PISA Documentation

PISA

Mar 28, 2023

Contents:

Performant Indexes and Search for Academia

CHAPTER 1

Description

PISA is a text search engine able to run on large-scale collections of documents. It allows researchers to experiment with state-of-the-art techniques, allowing an ideal environment for rapid development.

Some features of PISA are listed below:

- Written in C++ for performance
- Indexing, parsing & sharding capabilities
- Many index compression methods supported
- Many query processing algorithms supported
- Implementation of document reordering
- Free and open-source with permissive license

Note: PISA is still in its **unstable release**, and no stability or backwards-compatibility is guaranteed with each new version. New features are constantly added, and contributions are welcome!

1.1 Getting Started

1.1.1 Building the code

1.1.1.1 Requirements

To compile PISA, you will need a compiler supporting at least the C++17 standard. Our continuous integration pipeline compiles PISA and runs tests in the following configurations:

- Linux:
  - GCC, versions: 9, 10, 11, 12
  - Clang 11

- MaxOS:
  - XCode 13.2

Supporting Windows is planned but is currently not being actively worked on, mostly due to a combination of man-hour shortage, prioritization, and no core contributors working on Windows at the moment. If you want to help us set up a Github workflow for Windows and work out some issues with compilation, let us know on our Slack channel.

1.1.1.2 Dependencies

Most build dependencies are managed automatically with CMake and git submodules. However, several dependencies still need to be manually provided:

- `CMake >= 3.0`
- `autoconf`, `automake`, `libtool`, and `m4` (for building `gumbo-parser`)
- OpenMP (optional)

1.1.1.3 Building

The following steps explain how to build PISA. First, you need the code checked out from Github. (Alternatively, you can download the tarball and unpack it on your local machine.)

```
$ git clone https://github.com/pisa-engine/pisa.git
$ cd pisa
```

Then create a build environment.

```
$ mkdir build
$ cd build
```

Finally, configure with CMake and compile:

```
$ cmake ..
$ make
```

Build Types

There are two build types available:

- `Release` (default)
- `Debug`
- `RelWithDebInfo`
- `MinSizeRel`

Use `Debug` only for development, testing, and debugging. It is much slower at runtime.

Learn more from CMake documentation.

Build Systems

CMake supports configuring for different build systems. On Linux and Mac, the default is Makefiles, thus, the following two commands are equivalent:

```
$ cmake -G ..
$ cmake -G "Unix Makefiles" ..
```

Alternatively to Makefiles, you can configure the project to use Ninja instead:

```
$ cmake -G Ninja ..
$ ninja # instead of make
```

Other build systems should work in theory but are not tested.

1.1.2 Testing

You can run the unit and integration tests with:

```
$ ctest
```

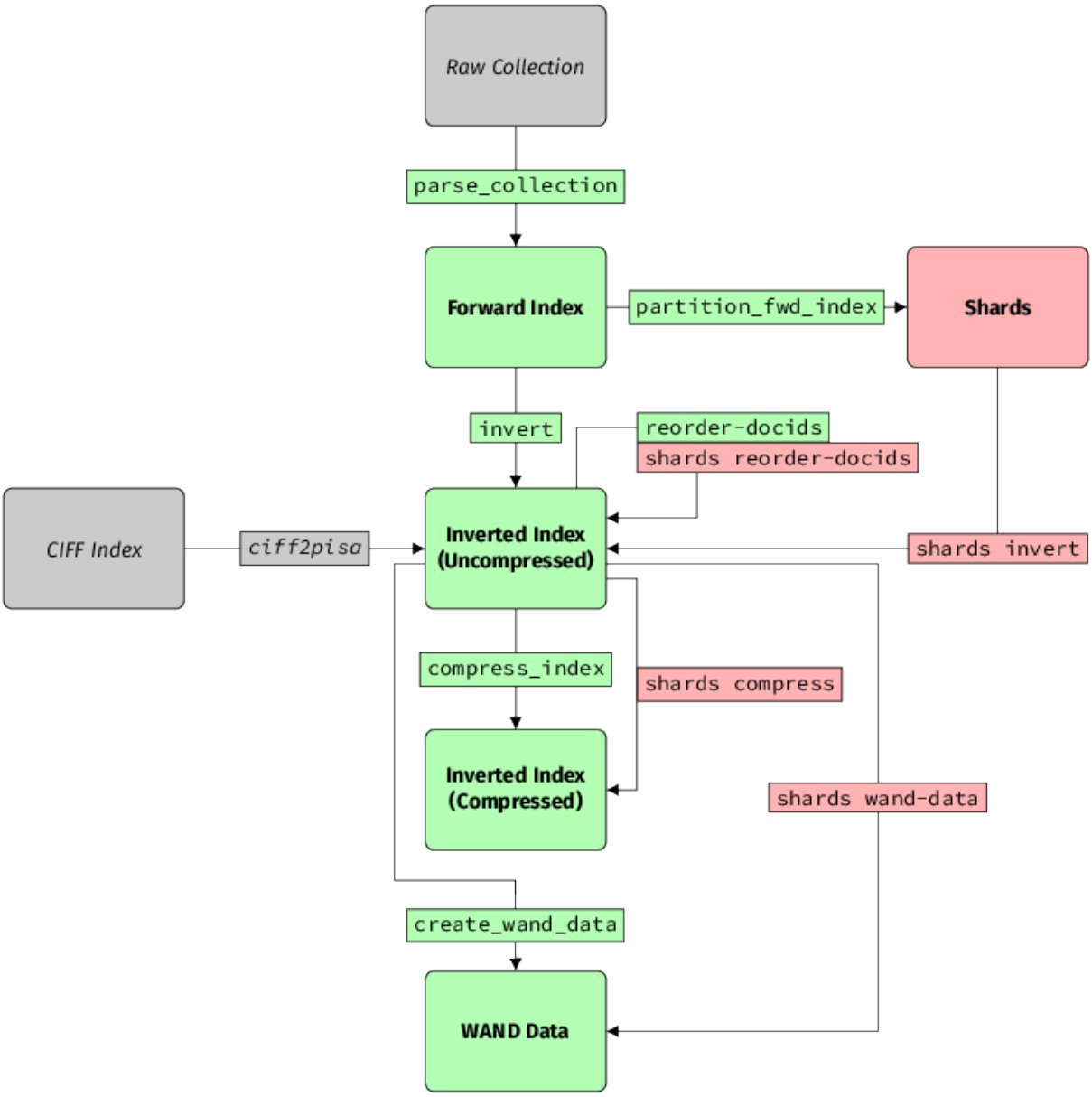
The directory `test/test_data` contains a small document collection used in the unit tests. The binary format of the collection is described in a following section. An example set of queries can also be found in `test/test_data/queries`.

1.1.3 PISA Regression Experiments

- Regressions for Disks 4 & 5 (Robust04)

1.2 Indexing Pipeline

This section is an overview of how to take a collection to a state in which it can be queried. This process is intentionally broken down into several steps, with a bunch of independent tools performing different tasks. This is because we want the flexibility of experimenting with each individual step without recomputing the entire pipeline.



1.2.1 External Resources

1.2.1.1 Raw Collection

The *raw collection* is a dataset containing the documents to index. A collection is encoded in one of the supported formats that stores a list of document contents along with some metadata, such as URL and title. The `parse_collection` tool takes a collection as an input and parses it to a forward index (see *Forward Index*). See Parsing for more details.

1.2.1.2 CIFF Index

This is an inverted index in the Common Index File Format. It can be converted to an uncompressed PISA index (more information below) with the *‘‘ciff2pisa‘* <https://github.com/pisa-engine/ciff>‘_ tool.

1.2.2 Forward Index

A *forward index* is the output of the `parse_collection` tool. It represents each document as a list of tokens (terms) in the order of their appearance. To learn more about parsing and the forward index format, see Parsing.

1.2.3 Inverted Index

An inverted index is the most fundamental structure in PISA. For each term in the collection, it contains a list of documents the term appears in. PISA distinguishes two types of inverted index.

1.2.3.1 Uncompressed / Binary Collection

The uncompressed index stores document IDs and frequencies as 4-byte integers. It is an intermediate format between forward index and compressed inverted index. It is obtained by running `invert` on a forward index. To learn more about inverting a forward index, see Inverting. Optionally, documents can be reordered with `reorder-docids` to obtain another instance of uncompressed inverted index with different assignment of IDs to documents. More on reordering can be found in Document Reordering.

1.2.3.2 Compressed

An uncompressed index is large and therefore before running queries, it must be compressed with one of many available encoding methods. It is this compressed index format that is directly used when issuing queries. See Compress Index to learn more.

1.2.4 WAND Data

This is a special metadata file containing additional statistics used during query processing. See Build additional data.

1.2.5 Shards

PISA supports partitioning a forward index into subsets called *shards*. Structures of all shards can be transformed in bulk using `shards` command line tool. To learn more, read Sharding.

1.3 Parsing

A *forward index* is a data structure that stores the term identifiers associated to every document. Conversely, an *inverted index* stores for each unique term the document identifiers where it appears (usually, associated to a numeric value used for ranking purposes such as the raw frequency of the term within the document).

The objective of the parsing process is to represent a given collection as a forward index. To parse a collection, use the `parse_collection` command:

```
parse_collection - parse collection and store as forward index.
Usage: parse_collection [OPTIONS] [SUBCOMMAND]

Options:
  -h,--help                   Print this help message and exit
  -L,--log-level TEXT:{critical,debug,err,info,off,trace,warn}=info
                              Log level
  -j,--threads UINT           Number of threads
  --tokenizer TEXT:{english,whitespace}=english
                              Tokenizer
  -H,--html UINT=0            Strip HTML
  -F,--token-filters TEXT:{krovetz,lowercase,porter2} ...
                              Token filters
  --stopwords TEXT            Path to file containing a list of stop words to filter␣
↪out
  --config TEXT               Configuration .ini file
  -o,--output TEXT REQUIRED   Forward index filename
  -b,--batch-size INT=100000  Number of documents to process in one thread
  -f,--format TEXT=plaintext  Input format

Subcommands:
  merge                       Merge previously produced batch files. When parsing␣
↪process was killed during merging, use this command to finish merging without␣
↪having to restart building batches.
```

For example:

```
$ mkdir -p path/to/forward
$ zcat ClueWeb09B/*/*.warc.gz |   # pass unzipped stream in WARC format
    parse_collection \
    -j 8                          # use up to 8 threads at a time
    -b 10000                      # one thread builds up to 10k documents in memory
    -f warc                       # use WARC
    -F lowercase porter2          # lowercase and stem every term (using the Porter2␣
↪algorithm)
    --html                        # strip HTML markup before extracting tokens
    -o path/to/forward/cw09b
```

In case you get the error `-bash: /bin/zcat: Argument list too long`, you can pass the unzipped stream using:

```
$ find ClueWeb09B -name '*.warc.gz' -exec zcat -q {} \;
```

The parsing process will write the following files:

- `cw09b`: forward index in binary format.
- `cw09b.terms`: a new-line-delimited list of sorted terms, where term having ID N is on line N, with N starting from 0.
- `cw09b.termlex`: a binary representation (lexicon) of the `.terms` file that is used to look up term identifiers at query time.
- `cw09b.documents`: a new-line-delimited list of document titles (e.g., TREC-IDs), where document having ID N is on line N, with N starting from 0.
- `cw09b.doclex`: a binary representation of the `.documents` file that is used to look up document identifiers at query time.

- `cw09b.urls`: a new-line-delimited list of URLs, where URL having ID N is on line N, with N starting from 0. Also, keep in mind that each ID corresponds with an ID of the `cw09b.documents` file.

1.3.1 Generating mapping files

Once the forward index has been generated, a binary document map and lexicon file will be automatically built. However, they can also be built using the `lexicon` utility by providing the new-line delimited file as input. The `lexicon` utility also allows efficient look-ups and dumping of these binary mapping files.

Examples of the `lexicon` command are shown below:

```
Build, print, or query lexicon
Usage: lexicon [OPTIONS] SUBCOMMAND

Options:
  -h,--help                   Print this help message and exit
  -L,--log-level TEXT:{critical,debug,err,info,off,trace,warn}=info
                              Log level
  --config TEXT               Configuration .ini file

Subcommands:
  build                       Build a lexicon
  lookup                      Retrieve the payload at index
  rlookup                     Retrieve the index of payload
  print                       Print elements line by line
```

For example, assume we have the following plaintext, new-line delimited file, `example.terms`:

```
aaa
bbb
def
zzz
```

We can generate a lexicon as follows:

```
./bin/lexicon build example.terms example.lex
```

You can dump the binary lexicon back to a plaintext representation:

```
./bin/lexicon print example.lex
```

It should output:

```
aaa
bbb
def
zzz
```

You can retrieve the term with a given identifier:

```
./bin/lexicon lookup example.lex 2
```

Which outputs:

```
def
```

Finally, you can retrieve the id of a given term:

```
./bin/lexicon rlookup example.lex def
```

It outputs:

```
2
```

NOTE: This requires the initial file to be lexicographically sorted, as `rlookup` uses binary search for reverse lookups.

1.3.2 Supported stemmers

- Porter2
- Krovetz

Both are English stemmers. Unfortunately, PISA does not have support for any other languages. Contributions are welcome.

1.3.3 Supported formats

The following raw collection formats are supported:

- `plaintext`: every line contains the document's title first, then any number of whitespaces, followed by the content delimited by a new line character.
- `trectext`: TREC newswire collections.
- `trecweb`: TREC web collections.
- `warc`: Web ARChive format as defined in the format specification.
- `wapo`: TREC Washington Post Corpus.

In case you want to parse a set of files where each one is a document (for example, the collection wiki-large), use the `files2trec.py` script to format it to TREC (take into account that each relative file path is used as the document ID). Once the file is generated, parse it with the `parse_collection` command specifying the `trectext` value for the `--format` option.

1.4 Inverting

Once the parsing phase is complete, use the `invert` command to turn a *forward index* into an *inverted index*:

```
Constructs an inverted index from a forward index.
Usage: invert [OPTIONS]

Options:
  -h,--help                   Print this help message and exit
  -i,--input TEXT REQUIRED    Forward index basename
  -o,--output TEXT REQUIRED   Output inverted index basename
  --term-count TEXT           Number of distinct terms in the forward index
  -j,--threads UINT           Number of threads
  --batch-size UINT=100000    Number of documents to process at a time
  -L,--log-level TEXT:{critical,debug,err,info,off,trace,warn}=info
                              Log level
  --config TEXT               Configuration .ini file
```

For example, assuming the existence of a forward index in the path `path/to/forward/cw09b`:

```
$ mkdir -p path/to/inverted
$ ./invert -i path/to/forward/cw09b -o path/to/inverted/cw09b --term-count `wc -w <␣
↪path/to/forward/cw09b.terms`
```

Note that the script requires as parameter the number of terms to be indexed, which is obtained by embedding the `wc -w < path/to/forward/cw09b.terms` instruction.

1.4.1 Inverted index format

A *binary sequence* is a sequence of integers prefixed by its length, where both the sequence integers and the length are written as 32-bit little-endian unsigned integers. An *inverted index* consists of 3 files, `<basename>.docs`, `<basename>.freqs`, `<basename>.sizes`:

- `<basename>.docs` starts with a singleton binary sequence where its only integer is the number of documents in the collection. It is then followed by one binary sequence for each posting list, in order of term-ids. Each posting list contains the sequence of document-ids containing the term.
- `<basename>.freqs` is composed of a one binary sequence per posting list, where each sequence contains the occurrence counts of the postings, aligned with the previous file (note however that this file does not have an additional singleton list at its beginning).
- `<basename>.sizes` is composed of a single binary sequence whose length is the same as the number of documents in the collection, and the i-th element of the sequence is the size (number of terms) of the i-th document.

1.4.1.1 Reading the inverted index using Python

Here is an example of a Python script reading the uncompressed inverted index format:

```
import os
import numpy as np

class InvertedIndex:
    def __init__(self, index_name):
        index_dir = os.path.join(index_name)
        self.docs = np.memmap(index_name + ".docs", dtype=np.uint32,
              mode='r')
        self.freqs = np.memmap(index_name + ".freqs", dtype=np.uint32,
              mode='r')

    def __iter__(self):
        i = 2
        while i < len(self.docs):
            size = self.docs[i]
            yield (self.docs[i+1:size+i+1], self.freqs[i-1:size+i-1])
            i += size+1

    def __next__(self):
        return self

for i, (docs, freqs) in enumerate(InvertedIndex("cw09b")):
    print(i, docs, freqs)
```

1.5 Sharding

We support partitioning a collection into a number of smaller subsets called *shards*. Right now, only a forward index can be partitioned by running `partition_fwd_index` command. For convenience, we provide `shards` command that supports certain bulk operations on all shards.

1.5.1 Partitioning collection

We support two methods of partitioning: random, and by a defined mapping. For example, one can partition collection randomly:

```
$ partition_fwd_index \
    -j 8                            # use up to 8 threads at a time
    -i full_index_prefix \
    -o shard_prefix \
    -r 123                            # partition randomly into 123 shards
```

Alternatively, a set of files can be provided. Let's assume we have a folder `shard-titles` with a set of text files. Each file contains new-line-delimited document titles (e.g., TREC-IDs) for one partition. Then, one would call:

```
$ partition_fwd_index \
    -j 8                            # use up to 8 threads at a time
    -i full_index_prefix \
    -o shard_prefix \
    -s shard-titles/*
```

Note that the names of the files passed with `-s` will be ignored. Instead, each shard will be assigned a numerical ID from `0` to `N - 1` in order in which they are passed in the command line. Then, each resulting forward index will have appended `.ID` to its name prefix: `shard_prefix.000`, `shard_prefix.001`, and so on.

1.5.2 Working with shards

The `shards` tool allows to perform some index operations in bulk on all shards at once. At the moment, the following subcommands are supported:

- invert,
- compress,
- wand-data, and
- reorder-docids.

All input and output paths passed to the subcommands will be expanded for each individual shards by extending it with `.<shard-id>` (e.g., `.000`) or, if substring `{}` is present, then the shard number will be substituted there. For example:

```
shards reorder-docids --by-url \
    -c inv \
    -o inv.url \
    --documents fwd.{}.doclex \
    --reordered-documents fwd.url.{}.doclex
```

is equivalent to running the following command for every shard `XYZ`:

```
reorder-docids --by-url \
    -c inv.XYZ \
    -o inv.url.XYZ \
    --documents fwd.XYZ.doclex \
    --reordered-documents fwd.url.XYZ.doclex
```

1.6 Compress Index

1.6.1 Usage

To create an index use the command `compress_inverted_index`. The available index types are listed in `index_types.hpp`.

```
Compresses an inverted index
Usage: compress_inverted_index [OPTIONS]

Options:
  -h,--help                   Print this help message and exit
  -c,--collection TEXT REQUIRED
                              Forward index basename
  -o,--output TEXT REQUIRED   Output inverted index
  --check                     Check the correctness of the index
  -e,--encoding TEXT REQUIRED Index encoding
  -w,--wand TEXT Needs: --scorer
                              WAND data filename
  -s,--scorer TEXT Needs: --wand --quantize
                              Scorer function
  --bm25-k1 FLOAT Needs: --scorer
                              BM25 k1 parameter.
  --bm25-b FLOAT Needs: --scorer
                              BM25 b parameter.
  --pl2-c FLOAT Needs: --scorer
                              PL2 c parameter.
  --qld-mu FLOAT Needs: --scorer
                              QLD mu parameter.
  --quantize Needs: --scorer  Quantizes the scores
  -L,--log-level TEXT:{critical,debug,err,info,off,trace,warn}=info
                              Log level
  --config TEXT               Configuration .ini file
```

For example, to create an index using the optimal partitioning algorithm, using the test collection, execute the command:

```
$ ./bin/compress_inverted_index -t opt \
    -c ../test/test_data/test_collection \
    -o test_collection.index.opt \
    --check
```

where `test/test_data/test_collection` is the *basename* of the collection, that is the name without the `.{docs,freqs,sizes}` extensions, and `test_collection.index.opt` is the filename of the output index. `--check` will trigger a verification step to check the correctness of the index.

1.6.2 Compression Algorithms

1.6.2.1 Binary Interpolative Coding

Binary Interpolative Coding (BIC) directly encodes a monotonically increasing sequence. At each step of this recursive algorithm, the middle element m is encoded by a number $m\ \ l\ \ p$, where l is the lowest value and p is the position of m in the currently encoded sequence. Then we recursively encode the values to the left and right of m. BIC encodings are very space-efficient, particularly on clustered data; however, decoding is relatively slow.

To compress an index using BIC use the index type `block_interpolative`.

> Alistair Moffat, Lang Stuiver: Binary Interpolative Coding for Effective Index Compression. Inf. Retr. 3(1): 25-47 (2000)

1.6.2.2 Elias-Fano

Given a monotonically increasing integer sequence S of size n, such that (S_{n-1} < u), we can encode it in binary using (lceillog urceil) bits. Elias-Fano coding splits each number into two parts, a low part consisting of (l = lceillog frac{u}{n}rceil) right-most bits, and a high part consisting of the remaining (lceillog urceil - l) left-most bits. The low parts are explicitly written in binary for all numbers, in a single stream of bits. The high parts are compressed by writing, in negative-unary form, the gaps between the high parts of consecutive numbers.

To compress an index using Elias-Fano use the index type `ef`.

> Sebastiano Vigna. 2013. Quasi-succinct indices. In Proceedings of the sixth ACM international conference on Web search and data mining (WSDM '13). ACM, New York, NY, USA, 83-92.

1.6.2.3 MaskedVByte

> Jeff Plaisance, Nathan Kurz, Daniel Lemire, Vectorized VByte Decoding, International Symposium on Web Algorithms 2015, 2015.

1.6.2.4 OptPFD

> Hao Yan, Shuai Ding, and Torsten Suel. 2009. Inverted index compression and query processing with optimized document ordering. In Proceedings of the 18th international conference on World wide web (WWW '09). ACM, New York, NY, USA, 401-410. DOI: https://doi.org/10.1145/1526709.1526764

1.6.2.5 Partitioned Elias Fano

> Giuseppe Ottaviano and Rossano Venturini. 2014. Partitioned Elias-Fano indexes. In Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval (SIGIR '14). ACM, New York, NY, USA, 273-282. DOI: https://doi.org/10.1145/2600428.2609615

1.6.2.6 QMX

Quantities, Multipliers, and eXtractor (QMX) packs as many integers as possible into 128-bit words (Quantities) and stores the selectors (eXtractors) separately in a different stream. The selectors are compressed (Multipliers) with RLE (Run-Length Encoding).

To compress an index using QMX use the index type `block_qmx`.

Andrew Trotman. 2014. Compression, SIMD, and Postings Lists. In Proceedings of the 2014 Australasian Document Computing Symposium (ADCS '14), J. Shane Culpepper, Laurence Park, and Guido Zuccon (Eds.). ACM, New York, NY, USA, Pages 50, 8 pages. DOI: https://doi.org/10.1145/2682862.2682870

1.6.2.7 SIMD-BP128

Daniel Lemire, Leonid Boytsov: Decoding billions of integers per second through vectorization. Softw., Pract. Exper. 45(1): 1-29 (2015)

1.6.2.8 Simple8b

Vo Ngoc Anh, Alistair Moffat: Index compression using 64-bit words. Softw., Pract. Exper. 40(2): 131-147 (2010)

1.6.2.9 Simple16

Jiangong Zhang, Xiaohui Long, and Torsten Suel. 2008. Performance of compressed inverted list caching in search engines. In Proceedings of the 17th international conference on World Wide Web (WWW '08). ACM, New York, NY, USA, 387-396. DOI: https://doi.org/10.1145/1367497.1367550

1.6.2.10 StreamVByte

Daniel Lemire, Nathan Kurz, Christoph Rupp: Stream VByte: Faster byte-oriented integer compression. Inf. Process. Lett. 130: 1-6 (2018). DOI: https://doi.org/10.1016/j.ipl.2017.09.011

1.6.2.11 Varint-G8IU

Alexander A. Stepanov, Anil R. Gangolli, Daniel E. Rose, Ryan J. Ernst, and Paramjit S. Oberoi. 2011. SIMD-based decoding of posting lists. In Proceedings of the 20th ACM international conference on Information and knowledge management (CIKM '11), Bettina Berendt, Arjen de Vries, Wenfei Fan, Craig Macdonald, Iadh Ounis, and Ian Ruthven (Eds.). ACM, New York, NY, USA, 317-326. DOI: https://doi.org/10.1145/2063576.2063627

1.6.2.12 VarintGB

Jeffrey Dean. 2009. Challenges in building large-scale information retrieval systems: invited talk. In Proceedings of the Second ACM International Conference on Web Search and Data Mining (WSDM '09), Ricardo Baeza-Yates, Paolo Boldi, Berthier Ribeiro-Neto, and B. Barla Cambazoglu (Eds.). ACM, New York, NY, USA, 1-1. DOI: http://dx.doi.org/10.1145/1498759.1498761

1.7 Query an index

1.7.1 Usage

```
Benchmarks queries on a given index.
Usage: queries [OPTIONS]

Options:
  -h,--help                   Print this help message and exit
  -e,--encoding TEXT REQUIRED Index encoding
  -i,--index TEXT REQUIRED    Inverted index filename
  -w,--wand TEXT              WAND data filename
  --compressed-wand Needs: --wand
                              Compressed WAND data file
  --tokenizer TEXT:{english,whitespace}=english
                              Tokenizer
  -H,--html UINT=0            Strip HTML
  -F,--token-filters TEXT:{krovetz,lowercase,porter2} ...
                              Token filters
  --stopwords TEXT            Path to file containing a list of stop words to filter␣
↪out
  -q,--queries TEXT           Path to file with queries
  --terms TEXT                Term lexicon
  --weighted                  Weights scores by query frequency
  -k INT REQUIRED             The number of top results to return
  -a,--algorithm TEXT REQUIRED
                              Query processing algorithm
  -s,--scorer TEXT REQUIRED   Scorer function
  --bm25-k1 FLOAT Needs: --scorer
                              BM25 k1 parameter.
  --bm25-b FLOAT Needs: --scorer
                              BM25 b parameter.
  --pl2-c FLOAT Needs: --scorer
                              PL2 c parameter.
  --qld-mu FLOAT Needs: --scorer
                              QLD mu parameter.
  -T,--thresholds TEXT        File containing query thresholds
  -L,--log-level TEXT:{critical,debug,err,info,off,trace,warn}=info
                              Log level
  --config TEXT               Configuration .ini file
  --quantized                 Quantized scores
  --extract                   Extract individual query times
  --safe Needs: --thresholds  Rerun if not enough results with pruning.
```

Now it is possible to query the index. The command `queries` treats each line of the standard input (or a file if `-q` is present) as a separate query. A query line contains a whitespace-delimited list of tokens. These tokens are either interpreted as terms (if `--terms` is defined, which will be used to resolve term IDs) or as term IDs (if `--terms` is not defined). Optionally, a query can contain query ID delimited by a colon:

```
      Q1:one two three
      ~~ ~~~~~~~~~~~~
query ID         terms
```

For example:

```
$ ./bin/queries \
    -e opt                          # index encoding
    -a and                          # retrieval algorithm
    -i test_collection.index.opt    # index path
    -w test_collection.wand         # metadata file
    -q ../test/test_data/queries    # query input file
```

This performs conjunctive queries (`and`). In place of `and` other operators can be used (see *Query algorithms*), and also multiple operators separated by colon (`and:or:wand`), which will run multiple passes, one per algorithm.

If the WAND file is compressed, append `--compressed-wand` flag.

1.7.2 Build additional data

To perform BM25 queries it is necessary to build an additional file containing the parameters needed to compute the score, such as the document lengths. The file can be built with the following command:

```
$ ./bin/create_wand_data \
    -c ../test/test_data/test_collection \
    -o test_collection.wand
```

If you want to compress the file append `--compress` at the end of the command. When using variable-sized blocks (for VBMW) via the `--variable-block` parameter, you can also specify lambda with the '-l

`or–lambda flags. The value of lambda impacts the mean size of the variable blocks that are output. See the VBMW paper (listed below) for more details. If using fixed-sized blocks, which is the default, you can supply the desired block size using the-b or–block-size ' arguments.

1.7.3 Query algorithms

Here is the list of the supported query processing algorithms.

1.7.3.1 AND

Unranked (`and`) or ranked (`ranked_and`) conjunction.

1.7.3.2 OR

Unranked (`or`) or ranked (`ranked_or`) union.

1.7.3.3 MaxScore

> Howard Turtle and James Flood. 1995. Query evaluation: strategies and optimizations. Inf. Process. Manage. 31, 6 (November 1995), 831-850. DOI=http://dx.doi.org/10.1016/0306-4573(95)00020-H

1.7.3.4 WAND

> Andrei Z. Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Zien. 2003. Efficient query evaluation using a two-level retrieval process. In Proceedings of the twelfth international conference on Information and knowledge management (CIKM '03). ACM, New York, NY, USA, 426-434. DOI: https://doi.org/10.1145/956863.956944

1.7.3.5 BlockMax WAND

Shuai Ding and Torsten Suel. 2011. Faster top-k document retrieval using block-max indexes. In Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval (SIGIR '11). ACM, New York, NY, USA, 993-1002. DOI=http://dx.doi.org/10.1145/2009916.2010048

1.7.3.6 BlockMax MaxScore

1.7.3.7 Variable BlockMax WAND

Antonio Mallia, Giuseppe Ottaviano, Elia Porciani, Nicola Tonellotto, and Rossano Venturini. 2017. Faster BlockMax WAND with Variable-sized Blocks. In Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '17). ACM, New York, NY, USA, 625-634. DOI: https://doi.org/10.1145/3077136.3080780

1.8 Document Reordering

PISA supports reassigning document IDs that were initially assigned in order of parsing. The point of doing it is usually to decrease the index size or speed up query processing. This part is done on an uncompressed inverted index. Depending on the method, you might also need access to some parts of the forward index. We support the following ways of reordering:

- random,
- by a feature (such as URL or document TREC ID),
- with a custom-defined mapping, and
- recursive graph bisection. All of the above are supported by a single command `reorder-docids`. Below, we explain each method and show some examples of running the command.

1.8.1 Reordering document lexicon

All methods can optionally take a path to a document lexicon and make a copy of it that reflects the produced reordering.

```
reorder-docids \
    --documents /path/to/original/doclex \
    --reordered-documents /path/to/reordered/doclex \
    ...
```

Typically, you will want to do that if you plan to evaluate queries, which will need access to a correct document lexicon.

NOTE: Because these options are common to all reordering methods, we ignore them below for brevity.

1.8.2 Random

Random document reordering, as the name suggests, randomly shuffles all document IDs. Additionally, it can take a random seed. Two executions of the command with the same seed will produce the same final ordering.

```
reorder-docids --random \
    --collection /path/to/inv \
    --output /path/to/inv.random \
    --seed 123456789 # optional
```

1.8.3 By feature (e.g., URL or TRECID)

An index can be reordered according to any single document feature, such as URL or TRECID, as long as it is stored in a text file line by line, where line `n` is the feature of document `n` in the original order.

In particular, our collection parsing command produces two such feature files:

- `*.documents`, which is typically a list of TRECIDs,
- `*.urls`, which is a list of document URLs.

To use either, you simply need to run:

```
reorder-docids \
    --collection /path/to/inv \
    --output /path/to/inv.random \
    --by-feature /path/to/feature/file
```

1.8.4 From custom mapping

You can also produce a mapping yourself and feed it to the command. Such mapping is a text file with two columns separated by a whitespace:

```
<original ID> <new ID>
```

Having that, reordering is as simple as running:

```
reorder-docids \
    --collection /path/to/inv \
    --output /path/to/inv.random \
    --from-mapping /path/to/custom/mapping
```

1.8.5 Recursive Graph Bisection

We provide an implementation of the *Recursive Graph Bisection* (aka *BP*) algorithm, which is currently the state-of-the-art for minimizing the compressed space used by an inverted index (or graph) through document reordering. The algorithm tries to minimize an objective function directly related to the number of bits needed to store a graph or an index using a delta-encoding scheme.

Learn more from the original paper:

> L. Dhulipala, I. Kabiljo, B. Karrer, G. Ottaviano, S. Pupyrev, and A. Shalita. Compressing graphs and indexes with recursive graph bisection. In Proc. SIGKDD, pages 1535–1544, 2016.

In PISA, you simply need to pass `--recursive-graph-bisection` option (or its alias `--bp`) to the `reorder-docids` command.

```
reorder-docids --bp \
    --collection /path/to/inv \
    --output /path/to/inv.random
```

Note that `--bp` allows for some additional options. For example, the algorithm constructs a forward index in memory, which is in a special format **separate from the PISA forward index** that you obtain from the `parse_collection` tool. You can instruct `reorder-docids` to store that intermediate structure (`--store-fwdidx`), as well as provide a previously constructed one (`--fwdidx`), which can be useful if you want to reuse it for several runs with different algorithm parameters. To see all available parameters, run `reorder-docids --help`.

1.9 Threshold estimation

Currently it is possible to perform threshold estimation tasks using the `kth_threshold` tool. The tool computes the k-highest impact score for each term of a query. Clearly, the top-k threshold of a query can be lower-bounded by the maximum of the k-th highest impact scores of the query terms.

In addition to the k-th highest score for each individual term, it is possible to use the k-th highest score for certain pairs and triples of terms.

To perform threshold estimation use the `kth_threshold` command:

```
A tool for performing threshold estimation using the k-highest impact score for each␣
↪term, pair or triple of a query. Pairs and triples are only used if provided with --
↪pairs and --triples respectively.
Usage: ./bin/kth_threshold [OPTIONS]

Options:
  -h,--help                   Print this help message and exit
  -e,--encoding TEXT REQUIRED Index encoding
  -i,--index TEXT REQUIRED    Inverted index filename
  -w,--wand TEXT REQUIRED     WAND data filename
  --compressed-wand Needs: --wand
                              Compressed WAND data file
  --tokenizer TEXT:{english,whitespace}=english
                              Tokenizer
  -H,--html UINT=0            Strip HTML
  -F,--token-filters TEXT:{krovetz,lowercase,porter2} ...
                              Token filters
  --stopwords TEXT            Path to file containing a list of stop words to filter␣
↪out
  -q,--queries TEXT           Path to file with queries
  --terms TEXT                Term lexicon
  --weighted                  Weights scores by query frequency
  -k INT REQUIRED             The number of top results to return
  -s,--scorer TEXT REQUIRED   Scorer function
  --bm25-k1 FLOAT Needs: --scorer
                              BM25 k1 parameter.
  --bm25-b FLOAT Needs: --scorer
                              BM25 b parameter.
  --pl2-c FLOAT Needs: --scorer
                              PL2 c parameter.
  --qld-mu FLOAT Needs: --scorer
                              QLD mu parameter.
  -L,--log-level TEXT:{critical,debug,err,info,off,trace,warn}=info
                              Log level
  --config TEXT               Configuration .ini file
  -p,--pairs TEXT Excludes: --all-pairs
                              A tab separated file containing all the cached term␣
↪pairs
  -t,--triples TEXT Excludes: --all-triples
```

(continues on next page)

(continued from previous page)

```
                              A tab separated file containing all the cached term␣
↪triples
  --all-pairs Excludes: --pairs
                              Consider all term pairs of a query
  --all-triples Excludes: --triples
                              Consider all term triples of a query
  --quantized                 Quantizes the scores
```

`--all-pairs` and `--all-triples` can be used if you want to consider all the pairs and triples terms of a query as being previously cached.