LookItUp!!

Team Members
Yaswanth Orru - 160050066
Shantanu Kumar - 160050069
Lakshmi Narayana - 160050080
Sai Kalyan S - 160050095

Abstract

We have implemented a basic search engine which will search ittb domain for given words and give the top ten pages in the domain which match with the search query the most, where the matching factor is based on factors like page rank of the page, tf*idf value of the (word,page) pair and also, preference has been given for pages with search tokens appearing in the title of the page.

Project mainly contains five parts
1. Web Crawling
2. Page Ranking
3. Indexing generating
4. Query processing
5. Ranking the pages having the search tokens based on above calculated values
Crawling

We have used Crawler4J for web crawling. We will parallely run multiple threads of crawlers each starts with a seed url recursively crawls urls present in that page and simultaneously update two key-value paired relations one for storing the child urls for a particular url i.e the HyperLink Graph (Document ID(key) → Document id of the child (value)) for every child of the key which is stored as a file in local storage, another for storing page rank for a url (Document ID(key) → Url, No.of clicks) stored in mongoDB. This Document Id will be used
for indexing and will be unique for each URL. Apache Tikka used to parse binary files like pdf, ppt, txt, doc etc.

**Indexing**

We will use inverse indexing for indexing the words we get after crawling. Apache Spark for parsing the document obtained from crawling threads into words which involves removing stopping words list(a, an, the), stemming the tokens, lowercasing all the tokens. Our key value pair for storing indices is

\[
\text{Stemmed Word(key)} \rightarrow \left[ \left[ \text{DocumentID}, \left[ \text{positions of all occurrences of that word}, \text{no. of occurrences in title, no. of occurrences in total} \right], \ldots \right] \right].
\]

Example :-

Document 1: Title1(Title) web retrieval web search information(body)
Document 2: search engine web ranking(Title) body1(body)

Indexing structure assuming only above two documents, the entry for web in our indices database will be

\[
\{ \text{‘key’ : ‘web’, ‘index’ : [ [1, [0, 2], 0, 2], [2, [2], 1, 1] ] } \right].
\]

→ We have used MongoDB for storing all the key value pairs.
Web-Page Ranking

Page Ranking

Uses HyperLink Graph and Calculates Rank Of Page.

Algorithm:

1. Initialize Rank of each page to 1.0
2. On each iteration of Page “x”, it sends a contribution of \( \text{rank}(x) \div \text{num neighbors}(x) \) to neighbours.
3. Set each pages rank to \( 0.15 + 0.85 \times \text{contributions Received} \).

Iterate steps 2 and 3 multiple time till rank of each page converges

Intuition:

Think of a web surfer who can jump to any children links with probability \( s \). And jump to any random page(including children) with probability 1-s. And assume his initial probability vector \( \mathbf{v} \) (whose each element denotes the probability of being in a page). And \( O(j) \) = number of children for a page \( j \).

Let \( G \) be a NxN matrix where \( G(i,j) = 1/O(j) \) if \( i \) is a child of \( j \), else 0.
And \( R \) also be a NxN matrix where \( R(i,j) = 1/N \) for all \( i,j \)

Then, probability vector after one jump would be, \( (sG + (1-s)R)\mathbf{v} \).

So, After \( n \) jumps its, \( (sG + (1-s)R)^n\mathbf{v} \). Let \( (sG + (1-s)R) \) be \( A \). Then, as \( n \) tends to infinity, \( A^n\mathbf{v} = A\mathbf{v} \). And \( A\mathbf{v} \) is the limiting prob. Vector or pagerank vector.

So, \( PA = A \). Hence, \( P \) is the eigenvector of \( A \) with eigenvalue 1.

TFIDF:

- It represents relative importance of a word in a document, It’s product of two statistics TF(term frequency) and IDF (Inverse Document Frequency).
- \( tf(t,d) \) is no. of times term “t” occurs in document “d”
- \( idf(t,D) = \log(N \div (1 + |d \in D : t \in d|)) \) where \( D \) is total Documents
- \( tfidf(t,d,D) = tf(t,d) \times idf(t,D) \)
- It is calculated during Indexing process and stored in “Mongo DB”.

-
Scoring of Documents is explained in Querying.

- As we know the documents that are related to a certain search string, web-page ranking is to rank the pages in relevance order.
- One way to do so is based on the no. of links that are mentioned in a link. The figure gives an idea of how we are going to implement this.

Source: Google Page Rank Algorithm

- Apart from these, We will use metrics like, location of queried string(Title/body), Whether the whole query string is present in the document or part of it, No.of Clicks, History of page, font size of the search text in the document etc.,(All these can be extracted from the key to index mapping obtained earlier).
- We would like to tune the weightages to each of the coefficients derived from each algorithm and arrive at final weightages that give the best results.
Querying

Steps:
1. Read input string “s”
2. Stem “s” and remove stopwords.
3. Split “s” by whitespaces to obtain terms $T = t_1, ..., t_k$
4. Find 10 pages with highest value of $score(T, d, D) = PageRank(d) \cdot \sum_{t \in T} tfidf(t, d, D)$
5. Also add the weightage of the word being present in the title of the document than in the body.
6. Print Result along with text matched in Document and repeat Step 1.

Testing

- We will test ranking algorithms individually, how inserting a page changes the values of page rank and tf*idf techniques.
- Finally we will check the result of our ordering of web pages with the expected output.

Softwares Used

1. Crawler 4J (for crawling)
2. Apache Spark (for MapReduce)
3. Apache Tikka (for parsing Binary files)
4. Apache Lucene (for stemming and removing stopwords)
5. Google Suggest API for providing suggestions while typing
6. Apache Tomcat for backend server.

Collections in MongoDB

1. urlsdb - |ObjectId|UrlName|No.Of.Clicks|
2. Hyperlink_graph - |ObjectId|Parent|Child|
3. spark - |ObjectId|key|index|
4. pageranks - |ObjectId|docID|rank|
5. Tfidf - |ObjectId|word|docID|tfidf|