ENHANCING THE QUEUING PROCESS FOR YIOOP'S SCHEDULER

Gargi Sheguri

Committee:

Dr. Chris Pollett (Advisor)

Dr. Robert Chun

Dr. Ben Reed

AGENDA

- Introduction
- Background
- Preliminary Work
- Deliverable#1: Bonus Factors
- Deliverable#2: SERP Freshness
- Deliverable#3: Improving Queries
- Conclusion

INTRODUCTION

- Yioop is a PHP-based, open-source web search engine
- Aim: Improve the overall quality of search results generated for a user query
- There are three major processes in search: **crawling, indexing,** and **retrieval**
- This project works on improving the indexing and retrieval processes
- Deliverables:
  - Uplifting certain results by incorporating new bonus factors
  - Improving the overall "freshness" of the SERP (Search Engine Results Page) with latest results
  - Increasing the number of results generated and making lookup faster

BACKGROUND

High-Level Components of Yioop Search

- **Fetchers:**
  - Download web pages
  - Perform initial parsing of downloaded content

- **Scheduler:**
  - Creates batches of URLs to be crawled next by priority

- **Indexer:**
  - Handles pre-processing and storage of documents in index

- **Query Processor:**
  - Processes users' search queries
  - Handles lookup

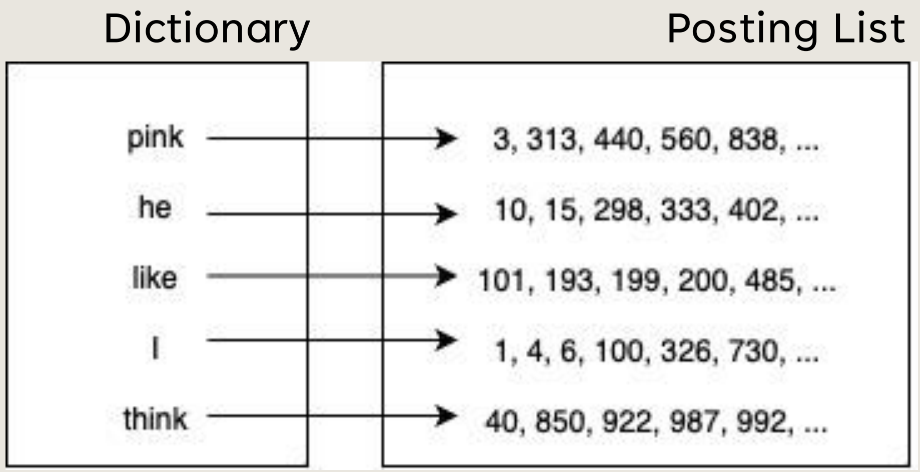
BACKGROUND

Document-at-a-time Processing

- Each document in the index is treated as an independent entity
- Scores are provided to individual documents based on the search terms
- The matched documents are sorted and the top *k* results are returned to form the SERP

Inverted Indexing

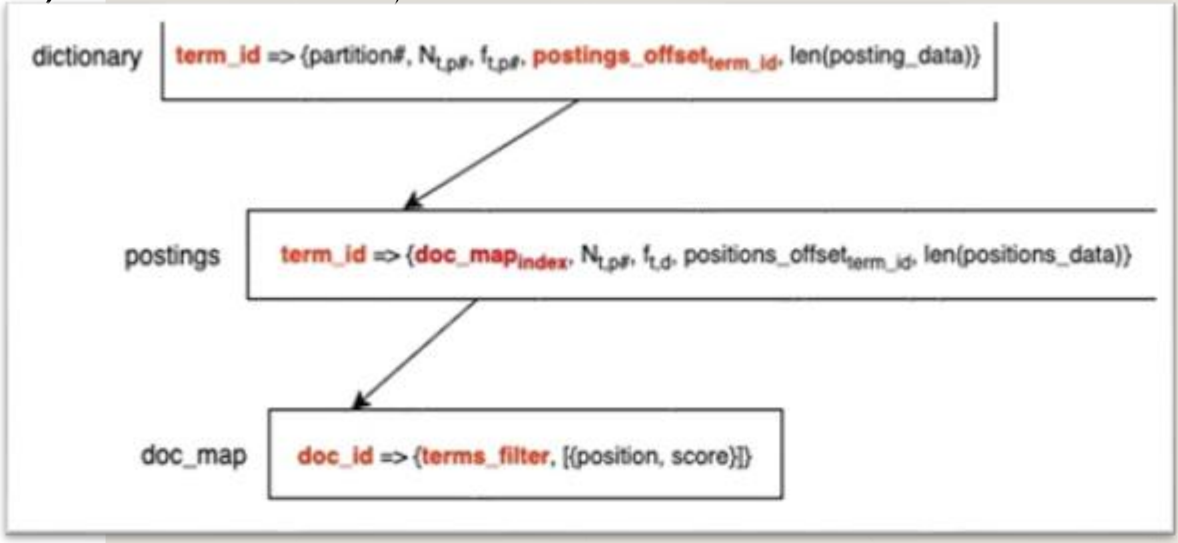
- Mapping terms to the documents they appeared in
- Each term (key) points to a list of positions of documents in the index
- Fast and efficient retrieval in large indexes



BACKGROUND

Indexing in Yioop

- Yioop uses a directory of files to store its inverted index
- This is because the entire index is too large to fit into main memory
- Index is thus divided into several independent partitions
- Important index components:
  - **documents**
    - Stores info about partition, document summaries, compression formats used
  - **positions_doc_map**
    - Holds serially numbered directories (for each partition)
    - Made up of:
      - **doc_map**
        - Scoring information for constituent terms
      - **positions**
        - Locations of term in documents
      - **postings**
        - Posting lists for term
  - **dictionary**
    - B+ tree mapping term_id to cumulative posting list



BACKGROUND

Meta Keywords in Yioop

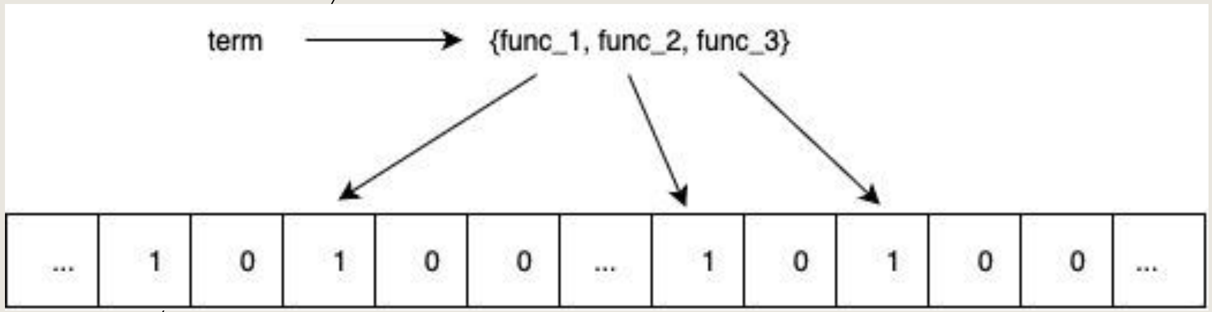
- Query parameters denoted as <key:value> pairs
- Can be included in search queries to filter results by specific criteria
- Yioop automatically adds certain meta keywords to queries, such as *lang* and *safe*
- Users can specify additional meta keywords in the search query
- If the *no:guess* keyword is added to the query, no additional meta keywords are assumed from the search terms

```
public static $meta_words_list = ['\-i:', '\-index:',  '\-', 'class:',
    'class-score:', 'cld:', 'code:', 'color:', 'date:', 'dns:', 'duration:',
    'filetype:', 'guid:', 'hash:', 'host:', 'i:', 'info:', 'index:', 'ip:',
    'link:',  'lang:', 'layout:', 'location:', 'media:', 'modified:',
    'numlinks:', 'os:', 'path:', 'pubdate:', 'robot:', 'safe:', 'server:',
    'site:', 'size:', 'time:', 'u:', 'version:','weight:', 'w:'
    ];
```

BACKGROUND

Bloom Filters

- Aimed at creating memory-efficient data structure to check membership in sets
- Uses multiple hash functions to map member elements to positions in bit array
- Same functions are used to test for membership
- If the bits in the corresponding to the hash outputs are set, the element is present in the set
- Constant time complexity for insertion and membership testing
- **Primary advantage: no false negatives**



DELIVERABLE#1: BONUS FACTORS INSPIRED BY YANDEX

What are bonus factors?

- Boost certain results in SERP ranking
- Add "bonus" scores to documents meeting certain criteria in index
- Aim: Improving Click-Through-Rate (CTR)
- Eg. Documents wherein the query terms appearing in the page title or URL get bonus scores

What is Yandex Search?

- Russian multinational IT company, most popular for its search engine
- Often considered to be Russia's equivalent to Google
- Close to 45GB of source code was leaked in January 2023
- Leak revealed more than 1,920 search factors used by Yandex

DELIVERABLE#1: BONUS FACTORS INSPIRED BY YANDEX

NUM_SLASHES_BONUS

- Based on Yandex Search's FI_NUM_SLASHES bonus
- Boosts results with lesser '/' in URL
- Idea: the further a page is from the "home" page, the less important it is

WIKI_BONUS

- Based on Yandex Search's FI_IS_WIKI bonus
- Boosts Wikipedia page results
- Idea: Wikipedia pages tend to be more relevant than other results

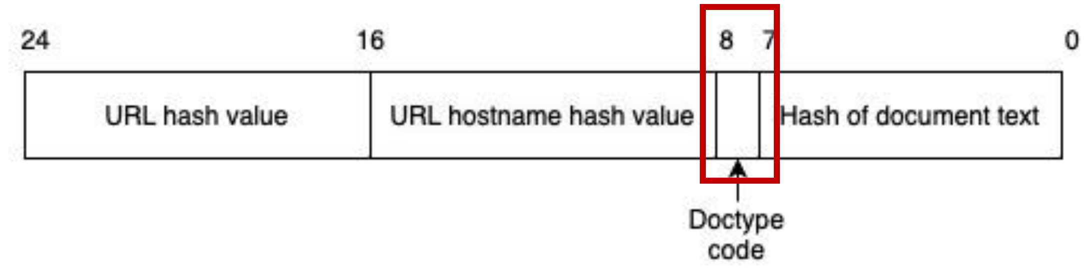
DELIVERABLE#1: BONUS FACTORS INSPIRED BY YANDEX

Why Wikipedia?

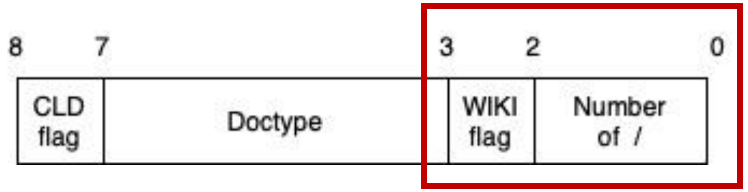
- Most popular web search engines boost Wikipedia results
- Reliable and trustworthy source of information due to collaborative nature
- Pages usually have a list of references, easy to verify accuracy
- Updated frequently
- Exhaustive range of topics

IMPLEMENTATION

- <u>DOC_ID format in Yioop:</u>
  - Length: 24 bytes
  - Uniquely identifies an indexed entry
  - Doctype code:
    - Holds descriptive information
    - Represents type of document (eg. binary, image, text, etc)



- <u>Modifying doctype code:</u>
  - Length: 1 byte
  - Number of Slashes representation:
    - 0: Between 0 and 1 slashes
    - 1: Between 2 and 4 slashes
    - 2: Between 5 and 6 slashes
    - 3: 7 or more slashes



IMPLEMENTATION

How does it work?

- When the score for a matched document is being calculated, the code calls specific functions to check for additional bonuses
- If the 3rd bit of the doctype code of a found *doc_id* is set, the WIKI_BONUS is added to the final score
- A fraction of the total NUM_SLASHES_BONUS is added the final score based on the count of '/' in the document URL
- The first two bits of the doctype code of a found *doc_id* are extracted to find this fraction
- The number NUM_SLASHES_BONUS is divided by this extracted value and added to the final score
- Thus, the bonus factor added is inversely proportional to the depth of the page

EXPERIMENTATION: SETUP

Tested Crawl Sizes
300024
557808
600110
1868398

Tested Searches
google
apple
wikipedia
yahoo no:guess
verizon
weather
ebay lang:en
site:google.com
site:apple.com
site:pinterest.com lang:en

EXPERIMENTATION: WIKI_BONUS

Tested Weights
Bonus = 5
Bonus = 1
Bonus = 0.75
Bonus = 0.5
Bonus = 0.25

- > 0.5 values boosted Wikipedia too far up
  - Wikipedia results appeared in the top three results on searching for *google, verizon, weather, apple*
  - It even appeared as the top result in some cases
  - It beat URLs from the corresponding domain sites
- = 0.5 gave the best results
  - The domain site appeared as the top result
  - Wikipedia results appeared after seemingly more important URLs, but in the top 10
- < 0.5 did not boost some Wikipedia results enough
  - Wikipedia results ranked lower than some deep-nested URL subdirectories
  - Sometimes Wikipedia results didn't make it into the top 10

EXPERIMENTATION:NUM_SLASHES_BONUS

Tested Weights
Bonus = 2 Buckets = {0-2, 3-4, 5-7, 8+}
Bonus = 1 Buckets = {0-1, 2-4, 3-6, 7+}
Bonus = 1 Buckets = {0, 1-2, 3-4, 5+}
Bonus = 0.5 Buckets = {0, 1, 2, 3+}
Bonus = 0.5 Buckets = {0-1, 2-4, 5-6, 7+}
Bonus = 0.5 Buckets = {0-2, 3-4, 5+}

- > 0.5 values worsened the SERP ranking: domain sites came up higher than any nested pages, even if the latter were more relevant
  - Eg. *www.verizon.com* and *www.ebay.com* came up higher than *www.apple.com/products/…*, *www.apple.com/support*…, etc. for a search on *apple*
- = 0.5 gave the best results
  - Bucket range {0-1, 2-4, 5-6, 7+} gave better results than {0, 1, 2, 3+} and {0-2, 3-4, 5+}
- < 0.5 did not affect the prior Yioop results noticeably
  - Deeper-nested URLs (nested in 3+ subdirectories) did not appear in the expected order of importance

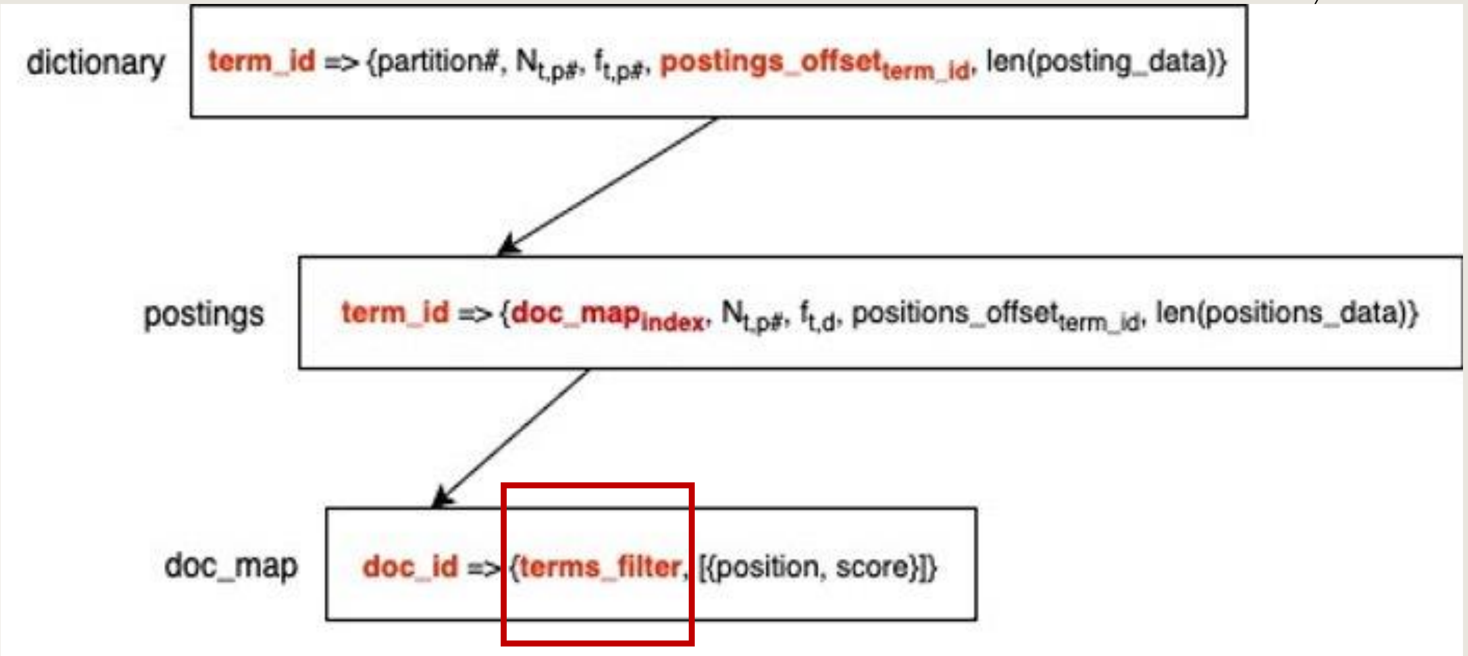
DELIVERABLE#2: IMPROVING SERP FRESHNESS

- Yioop uses a Bloom filter to keep track of URLs that have been crawled to avoid repetition
- This filter is cleared periodically to avoid space inefficiency and keep up with sites that are updated frequently
- Problem:
  - Multiple versions of a page result
  - Yioop considers the first-crawled version to be the most important
  - "Stale" results might come up
  - The SERP might not include the latest version of a result
  - The latest version of a page might not contain the search query terms
- Lookup has to be modified to show updated results

IMPLEMENTATION

<u>Terms Bloom filter:</u>

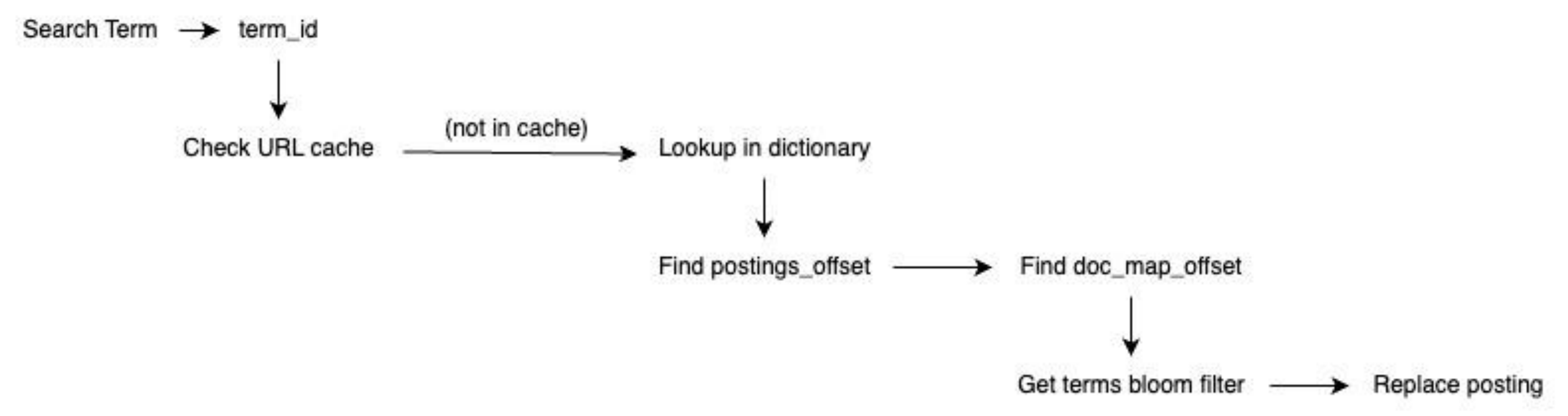
- Indexing logic is modified to include the top 300 words present in a document via a Bloom filter
- This will help during lookup, to confirm that the search term exists in the indexed document
- The Bloom filter is added to *doc_map* entries



IMPLEMENTATION

<u>Finding the most recent version of a result:</u>

- Lookup comprises of word iterators to fetch documents associated with a single search term
- The WordIterator class constructor now accepts a flag marking whether the most recent version of a result needs to be looked up
- A cache of URLs and the positions of their latest versions in the index are maintained to improve lookup time



EXPERIMENTATION: SETUP

Tested Crawl Sizes
1128038
1793491
1500000

Tested Searches
mobile
horse
mountain
goodread book
yahoo
weather.com
site:https://www.google.com/
site:https://www.wikipedia.org/
bestseller no:guess

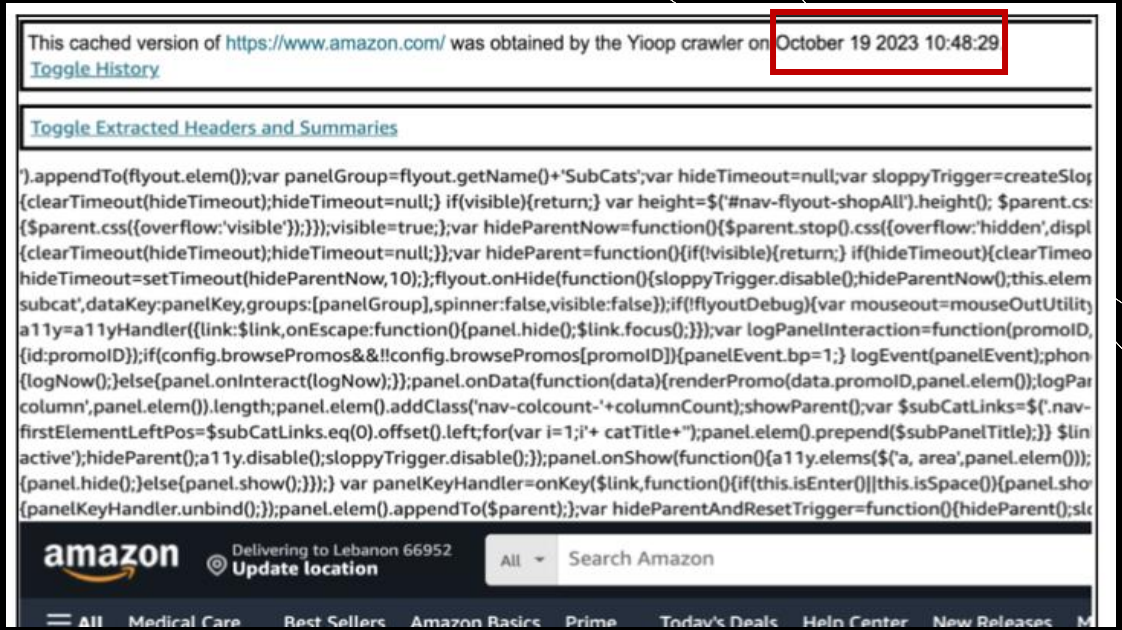
EXPERIMENTATION: OBSERVATIONS

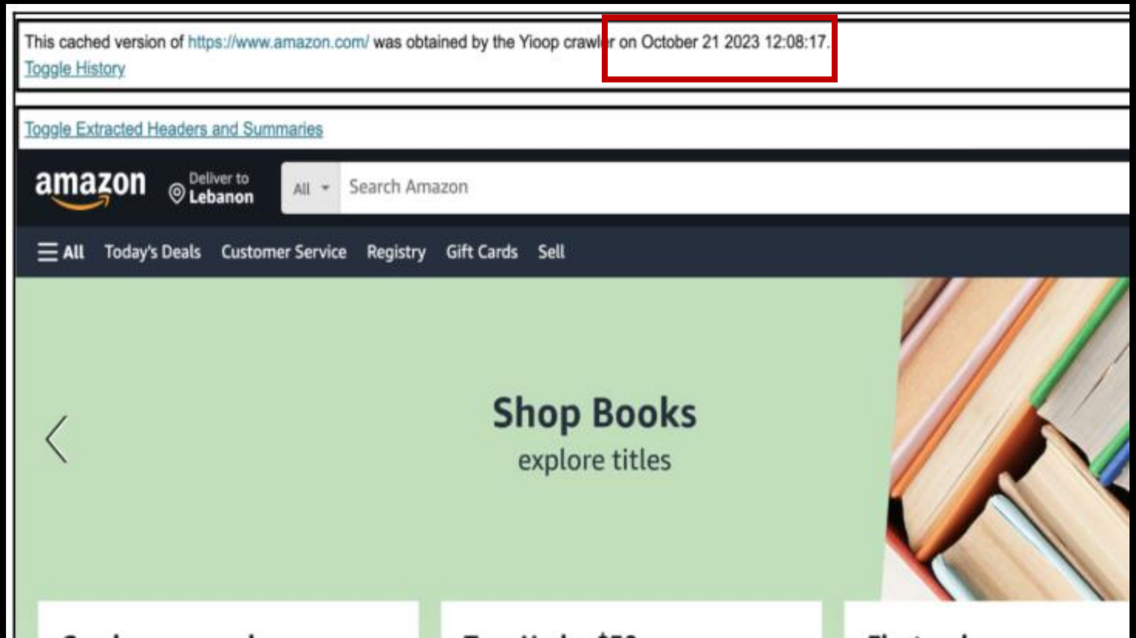
Search	Original Time (ms)	Crawl#1 (ms)	Crawl#2 (ms)
mobile	801	846	866
mountain	688	693	693
goodread book	551	688	691
site:https://www.google.com/	1013	1102	1099
horse	440	502	511
yahoo	399	410	502

Takeaways:

- Response generation time did not increase dramatically for most queries
- Search time increased by 0.1s for *goodread book* and *horse*
- Overall, tradeoff (increased lookup time v/s freshness) seems fair and does not diminish efficiency

EXPERIMENTATION: OBSERVATIONS





DELIVERABLE#3: IMPROVING SEARCH QUERIES

Conjunctive v/s disjunctive queries:

- Conjunctive queries separate search terms using AND operators
- Disjunctive queries separate search terms using OR operators

- Why use disjunctions?
  - Broadened search scope
  - Search for synonyms
  - Diverse set of results

Search Query	*chatgpt openai*
Conjunctive Query Equivalent	*chatgpt AND openai*
Disjunctive Query Equivalent	*chatgpt OR openai*

- Quotes and terms separated by '&' operators are retained as conjunctions
- Disjunctive terms/phrases are separated by '|' operators

DELIVERABLE#3: IMPROVING SEARCH QUERIES

Query Processing with Heaps:

- Consider that the top *k* documents are to be returned, the number of search terms is *n*, and there are *m* matching documents in the index
- The overall time complexity of retrieving and sorting *m* documents is $\theta(m*n + m*\log m)$
- Issues:
    - Unnecessary cost of sorting *m* documents
    - Need to go through all *n* terms to calculate score, even if document text doesn't contain them
- Heaps efficiently overcome these issues

DELIVERABLE#3: IMPROVING SEARCH QUERIES

MaxScore:

- Idea: Finding an upper bound on a term's overall contribution
- Yioop uses Divergence-from-Randomness to score documents

$$\mathrm{DFR}_t = \frac{\log\left(1 + \frac{l_t}{N}\right) + f_{t,d} \log\left(1 + \frac{N}{l_t}\right)}{f_{t,d} + 1}$$

- The maximum relevance score that a document can achieve for a query is:

$$1 + \log_2\left(1 + \frac{N}{l_t}\right)$$

  where N is the total number of documents in the index and l_t is the total number of occurrences of the search term

- Using this relevance calculation, the MaxScore that a document containing (only) the search term in question can possibly achieve is given as:

$$1 + \log_2\left(1 + \frac{N}{l_t}\right) + \mathrm{DR}_{\max}$$

where DR_{max} is the maximum Doc Rank score that can be achieved

IMPLEMENTATION

Converting to disjunctive:

- The search string is divided by whitespaces into multiple disjuncts
- Each disjunct is treated as an independent search query
- Meta words are tacked onto each query
- Documents matching each query are retrieved
- A UnionIterator instance is used to score, combine, and sort the top results obtained from each WordIterator

IMPLEMENTATION

Maintaining heaps:

- Query processing now includes heaps to make search more efficient
- Three min heaps used:
  - **Results heap:** Maintains top *k* documents found until now
  - **Search terms heap:** Maintains query terms being searched for
  - **Low-scoring terms heap:** Maintains query terms with low MaxScore values

IMPLEMENTATION

- The **search terms heap** is used to find the position of the next document matching the search criteria
- The corresponding score is calculated for this document (a sum of its DocRank and relevance scores for each search term appearing in it)
- If the found score is greater than the current k^{th} best score in the results heap, it is inserted into the **results heap** (and reheap is invoked)
- Before looking for the next matching document, any terms on the search terms heap with a MaxScore value lower than the current k^{th} best score in the results heap are maintained in the **low-scoring terms heap** instead
- These terms are not used for lookup, but their relevance scores are added to the appropriate total document score

EXPERIMENTATION: SETUP

Tested Crawl Sizes
1128038
1793491
1500000

	Conjunctive Query	Disjunctive Query
1.	*justin trudeau lang:en safe:true*	*justin lang:en safe:true | trudeau lang:en safe:true | justin-trudeau lang:en safe:true*
2.	*prime lang:en*	*prime lang:en*
3.	*prime minist safe:true lang:en*	*prime safe:true lang:en | minister safe:true lang:en | prime-minist safe:true lang:en*
4.	*prime-minist no:guess*	*prime-minist no:guess*
5.	*google verizon pinterest safe:true*	*google safe:true | verizon safe:true | pinterest safe:true*
6.	*“google verizon” lang:en safe:false*	*“google verizon” lang:en safe:false*
7.	*apple & mac lang:en safe:true*	*apple & mac lang:en safe:true*
8.	*weather site:weather.com lang:en safe:true*	*weather site:weather.com lang:en safe:true*
9.	*lang:en media:news w:1 -i:100 #1#*	*lang:en media:news w:1 -i:100 #1#*
10.	*lang:en media:news w:1 -i:100 safe:true*	*lang:en media:news w:1 -i:100 safe:true*
11.	*sand beach california safe:true*	*sand safe:true | beach safe:true | california safe:true*
12.	*chatgpt gpt4 openai safe:true lang:en*	*chatgpt safe:true lang:en | gpt4 safe:true lang:en | openai safe:true lang:en*
13.	*“chatgpt openai” & gpt4 safe:true lang:en*	*“chatgpt openai” & gpt4 safe:true lang:en*

EXPERIMENTATION: OBSERVATIONS

Conjunctive Query	Disjunctive Query	Crawl#1 Conj	Crawl#1 Disj	Crawl#2 Conj	Crawl#2 Disj	Crawl#3 Conj	Crawl#3 Disj
google verizon pinterest safe:true	google safe:true | verizon safe:true | pinterest safe:true	0	412	0	200	1	290
prime-minister no:guess	prime-minister no:guess	2	2	2	2	0	0
prime minister	prime | minister | prime-minister	12	513	9	634	1	501
apple & mac lang:en safe:true	apple & mac lang:en safe:true	14	14	21	21	11	11
lang:en media:news w:1 -i:100 #1#	lang:en media:news w:1 -i:100 #1#	542	542	564	564	1110	1110
sand beach california safe:false	sand safe:false | beach safe:false | california safe:false	0	307	2	399	0	299
"chatgpt openai" & gpt4 safe:true lang:en	"chatgpt openai" & gpt4 safe:true lang:en	0	0	0	0	0	0
justin trudeau lang:en safe:true	justin lang:en safe:true | trudeau lang:en safe:true | justin-trudeau lang:en safe:true	2	49	0	32	11	81

EXPERIMENTATION: COMPARISON WITH POPULAR WEB SEARCH ENGINES

- Human factors used to judge relevance:
  - Relevance of top 10 search results for the query
  - Verified sources and content quality of the top 10 search results for the query
  - Overall recency of pages

Tested Searches
prime minister
apple mac
goodread book
election potus america
california earthquake

- Categorization:
  - True Positive: Relevant websites making it to the top 10 results
  - False Positive: Irrelevant websites making it to the top 10 results
  - True Negative: Irrelevant websites in the top 20 results that did not make it into the top 10 results
  - False Negative: Relevant websites in the top 20 results that did not make it into the top 10 results

- Note:
  The term “irrelevant” is misleading in this context: this experiment considers results that did not crack the top 10 (or usually the first page of results) as less relevant to the search query despite being appropriate responses primarily because the odds of them being clicked on are considerably low

EXPERIMENTATION: COMPARISON WITH POPULAR WEB SEARCH ENGINES

Observations

	Actual Relevant	Actual Irrelevant
Predicted Relevant	34	16
Predicted Irrelevant	41	9

Yioop (Conjunctive)

	Actual Relevant	Actual Irrelevant
Predicted Relevant	45	5
Predicted Irrelevant	15	35

Google

	Actual Relevant	Actual Irrelevant
Predicted Relevant	30	20
Predicted Irrelevant	7	43

Yioop (Disjunctive)

	Actual Relevant	Actual Irrelevant
Predicted Relevant	40	10
Predicted Irrelevant	25	25

Yandex

	Precision	Recall	F1 Score
Yioop (Conjunctive)	0.68	0.453	0.544
Yioop (Disjunctive)	0.60	0.81	0.689
Google	0.90	0.75	0.818
Yandex	0.80	0.615	0.695

EXPERIMENTATION: COMPARISON WITH POPULAR WEB SEARCH ENGINES

Observations

Search Query	Google Overlapping Results (Yioop Conjunctive)	Yandex Overlapping Results (Yioop Conjunctive)	Google Overlapping Results (Yioop Disjunctive)	Yandex Overlapping Results (Yioop Disjunctive)
prime minister	10	7	8	13
apple mac	13	15	9	11
goodread book	11	14	10	12
election potus america	6	4	15	13
california earthquake	8	11	6	6

EXPERIMENTATION: COMPARISON WITH POPULAR WEB SEARCH ENGINES

Takeaways:

- (Yioop's) conjunctive logic precision surpasses disjunctive logic
- Top 3 results of conjunctive logic SERP appear in top 10 results of disjunctive logic SERP
- Disjunctive logic has a low False Positives score, while conjunctive logic has a high False Positives score
  - The second page of the disjunctive logic SERP almost always comprises of results that are less relevant to the query than the top 10 results
- Disjunctive logic gives better results for search terms that are seemingly meaningful to each other (such as searching for synonyms)
  - *Eg. apple mac, election potus america,* and *goodread book*
- For seemingly unrelated search terms, conjunctive logic results were better than disjunctive logic results
  - Eg. Searching for *prime minister* also included a few Amazon Prime pages in the top 20 results
- By comparison, Google and Yandex's SERP results for each of the queries more in tune with the expected results. However, as this experimentation of Yioop was done on a limited index (of approximately 1500000 documents), it is unfair to compare the quality of search results.

CONCLUSION

- Implemented new bonus factors to improve the relevance of search results
- Improved SERP freshness by ensuring that only latest-crawled versions of result pages are offered
- Increased and diversified the search results' space by using disjunctive queries
- Improved lookup time by using heaps and MaxScore calculation in query processing

REFERENCES

C. Pollett, Yioop Search Engine Ranking Mechanisms. https://www.seekquarry.com/p/Ranking.

M. King, "Yandex scrapes Google and other SEO learnings from the source code leak," SearchEngineLand, 2023. https://searchengineland.com/yandex-leak-learnings-392393.

C. Pollett, "Lecture Slides for CS267, 2022", San Jose State University, Accessed: October 23, 2023. [Online]. Available: https://www.cs.sjsu.edu/faculty/pollett/2671.22s.

C. Shepard, “10 Illustrations of How Fresh Content May Influence Google Rankings”, Moz blog, 2016. https://moz.com/blog/google-fresh-factor-new.

G. Amati and C. Rijsbergen, “Probabilistic Models of Information Retrieval based on Measuring the Divergence from Randomness”, ACM Transactions of Information Systems (TOIS), 2022.

B. Bloom. "Space/time trade-offs in hash coding with allowable errors." *Communications of the ACM* 13, no. 7, 1970. pp 422-426.

THANK YOU!

Huge shout out to: Dr. Chris Pollett