Saryshli

---

contributors

> Ahmed Asaad Darwish Ahmed Mohamed Mostafa Ahmed Mohamed Ahmed Lotfy Omar Fareed Abdelaty

---

Introduction

Our search engin **Saryshli** is based on three main components the **indexer**, the **crawler**, the **interface**.

Crawler

- **Description**

  > The job of the crawler is to start with a seed set, a small collection of websites and use them to discover a much larger pool of websites that can be stored in our databases and include in our search results, the algorithm goes as follows:

  - A **Queue** includes the initial seed set.
  - We take the number of **threads** from the user and then check whether there was an uncompleted run before or not. if the last run was interrupted, we will **re-crawl** from the last crawled link.
  - Each **thread** dequeues the links from the queue and check for the **robots.txt** and for **compact string** for the fetched link.
  - If the link passed the past tests, it will crawl through the document and store **url**, **compact string**, **title**, **popularity** of this document in the database.
    - **compact string**: it is a unique string for each url to check if we visited this url before or not, Our compact string consists of the **first quarter** and the **last quarter** of the page.
    - **Robots.txt**: we get this file from the link, and check whether our bot can connect to this link or not.
    - **popularity**: it is an integer which indicates how many pages refers to this link, which increases its priority in the links queue.

Indexer

- **Description**

> The job of the **indexer** is to index the document in the database to make it easy and fast to search for specific sentences without having to search in any page.

- we create a **hash table** to store information about each world.
- fetching the links from the crawler database and loading the web pages using **Jsoup** library.
- we are iterating over the whole page character by character to extract words
- for each word we ask if it's not a stop word and then making **stemming** to the word.
  - **stop words**: like **is**, **am**, **he**, ..**etc**
- now we increase the priority of that word in the page and store the positions of occurrences in the page.
- iterating through the **hash table** and store the information of each word in the database.

Ranker

- **Description**

> The job of the **ranker** is to rank urls to return different urls for each page due to the **ranking equation**

- Retrieve all urls related to searching sentence
- Save all urls in a **HashMap**<url, **urlInfo**>
- **urlInfo** contains all words in this url and all related information to this word
  - *weight* : summation of the weights of the word locations
  - *ITF* : Item term frequency
  - *occurs-at* : all positions of the word occurrences
- Save number of complete sentences for each url in a HashMap<url , Integer>
- Choosing the links due to **ranking equation**
  - *ranking-equation* for each url **1 * totalWeight + 7 * numOfWordsInTheDocument + 10 * total_ITF + 4 * numOfCompletedSentences + 10 * popularity**

Interface

- **Description**

> The **interface** is responsible for displaying the search environment for the user and display the results using **pagination**.

- To handle the request we use **Jzava Servlets**.
- We host the home page and the results page using **Tom Cat**.
- We make **FrontendHandler** which is a java class to handle the request and return the result page.
- Also we provide a **Voice search**.