Web Search Engine for Computer Science

Description

In this project, I crawl Wikipedia webpages for Computer Science first. And then, these fetched webpages are used to build a small web search engine by using multiple methods mentioned in the CS3245.

Crawling

The whole pipeline of crawling can be shown as Fig. 1, which is originally posted at [Lecture 12: Crawling and Link Analysis]

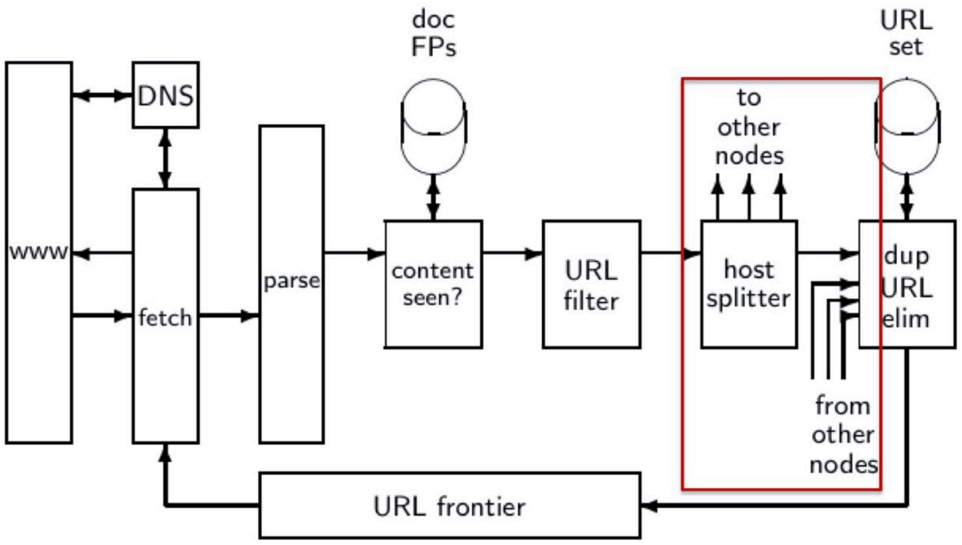


Figure 1 Crawling architecture

To make a small search engine for Computer Science with high webpage quality, I set `https://en.wikipedia.org/wiki/Outline\_of\_computer\_science` as a root link for crawling.

After fetching and parsing webpages, we should check the category of this webpage whether it is related to computer science. At the bottom of each wiki page, there is a Categories tag that can easily be selected for filtering out non computer science related pages.

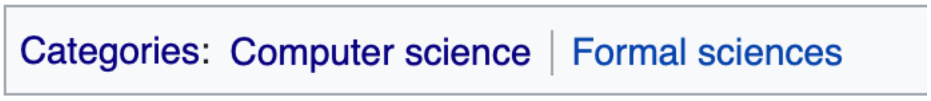

Categories: Computer science | Formal sciences

Figure 2 Categories

Based on that, I set some keywords for filtering webpages. Here are some shorten words of categories.

```
category_list = ['info', 'com', 'tech', 'sci', 'dev',
'alg', 'learn', 'eng', 'math', 'oper', 'data']
```

After that, we should also check the content of the webpage whether we have seen it in the previously fetched webpages. All we want do is avoiding crawl duplicated webpages and reduce the size of crawled data.

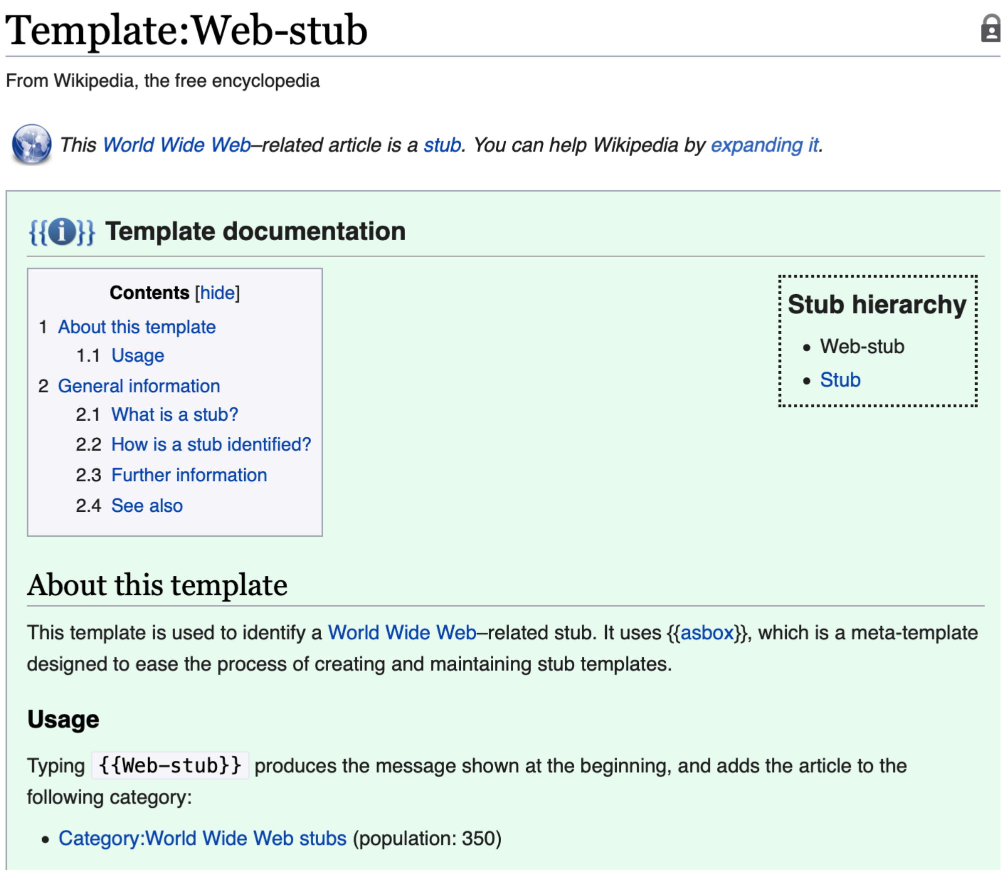
There are two scenarios of duplicated content:

1. different URLs but have the same content

- https://en.wikipedia.org/wiki/Internet_Movie_Database
- https://en.wikipedia.org/wiki/IMDb

2. different URLs but the content almost the same

- https://en.wikipedia.org/wiki/Template:Web-stub
- https://en.wikipedia.org/wiki/Template:KosciuskoCountyIN-geo-stub



Template:Web-stub

From Wikipedia, the free encyclopedia

This World Wide Web–related article is a stub. You can help Wikipedia by expanding it.

Template documentation

Contents [hide]

1 About this template
  1.1 Usage
2 General information
  2.1 What is a stub?
  2.2 How is a stub identified?
  2.3 Further information
  2.4 See also

Stub hierarchy

- Web-stub
- Stub

About this template

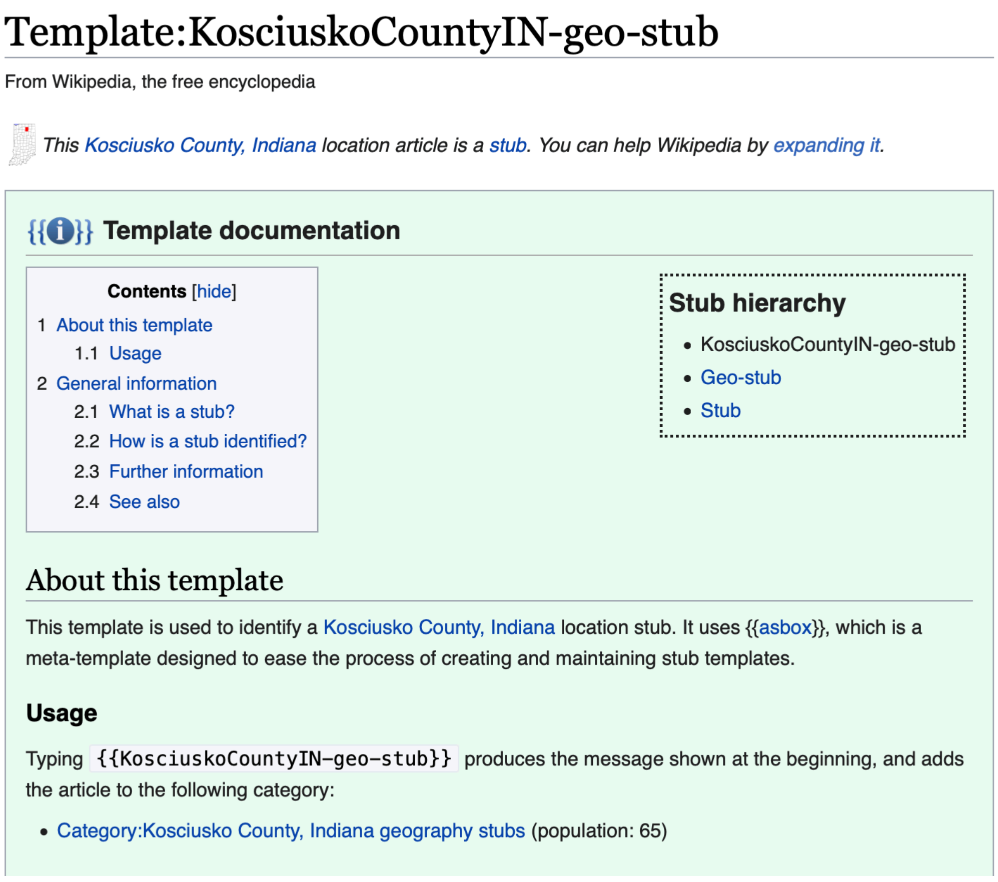
This template is used to identify a World Wide Web–related stub. It uses {{asbox}}, which is a meta-template designed to ease the process of creating and maintaining stub templates.

Usage

Typing `{{Web-stub}}` produces the message shown at the beginning, and adds the article to the following category:

- Category:World Wide Web stubs (population: 350)

Figure 3 Webpage screenshot


Template:KosciuskoCountyIN-geo-stub

From Wikipedia, the free encyclopedia

This Kosciusko County, Indiana location article is a stub. You can help Wikipedia by expanding it.

Template documentation

Contents [hide]

1 About this template

1.1 Usage

2 General information

2.1 What is a stub?

2.2 How is a stub identified?

2.3 Further information

2.4 See also

Stub hierarchy

- KosciuskoCountyIN-geo-stub
- Geo-stub
- Stub

About this template

This template is used to identify a Kosciusko County, Indiana location stub. It uses {{asbox}}, which is a meta-template designed to ease the process of creating and maintaining stub templates.

Usage

Typing `{{KosciuskoCountyIN-geo-stub}}` produces the message shown at the beginning, and adds the article to the following category:

- Category:Kosciusko County, Indiana geography stubs (population: 65)

Figure 4 Webpage screenshot

With many hyperlinks in the webpages, not only do we use these links to build an A matrix in the PageRank part, but also they are used for further crawling. Since we only care about wiki webpages, we can filter out other host links and drop some danger links with robots templates.

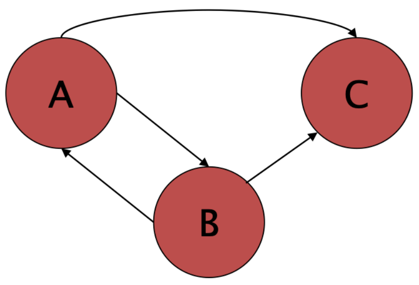
With all actions above, we save `webpage.url`, `webpage.title`, `webpage.content`, and `webpage.outlinks(hyperlinks)` to the file, which will be processed in the indexer part.

Used Tech

1. Simhash
   To finish the content check part, I use an external library called `SimHash`. The key part of `SimHash` is to convert weights of each term into hash value and get a fixed length of fingerprint of each document.

PageRank

With fetched URLs and their hyperlinks in the crawling part, we can easily build an A matrix by using Markov chains and refine it by using Teleporting to avid `Dead End` and `Spider Trap`



Try this: Calculate the matrix A_{ik} using 10% chance of teleportation.

$$A_{ik}: \begin{array}{cccc} & A & B & C \\ A & 1/30 & 29/60 & 29/60 \\ B & 29/60 & 1/30 & 29/60 \\ C & 1/3 & 1/3 & 1/3 \end{array}$$

Figure 5 Matrix A

The Fig. 5 is originally posted at Lecture 12: Crawling and Link Analysis

After that, we calculate `array a` by using pagerank algorithm to represent the probability of each webpage. These probabilities can be used in the searcher part.

```
a = array([[8.30168216e-05, 8.30168216e-05,
1.57926651e-04, ... 3.82857116e-04, 3.82857116e-04,
4.64916368e-04]])
```

Indexer

The structure for storing data can be shown as Fig. 6

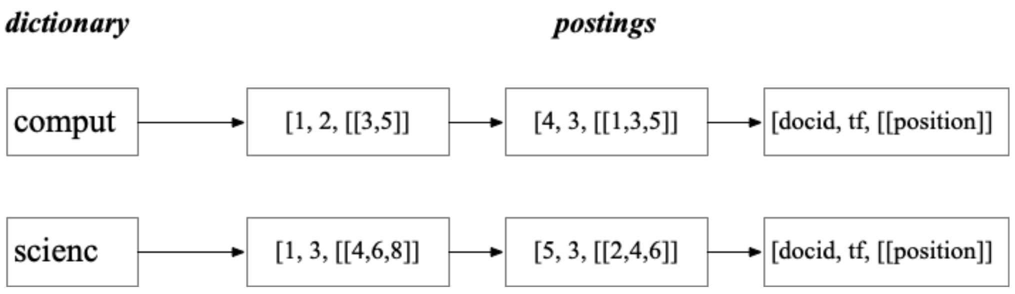


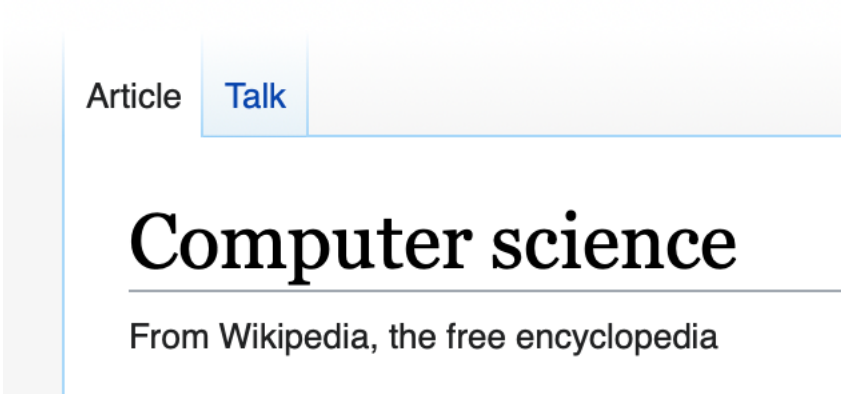
Figure 6 Indexer structure

For each fetched webpages, remove all punctuations from title and content, stem the words and change all characters to lower case. Store the doc_id, term frequency, and position to the postings. (position will be used for phrasal query). To build an A matrix, we fetch hyperlinks of each webpage and assign value corresponding to doc_id and index of hyperlinks.

After processing webpages, we refined the `A matrix` and calculate `array a`. And then save all things (array a, dictionary, and postings) to the file.

Used Tech

1. zone and field



Computer science

From Wikipedia, the free encyclopedia

Figure 7 Title

As we can see, in each webpage, title represents the important information about this webpage. Thus, indexing them is also important when users want to search something like “find docs with merchant in the title zone and matching the query gentle rain”

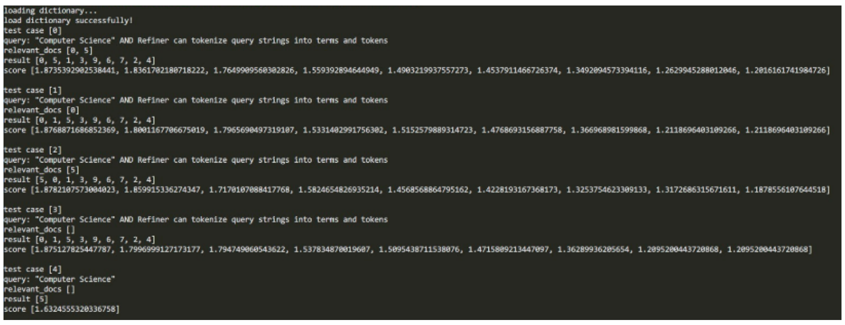
As for the field part, I don’t see any useful information like data_published or court, so I don’t process them.

Searcher

Given the user’s input as a query, we tokenize it first. With dictionary.txt and postings.txt stored before, we construct the query vector and document vector by using the VECTOR SPACE MODEL. After calculating the similarity between each document and query, we normalize and combine them with the array value. In general, I just treat the normalized similarity as probability and calculate the average. Finally, all related webpages will be shown with sorted order.

Used Tech

1. query expansion
   In query expansion, additional input is given based on synonyms. After tokenizing the query in the searcher part, I call _expand () function to add synonyms.
   Within _expand() function, for each token, we call wordnet.synsets(token) to get a set of synonyms. In order to get more precise answers, we use a dictionary to check all synonyms. If they exist in the dictionary, add them to the query.
   Given the query computer science, after expansion, the result is calculate computer science skill.
2. relevance feedback
   In this part, we use Rocchio Algorithm to do it. Assume that top 5 documents in the original result is highly relevant to the query, we use these doc vectors to modify the query vectors and search it again.



```
loading dictionary...
load dictionary successfully!
test case [0]
query: "Computer Science" AND Refiner can tokenize query strings into terms and tokens
relevant_docs [0, 5]
result [0, 5, 1, 3, 9, 6, 7, 2, 4]
score [1.8735392902538441, 1.8361702180718222, 1.7649909560302826, 1.559392894644949, 1.4903219937557273, 1.4537911466726374, 1.3492094573394116, 1.2629945288012046, 1.2016161741984726]

test case [1]
query: "Computer Science" AND Refiner can tokenize query strings into terms and tokens
relevant_docs [0]
result [0, 1, 5, 3, 9, 6, 7, 2, 4]
score [1.8768871686852369, 1.8001167706675019, 1.7965690497319107, 1.5331402991756302, 1.5152579889314723, 1.4768693156887758, 1.366968981599868, 1.2118696403109266, 1.2118696403109266]

test case [2]
query: "Computer Science" AND Refiner can tokenize query strings into terms and tokens
relevant_docs [5]
result [5, 0, 1, 3, 9, 6, 7, 2, 4]
score [1.8782107573004023, 1.859915336274347, 1.7170107088417768, 1.5824654826935214, 1.4568568864795162, 1.4228193167368173, 1.3253754623309133, 1.3172686315671611, 1.1878556107644518]

test case [3]
query: "Computer Science" AND Refiner can tokenize query strings into terms and tokens
relevant_docs []
result [0, 1, 5, 3, 9, 6, 7, 2, 4]
score [1.875127825447787, 1.7996999127173177, 1.794749060543622, 1.537834870019607, 1.5095438711538076, 1.4715889213447097, 1.36289936205654, 1.2095200443720868, 1.2095200443720868]

test case [4]
query: "Computer Science"
relevant_docs []
result [5]
score [1.6324555320336758]
```

Figure 8 Relevance Feedback

As you can see in Fig. 8, if the relevant docs are 0 and 5, the result changes and top 2 results are 0,5. All scores of documents are also changed.

Limitation

1. Less crawling data
   In this mini-project, I just crawl 2000 webpages about Computer Science. With a large corpus in Wikipedia, there are plenty of unprocessed CS-related pages. For example, in my dataset, I don’t even have a wiki about “Information Retrieval” and “Computer Science”. Therefore, the result of them may not be very convincing.
2. Not satisfied A matrix

To build a good A matrix, every fetched webpage should contain as many hyperlinks as possible. However, when I check the A matrix, only 127 webpages have values, which means that all rest sites are dead ends. With this drawback, values in the array are not very meaningful.

Author

👤 Evan - Crawling, PageRank, Indexer, query expansion

- Twitter: @NavePnow(https://twitter.com/NavePnow)
- Github: @NavePnow(https://github.com/NavePnow)

👤 Rulin - Searcher, relevance feedback

- Github: @XJDKC (https://github.com/XJDKC)

Reference

- SimHash
- WordNet
- CS3245
- CS3245-NUS-HW3
- CS3245-NUS-HW4