Course Project Report CS101 Project

Google @ IITB

TEAM ID:454

Akash Doshi, 140010008
Ch.Jyothi Durga Prasad, 140010042
Aditya Bhosale, 140010009
Mainak Majumdar, 14D110022

Table Of Contents

1. **Introduction**:

Nowadays when most people are asked a question, they never bother to rack their brains for an answer, or go to a nearby library. They just “Google” it. Searching on Google, browsing through its gigantic number of search results, so conveniently arranged such that the result you require is generally on the first page, is so regular that raraely do we stop to pause and think on how exactly does this search engine function?

Our project delves deep into the intricate machinery behind this amazingly efficient search engine, incorporating the now famous “page rank” algorithm in its most primitive form as given in none other than Larry Page and Sergey Brin’s PHD Thesis at Stanford University. We store a large number of websites in a hash table: which is one of the precursors to the “Big Data” revolution that has now swept the computing world, also make use of an enormous Auto Spell Checker, finally displaying the results to the world in a GUI very similar to Windows 2000, written entirely in C++!!

2. Problem Statement:

- We are designing a search engine to closely model Google and its page rank algorithm.
- Using a web crawler we are obtaining all the hypertext references and their titles from the IITB main webpage, convert it into a text file and read that text file in a c++ program.
- The keywords and their “n-grams”, obtained from modification of the url’s, will be stored in the form of an array of linked lists - “hash table”.
- When the user enters a search ,the search will be n-grammed and each n-gram’s hash value will be extracted and the word will be searched for in the corresponding index of the array.
- On success the user will be shown a list of urls stored under the keyword in decreasing order of authoritativeness. In case the keyword is not found, the user will be redirected to a google search which jQueries the string entered by the user concatenated with ‘iitb’.

3. Requirements:

Software Requirement :

Windows Operating System: Windows 7 or later.

windows.h library : For the ShellExecute function and running graphics associated with the application.

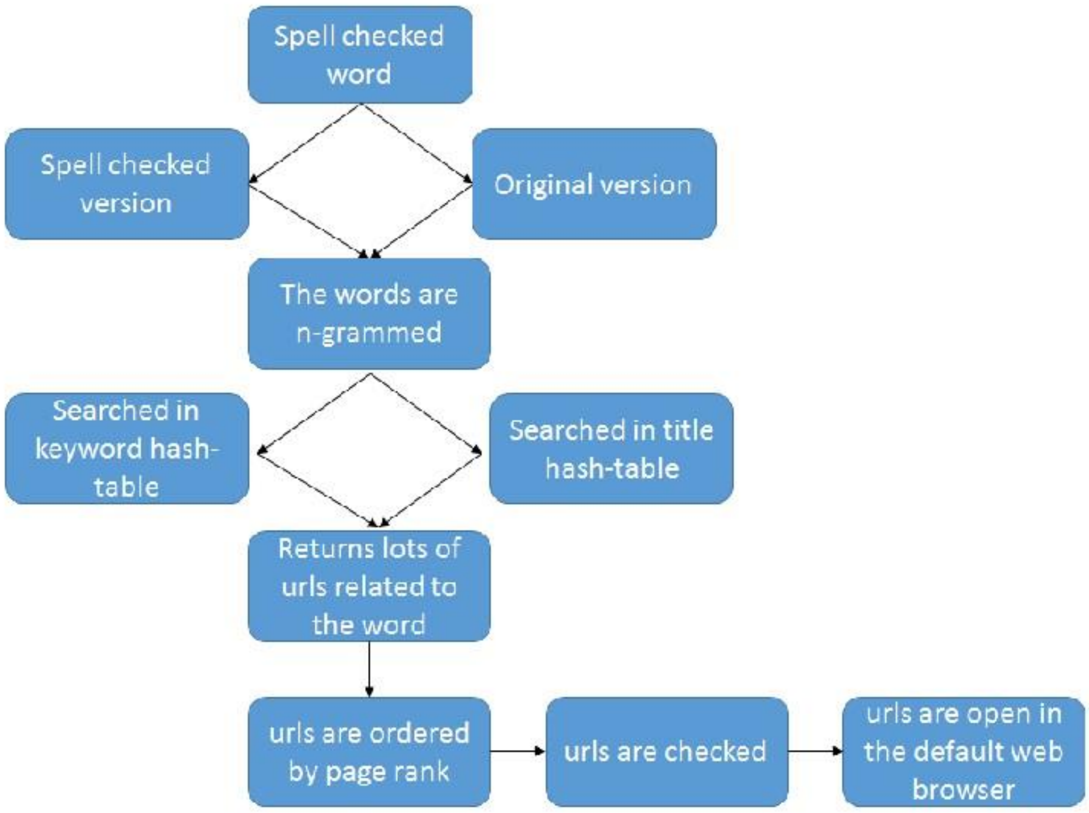
Minimum screen resolution required: 1080 * 720 pixel.

4. Implementation:

Taking input from user: The user in provided with a runtime text input field which stores the input in a string variable when the user clicks the search button.

Spell Checker:

A large dictionary stored in the form of a text file is read into a hash table.When the user enteres a search, each letter, starting fro the last, is successively replaced by all 25 characters and the word obtained checked for existence in the dictionary. If found the word is returned. If the original spelling was correct, itself is returned else an empty string is returned to the calling function.


Spell checked word
Spell checked version
Original version
The words are n-grammed
Searched in keyword hash-table
Searched in title hash-table
Returns lots of urls related to the word
urls are ordered by page rank
urls are checked
urls are open in the default web browser

Web crawler:

1. Using urllib library in python, we can work with webpages and their source code so as to extract all the links on the page.

2. We loop the program to find the list of all links on the pages whose urls we extracted in the first step.

3. Then we check if the sites belong to the IITB domain by checking if they have the 'iitb.ac.in' string in them

4. Later we extract the titles of these pages to make a list of keywords for that page.

Hashing and Searching:

1. Using the textfile produced by the web crawler described above, read the URLs of the textfile into an arr, and the titles of the webpages into a title array

2. Since the list is of websites in IITB domain, we first scan each URL for the string http://www.iitb.ac.in or http://www1.iitb.ac.in, remove it and store the rest of the string in a hash table. The titles do not need any modifications and are straight away pushed into a separate hash table.

3. We do this by computing the ASCII value of each alphabet in the string, summing it up and finding its modulo with the size of the hash table i.e. arr. This assigns each string a row in which it will be stored in one of the nodes of the linked list.

4. Now we n-gram each string. This involves breaking up a string into its substrings from length three onwards. We feed each n-gram of both the keyword and its title into the hash table, every time checking if the word is already present. If so, only the integer corresponding to the array index of the url is pushed under the corresponding node, else a new node is created as described forthwith.

4. If the linked list is empty, we store the string in the first node, else we traverse the extent of the linked list and store it there.

5. For IITB websites which do not fall into category 2 we remove common words like http://, www. etc. and store them using rules of category 3&4.

6. User now enters the search into a console based application which accepts the input from the user in a graphic window.

7. This entered sting, after undergoing a spell check(described above) is now subjected to an extensive search:

i)First the search is split up into words, following which each word is then n-grammed. Each of these entities the undergoes the following.

i) It is converted to lower case, and its hash value computed. It is then searched for in the corresponding linked list by traversal and
compared. If found successfully, the website corresponding to it is opened in the web browser.

ii) If above fails, the entered search is compared with the titles of the HTML page, along with its n-grams, extracted by the web crawler using the same principles discussed above.

iii)At the same time, everytime a keyword is found, the array indices of all the url's under the node are pushed onto a vector. When search is completed , these are all displayed as buttons on the graphic screen in order of their page rank.

iv) If all above fail, the user will be redirected to a google search which jQueries the string entered by the user concatenated with 'iitb'.

Implementation of a User Friendly Graphics Interface:

- We have created buttons , titles and text box fields , to provide Previous and Next Page functionality, accept the search, input of the spell checker and hypertext references to the websites . One can also clear his search and write the new search without having to close the application.
- For the above, we have made use mainly of the fact that in win32 GUI, every form of input , be it from the keyboard or the mouse results in a message being sent to the CALLBACK function, and by appropriate define statements in the pre-processor , we can adjust these values to check for them correctly when received
- Every time a next or previous page command is encountered, or the user enters a new search, all the previous buttons are deleted, however the dictionary, hash tables and page ranks are not deleted from the memory, making the search superfast.

5. Testing Strategy and Data:

The following kind of data are treated as described below by our search engine:

- All string of length greater than two will be able to undergo the entire search algorithm.
- If any digits are entered, they are ignored both in the spell checker and the spell corrected word but are later concatenated back into the search.
- If any other non alpha numeric character is present, it is simply ignored
- In case a string does not exist in the dictionary or the domain of our database, user is redirected to Google.

6. Discussion of System:

A) What all worked as per plan?

- Were able to implement page rank algorithm in Python
- Were able to store large amount of data in hash table and successfully scan via basic algorithms of traversal, insertion and deletion in a linked list.
- Were able to make search case insensitive as promised.
- Were able to implement a user friendly graphic interface in Win32 GUI as discussed in SRS.
- Were able to provide a default JQuery to Google in case of nonexistence in IITB Domain.

B) What we added more than discussed in SRS?

- An extensive SpellChecker was implemented in C++.A large database of 1,43,000 was stored in a hash table and each word entered was checked for the closest word in the Dictionary as explained before.

- The search has been made highly tolerant to numbers and non-alpha numeric characters, returning desired results as explained before.
- A highly sophisticated graphic system was developed which also had Previous and Next Page buttons, Number of results found and page number displayed, by making appropriate use of the SendMessage() function in the CALLBACK method.

(C) Challenges faced and how they were overcome:

- Google page rank algorithm ranks pages depending on how many pages give link to that page and also takes into consideration the authoritativeness of the pages that give the link.
- However, there's a problem of dangling nodes (also called sink) ie. Pages which have only incoming links without any going out. This problem is solved by transforming the original matrix into what is called the Markov matrix which replaces the columns whose all entries are zero by 1/n (n is the number of pages in the database)
- Another problem is that of disconnected graph, meaning an isolated set of links in the database. In this case, the eigenvalue equation gives ambiguous result (more than one eigenvector for one eigenvalue). To solve this problem, we take into consideration a damping factor (0.85 in our case)
  $G = (1-d)M + dS$ G is called the Google matrix
  M is the Markov matrix
  d is the damping factor
  n is the total number of pages in the database
  S is an nxn matrix with all entries = 1/n
- While designing the spell checker, we faced a problem in handling how to incorporate the change in word to the correct spelling. This is because every time a button is clicked a message is sent. In the message, the "msg" part redirects it to the WM_COMMAND case following which the "wParam" parameter determines the case inside the switch (wParam) to be executed. If the action based on a button is inside the case, it would never be reached. Hence we wrote a default SendMessage() function in case of no change and utilized a fall-through in case command within a switch to appropriately handle the message.
- Figuring out how to most effectively store the websites to be displayed , compiled from the search results of

various substrings was difficult. For this we expanded the struct definition to also include a vector of integers, which were the array indices of the website. When this substring was found, we pushed all its array indices onto a globally declared vector, continuously ensuring that the index was not already present.When the last substring from the search was extracted, we would send a message to the function to sort the global vector in order of page rank and offload it onto the graphics interface.

7. Future Work:

- Currently on searching for “computer science” no CSE website is returned as neither the website nor the title contain computer rather they contain “CSE”. If we were also able to hash the text displayed on the webpage this problem could be solved, but implementation of “Big Data” structures on this scale is not feasible.
- The implemented page rank algorithm depends only on the number of hypertext references on a page. By obtaining other data, like number of people having visited a page, standard of web designing incorporated in the page, a more powerful page rank algorithm can be obtained akin to the ones used today by popular search engines.
- As soon as the user enters about three alphabets, a drop down menu can be displayed (using the CBS_DROPDOWNLIST function in Win32) to show him the possible search words. This can be done as our hash table also stores the n-grams of every word and hence the word itself can be traced.
- To provide for reusability of this code, an algorithm has been developed to provide the contents of the websites and the dictionary in a read-only mode to a future user. This user will be provided with an authorization code for a Dropbox account which contains these files. On entering the code, the program will search that Dropbox account for the requisite files, and read them into the program, without showing the user it’s contents.

8. Conclusions:

Our project has the unique feature of being always capable of improvement and future innovation. Indeed, even Google is always getting better. A few years back Google introduced the feature that on typing only part of the search all the websites corresponding to the most probable completion are displayed. Many other such small intricacies can all be easily implemented in our code.

9. References:

Web crawler documentation:
http://www.crummy.com/software/BeautifulSoup/bs4/doc/
https://docs.python.org/2/library/urllib.html
https://docs.python.org/2/library/urlparse.html
Opening an executable from C++ code:
https://msdn.microsoft.com/en-us/library/windows/desktop/bb762153%28v=vs.85%29.aspx
Graphics interface using Win32 GUI:
http://sourceforge.net/projects/perl-win32-gui/
http://www.winprog.org/tutorial/start.html
https://msdn.microsoft.com/en-us/library/windows/desktop/ms632679%28v=vs.85%29.aspx
An Introduction to Programming through C++ by Abhiram G Ranade
www.edx.org **: Harvards CS50 Course Introduction to Computer Science**

Github Link to project: https://github.com/akashsdoshi/Google-IITB

YouTube Link for demonstration of running project: https://youtu.be/zgbtK53-cOQ