisors, namely **Dr. Arnav Bhavsar, Prof. B. D. Chaudhary** and **Dr. Satyajit Thakor**. During the review meetings there were many precious points raised by the committee that greatly helped me in understanding the core concepts of the project. I would also like to thank them for reading previous drafts of this dissertation and providing many valuable comments that improved the presentation and contents of this dissertation.

Getting through my dissertation required more than academic support and I have many people to thank for listening to me and at times having to tolerate me over the past two semesters. I cannot begin to express my gratitude and appreciation for their friendship. There are many who have been unwavering in their personal and professional support during the time I spent at the college in my final year. I would also like to thank my family for supporting me every time I felt demotivated.

Sahil Mutneja

Abstract

This thesis will talk about the prototype of a search engine that takes specialization as input and returns back the list of relevant faculty members as output in the order of priority. Crawling, indexing and ranking are basically the three things that this model will be performing in order to return the results. The crawler in totality will be covering a total of six IIT's in its domain. The results that the search engine will return will be from these institutes only. The crawled pages will then be saved in the index where for each of the word we will have the location of its occurrence i.e. URL of the document. In addition to the location, count of the word in the link will also be saved. This will be used by the ranker to compute the order the results will be displayed in. Finally it will be the user who will enter the query and based on the keywords used in the query relevant links will be returned. The relevance of the links will be decided by the number of occurrences of these keywords in these links. A GUI for the engine will help the user to interact with the system and analyse the results more efficiently.

Keywords: *World Wide Web, Search Engines, Information Retrieval, Inverted Index*

Contents

Abstract	iii
Abbreviations	vii
List of Figures	1
1 Introduction	2
1.1 Decomposing the problem	2
1.2 Objective and scope of the work	3
2 Search engine	5
2.1 How Search Engines Work	6
2.2 Search engine modules	7
2.2.1 Document processing	7
2.2.2 Query Processor	10
2.2.3 Search and matching function	11
2.2.4 Ranking	12
3 Search engine for querying academic experts	16
3.1 Input to search engine	19
3.2 Database of URLs based on crawled content	20
3.2.1 Crawled Content	21
3.2.2 Indexing the Crawled Content	23
3.2.3 Optimization of Lookup Operation	25
3.3 Ranking mechanism	26

3.4	Final retrieved results	27
3.4.1	Issues to display content	27
4	Exerimental Studies and Results	30
5	Conclusion and Future Work	34
	References	36

Abbreviations

CGI	- Common Gateway Interface
CSS	- Cascading Style Sheets
DS	- Data struture
HTML	- HyperText Markup Language
II	- Inverted Index
IDF	- Inverse Document Frequency
JS	- Javascript
NLP	- Natural Language Processing
SE	- Search Engine
SEO	- Search Engine Optimization
TF	- Term Frequency
URL	- Universal Resource Locators

List of Figures

2.1	Illustration of PageRank	15
3.1	Block diagram of search engine	17
3.2	Flow diagram of search engine	18
3.3	Use case diagram of search engine	19
3.4	Text box to get user query	20
3.5	Seed page for IIT Mandi	22
3.6	List showing faculty URLs	28
3.7	List showing other relevant URLs	29
4.1	Look up time for small set	32
4.2	Look up time for large set	33

Chapter 1

Introduction

The problem we are addressing here is to build a search engine that will provide the user with a list of experts when queried against a specialization. The experts that will be displayed will be from the premium institutes such as IITs spanning across India. The list of professors will be such that the most relevant professors will be shown towards the top and so on in decreasing preference. The result will comprise of the links of web pages of the specialist along with the departments they belong in the respective institute.

1.1 Decomposing the problem

The problem of building a search engine can be decomposed into three major subparts, written below, solving which will solve our problem.

- **Web Crawling** : The work of crawler is to recursively discover new and new pages, starting from the ones hard coded into the script and filter the ones which are most appropriate for the purpose of our meta search engine. It also makes sure that the spam pages are avoided when the crawler forward the pages to the indexer.
- **Indexing** : In this phase, the forwarded content from the crawler is now analysed and gets saved in the data structure, where for each of the encountered word we will save the link of the page we saw the word in. In addition to the location of the word, count of the word in the respected document will also be recorded. Now, for each of the

encountered word we will be having links of the pages the word is associated with along with the count in that given page.

- **Ranking and Searching** : This part deals with computing the rank of the pages we have in our indexer. The algorithm that we use out here is modified inverted index which considers word frequency for rating purpose. More the frequency in a given page, higher will be its relative rank. Once we have the ranks computed with us, which is done behind the scenes, we come to the part where user queries something. The queried phrase gets split into words and then gets looked up into the indexer from which links get returned. These links will then get sorted on the basis of the ranks search engine now have and finally gets displayed to the user.

Out of the things mentioned above, crawling and indexing are the things that are done offline. The output of the same are saved in a text file which will be later referred to by the ranker. The real time query as a result is quick and takes constant amount of time to return the links per query searched.

1.2 Objective and scope of the work

The project aims at building a search engine that performs specific sort of work on a large scale. The problem with building a search engine on such a big level are the vast number of intricacies that comes with it. In order to meet with the requirements and also build a functional search engine we will be doing the following mentioned things that will provide a base for further expansion of the project. The scope and limitations are as follows :

- As the title of the thesis tells us, we are building a search engine that is specific to our cause. The scope here is well defined, build a search engine that responds to the query in the form of specialization. The output will consist of the links of the concerned professors from the list of institutes hard coded into the system.
- The number of institutes that are included are six and for others only the department pages are included. Increasing the number of institutes requires large crawling power

and higher computation power of the system in hand. This in turn will require large computation power and more memory, hence larger number of machines syncing together in a distributed environment.

- The search results will comprise only of the links of the concerned professors and faculty members. It will not consist of other different pages linked with the professors namely, their facebook pages, their social networking profiles and other academic work they are associated with namely their research publication and other related work in the form of pdfs and docs. To include these as well, crawler will be required to crawl a large proportion of the web, which not only being not feasible for the scope of our project but will also require a large computation power.
- As the number of pages crawled are limited, the conventional ranking mechanism will not be suitable for this case. In spite of using that or any other high scale ranking algorithm namely page rank, hubs and authorities etc. which requires a great amount of research and groundwork, we will be sticking to the inversion frequency algorithm and similar sort of works in order to rank them in order.

Chapter 2

Search engine

A search engine [1] is basically a computer program that gets run on a server and is used to search data spread across the internet for some specified search query. Search engine in its working comprises of spiders which are also referred to as bots. The work of spider is to roam around the wide web graph and search for various universal resource locators (URLs) and keywords. Roaming around the web allows the crawler to gather information and send the collected data back to the program so that the program can now classify and categorise the data in a form where for every keyword found, mapping with the corresponding links is done. The main advantage of having this database is that if now a user searches for something then there is no need to repeat the complete process of crawling the entire web. In spite of what happens is that the query gets directly searched in the database and results in the form of relevant URLs get returned. In the search engine industry over the last decade several significant changes have been noticed and thereafter taken into development process. A user can now easily get a ton of information about almost anything by merely entering a few keywords that the user is interested in. In short, summarising the working of search engine one can say that it first gathers information by crawling the wide network of documents, then save the content from the documents in a format that depends on the purpose of search engine and finally retrieve and display results in the form of URLs based on numerous factors and sorting them based on these factors.

2.1 How Search Engines Work

On the vast expanse of the World Wide Web (WWW) its really difficult to get specific information relevant to our query. To make this thing easy for the users search engines come into picture. Without the presence of search engine its almost impossible to locate anything on the web unless we know the exact location where the content is actually located which is not how internet works. When people refer to search engine, they basically refer to the large database from which relevant results gets returned from to the user. Based on the manner search engine maintains their database they are classified into three types :

1. Powered by robots also known as crawlers, ants or spiders.
2. Powered by human submissions
3. Combination of the two

Crawler based search engines use the program known as crawler for its working. It roams around the web in order to visit various websites, read the site's metatags and gather the information on the actual site. The most important thing it does to expand itself is by recursively following the links that the site connects to (links present in the content of the page) and performing indexing on all the linked websites. The crawler then returns the information back to the central repository where the data is indexed. In order to have updated content in the database it periodically visit websites and update the changes in the indexer if encountered any. The frequency of this automated rechecking of the websites is something which is decided by the administrator of the search engine. Human powered search engines as the name implies is dependent on the content published by the humans for the purpose of the engine. Only the submitted information is indexed and subsequently catalogued. A search engine or information retrieval (IR) system comprises basically of four different modules :

- Document processing
- Query processing
- Functions for search and matching

- Ranking techniques

In the next section we will be taking into account these modules [2] which forms the basis of any search engine.

2.2 Search engine modules

2.2.1 Document processing

This initial phase deals with processing and parsing the documents the user will be searching against. Given below are the steps that are performed by the document processor in order to classify the meaningful sections from the pages. Depending on the processor some or all of the steps are followed :

- **Input processing** : While crawling the web there will be different sort of documents that will be encountered by the crawler. Some of the documents out of these will be relevant and some will not be relevant. In order to standardize the various formats so as to further simplify the processing of documents we use input processing technique. It merges all the data into a data structure that will serve the purpose of the search engine. The need of this is of relative importance as well formed and consistent format is directly related to the extent of sophistication in document processing. More properly the data is organised and processed, more effective will be the way lookup operation will be carried out in the later course.
- **Potential elements for indexing** : Indexer is responsible for building a database from which the search queries will be answered from. The design of data structure depends on the word term. Depending upon the domain of the project, the definition of the term can be anything. There is another concept that comes here which is known as tokenizer. This program is used to define the term which will be suitable for indexing. Depending on the search engine there are different rules that document processor will execute.

- Removing stop words from the documents** : There are various documents that are crawled and further forwarded to the indexer stage. This step will go through the entire document and for every word encountered, it will add to the database the keyword mapped with the URL of the document. In order to save system resources some of the terms are eliminated and hence are not considered for further processing. These are the terms that have little or almost no value in finding the useful documents in response to the user query. Such words are known as stop words and may comprise up to 40 percent of the document under consideration. Some of the examples of stop words are *a, the, and, but, in, over, he, it etc..*. One option in order to not include these words is to modify the document and remove the presence of all the stop words and other option which is widely used is to ignore these words at the time we are updating our database. Hence all the stop words will be eventually deleted from the final database.
- Term stemming** : This is one of the thing that helps in giving the indexer a better shape. It reduces the number of unique words present in the index by removing the variations of a word present in the document. It works by recursively removing the word suffixes in layer by layer order of processing. Talking about the efficiency, it helps in reducing the storage size of the index and hence speeds up the lookup operation. In terms of effectiveness of stemming, it reduces all forms of the word to the stemmed or base form. So for all the variations present in the document only one will be forwarded to the index data structure.

With positive comes negative aspect of stemming as well. Whenever stemming is done related terms are not included in the index data structure. This will result in a loss of precision which would not have happened if search engine would have included the variant forms as well. In case a user asks for something, for example, lets say user asks for analysis then only the stemmed term results will be displayed to the user which is not something that the user has asked for in the first place. Inspite of querying the actual word, stemmed word results will be returned which in some cases is useful and in some not. Results for analy- will be displayed even if the user asks for

something like analysis, analysing, analyzes, analyzed and analyzer.

- **Index entries extraction** : Now after completing the above mentioned steps all the entries gets extracted and separately monitored by the document processor. These terms are then inserted and stored in an inverted file that lists the location of occurrence of the keyword along with the frequency of occurrence.
- **Term weight assignment** : For each of the term present in the index file some weight gets assigned to it. One of the easiest way to do this is to assign a binary weight of 0 if the term is absent and 1 if the term is present. Changing the techniques used for weighting mechanism will directly affect the complexity of the search engine. Analysing the frequency with which a term appears in a document creates more complex weighting. On top of it length normalization will add more sophistication. For optimal weighting comes the concept of term frequency (TF) and inverse document frequency (IDF). This algorithm deals with the frequency with which term is present in a document. After getting the count within the document it looks in the entire database for its frequency. After comparing both these counts it finally evaluates the term.

The ranking takes two ideas into account for weighting. The term frequency in the given document and the inverse document frequency of the term in the entire database. The term frequency in the given document shows how important the term is in the given document whereas the inverse document frequency measures how generally the term occurs across various documents. If a words appears more frequently in a document then a high score is given to the word and if the word appears in many documents then a low score is given to the word. Words appearing frequently in a single document will be scaled up whereas commonly occuring word such as *the, is, are, for* will be scaled down.

- **Index creation** : The index or inverted file is the internal data structure that is used to store the vast amount of data search engine deals with. Depending upon the type of search engine there can be various type of data structures that can be taken into

practice. The entries that are present can range from a very simple listing of every alphanumeric sequence to a more linguistically complex list of entries. With each of the entry comes the mapping to the list of documents where it refers to. The document will be represented in the form of URL from which the page gets accessed. More and more complexities can be added to the index to create more functionalities for the user. But the issue that arises is of space i.e. more the information one want to store, the slower will become the lookup operation. A line hence needs to be drawn between the two so as to maximise the user search experience.

2.2.2 Query Processor

Query processing has a total of six possible steps that could be performed on the query user searches for. Depending on the search engine and how the back end processing is carried out these steps could be cut short and at any number of step the system can proceed to match the query to the inverted file. More steps and more documents make the process more expensive for processing in terms of computational resources and responsiveness. However, the longer the wait for results, the higher the quality of results. Thus, search system designers must choose what is most important to their users - time or quality. The steps in query processing are :

- **Tokenize the query terms** : As soon as the user inputs a query the search engine whether a keyword based system or a full natural language processing (NLP) system must tokenize the query stream i.e., break it down into understandable segments. Usually a token is defined as an alphanumeric string that occurs between white space and punctuation. Each of the significant word present in the query is treated as a token.
- **Parsing** : Since users may employ special operators in their query including proximity, boolean or adjacency operators, the system needs to parse the query first into query terms and operators. These operators may occur in the form of reserved punctuation (e.g., quotation marks) or reserved terms in specialized format (e.g., AND, OR). In the case of an NLP system, the query processor will recognize the operators implicitly in the language used no matter how the operators might be expressed (e.g., prepositions,

conjunctions, ordering).

At this point, a search engine may take the list of query terms and search them against the inverted file. In fact, this is the point at which the majority of publicly available search engines perform the search.

- **Query creation** : Depending on how the system does its matching search engine creates a query representation. If a statistically based matcher is used, then the query must match the statistical representations of the documents in the system. Good statistical queries should contain many synonyms and other terms in order to create a full representation. If a Boolean matcher is utilized, then the system must create logical sets of the terms connected by AND, OR or NOT.
- **Weighting the query term** : The final step in this phase computes weights for the terms present in the query. Different weight is assigned to the query term depending on how much weight we want to assign to it and how much weightage we need to give to various terms.

2.2.3 Search and matching function

Depending on the problem search engine is based on, different models of information retrieval systems are used. Different models carry out their search and matching functions differently and it all depends on the domain search engine works in. Searching the inverted index file in accordance to the searched query is simply referred to as matching. This sort of matching is basically a standard binary search and will keep on processing until keyword is finally figured out to be present or absent.

While the computational processing required for simple, unweighted, non boolean query matching is far simpler than when the model is an natural language processing (NLP) based query within a weighted, boolean model, it also follows that the simpler the document representation, the query representation and the matching algorithm, the less relevant the results except for very simple queries, such as one word, non ambiguous queries seeking the most

generally known information.

Based on some factors namely term frequency, TF/IDF, query term weights and boolean logic fulfillment scores are given to the documents. This score basically is a measure of similarity of the query term with the document. Based on this score a set of documents or pages is formed. This scoring algorithm is based on the way keywords present in the search query is organised in the document. Some of the search engines inspite of using this technique works on the technique that doesnt look at the content of the document rather looks at the relation between various documents and the past retrieval history of documents/pages.

After we are done calculating the similarity of every document present in the subset of documents, search engine presents an ordered list against every query searched for by the user. The complexities while ordering the links relevant to the query depends on the model used by the system. It also depends on the query weighting mechanism and richness of the document.

2.2.4 Ranking

The most important thing that makes the search engine useful for users is the way it shows the results. The ordering of URLs is the deciding factor that makes a search engine useful. The URL is basically the location of document or a web page on the server. Clicking the link will allow the user to access the document. Some of the search engines provide a platform to the users to add feedback to the way results are obtained. Search engine accepts the suggestion and in real time updates the results in order to meet the user demand. Based on the suggestion received by the user, query gets modified and revised query gets looked up. So now rather than searching the original query, a new query gets searched into the database. Now the results that will be shown will meet the user demand. Some of the techniques that search engine use to rank various pages present in the database are mentioned below :

- **Term frequency** : Term frequency is one of the widely used technique used by the search engines today to rank the pages retrieved from the index. Once all the relevant

documents are returned from the database based on the query entered by the user, the keywords frequency in the URL will decide which link will be shown at the top. Along with the link, cumulative count based on the keywords present in the query will be calculated. Links will now be sorted based on these calculated cumulative count. Higher the count, more the rank and hence more towards the top result will be displayed. The eventual ordering will be decided by the rank based on the count calculated.

- **Location of keywords** : Other than the frequency of the keywords in the document, results are also dependent on the location at which the keywords mentioned in the query are present. Preference is given to words found in the metadata, title or lead paragraph of the document. The location at which these keywords are present indicates its significance to the document in which it is present. A high weight will be given to a document if the keyword is present in the title of the page as it represents the essence of the document. If keyword is present in the body of the document, comparatively lower weight will be given to the document. On the same grounds it can also be said that a keyword present in the first paragraph or section heading is more likely to make the document relevant.
- **Link analysis** : Search engines in today's world have incorporated in them various different and advanced features for weighting and ranking pages. Link analysis is now done based on how well connected various pages in the network are. Not only this hubs and authorities not only considers the inlinks and outlinks of various pages but also considers how important page is sending a outlink and how good is the content of the page where inlink is coming into. In order to get an insight into the ranking mechanism lets look at pagerank. It is one of the widely used algorithm that is used by search engines today. This algorithm analyses the inlinks and outlinks of every page and assigns a rank to it. These ranks are then sorted in decreasing order and finally displayed to the user. Pagerank algorithm using mathematical equation is described below [3].

Let x be a web page. Then

- $L(x)$ is the set of websites that link to x
- $C(y)$ is the out-degree of page y
- α is probability of random jump
- N is the total number of websites

$$PR(x) := \alpha \left(\frac{1}{N} \right) + (1 - \alpha) \sum_{y \in L(x)} \frac{PR(y)}{C(y)}$$

Note that the pageranks form a probability distribution over web pages, so the sum of all web pages pageranks will be one. Pagerank or $PR(A)$ can be calculated using a simple iterative algorithm and corresponds to the principal eigenvector of the normalized link matrix of the web. The pagerank concept is well defined in Figure 2.1.

In the figure, the alphabets are depicting pages or URL and the lines depict the in-link and outlink from the pages. An arrow pointing from A to B means, in the content of page A there is an URL that represents page B. The pagerank states that if the number of inlinks of a page is more, its page rank will be higher. The percentage in the sphere represents the rank of the page. Greater the percentage of a page, more the page rank value and hence higher is the probability a user will want to access this page.

- **Date of publication** : One of the things that is taken into consideration while returning results to the user is how recently the article is originally published. Search engines assume that more recent the article, the greater will be its relevance and more probable it is that the user wants it.
- **URL popularity** : In order to determine the relevance of a page, search engine usually tends to add the popularity of the URL as a factor as well. Popularity of the URL is decided by the number of clicks the URL has received. The number of clicks basically means number of times the website is visited by various users using internet. More the number of visits, higher will be the relevance of the link. It is only useful when we are comparing two documents who have equally matched our search criterion.

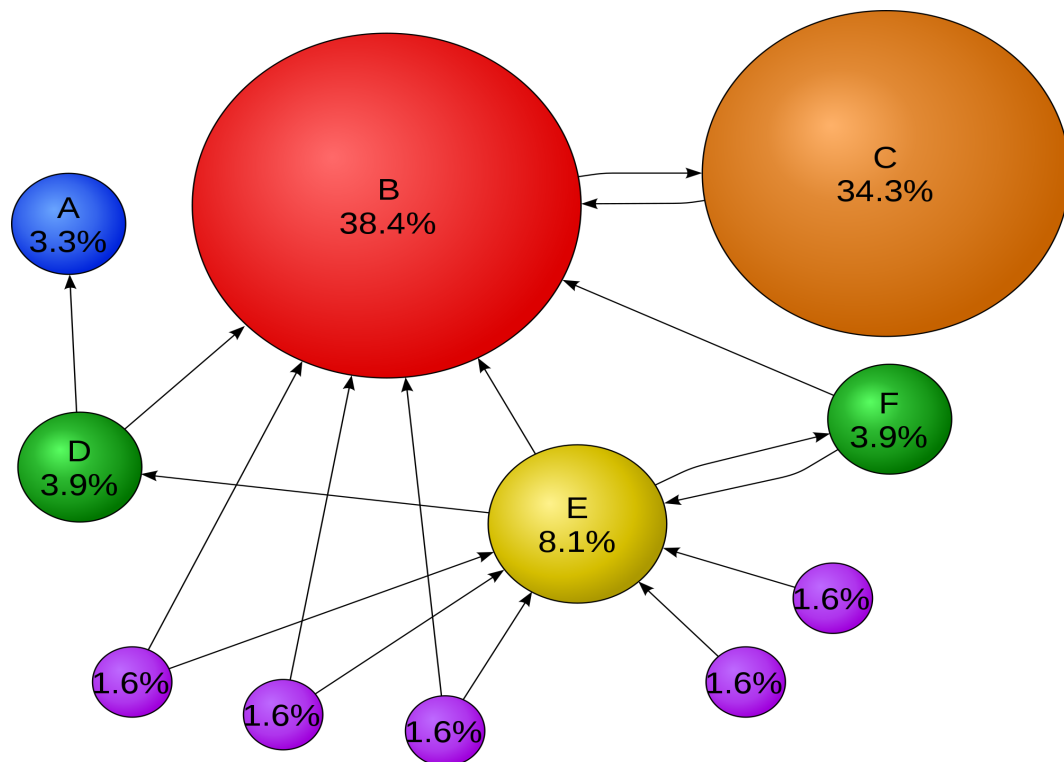


Fig. 2.1: Illustration of PageRank

- Occurrence vs the length of document :** If two documents contains a certain keyword with equal frequency then the one with lesser length will be more relevant to the search engine. The ratio of term frequency to the length of the document needs to be higher if a document needs to be classified as more relevant. Higher ratio will signify the presence of search query in a shorter set of text in the document.
- Proximity of query terms :** Search query can be composed of a single word or can be a multiple worded phrase as well. In order to fetch the relevant documents other than the above mentioned techniques, search engines also use technique that checks for the distance between the keywords present in the query. Lesser the proximity of words inside the document more will be the chance that the document meets the required demand of the user. If the words are far apart chances are there is no relation to what the phrase intended to ask in the first place.

Chapter 3

Search engine for querying academic experts

This section deals with the technical explanation of the project i.e. process of building a search engine application that responds to the user query given in the form of specialization. The output will be of the form of relevant faculty members sorted in the order of priority. In order to make the working of the search engine more clear, a diagram depicting the various building blocks of the engine is shown in Figure 3.1. The model that is achieved out here involved research and study of the existing search engines and in depth analysis of the way search engines works. The figure tells us about the various stages, both front end and back end, that undergoes from the point a query is entered to the point results are returned to the user.

The figure clearly shows various stages search engine undergoes. Starting from the user entering a query into the graphical user interface (GUI), the query going to the dictionary where data is stored in the form of key, value pair i.e. term, list of URLs form. This data comes from the initial preprocessing done by the system and involves crawling of pages and indexing of those crawled web pages. Once in the back end we have the list of URLs, next step is to sort them on the basis of frequency of the keywords we searched for. After sorting is done final step is to return the modified list and show the same on the GUI in two different sets. First one of which is faculty URLs and the second set is the URLs of the related pages and department pages.

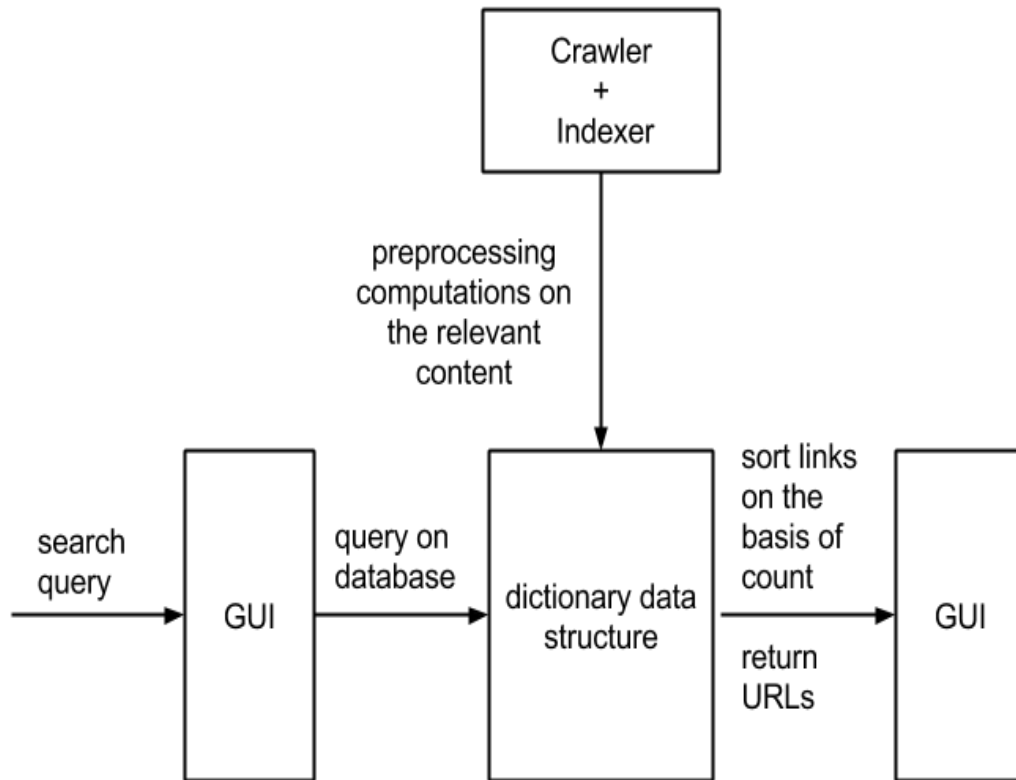


Fig. 3.1: Block diagram of search engine

In order to make the back end of our search engine more clearer, a flow diagram of the same is shown in Figure 3.2. The figure tells us about the various stages the query entered by the user undergoes till the final result in the form of links is displayed.

The first step is query validation. The entered string is processed and only the relevant words are taken into consideration for further processing. The second step is the searcher. The words that are forwarded from the previous step are now looked up in the index data structure and the relevant links are loaded into the memory. The third step is the ranking. The links that we have in memory already have a rank associated with them. Now this pair of link and rank is sorted so that the link with the maximum rank is placed first and so on. The final step is to return the links in the order formed in the previous step.

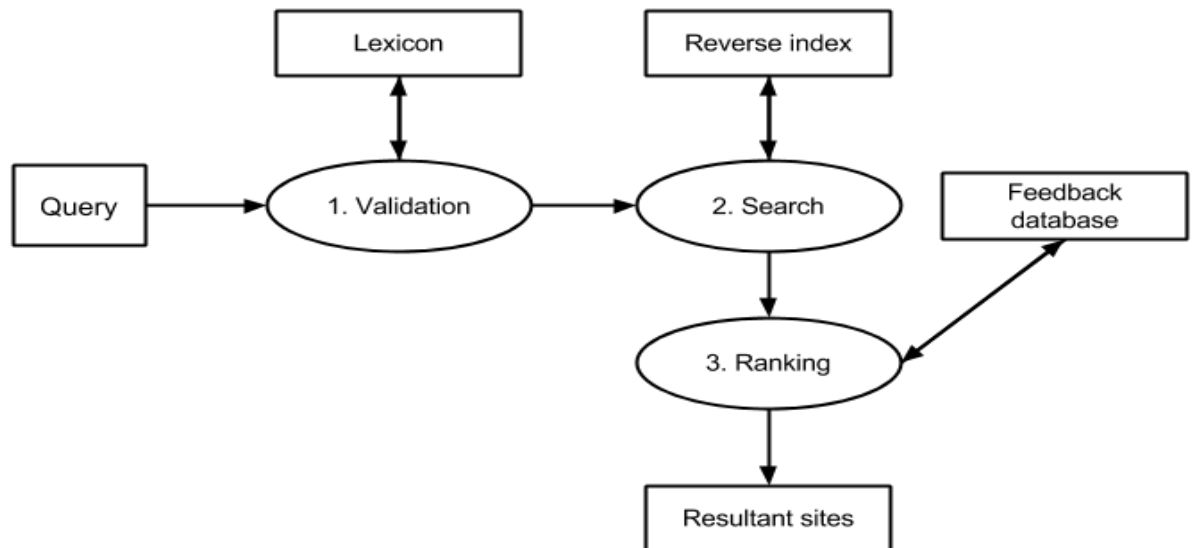


Fig. 3.2: Flow diagram of search engine

In addition to the flow diagram, a use case diagram is shown in Figure 3.3. It tells us about the interaction between the users and different cases a user is involved in. As depicted in the figure, there are three types of user that interacts with this search engine. The first one is the user that will access the engine. This user will submit a query which will be further processed and finally results will be displayed. The second type of user is the web developer. This user will write the scripts and will host the search engine on the web server platform to make it accessible to everyone. The third type of user is the admin. This user decides the functionalities, algorithms, technologies and the rules to be followed while building the search engine.

In the sections below we will be talking about the working of the search engine in the order

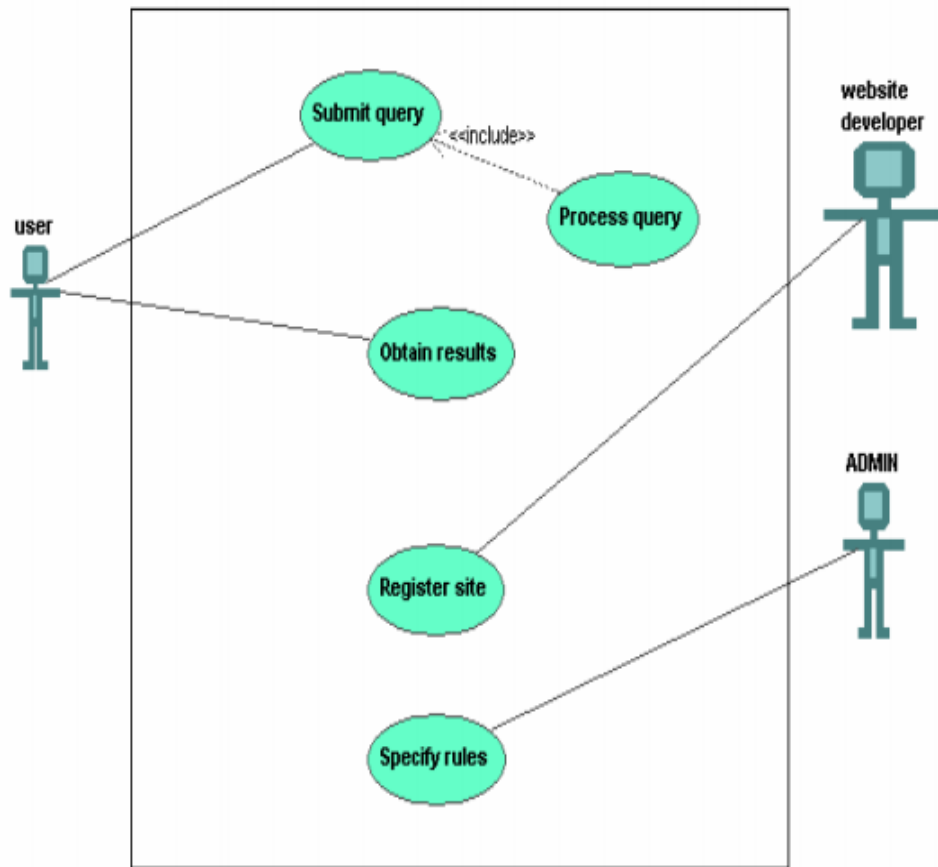


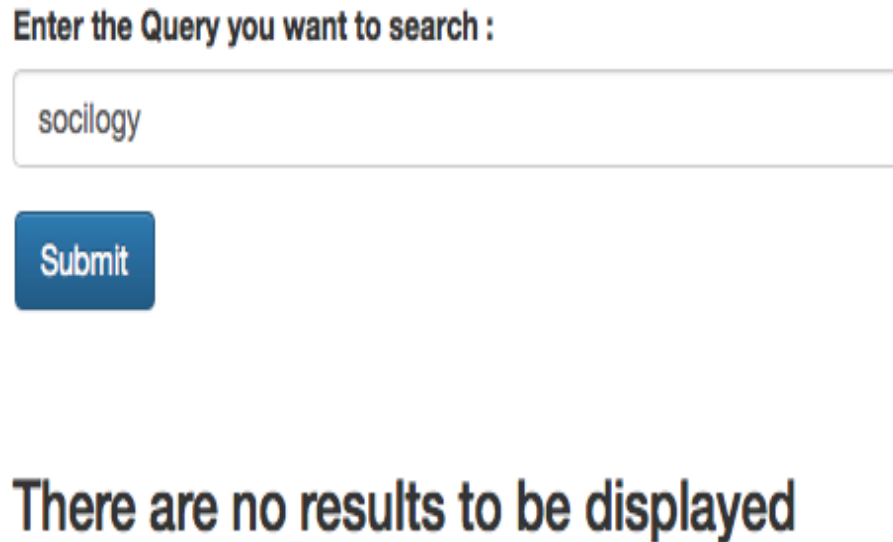
Fig. 3.3: Use case diagram of search engine

user will interact with the system. The entire description will be divided into four sections namely GUI where user will enter the query, database where the data in the form of urls and term frequency will be obtained from, ranking of the retrieved links based on the count of terms and finally the GUI which will show back the results to the user. The database part will be further divided into two sub sections, crawler and indexer.

3.1 Input to search engine

This section will talk about the interface that will be provided to the users for querying specializations. The user will be provided with a medium where a text box as shown in Figure 3.4 will be displayed. The user will enter the specialization in this text box. Some of

the examples of search query user will be giving to the system are machine learning, pattern recognition, computer vision, data structures and algorithms and so on.



Enter the Query you want to search :

Submit

There are no results to be displayed

Fig. 3.4: Text box to get user query

The technology used for creating this web page is hypertext markup language (HTML), cascading style sheets (CSS) and javascript (JS). HTML is used to create the actual content of the page i.e. HTML defines the basic structure and the contents of a website. CSS on the other hand is responsible for the design of the webpage i.e. how every thing looks and where it is on the page. On top of the content and design comes javascript which is responsible for everything that has to change or get animated after the website gets loaded. For the given case its only CSS and HTML which is in action, but once we have the results to be retrieved from the server JS will also come into picture along with common gateway interface (CGI).

3.2 Database of URLs based on crawled content

After user enters the query comes the part where results needs to be retrieved. In this phase, searched query gets split into keywords which in turn gets looked up into the dictionary data structure where for each of the keyword mapping is done with the list of URLs along

with its count in this document. In order to get this database there is crawling and indexing that is done on a set of documents. In the present case the list of documents varies around six IITs. The database will comprise of keywords from all the relevant URLs that will be encountered by the crawler. In order to more efficiently describe how this database is formed and stored in dictionary data structure, let's look at the two discrete steps separately. In the first subsection we will be talking about the crawler and in the second subsection indexer is defined along with the way data is stored in the dictionary data structure.

3.2.1 Crawled Content

Crawler work is to recursively visit websites that can be reached from a given website and maintain a record of it. The initial pages are termed as seed pages. These are the pages from which exploration of rest of the pages begins. For the case of IIT Mandi, seed page is shown in Figure 3.5. Depending on the number of institutes that needs to be taken into consideration, number of seed pages varies. So, all the start pages are taken in a list(array of URLs). Now these links get explored one by one and all the relevant pages of that particular institute gets explored. So if IIT Roorkee seed page is considered, all the relevant pages that comes under the domain of IIT Roorkee will be explored and further processed. The link which is taken into consideration is analysed for its HTML content and all the anchor tags (links/URLs) present in the HTML of the page gets stored in the queue. In this way the queue keeps on growing with every new page found. As queue follows first in first out (FIFO) ordering, the links get selected from the queue and checked whether it satisfies the purpose of the project or not i.e. relevance of the link is checked. For our purpose, pdf, social profile of the faculty is not something that will serve our purpose. If the link satisfies the constraints, it is taken into consideration else its discarded. The popping operation of the queue keeps on going until there are elements left in the queue.

For the search engine we are dealing with, the content we are crawling is limited to six IITs and for the rest of the IITs we have taken into account the departments of the same. The IITs whose faculty members we have considered are IIT Mandi, IIT Roorkee, IIT Bhubaneswar, IIT Patna, IIT Gandhinagar and IIT Jodhpur. Adding more aspects to the crawler will require

Faculty

- [School of Computing and Electrical Engineering](#)
- [School of Basic Sciences](#)
- [School of Engineering](#)
- [School of Humanities and Social Sciences](#)

Fig. 3.5: Seed page for IIT Mandi

higher processing power and more computation power of the system. The search engine here deals only with the official web pages of the relevant faculty members. The thing we are not crawling and hence not including in our results is the related documents of the concerned faculty members. The main focus out here is to build a basic system/prototype where we can see the power of searching and retrieving results of some specialization on a small scale. This basic prototype can act as a platform on which further extension can be made and thereafter can be used at a full fledged larger scale.

Towards the end of the crawler, we also updated a data structure (graph) of the data type list of lists. For each of the link we added to this graph all the outlinks that could be found on this page. This for the information retrieval we are dealing with is not useful, but it can be used for analysing the graph structure formed once we extend the domain of the crawler. This will help in analysing the connectivity of various components in the graph and get relationship between them.

3.2.2 Indexing the Crawled Content

Crawler work is to explore all the relevant pages and forward them to indexer data structure for further processing. On getting the link, next step is to analyse the HTML content of the page. This is done by using a module known as `URLlib` [4]. This module provides a high level interface for fetching data across the world wide web. In particular, the ***URLopen()*** function is similar to the built in function ***open()***, but accepts URLs instead of filenames. It opens the link, checks for the content and return to the user the content of the page if found, else it returns the error code along with the message.

After confirming the validity of the link and extracting the content of the page in raw format, the next step is to format the page in such a manner that simplifies our further processing. The page will be formatted on the basis of tags via using another module of python named **BeautifulSoup** [5]. BeautifulSoup is a package mostly used for parsing HTML and XML documents. It creates a parse tree for parsed pages that can be used to extract data from HTML, which is useful for web scraping. It helps in extracting all the anchor tags and then do whatever operation is required which in turn is a part of crawler.

After the content of the page is in a formatted manner, we send it for indexing purpose. In this stage we use a dictionary data structure where the key is a keyword and the value in itself is a dictionary. The value here comprises of two parts i.e. key and a value. The key is the URL of the page in which the keyword is encountered and the value is the number of times the keyword appears in this link. At the time of the addition of a new keyword, there are a total of three cases that arises and different operations are done to deal with it.

- If the word is already in the index and URL is not there in its value then URL is added and the count of the word in the URL is initiated to 1.
- If the word is in index data structure and URL is also there in its value then the count of the word in the URL is incremented by 1.
- If the word is not there in the index, new key is generated and a dictionary corresponding to it is also formed. The dictionary now has the URL as the key and value 1

assigned to it.

After the indexer part is over, comes again the part of the crawler which now adds the link encountered in the page to the queue. While doing so it has to check various conditions mentioned below in order for it to work properly and not go into infinite recursion.

- The links that will be obtained can be of two types, *absolute URLs and relative URLs*. An **absolute URL** [6] contains all the information necessary to locate a resource. A **relative URL** locates a resource using an absolute URL as a starting point. In effect, the complete URL of the target is specified by concatenating the absolute and relative URLs. So it is to be taken care that relative URL is properly translated to absolute URL and then only gets saved in the queue.
- After we have the URL we need to check whether the link is already traversed or not. For that we have to check the list which contains all the URLs of the pages that are already traversed. If the URL is there in the list, nothing will be done. If the URL is not in the queue it will be added to the end of the list.
- While adding data to the list we also have to take care whether the link is relevant to our study or not. In order to do so we will be checking for the presence of several keywords in the link namely docs, pdf, jpg and so on. If these things are found to be present in the link, the link will be further not taken into consideration.

This section talks about how indexer works and how the HTML of the page is decomposed and stored in the dictionary. There are various ways to organise the large amount of data this search engine will be dealing with and making it arrange in a manner that optimizes the insertion time as well as lookup time is the main objective. As search engine primary goal is to retrieve results in shortest amount of time, the lookup operation time needs to be as optimized as possible. Next subsection will talk about the various aspects of the dictionary data structure along with its functionalities.

3.2.3 Optimization of Lookup Operation

One of the major thing to consider out here is to analyse the importance of the cost of lookup operation. The main part from the point of view of user is getting the search results fast. In order to do so we need to decide which data struture can serve our purpose and the most important determining factor is the lookup time i.e. how fast a data can be retrieved from the data structure. After researching on various data structures namely stack, queue, tree, balanced binary search trees, dictionary, heaps, tries and various others, dictionary was taken as the data structure to act for the same. Due to the way this data structure works i.e. probing, a constant lookup time operation is carried out. When a user enters a query, the decomposed keyword gets looked up in the indexer data structure in constant amount of time and results are thereafter returned from it. The amortized lookup run time is $O(1)$ which is constant. This is of prime importance as its the real time query that makes the search engine useful.

Adding to it, the recomputation of the indexer is avoided by using a module named pickle [7]. This module allows user to save any data structure computed in the script to a file on the disk. This allows to reuse the file again and again, hence saving a lot of time in recomputation of the same thing as the pages of the faculty will be undergoing almost no change in the span of days or may be months. In order to reflect the changes, a spider will be run in the background which will be crawling the web and changing the content of the file if there is any change encountered in the websites of the concerned faculties.

The spider is nothing but a bash script that gets run after a definite period of time, in our case it being 20 days. The work of bash script is to execute the crawler which in turn calls the indexer script. The changes encountered via this in the faculty websites are thereafter updated in the pickle file mentioned above. This helps to completely automate the working of search engine without any user interference in the entire process.

3.3 Ranking mechanism

In the above section we looked at how data is organized in the dictionary data structure. So once the user searches for some specialization a lookup operation is done in this database and all the relevant links along with the combined count is returned. The returned structure is also a dictionary where key is the URL and value is the count (frequency of occurrence in URL) based on which final ordering will be based. This dictionary will be sorted based on the count and finally the modified and sorted list of URLs will be displayed to the user. To make the concept more clear we will demonstrate the same via an example. But before diving into the example we will look into the concept known as inverted index.

The pages that we are considering doesn't form a network that can use the conventional ranking mechanisms namely page rank and hubs and authorities and all. In order to rank the web pages we have in our record, we will use the concept of inverted index. An inverted index [8] is a data structure that we build while parsing the documents covered under the domain of the crawler. Our search queries will be based on these documents only. For every word encountered in the document, it maintains a list of URLs where the word is found. Given a query, this inverted index data structure will return the list of documents relevant for the query. Each term is a key in the index whose value will be the list of documents also known as the posting list which signifies the list of documents where the term appears in. To illustrate with an example, if we have the following documents:

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]. In addition to it, we will also be keeping a record of the frequency of the word in the document and based on this we

will rank our pages. The amount of information we keep in our index will grow as we add more functionality to our search engine. Once we know the links that the user is interested in, we only have to sort the results based on the ranks attached to the links. We will then return the sorted order of links obtained.

3.4 Final retrieved results

As discussed earlier about the GUI, now we will be talking about the way final results are displayed to the user. In the previous section we talked about the ordered list of links that we will be displaying to the user. Now for the case of project, the list we have is divided into two sets. In the first set we have all the faculty members from the six institutes that we have crawled. These web pages are the official pages hosted on the institutes main server. The second set will consist of web pages of departments the specialization is related to. In addition to this, it will also contain all the relevant web pages that were crawled starting from the seed page and have institute domain name present in the URL. Figure 3.6 shows the list of URLs of faculty members and Figure 3.7 shows other relevant URLs for a given specialization query.

3.4.1 Issues to display content

The thing we are doing here is dynamically fetching content from the file on the disk as discussed earlier. Once we have the relevant content we have to sort the links on the basis of count of keywords. This is the dynamic part we will be dealing with. This part is done once we have the query from the user. So a script needs to be run which will be fetching all the relevant links from the database file and then will sort it on the basis of count. Now, the issue is, how to combine the working of the script with the web page which will be displayed to the user?

In order to solve this problem a method known as common gateway interface (CGI) is used. Basically when implemented on a web server, it provides an interface between the web server and the script that will generate the dynamic content. These programs are known as CGI scripts and 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-rajb-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

Fig. 3.6: List showing faculty URLs

Show	10	entries	Search:	<input type="text"/>
S.No.	Other Relevant Web Pages			
11	http://www.iitp.ac.in/index.php/schools-and-centers/engineering/mechanical-engineering/people/faculty/dr-probir-saha.html			
12	http://www.iitp.ac.in/index.php/schools-and-centers/engineering/mechanical-engineering/people/faculty/dr-sudhanshu-sekhar-panda.html			
13	http://www.iitp.ac.in/index.php/schools-and-centers/engineering/mechanical-engineering/people/faculty/dr-mohd-kaleem-khan.html			
14	http://www.iitp.ac.in/index.php/schools-and-centers/engineering/electrical-engineering/people/faculty/dr-ranjan-kumar-behera.html			
15	http://www.iitgn.ac.in/faculty.htm			
16	http://www.iitp.ac.in/index.php/schools-and-centers/school-of-basic-sciences/mathematics/people/faculty/414-dr-debashree-guha-adhya.html			
17	http://www.iitp.ac.in/index.php/schools-and-centers/engineering/computer-science-a-engineering/people/faculty/dr-ashok-singh-sairam.html			
18	http://www.iitp.ac.in/index.php/schools-and-centers/engineering/mechanical-engineering/people/faculty/dr-somnath-sarangi.html			
19	http://faculty.iitmandi.ac.in/~arnav			
20	http://www.iitp.ac.in/index.php/schools-and-centers/engineering/mechanical-engineering/people/faculty/dr-manabendra-pathak.html			
Showing 11 to 20 of 77 entries		Previous	1	2
			3	4
			5	...
			8	Next

Fig. 3.7: List showing other relevant URLs

Chapter 4

Exerimental Studies and Results

One of the studies that was done during the course of the project was to analyse which data structure to use in order to optimize the insertion time and at the same time optimise the lookup time. In order to do so there were various data structures that we took into consideration and finally opted for dictionary data structure which stores data in the format of key value pair. The first half of this section we will be talking about dictionary data struture and will be comparing its run time and insertion time with other data strutures. We will be using list for our comparison purpose. But before that lets look at how is dictionary implemented and what are its main features. Given below are some of the highlights of the same.

- Python dictionaries are implemented as **hash tables**.
- Hash tables must allow for hash collisions i.e. even if two keys have same hash value, the implementation of the table must have a strategy to insert and retrieve the key and value pairs unambiguously.
- Dictionary uses open addressing to resolve hash collisions. In this method hash collision is resolved by **probing** i.e. searching through alternate locations in the array (the probe sequence) until either the target record is found, or an unused array slot is found, which indicates that there is no such key in the table.
- Python hash table is just a contiguous block of memory (sort of like an array, so you can do $O(1)$ lookup by index.

- Each slot in the table can store one and only one entry.
- Probing just means it searches slot by slot to find an empty slot. Technically we could just go one by one, $i+1$, $i+2$, and so on and use the first available one (that's linear probing). But for some reasons, CPython uses random probing. In random probing, the next slot is picked in a pseudo random order. The entry is added to the first empty slot. The main point to take from here is that the slots are probed until first empty slot is found.
- While adding entries to the table following things happens :
 - When adding entries to the table, we start with some slot, i that is based on the hash of the key. CPython uses initial $i = \text{hash}(\text{key})$ and mask where $\text{mask} = \text{PyDictMINSIZE} - 1$. Just note that the initial slot, i , that is checked depends on the hash of the key.
 - If that slot is empty, the entry (hash , key , value) is added to the slot.
 - If the slot is occupied, CPython (and even PyPy) compares(==) the hash AND the key of the entry in the slot against the key of the current entry to be inserted. If both match, then it thinks the entry already exists, gives up and moves on to the next entry to be inserted. If either hash or the key don't match, it starts probing.
- The same thing happens for lookups, just starts with the initial slot i (where i depends on the hash of the key). If the hash and the key both don't match the entry in the slot, it starts probing, until it finds a slot with a match. If all slots are exhausted, it reports a fail.

In the above part we saw how the dict is implemented and how it is able to give a constant lookup time. Now on top of this we will compare the results we obtained when we use list and when we use dictionary for the index implementation.

Lookups in lists are $O(n)$ whereas lookups in dictionaries are amortized $O(1)$ where n is the total number of items present in the data structure. Dictionaries in Python are hashable. This means that a hash table is calculated whenever you create or modify a dictionary. Your collection is essentially split up into buckets of equal size, each with a unique identifier,

the hash. The size of buckets remains constant no matter how large your dictionary is, you simply have more buckets to deal with. The time when you ask Python x in S , it will calculate the hash of x and access that bucket directly. It then searches through the bucket as it would through a list. Hence, this search time is independent of the size of your dictionary. For the experiment purpose list and dictionaries of variable size were created and searched through noting down the time it took for it. For each of the size, the search time was averaged over 100 iterations. Given below are the two figures that demonstrates the same. In Figure 4.1 the number of elements that are considered are small whereas in Figure 4.2 the sample size is quite significant which is what we deal with for our search engine.

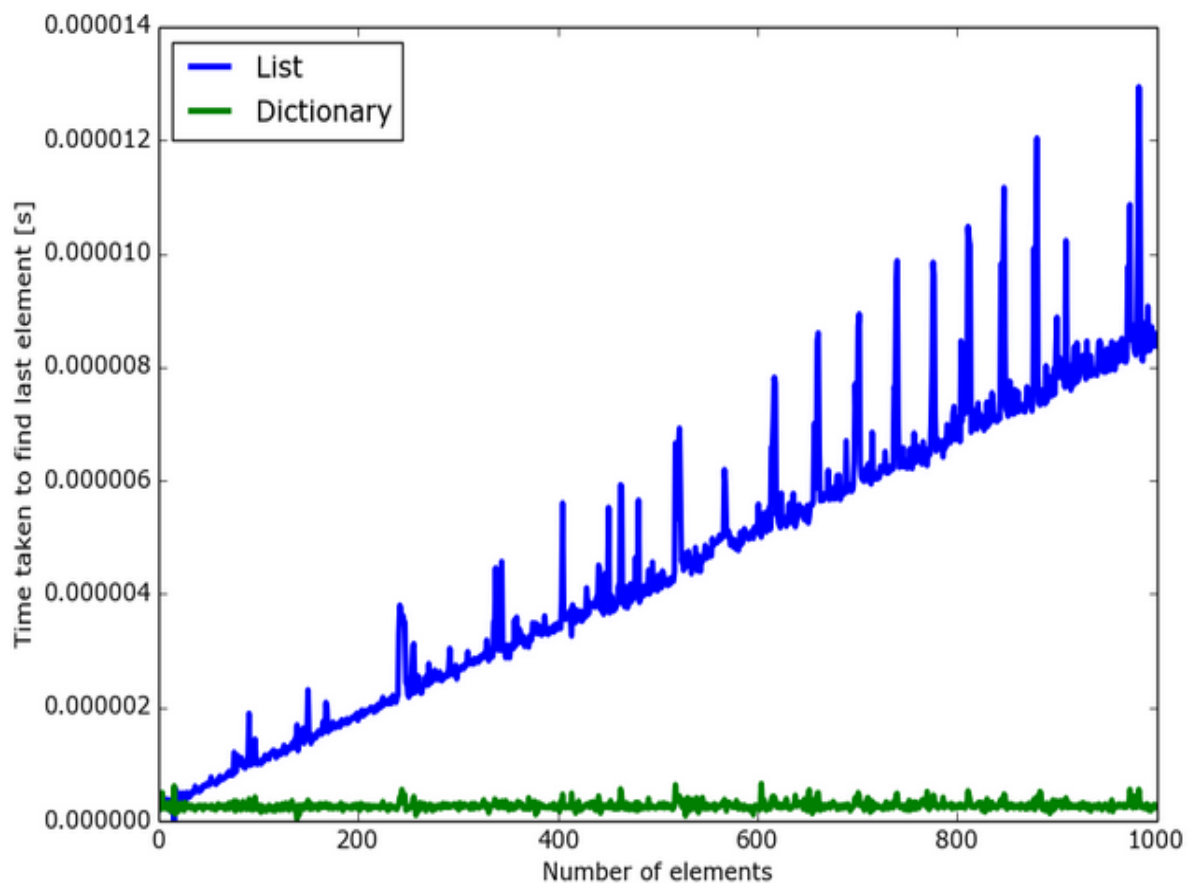


Fig. 4.1: Look up time for small set

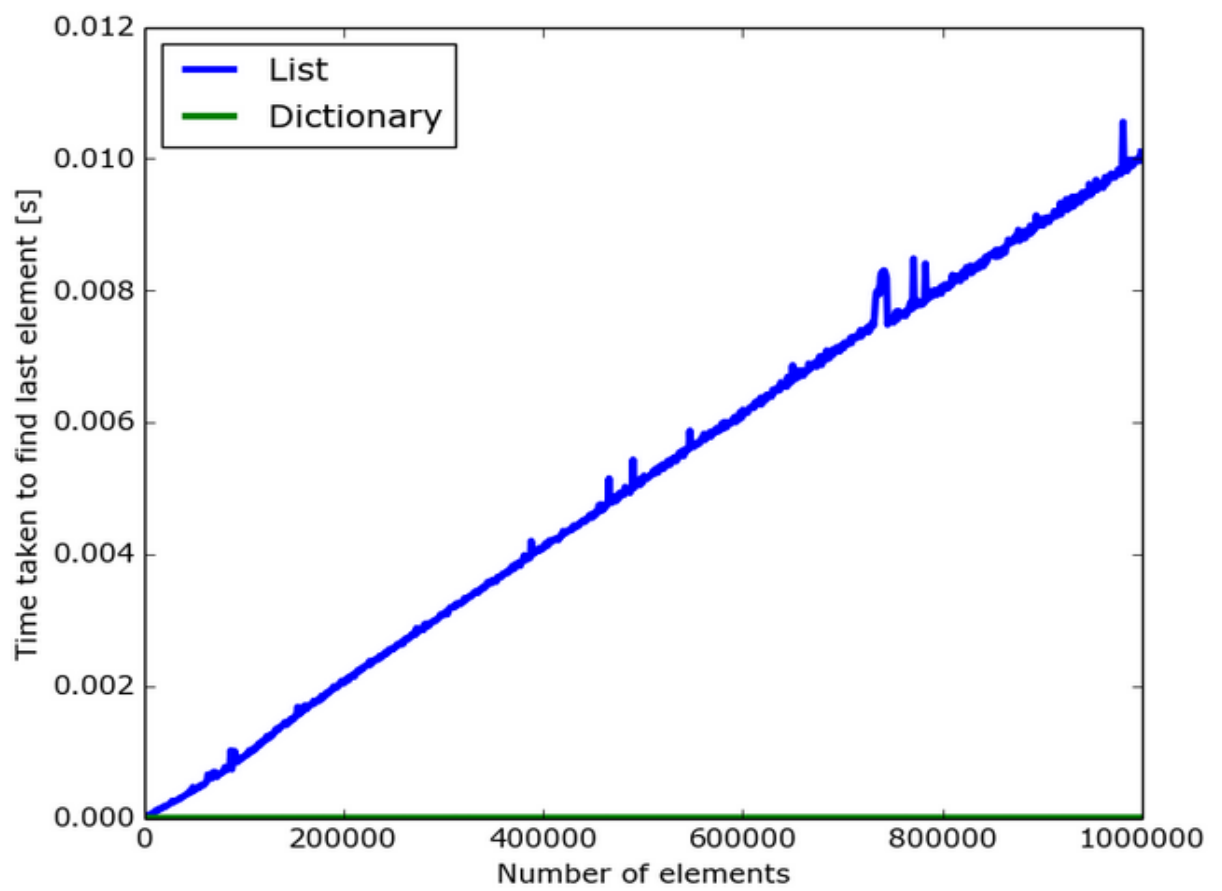


Fig. 4.2: Look up time for large set

Chapter 5

Conclusion and Future Work

Till now we have looked at the working of prototype of a search engine in which on entering specialization results will be obtained from the limited number of premium institutes that are covered in the domain of the crawler. Being a prototype it lacks few of the things which once taken into consideration, it can be used at a much larger scale and even for commercial purpose as well. Its basically just like any other search engine but with limited amount of crawled content. A proper interface is designed which will be responding to user's query. The query that it supports is specialization of some sort. For example, machine learning, pattern recognition, computer vision, image processing, speech technology are some of the search queries it supports. Searching for these will return the list of the urls of concerned faculties from the limited number of institutes we have in our crawled database system. On top of this, still there are many things that can be worked on in order to make it useful for the masses. Being just a prototype there are many features it will be missing. Mentioned below are some of the points that could be worked on in near future in order to enhance its functionalities.

- **Ranking Techniques** : Currently we are using modified inverted index for the purpose of ranking. It takes into account the frequency of the words and then sorts the urls on the basis of these ranks. This is something which is useful when used on a small scale. Extension to ranking techniques can be made once we include more number of institutes which in turns increases the crawled content. This increase will in turn increase the connectivity between various pages and this is something we will be

taking advantage of. The algorithm that utilizes the property of inlink and outlink from a page is known as page rank algorithm. We can incorporate this into our ranking script to further enhance the way the results get displayed to the user.

- **Extending the Domain** : As already mentioned in the system anatomy section, we are covering only a limited number of premium institutes i.e. a total of 6 IIT's with each of the faculty member and for the rest of IIT's we are covering only the department pages. Target audience for this prototype will be a small number. Once this is found to be fault proof we can add more number of institutes. With more number of institutes comes the requirement of greater computation power of the crawler and more space for the indexer.

In addition to this results can also include other related materials namely pdf, docs, re-search materials and other important stuff the professors are associated with. This can act as an extension where not only the concerned faculty members will be presented in the final result list but also the related work will be shown in the final result list.

- **GUI Enhancement** : This part deals with enhancing the existing application setup and make it more feature equipped and more user friendly. Many more features that enhance the user usability could be taken into consideration and hence can be explored further.

The same has been uploaded on the server and can be accessed by anyone by following the link http://sahilmutneja.com/cgi-bin/search_engine.py. Users can check its working by entering their query in the text box provided. The results will be displayed in the section below the box showing it in two parts, first one will show the url of the faculty members and the second one will show the url of the departments that are concerned with the specialization. The user can comment on the working and give suggestion for the improvement on the following link <http://sahilmutneja.com/contact.php>.

References

- [1] From Wikipedia, the free encyclopedia, “Web search engine.”
- [2] Elizabeth Liddy, Director of the Center for Natural Language Processing Professor, School of Information Studies, Syracuse University , “How a Search Engine Works,” May 2001.
- [3] Sergey Brin and Lawrence Page, “The Anatomy of a Large-Scale Hypertextual Web Search Engine.”
- [4] Python, Documentation, The Python Standard Library, Internet Protocols and Support , “urllib Open arbitrary resources by URL.”
- [5] Sergey Brin and Lawrence Page, Computer Science Department, Stanford University, Stanford, CA 94305, “Beautiful Soup Documentation.”
- [6] Web Developers Notes, “Relative and absolute urls.”
- [7] Documentation, The Python Standard Library, Data Persistence, “pickle Python object serialization.”
- [8] Elasticsearch: The Definitive Guide, Getting Started, Mapping and Analysis, Inverted Index, “Inverted index.”

Curriculum Vitae

Name: Sahil Mutneja

Date of birth: 26 August, 1992

Education qualifications:

- [2011 - 2015] Bachelor of Technology (B.Tech.),
Computer Science and Engineering,
IIT Mandi, Himachal Pradesh, India.

Permanent address:

House Number 2002/2, Sector 47-C,
Chandigarh, Punjab, India, 160047.
Ph: +918894404479