



The Wikipedia Goggle

Aapeli Vuorinen *oav2108*

Katie Kim *jk4534*

Molly McNutt *mrm2234*

<https://goggle.columbia.lol/>

The Wikipedia Goggle

The **Wikipedia Goggle** is a search engine for the English Wikipedia, using a trigram index and Google's original PageRank, implemented in modern C++.



The Wikipedia Goggle

Results returned in 0.108 s.

C++

... redirect cxx pp move vandalism small yes short description general purpose programming language use dmy dates date janu... [Read more...](#)

C++11

... short description edition of the c programming language standard distinguish c c standard revision multiple issues cond... [Read more...](#)

C++ Standard Library

... use dmy dates date december c standard library to edit this template go to template c standard library in the c program... [Read more...](#)

Microsoft Visual C++

... short description integrated development environment product by microsoft infobox software name visual c logo visual st... [Read more...](#)

Intel C++ Compiler

... short description compiler infobox software name intel oneapi compute acceleration oneapi dpc c compiler other names cs... [Read more...](#)

C++20

... short description edition of the c programming language standard multiple issues lead too short date december prose dat... [Read more...](#)

Boost (C++ libraries)

... short description c libraries infobox software title name boost logo boost png image name is enough logo size px logo a... [Read more...](#)



The Wikipedia Goggle

Results returned in 0.005 s.

Finland

... short description country in northern europe about the republic of finland pp vandalism small yes pp pc small yes pp mo... [Read more...](#)

Grand Duchy of Finland

... short description predecessor state of modern finland infobox former country native name smalldiv ubl native name fi su... [Read more...](#)

Sub-regions of Finland

... politics of finland subdivisions of finland finland is divided into sub regional units ref cite web title luokitus sub ... [Read more...](#)

Parliament of Finland

... short description supreme legislature of finland use dmy dates date january infobox legislature name parliament of finl... [Read more...](#)

Gulf of Finland

... short description arm of the baltic sea use dmy dates date january infobox body of water name gulf of finland image cap... [Read more...](#)

Finland national football team

... short description men s national association football team representing finland about the men s team the women s team f... [Read more...](#)

Lapland (Finland)

... for the province in sweden lapland sweden for the s pmi s pmi short description region of finland for the former munici... [Read more...](#)



The Wikipedia Goggle

Results returned in 0.007 s.

Wikipedia

... short description free multilingual online encyclopedia about the online encyclopedia wikipedia s home page main page t... [Read more...](#)

German Wikipedia

... short description german language edition of wikipedia use dmy dates date august infobox website name wiki favicon germ... [Read more...](#)

French Wikipedia

... short description french language edition of wikipedia use dmy dates date august expand french wikip dia en fran als da... [Read more...](#)

English Wikipedia

... short description english language edition of wikipedia redirect en wikipedia org the main page main page searching on ... [Read more...](#)

Spanish Wikipedia

... short description spanish language edition of wikipedia more citations needed date april infobox website name wiki favicon... [Read more...](#)

Basics of a search engine

Offline:

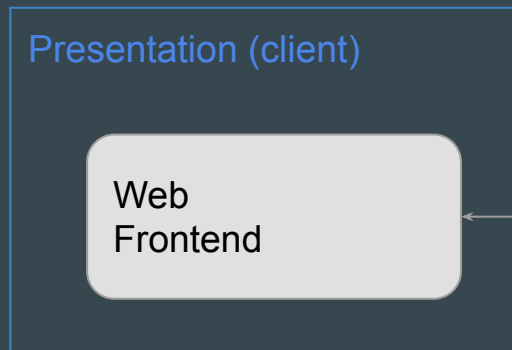
- 1. Indexing
 - Pre-process data into efficient data structures for querying and ranking

Online:

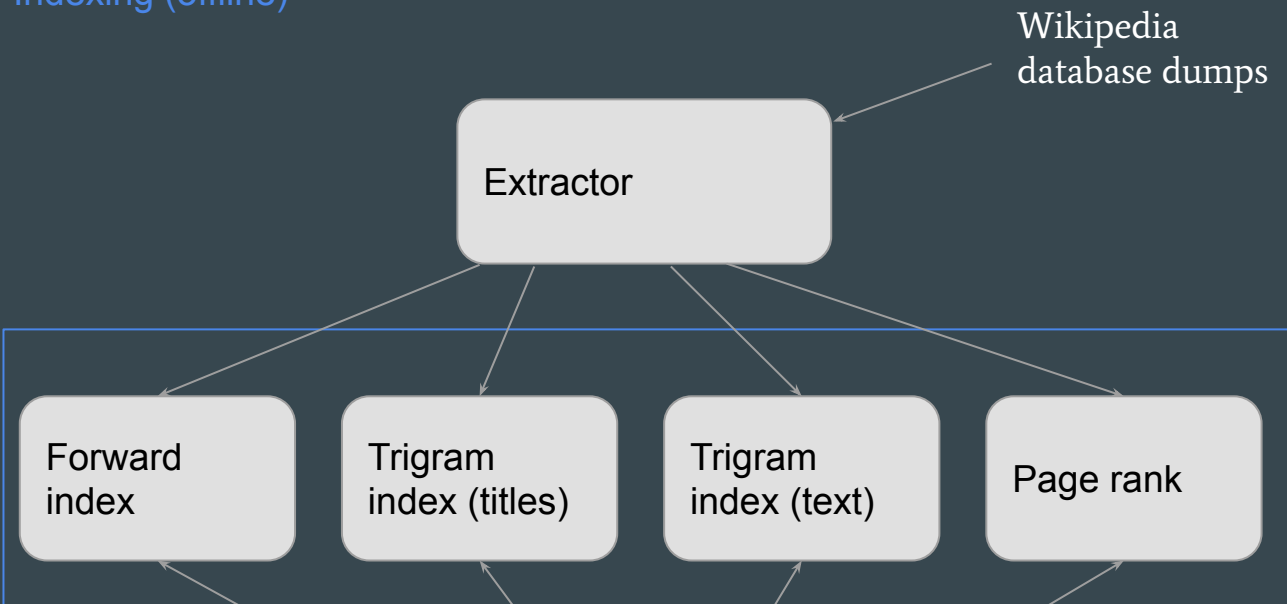
- 2. Information retrieval
 - Retrieve all documents matching a query
- 3. Ranking
 - Rank retrieved documents depending on query

Now do it in < 100 ms! - You need good data structures!

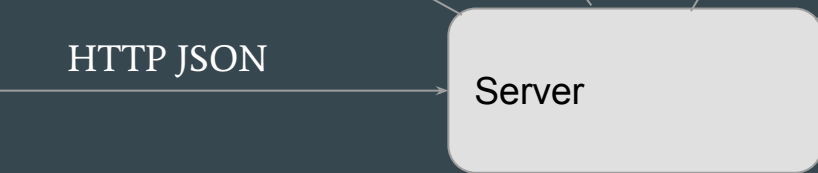
Architecture overview



Indexing (offline)

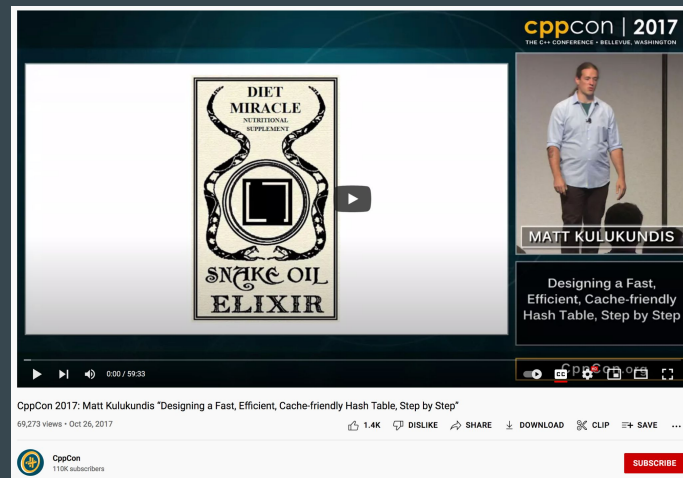


Querying (online)



Dependencies

- Googly stuff:
 - Bazel - build system
 - Abseil - hash maps, flags, etc
 - googlelog + googletest + googlebench
 - LevelDB - persistent key-value storage
 - Protocol buffers - structured data serialization (cf. flats by Bjarne)
- Non-G stuff
 - rapidxml by Marcin Kalicinski: <http://rapidxml.sourceforge.net/>
 - bzip2 by Julian Seward: <https://sourceware.org/bzip2/>
 - httplib by Yuji Hirose: <https://github.com/yhirose/cpp-httplib>
 - JSON escape code from @vog (SE): <https://stackoverflow.com/a/33799784> :)



Dump extraction & pre-processing

2022-04-21 11:45:28 **done** Recombine multiple bz2 streams

[enwiki-20220420-pages-articles-multistream.xml.bz2](#) 19.3 GB

[enwiki-20220420-pages-articles-multistream-index.txt.bz2](#) 230.0 MB

- Input: ~20 GB bzip2 “multistream” XML + “index.txt” file
 - Output: cleaned up documents
-
1. Extract chunk boundaries from “index.txt” file
 2. Read each chunk, decompress (bzip2)
 3. Parse XML (rapidxml)
 4. Extract wiki id, title, text, links (set)
 5. Strip text of extra markup, etc
 6. Ignore special pages (talk, templates, categories, etc)

Indexing as-we-go overview

(I'll talk about each individually)

- Trigram index for titles -> id
- Trigram index for text body -> id
- Forward index for id -> doc
- Some “magic” page goodness measurements (for a basis of PageRank)
- Hash map of target -> list of backlinks

Trigram indexes: principles

- **Reverse index** for text \rightarrow id
- Split text into trigrams (triplets of characters)
- a-z + space gives $27^3 \sim 20k$ trigrams
- For each trigram, store list of ids

To query:

- Get trigrams for query
- Intersect list of ids of these trigrams
- Lossy: find query in result

Implemented as

array of trigram_id \rightarrow std::vector<doc_id>

Indexing

hello world

↓
Trigrams

"_ _h", "_he", "hel",
 "ell", "llo", "lo_",
 "o_ _", "o_w", "_w",
 "_wo", "wor", "orl",
 "rld", "ld_", "d_ _"

Querying

hello \rightarrow "_ _h", "_he",
 "hel", "ell",
 "llo", "lo_",
 "o_ _"

Now find all docs that contain **all**
 of these trigrams

Forward index (DocIndex)

- Store document by id for avg $O(1)$ lookup
- Earlier a hash map of id \rightarrow Document
- Later moved to rely only on LevelDB (next slide)

Data persistence in LevelDB as Protocol Buffers

- LevelDB: a fast on-disk key-value store
 - i.e. bytes key -> bytes value
 - Implemented as log-structured merge tree (LSM tree)
 - We use it as a giant on-disk hash map :)
- DocIndex: docs/{id}
- Titles: trgm/titles/{trigram_ix}
- Text body: trgm/text/{trigram_ix}
- Pagerank: pr/pagerank

```
message Doc {  
  uint32 id = 1;  
  uint32 wiki_id = 5;  
  string title = 2;  
  string text = 3;  
  repeated string links = 4;  
}
```

```
message TrigramVec {  
  repeated uint32 docs = 1;  
}
```

```
message PagerankVec {  
  repeated float prs = 1;  
}
```

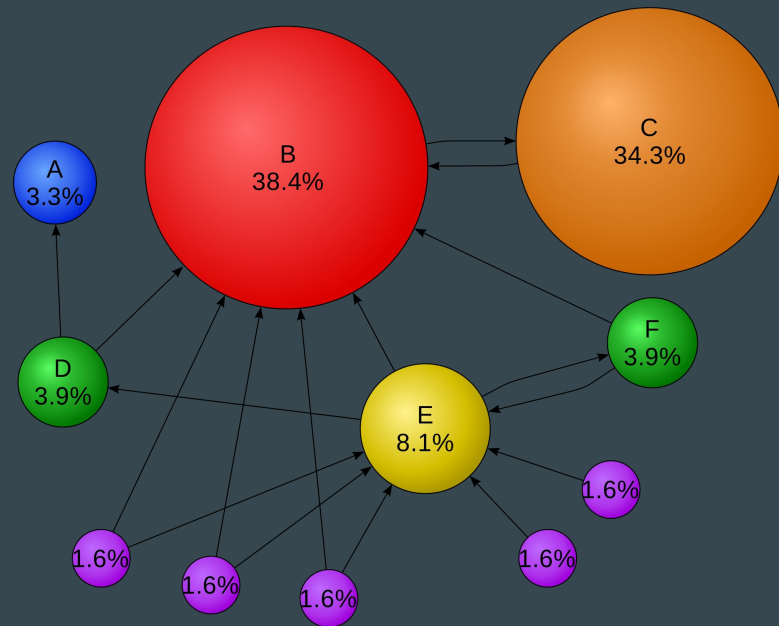
So we don't need to re-index each startup.

PageRank: principles

Random (web) surfer model:

1. A user starts from a random page
2. Randomly clicks on a link on that page
3. With probability $1-d$, stop and go back to 1.

PageRank is the steady-state prob. distribution of the page that the surfer is on. (Cf. Markov chains)



$$\mathbf{p} = (1 - d)\mathbf{g} + d\mathbf{B}\mathbf{p}$$

- d is the dampening factor
- \mathbf{g} is the “goodness” vector
- \mathbf{p} is the page rank vector
- \mathbf{B} is the transition matrix: $B_{ij} = 1/N_i$ if page i links to page j , and N_i is the number of links on page i

$$\mathbf{p} = ((1 - d)\mathbf{I} + d\mathbf{B})\mathbf{p}$$

$$p_i = (1 - d)g_i + d \sum_{j \in B_i} \frac{p_j}{N_j}$$

PageRank: implementation

- Generate hash map of target -> backlinks
 - `absl::flat_hash_map<std::string, std::vector<std::string>>`
 - Need to go through and switch page titles to page ids to get:
 - `absl::flat_hash_map<int, std::vector<int>>`
- Power iteration using single precision floats (~40 MB)
- Page goodness:
 - Try to weed out “List of ...” pages
 - Surprising number of lists of dates, e.g. “December 30”
 - Try to avoid “stub” pages, favor some length
 - Basically comes down to page length + number of links and their ratios and counts

Surprisingly fast: ~2 min on full 10 million Wiki pages to $< 1e-7$ error!

December 30	
From Wikipedia, the free encyclopedia	
December 30 is the 364th day of the year (365th in leap years) in the Gregorian calendar; one day remains until the end of the year.	
Contents [hide]	
1 Events	
1.1 Pre-1600	
1.2 1601–1900	
1.3 1901–present	
2 Births	
2.1 Pre-1600	
<ul style="list-style-type: none"> 1947 – Jeff Lynne, English singer-songwriter, guitarist, and producer 1947 – Steve Mix, American basketball player and coach 1948 – Jed Johnson, American interior designer and director (d. 1996) 1949 – David Bedford, English runner 1949 – Jerry Coyne, American biologist and author^{[17][18]} 1949 – Jim Flaherty, Canadian lawyer and politician, 37th Canadian Minister of Finance (d. 2016) 1950 – Timothy Mo, Chinese-English author 1950 – Lewis Shiner, American journalist and author 1950 – Bjarne Stroustrup, Danish computer scientist, created the C++ programming language 1950 – Matti Väinö, Finnish runner 1951 – Doug Allder, English footballer and coach 1951 – Chris Jasper, American musician, singer-songwriter, and producer 1951 – Nick Rose, English runner 1952 – June Anderson, American soprano and actress 1953 – Daniel T. Barry, American engineer and astronaut 	

Basic querying

1. Split query into trigrams
 2. Compute intersection of all documents that contain all trigrams
 3. Filter all returned documents (retrieved from DocIndex) using `string.find(query)`
 4. Continue searching until we've found `page_length` documents
- We actually first search the title index, then search full body
 - Indexes are sorted by page rank!

Optimization: setup

- Running on AMD Ryzen 7 5800X (32 MB L3 cache)
 - 96 GB DDR4 RAM to build indexes! (Building peaks at ~60 GB RAM)
 - ~45 GB disk for persistence, ~30 GB RAM once indexes built
-
- LLVM Clang 10.0.0-4ubuntu1
 - Ubuntu 20.04.4 LTS
-
- Indexing: 50 min for full Wiki (~10 million pages)
 - Page rank computation: ~2 min, to $< 1e-7$ error
 - Search: depends on query

Optimization: the query planner

- Initially:
 - sorted docs
 - $O(n^2)$ intersection, ~2.6 min!
- Binary search:
 - $O(n \log(n))$, ~1.6 sec (100x speedup)
- “Early break”: $< \text{pl docs}$, $> 80\%$ docs
- “Smart query planner”, ~0.45 sec (350x speedup)
 - For large lists: sequential strategy
 - For small list with large one: binary search strategy
- Separated text body from metadata
- Last bit parallelized

Searching for “hello world” in full Wikipedia corpus

```

1  std::vector<uint32_t> TrigramIndex::FindPossibleDocuments(
2      const std::string_view& query,
3      const std::unique_ptr<std::vector<float>>& pagerank,
4      std::function<bool(uint32_t)> check_doc, size_t page_size) {
5      auto trigrams = split_into_trigrams(query);
6      std::sort(trigrams.begin(), trigrams.end(), [&](auto a, auto b) {
7          return GetContainerAt(a).size() < GetContainerAt(b).size();
8      });
9      container_type remaining_docs{...};
10     for (auto& ix : trigrams) {
11         container_type& docs = GetContainerAt(ix);
12         if (docs.size() > 0.8 * pagerank->size()) {
13             LOG(INFO) << "More than 80%% docs for this trigram, breaking"; break;
14         }
15         if (remaining_docs.size() < page_size) {
16             LOG(INFO) << "Less than page size docs, breaking"; break;
17         }
18         if (remaining_docs.size() > 1000 &&
19             remaining_docs.size() > 0.05 * docs.size()) {
20             LOG(INFO) << "Using sequential strategy";
21             container_type intersection{};
22             std::set_intersection(docs.begin(), docs.end(), remaining_docs.begin(),
23                                 remaining_docs.end(),
24                                 std::back_inserter(intersection), doc_order);
25             remaining_docs = std::move(intersection);
26         } else {
27             LOG(INFO) << "Using binary search strategy";
28             auto it = remaining_docs.begin();
29             while (it != remaining_docs.end()) {
30                 if (!std::binary_search(docs.begin(), docs.end(), *it, doc_order)) {
31                     remaining_docs.erase(it);
32                 } else {
33                     ++it;
34                 }
35             }
36         }
37     }
38     std::vector<uint32_t> matches{};
39
40     for (auto&& doc_id : remaining_docs) {
41         if (check_doc(doc_id)) {
42             matches.push_back(doc_id);
43         }
44         if (matches.size() >= page_size) break;
45     }
46
47     return matches;
48 }

```

Optimization: the query planner

```

I20220430 08:29:18.675000 2385000 trigram.cc:220] Had to check 41 docs before finding 1 in 0 ms
I20220430 08:29:18.675001 2385000 trigram.cc:102] Have trigrams:
I20220430 08:29:18.675008 2385000 trigram.cc:104] Have trigram: lo , size: 1140260
I20220430 08:29:18.675096 2385000 trigram.cc:104] Have trigram: rld, size: 1909386
I20220430 08:29:18.675101 2385000 trigram.cc:104] Have trigram: orl, size: 2135213
I20220430 08:29:18.675107 2385000 trigram.cc:104] Have trigram: hel, size: 2212559
I20220430 08:29:18.675114 2385000 trigram.cc:104] Have trigram: o w, size: 2346584
I20220430 08:29:18.675122 2385000 trigram.cc:104] Have trigram: llo, size: 2511270
I20220430 08:29:18.675127 2385000 trigram.cc:104] Have trigram: ell, size: 3421354
I20220430 08:29:18.675133 2385000 trigram.cc:104] Have trigram: wor, size: 3811003
I20220430 08:29:18.675139 2385000 trigram.cc:104] Have trigram: ld , size: 3932145
I20220430 08:29:18.675145 2385000 trigram.cc:104] Have trigram: wo, size: 4183474
I20220430 08:29:18.675153 2385000 trigram.cc:104] Have trigram: he, size: 4539906
I20220430 08:29:18.675158 2385000 trigram.cc:104] Have trigram: h, size: 6196546
I20220430 08:29:18.675164 2385000 trigram.cc:104] Have trigram: w, size: 6270405
I20220430 08:29:18.675170 2385000 trigram.cc:104] Have trigram: o , size: 6332643
I20220430 08:29:18.675177 2385000 trigram.cc:104] Have trigram: d , size: 6402039
I20220430 08:29:18.675182 2385000 trigram.cc:104] Have trigram: , size: 6487373
I20220430 08:29:18.675648 2385000 trigram.cc:167] After trigram lo , have 1140260 docs left, took 0 ms
I20220430 08:29:18.675658 2385000 trigram.cc:136] Using sequential strategy
I20220430 08:29:18.727077 2385000 trigram.cc:167] After trigram rld, have 503364 docs left, took 51 ms
I20220430 08:29:18.727115 2385000 trigram.cc:136] Using sequential strategy
I20220430 08:29:18.754030 2385000 trigram.cc:167] After trigram orl, have 502055 docs left, took 26 ms
I20220430 08:29:18.754060 2385000 trigram.cc:136] Using sequential strategy
I20220430 08:29:18.781070 2385000 trigram.cc:167] After trigram hel, have 350714 docs left, took 27 ms
I20220430 08:29:18.781106 2385000 trigram.cc:136] Using sequential strategy
I20220430 08:29:18.812491 2385000 trigram.cc:167] After trigram o w, have 282096 docs left, took 31 ms
I20220430 08:29:18.812526 2385000 trigram.cc:136] Using sequential strategy
I20220430 08:29:18.838678 2385000 trigram.cc:167] After trigram llo, have 252016 docs left, took 26 ms
I20220430 08:29:18.838707 2385000 trigram.cc:136] Using sequential strategy
I20220430 08:29:18.869164 2385000 trigram.cc:167] After trigram ell, have 245368 docs left, took 30 ms
I20220430 08:29:18.869196 2385000 trigram.cc:136] Using sequential strategy
I20220430 08:29:18.901517 2385000 trigram.cc:167] After trigram wor, have 245346 docs left, took 32 ms
I20220430 08:29:18.901551 2385000 trigram.cc:136] Using sequential strategy
I20220430 08:29:18.935571 2385000 trigram.cc:167] After trigram ld , have 244661 docs left, took 34 ms
I20220430 08:29:18.935601 2385000 trigram.cc:136] Using sequential strategy
I20220430 08:29:18.978783 2385000 trigram.cc:167] After trigram wo, have 244540 docs left, took 43 ms
I20220430 08:29:18.978821 2385000 trigram.cc:136] Using sequential strategy
I20220430 08:29:19.019239 2385000 trigram.cc:167] After trigram he, have 243699 docs left, took 40 ms
I20220430 08:29:19.019270 2385000 trigram.cc:127] More than 80% docs for this trigram, breaking
I20220430 08:29:19.019274 2385000 trigram.cc:175] Matched 243699 docs in 344 ms
I20220430 08:29:19.019296 2385000 trigram.cc:192] About to check docs with 32 threads
I20220430 08:29:19.137099 2385000 trigram.cc:220] Had to check 1245 docs before finding 1 in 117 ms
I20220430 08:29:19.138181 2385000 goggle.cc:329] Searched for "hello world" with page size 10
I20220430 08:29:19.138200 2385000 goggle.cc:330] Took 465 ms

```

Searching for “hello world” in full Wikipedia corpus

```

1 std::vector<uint32_t> TrigramIndex::FindPossibleDocuments(
2     const std::string_view& query,
3     const std::unique_ptr<std::vector<float>>& pagerank,
4     std::function<bool(uint32_t)> check_doc, size_t page_size) {
5     auto trigrams = split_into_trigrams(query);
6     std::sort(trigrams.begin(), trigrams.end(), [&](auto a, auto b) {
7         return GetContainerAt(a).size() < GetContainerAt(b).size();
8     });
9     container_type remaining_docs {...};
10    for (auto& ix : trigrams) {
11        container_type& docs = GetContainerAt(ix);
12        if (docs.size() > 0.8 * pagerank->size()) {
13            LOG(INFO) << "More than 80%% docs for this trigram, breaking"; break;
14        }
15        if (remaining_docs.size() < page_size) {
16            LOG(INFO) << "Less than page size docs, breaking"; break;
17        }
18        if (remaining_docs.size() > 1000 &&
19            remaining_docs.size() > 0.05 * docs.size()) {
20            LOG(INFO) << "Using sequential strategy";
21            container_type intersection{};
22            std::set_intersection(docs.begin(), docs.end(), remaining_docs.begin(),
23                                remaining_docs.end(),
24                                std::back_inserter(intersection), doc_order);
25            remaining_docs = std::move(intersection);
26        } else {
27            LOG(INFO) << "Using binary search strategy";
28            auto it = remaining_docs.begin();
29            while (it != remaining_docs.end()) {
30                if (!std::binary_search(docs.begin(), docs.end(), *it, doc_order)) {
31                    remaining_docs.erase(it);
32                } else {
33                    ++it;
34                }
35            }
36        }
37    }
38    std::vector<uint32_t> matches{};
39
40    for (auto&& doc_id : remaining_docs) {
41        if (check_doc(doc_id)) {
42            matches.push_back(doc_id);
43        }
44        if (matches.size() >= page_size) break;
45    }
46
47    return matches;
48 }

```


Is it any good?

- When it works well
- When it doesn't work well
- Ranking trick
- Probably not good in general for full text search of large documents
- Great for title search



Anarchism

From Wikipedia, the free encyclopedia

For other uses, see [Anarchism \(disambiguation\)](#).

"Anarchist" and "Anarchists" redirect here. For other uses, see [Anarchist \(disambiguation\)](#).

Not to be confused with [Anarchy](#).

Anarchism is a [political philosophy](#) and [movement](#) that is sceptical of [authority](#) and rejects all involuntary, coercive forms of [hierarchy](#).^[1] Anarchism calls for the abolition of the [state](#), which it holds to be unnecessary, undesirable, and harmful. As a historically [left-wing](#) movement, placed on the farthest left of the [political spectrum](#), it is usually described alongside [communalism](#) and [libertarian Marxism](#) as the [libertarian](#) wing ([libertarian socialism](#)) of the [socialist movement](#), and has a strong historical association with [anti-capitalism](#) and [socialism](#).

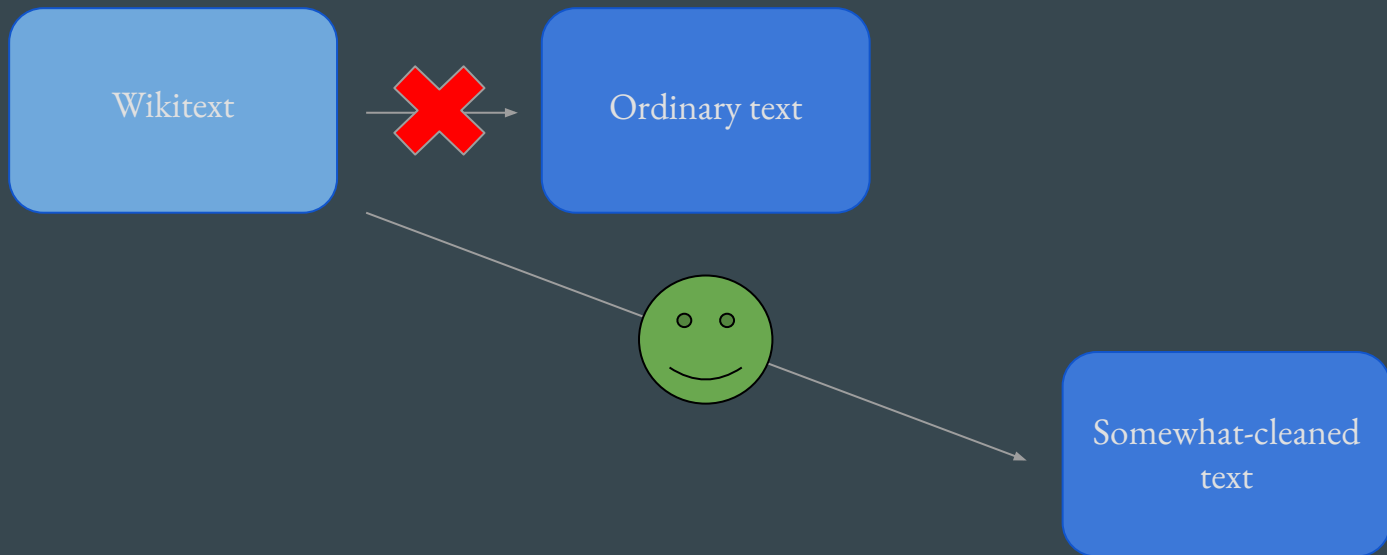
```
{{short description|Political philosophy and movement}}
{{other uses}}
{{redirect2|Anarchist|Anarchists|other uses|Anarchist (disambiguation)}}
{{distinguish|Anarchy}}
{{pp-semi-indef}}
{{good article}}
{{use British English|date=August 2021}}
{{use dmy dates|date=August 2021}}
{{anarchism sidebar}}
{{basic forms of government}}
'''Anarchism''' is a [[political philosophy]] and [[Political movement|movement]] that is sceptical of [[authority]] and rejects all involuntary, coercive forms of [[Social hierarchy|hierarchy]].{{sfn|Suissa|2019b|ps=: "...as many anarchists have stressed, it is not government as such that they find objectionable, but the hierarchical forms of government associated with the nation state."}} Anarchism calls for the abolition of the [[State (polity)|state]], which it holds to be unnecessary, undesirable, and harmful. As a historically [[left-wing]] movement, placed on the farthest left of the [[political spectrum]], it is usually described alongside [[Communalism (Bookchin)|communalism]] and [[libertarian Marxism]] as the [[libertarian]] wing ([[libertarian socialism]]) of the [[socialist movement]], and has a strong historical association with [[anti-capitalism]] and [[socialism]].
```



String cleaning measurements

We started out quite optimistic!

Then we realized just how slow regex – or linear scans – can be in practice.



String cleaning measurements

Measurements: runtime using Bazel run & using 1/1000th of total dump

Trade-off between tractable pre-processing time and quality of search results

- ParseXml w/ link extraction & removal of non-alphabetical characters:
 - 113281 ms averaged across 3 runs
 - A single for loop – $O(n)$
- ParseXml w/ link extraction & tolower():
 - 44886 ms averaged across 3 runs
 - A single for auto loop – $O(n)$
- ParseXml w/ link extraction & ad hoc regex:
 - 78k – 100k ms for a single document
 - Will not do.

Future improvements

- *Nothing. Goggle is perfect the way it is.*
- Remove extra markup instead of just ignoring it

V0.8: we extract and clean (by hand) 1k wiki pages and are able to do a basic trigram search on them, returning results in arbitrary order through a command line interface

V1.0: code to extract and clean a wikipedia dump, then search all wiki pages, returning results in arbitrary order through a web interface

v1.2: Allow for fuzzy search or basic regex search

References

- The Anatomy of a Large-Scale Hypertextual Web Search Engine, Sergey Brin, Larry Page, available at <http://infolab.stanford.edu/~backrub/google.html>
- The PageRank Citation Ranking: Bringing Order to the Web. Page, Lawrence and Brin, Sergey and Motwani, Rajeev and Winograd, Terry, available at <http://ilpubs.stanford.edu:8090/422/>
- PostgreSQL docs on Trigma

“Latency numbers every programmer should know”

memory:

sequential read (MT): 41 GB/s
sequential read (64B): 22 GB/s (2.7 ns)
sequential write (64B): 15 GB/s (4 ns)
random read (64B): 2.8 GB/s (20 ns)
random write (64B): 2.6 GB/s (22 ns)

disk:

sequential read (64KB): 7 GB/s (8 μ s)
sequential write (fsync) (8KB): 4 MB/s (1.8 ms)
sequential write (no fsync) (8KB): 1.4 GB/s (5 μ s)
sequential read (io uring) (2MB): 3.9 GB/s (506 μ s)
random read (8KB): 600 MB/s (13 μ s)

other:

mutexes per second: 17,531,612 (57 ns)
sha256 (64B): 150 MB/s (409 ns)
crc32 (64B): 3.6 GB/s (16 ns)
siphash (64B): 3.0 GB/s (19 ns)
tcp echo on localhost (64KB): 4.5 GB/s (13 μ s)