WAH SEARCH

https://github.com/harshsodi/WahSearch

Instructor: Dr. Luis Rueda

Memebers: Harsh Sodiwala - 105186258

Jiiqng Zhu - 105092951

Weiwei Cao - 105171487

MAIN FEATURES

- Searching
- Auto-complete
- Spelling checking

OVERVIEW OF DESIGN

FRONT-END DETAILS

Javascrpit/html/css(Vue.js, MVVM)

/suggestion?keywords=computer

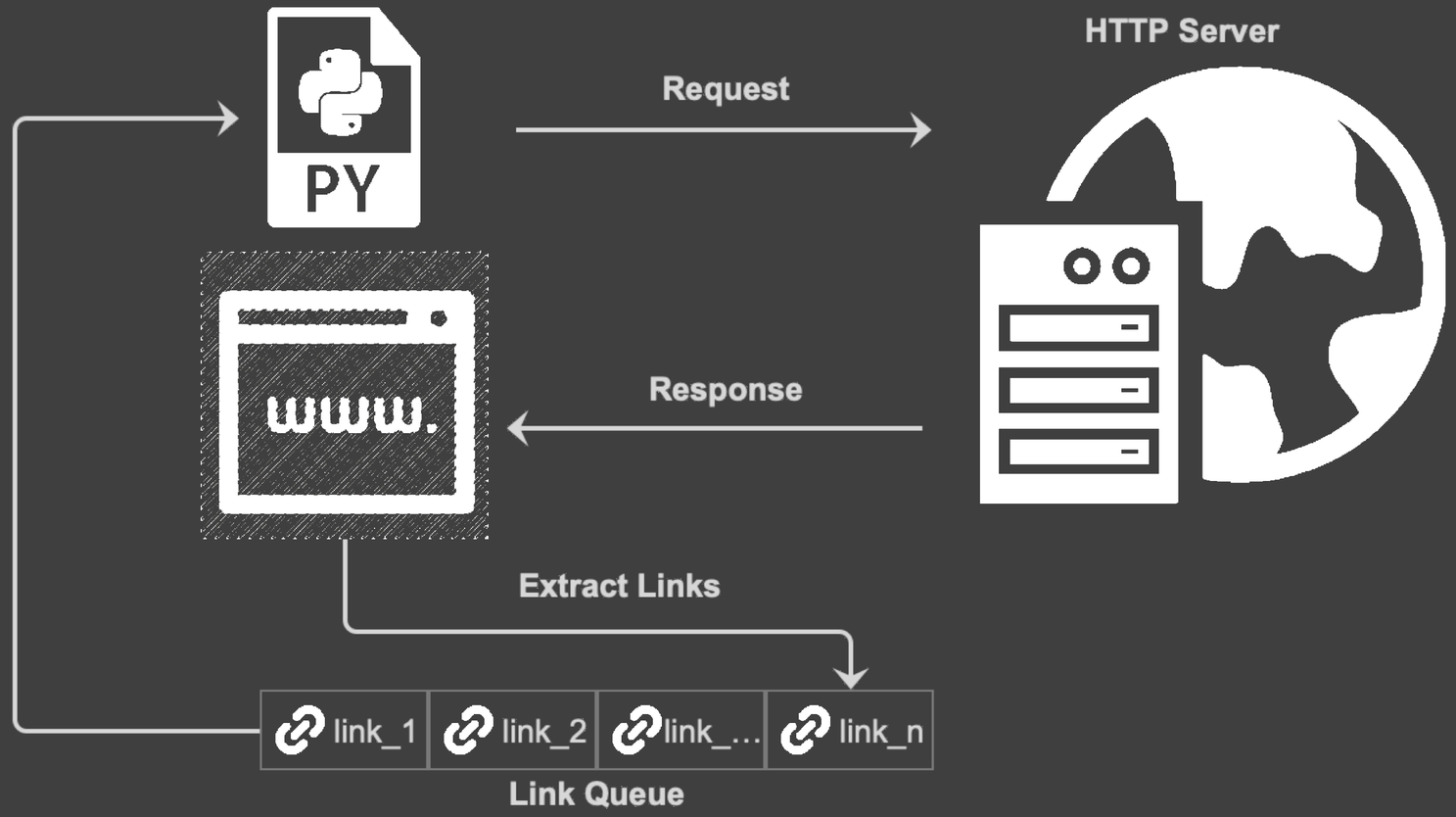
/search?keywords=computer&page_number=1&per_page=20

BACK-END DETAILS

DATA COLLECTION

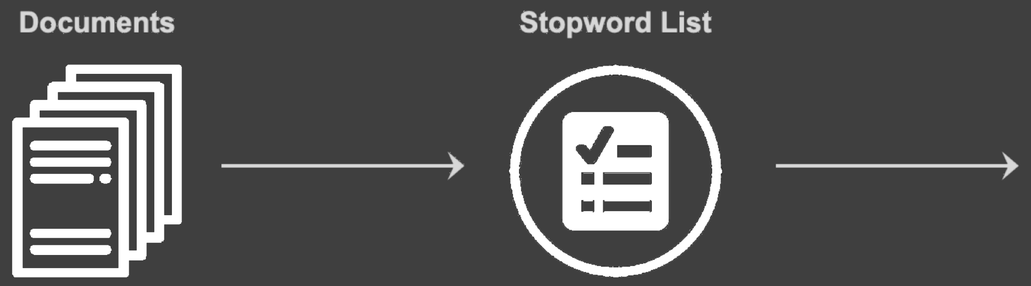
- The spider that will crawl the webpages using **Breadth-First-Search** strategy.

- For each webpage, a **Parser** will extract links from html tag.

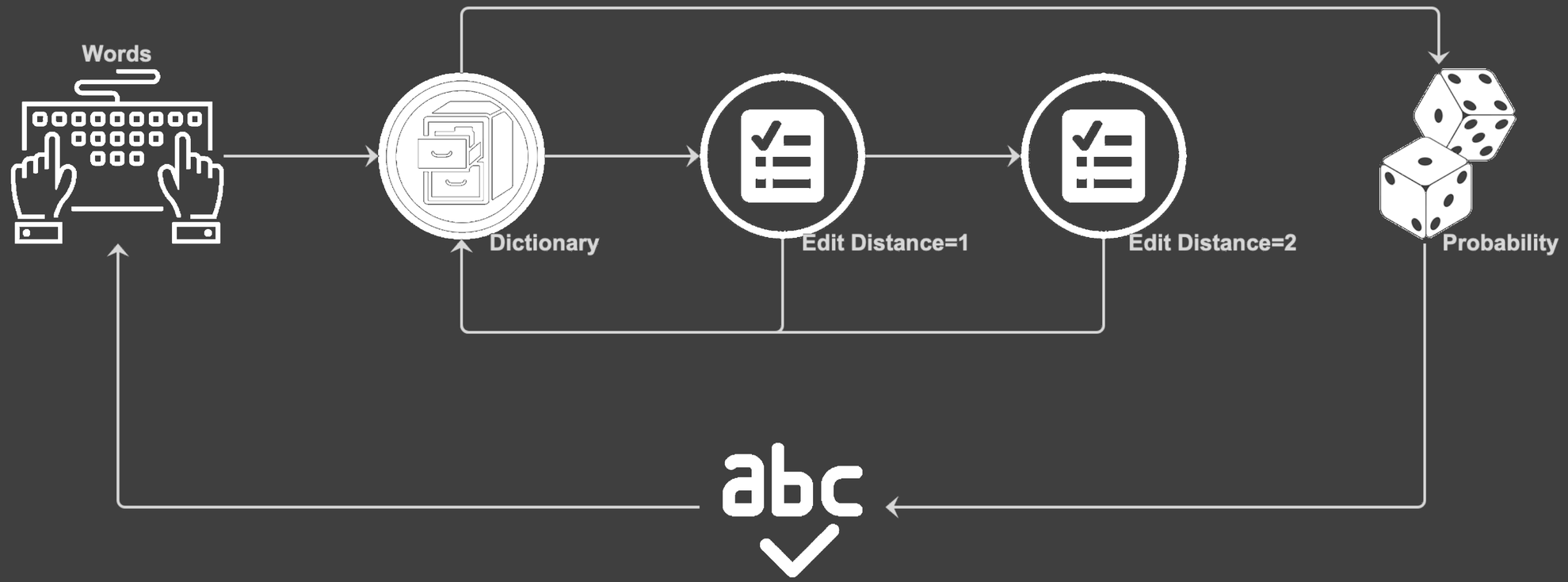


DATA COLLECTION

To support full text search over a set of documents, **Inverted Index** is used here.

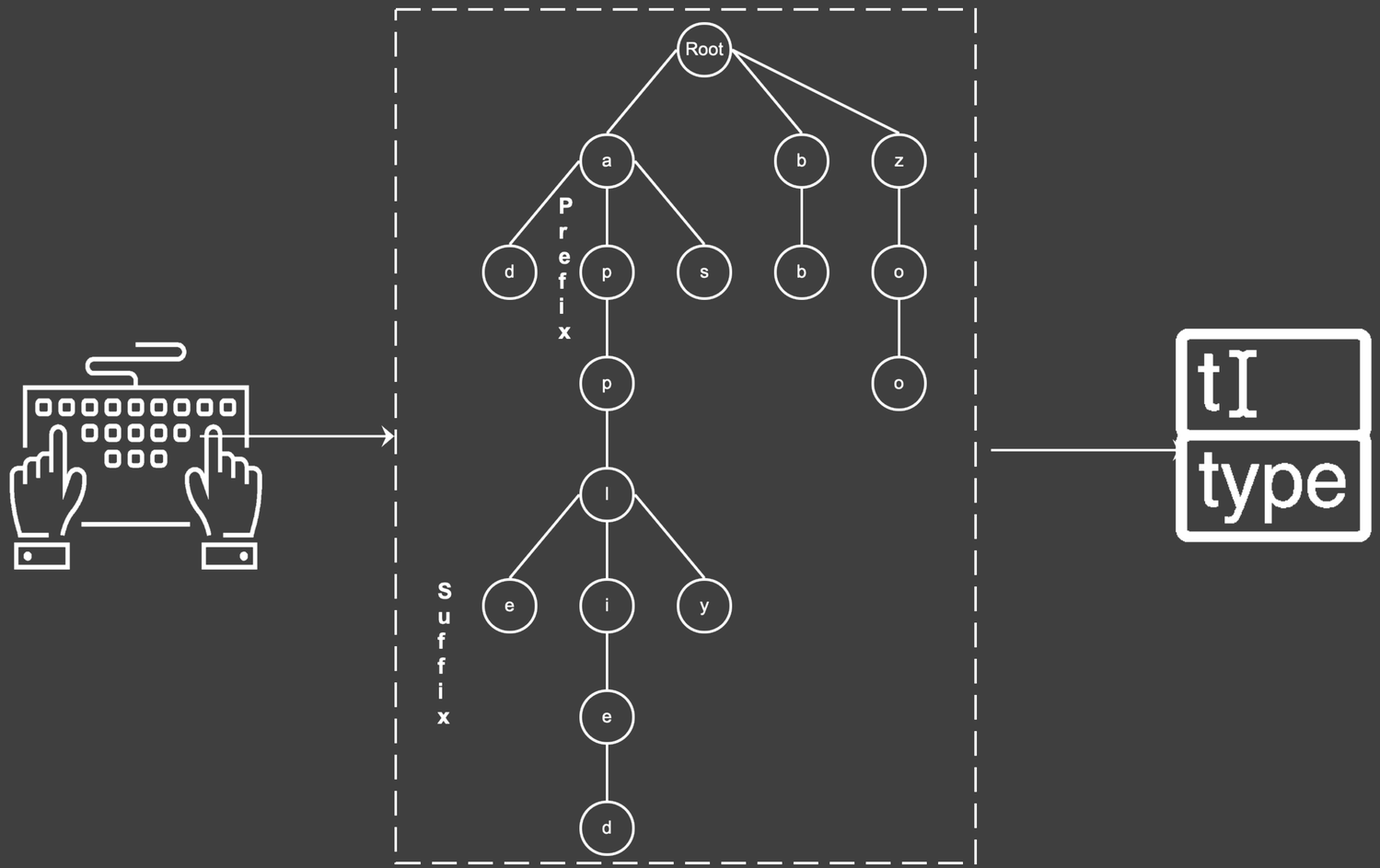


ID	Word	{Doc_id, Count}
001	master	{doc_1010, 100}, {doc_1011, 200}
002	applied	{doc_1010, 100}, {doc_0091, 30}
...	…	…
N	computer	{doc_1010, 100}



The algorithm for query checking it's the non-word errors correction based on **Edit Distance**.

SPELLING CHECKING



By using Trie, put all the words from the dictionary in terms of chars. Then we get the suffix according to the given prefix.

AUTO-COMPLETE

SEARCH MODULE

SCORING-BASED ON TF-IDF

Key idea of TF-IDF [Term Frequency - Inverse Document Frequency]

More frequent the word is, the more it represents the document.

The words that occurs in less documents in the corpus are considered more valuable keywords

$$tf\_idf_{word\_i,doc\_j}$$

$$tf\_idf_{word_i,doc_j} = tf_{word_i,doc_j} . idf_{word_i}$$

$$tf_{word_i,doc_j} = \frac{freq_{word_i,doc_j}}{||N_j||}$$

$$idf_{word_i} = log\left(\frac{N}{df_i}\right)$$

$freq_{word_i,doc_j}$ = $frequency\ of\ word_i\ in\ doc_j$

$||N_j||$ = $euclidean\ norm\ of\ doc_j$

N = $total\ number\ of\ documents\ in\ corpus$

df_i = $Number\ of\ documents\ in\ which\ word_i\ appears$

CALCULATION OF TF-IDF SCORE

PAGE SCORE

After calculating TF-IDF scores we have a matrix like below:
Each row is a vector (very very sparse) - the vector of tf_ifd socre dor that document with all the words in the corpus

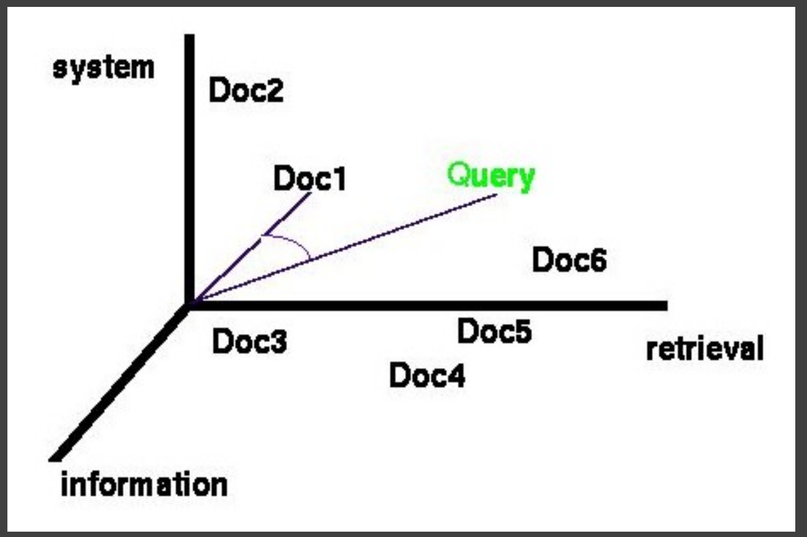
	word_1	word_2	word_3	...	word_n-2	word_n-1	word_n
doc_1	0	0.02	0		0	0.67	0
...							
doc_m	0	0	0.24		0	0	0.44

	word_1	master	word_2		applied	word_n-1	computing
q	0	0.8	0		0.76	0	0.34

The final score of the document - Cosine-similarity

The final score is given to a document as the amount of similarity between itself and the search query, which is calculated as the cosine angle between the document vector and query vector (red and blue in the table diagram)

$$score_{doc_i} = \frac{\overrightarrow{doc_i} \cdot \vec{q}}{|\overrightarrow{doc_i}| * |\vec{q}|} = \frac{\sum doc_{i_x} * \sum q_x}{\sqrt{\sum_{x=0}^{n} doc_{i_x}{}^2} * \sqrt{\sum_{x=0}^{n} q_x{}^2}}$$



The document with least cosine angle with query is the most relevant document. We find the top K pages using a slightly modified quick-select algorithm on the document ranks as follows.

	page_1	page_2	page_3	...	page_n-2	page_n-1	page_n
query	0	0.8	0	...	0.76	0	0.34

We have the page-score array as

- Find the K-th smallest element as in Quick-Select
- At this point we have K-1 elements on the left of the K-th element. Sort all of these using Quick-Sort and hence we have top K pages on O(n).
  [O(n) even after quick-sort because we sort only K pages irrespective of input size]

RANKING - USING QUICK SELECT

DATABASE BASED


QUESTION TIME