search-engine-indexer

search-engine-indexer is a tool for working with search term log files.

Dependencies

search-engine-indexer can be run either from a JAR file or from a Docker image.
We recommend running it from a JAR file for performance reasons.

JAR file dependencies

- Java

Docker image dependencies

- Docker

Usage

```
search-engine-indexer 0.2.0-SNAPSHOT

search-engine-indexer is a tool for working with search term log files

Usage
  java -jar search-engine-indexer-0.2.0-SNAPSHOT-standalone.jar [SUBCOMMAND] [OPTIONS]

Options
  -v, --version  Show version
  -h, --help     Show help

Subcommands
  generate
    --dictionary-file DICTIONARY_FILE
    --number-of-output-files NUMBER_OF_OUTPUT_FILES
    --human-bytes-to-write HUMAN_BYTES_TO_WRITE
    --output-directory OUTPUT_DIRECTORY

  process
    --input-directory INPUT_DIRECTORY
    --output-file OUTPUT_FILE
    --maximum-terms-in-memory MAXIMUM_TERMS_IN_MEMORY
```

JAR file

1. Download

   ```
   wget https://s3.amazonaws.com/search-engine-indexer/jars/search-engine-indexer-0.2.0-SN
   ```

2. Run

   ```
   java -jar search-engine-indexer-0.2.0-SNAPSHOT-standalone.jar --help
   ```

Docker image

Based on `openjdk:11.0.4-jre-slim` which is 204MB.

```
docker run mpereira/search-engine-indexer:0.2.0 --help
```

Implementation details

The search-engine-indexer has two main functionalities:

1. generating random unsorted search term log files
2. combining unsorted search term log files into a single alphabetically sorted search term log file

This section will focus on the latter.

The `search-engine-indexer.processor` namespace in `src/search_engine_indexer/processor.clj` has a `process-directory` function that does all the work. It takes as input an *input directory* that should contain search term log files and an *output file* path in which the alphabetically sorted search terms will be created. It also takes a *maximum terms in memory* parameter which sets an upper bound on the memory usage of the program. None of the program's in-memory objects can contain information about more than that amount of search terms.

The algorithm is:

```
For each input file
  For each line (search term) in the input file
     Increment in-memory sorted hash map entry for search term

     If in-memory sorted hash map has more search terms than maximum
         Flush in-memory sorted hash map to search term count files in disk

Aggregate in-disk flushed search term count files

For each aggregated search term count file, in alphabetical order
   Append the search term "count" times to the output file
```

First, it streams each input file sequentially (only one line will be in memory at a time) and accumulates counts in an in-memory sorted hash map. The sorted hash map keys are search terms and values are occurrence counts.

Example

```
{"pinakotek" 2
 "beer" 1}
```

The program keeps track of the total occurrence counts (in the case above it would be `3`) and flushes search term count files to disk when the in-memory sorted hash map is full. Flushing the above sorted hash map would result in an empty in-memory sorted hash map (`{}`) and the following two files on disk:

`pikakotek` contents:

```
pinakotek
2
```

`beer` contents:

```
beer
1
```

Assuming the in-memory sorted hash map is updated again:

```
{"pinakotek" 3
 "ludwig" 5}
```

Flushing it would result in the following disk on disk:

`pikakotek` contents:

```
pinakotek
2
3
```

`beer` contents:

```
beer
1
```

`ludwig` contents:

```
beer
5
```

- The `pikakotek` file got appended to with a `3`
- The `beer` file stayed the same since there were no `beer` occurrences in the sorted hash map
- The `ludwig` file was created with one occurrence count of `5`

After all intermediate per-search-term state files are written to disk, their counts are aggregated (summed) and written to disk again. The result of aggregating the files above would be:

`pikakotek` contents:

```
pinakotek
5
```

`beer` contents:

```
beer
1
```

`ludwig` contents:

```
beer
5
```

Disk reads and writes are all sequential and writes are all appends, which should yield high I/O throughput. Multiple intermediate state files are written to, which may cause disk seeks. This can be ameliorated by increasing the *maximum terms in memory* parameter, which will cause fewer intermediate state file writes.

Running a simulation

Even though you're free to clone the repository and build artifacts yourself, in this section we'll make use publicly available AWS S3 objects. The only requirements to follow through the steps are Java and UNIX utilities.

Even though there is a Docker image available for the search-engine-indexer, we'll run the JAR instead so as to not observe reduced IO performance due to doing operations inside a container.

Create a directory for the simulation

```
mkdir search-enginer-indexer-simulation
```

1. `cd` into it

   ```
   cd search-enginer-indexer-simulation
   ```

Download dictionary file

```
wget https://s3.amazonaws.com/search-engine-indexer/dictionaries/german_beer_brands.txt
```

1. Verify dictionary file size

   ```
   wc -l german_beer_brands.txt
   ```

   ```
   77 german_beer_brands.txt
   ```

2. Verify dictionary file contents

```
head german_beer_brands.txt
```

```
aktienbrauerei kaufbeuren
allgäuer brauerei
asgaard
augustiner-bräu
bayerische staatsbrauerei weihenstephan
berg brauerei
berliner pilsner
berliner weisse
bitburger brauerei
blue girl beer
```

Generate random search term log files

This command will create 10 unsorted search term log files in the `beer_brand_search_terms` directory. Their combined size will be around 500MiB and their contents will come from the `german_beer_brands.txt` dictionary.

```
java -jar search-engine-indexer-0.2.0-SNAPSHOT-standalone.jar \
     generate \
     --dictionary-file german_beer_brands.txt \
     --number-of-output-files 10 \
     --human-bytes-to-write 500MiB \
     --output-directory beer_brand_search_terms
```

```
Dictionary file:                                             german_beer_brands.txt
Number of unsorted search term log output files:             10
Human-readable number bytes to write across output files:    500MiB
Output directory for unsorted search term log output files:  beer_brand_search_terms

Read dictionary file with 77 terms
Created output directory '/Users/murilo/git/search-engine-indexer/search-enginer-indexer-sin
Writing 500MiB across 10 output files
Created '/Users/murilo/git/search-engine-indexer/search-enginer-indexer-simulation/beer_bran
Created '/Users/murilo/git/search-engine-indexer/search-enginer-indexer-simulation/beer_bran
Created '/Users/murilo/git/search-engine-indexer/search-enginer-indexer-simulation/beer_bran
Created '/Users/murilo/git/search-engine-indexer/search-enginer-indexer-simulation/beer_bran
Created '/Users/murilo/git/search-engine-indexer/search-enginer-indexer-simulation/beer_bran
Created '/Users/murilo/git/search-engine-indexer/search-enginer-indexer-simulation/beer_bran
Created '/Users/murilo/git/search-engine-indexer/search-enginer-indexer-simulation/beer_bran
Created '/Users/murilo/git/search-engine-indexer/search-enginer-indexer-simulation/beer_bran
Created '/Users/murilo/git/search-engine-indexer/search-enginer-indexer-simulation/beer_bran
Created '/Users/murilo/git/search-engine-indexer/search-enginer-indexer-simulation/beer_bran
524288119 bytes written in 9.52 seconds (55.06 MB/s)
```

Verify `generate` output: the `beer_brand_search_terms` directory

1. Size

   ```
   du -ah beer_brand_search_terms | sort -h
   ```

   ```
   52M beer_brand_search_terms/0.log
   52M beer_brand_search_terms/1.log
   52M beer_brand_search_terms/2.log
   52M beer_brand_search_terms/3.log
   52M beer_brand_search_terms/4.log
   52M beer_brand_search_terms/5.log
   52M beer_brand_search_terms/6.log
   52M beer_brand_search_terms/7.log
   52M beer_brand_search_terms/8.log
   52M beer_brand_search_terms/9.log
   514M    beer_brand_search_terms
   ```

   Looks like the combined size of all generated files is 514M as reported by `du`. Close enough!

2. `beer_brand_search_terms/0.log` contents

   ```
   head beer_brand_search_terms/0.log
   ```

   ```
   bitburger brauerei
   hofbräuhaus traunstein
   hasseröder
   störtebeker braumanufaktur
   einbecker brauerei
   allgäuer brauerei
   brauerei gold ochsen
   mecklenburgische brauerei lübz
   brauerei gold ochsen
   bayerische staatsbrauerei weihenstephan
   ```

   Looks random enough. Let's take a look at another generated file just to make sure it's really random.

3. `beer_brand_search_terms/1.log` contents

   ```
   head beer_brand_search_terms/1.log
   ```

   ```
   janssen & bechly brauerei
   paulaner brauerei
   löwenbräu brauerei
   veltins brauerei
   list of brewing companies in germany
   kronen
   janssen & bechly brauerei
   ```

```
hasseröder
st. erhard brauerei
privatbrauerei wittingen
```

Alright, looks good to me.

Process input directory with randomly generated search term log files (`beer_brand_search_terms`)

Now that we have a bunch of unsorted search term log files we can combine them into an alphabetically sorted search terms log file.

This command will read the unsorted search term log files in the `beer_brand_search_terms` directory and write their alphabetically sorted search terms to `beer_brand_search_terms.log` while loading at most `100000` (one hundred thousand) search terms (and even "search term"-derived data) in any of the program's in-memory objects.

`maximum-terms-in-memory` sets a hard upper bound on the memory usage of the program. Low values will result in having to write intermediate state to disk more often, but with the advantage of using less memory. High values will result in higher memory usage but less frequent disk writes.

The program will print a dot (`.`) every time it writes intermediate state to disk.

```
java -jar search-engine-indexer-0.2.0-SNAPSHOT-standalone.jar \
     process \
     --input-directory beer_brand_search_terms \
     --output-file beer_brand_search_terms.log \
     --maximum-terms-in-memory 100000
```

```
Input directory with unsorted search term log files:           beer_brand_search_terms
Output file to be created with sorted search terms:            beer_brand_search_terms.log
Maximum number of search terms that will be loaded in memory:  100000

Processing files in '/Users/murilo/git/search-engine-indexer/search-enginer-indexer-simulati
'/Users/murilo/git/search-engine-indexer/search-enginer-indexer-simulation/beer_brand_search
'/Users/murilo/git/search-engine-indexer/search-enginer-indexer-simulation/beer_brand_search
'/Users/murilo/git/search-engine-indexer/search-enginer-indexer-simulation/beer_brand_search
'/Users/murilo/git/search-engine-indexer/search-enginer-indexer-simulation/beer_brand_search
'/Users/murilo/git/search-engine-indexer/search-enginer-indexer-simulation/beer_brand_search
'/Users/murilo/git/search-engine-indexer/search-enginer-indexer-simulation/beer_brand_search
'/Users/murilo/git/search-engine-indexer/search-enginer-indexer-simulation/beer_brand_search
'/Users/murilo/git/search-engine-indexer/search-enginer-indexer-simulation/beer_brand_search
'/Users/murilo/git/search-engine-indexer/search-enginer-indexer-simulation/beer_brand_search
'/Users/murilo/git/search-engine-indexer/search-enginer-indexer-simulation/beer_brand_search
................................................................................
................................................................................
```

```
................................................................................
......................................
Term occurrences across all files in '/Users/murilo/git/search-engine-indexer/search-enginer
bitburger brauerei 361785
rhanerbräu 361396
brauerei kaiserdom 361197
kaiser bräu 361340
janssen & bechly brauerei 361436
allgäuer brauerei 361344
maisel brau bamberg 361833
warsteiner 361298
flensburger brauerei 361238
grafenwalder 361759
riegele 361593
stadtbrauerei spalt 361926
herrenhäuser brauerei 362955
cölner hofbräu früh 361246
staatliches hofbräuhaus in münchen 362137
köstritzer 361556
vitamalz 362119
diebels 362229
gaffel becker & co 362011
könig brauerei 361475
kulmbacher brauerei 363133
ganter brauerei 362291
koblenzer brauerei 361592
henninger brauerei 361975
radeberger brauerei 362361
fucking hell 361826
eck brauerei 362925
kronen 361306
jever brauerei 361465
dortmunder actien brauerei 361335
paulaner brauerei 362367
hacker-pschorr brauerei 362352
heinrich reissdorf 361952
augustiner-bräu 361084
pinkus müller 361765
rothaus 361821
fürstenberg brauerei 362942
hofbräuhaus traunstein 361885
hofbrauhaus arolsen 361563
brauerei gold ochsen 361317
krombacher brauerei 362392
asgaard 361576
oettinger brauerei 361047
```

```
berliner weisse 361244
privatbrauerei wittingen 362169
list of brewing companies in germany 362604
brauhaus am kreuzberg 362427
st. pauli girl 362812
wernesgrüner 360750
st. erhard brauerei 361367
brauerei zur malzmühle 362470
berliner pilsner 360501
berg brauerei 362180
dortmunder export 360860
veltins brauerei 361039
löwenbräu brauerei 361902
klosterbrauerei andechs 360792
gasthof herold 362595
bayerische staatsbrauerei weihenstephan 362417
könig ludwig schlossbrauerei 362034
bolten-brauerei 361136
störtebeker braumanufaktur 362091
hasseröder 361948
aktienbrauerei kaufbeuren 361275
kuchlbauer brauerei 361407
brauerei gebr. maisel 362642
blue girl beer 361557
freiberger brauhaus 362633
mecklenburgische brauerei lübz 362538
spaten-franziskaner-bräu 361510
herzoglich bayerisches brauhaus tegernsee 360992
licher brauerei 361457
holsten pils 362102
einbecker brauerei 361594
zötler brauerei 361620
erdinger 361485
g. schneider & sohn 361815
Wrote sorted search terms to '/Users/murilo/git/search-engine-indexer/search-enginer-indexer
533332638 bytes (27858110 search terms) processed in 30.02 seconds (17.76 MB/s)
```

We can see both the per-"search term" and total occurrence counts in the command output. The total number of search terms processed was 27858110.

Verify process output: `beer_brand_search_terms.log`

1. Size

   ```
   du -ah beer_brand_search_terms.log
   ```

```
   513M    beer_brand_search_terms.log
   ```

2. Content

   ```
   head beer_brand_search_terms.log
   ```

   ```
   aktienbrauerei kaufbeuren
   aktienbrauerei kaufbeuren
   aktienbrauerei kaufbeuren
   aktienbrauerei kaufbeuren
   aktienbrauerei kaufbeuren
   aktienbrauerei kaufbeuren
   aktienbrauerei kaufbeuren
   aktienbrauerei kaufbeuren
   aktienbrauerei kaufbeuren
   aktienbrauerei kaufbeuren
   ```

   This is good evidence that the file is actually alphabetically sorted.

3. Total search terms count

   ```
   cat beer_brand_search_terms.log | wc -l
   ```

   ```
   27858110
   ```

   `27858110` (~27.8 million) is the number of search terms shown in the end of the output for the "Process input directory" step, so this looks good.

4. Unique search terms count

   ```
   cat beer_brand_search_terms.log | uniq | wc -l
   ```

   ```
   77
   ```

   77 is the number of terms in the dictionary, so this is also looking good.

5. Unique search term counts

   ```
   uniq -c beer_brand_search_terms.log | head
   ```

   ```
   361275 aktienbrauerei kaufbeuren
   361344 allgäuer brauerei
   361576 asgaard
   361084 augustiner-bräu
   362417 bayerische staatsbrauerei weihenstephan
   362180 berg brauerei
   360501 berliner pilsner
   361244 berliner weisse
   361785 bitburger brauerei
   361557 blue girl beer
   ```

   It seems that the search terms were written in order, otherwise `uniq -c` wouldn't have worked. Also, comparing the output above with the output produced in the "Process input directory" step should demonstrate that search terms were written correctly.

That's it

In this simulation we generated random unsorted search term log files, combined them into an alphabetically sorted search term log file and verified that the outputs were correct. Feel free to run simulations with different parameters!

Just for fun, let's try a much smaller `maximum-terms-in-memory` value. Let's set it to `1000`. We'd expect reduced throughput given more frequent intermediate state writes to disk.

```
java -jar search-engine-indexer-0.2.0-SNAPSHOT-standalone.jar \
     process \
     --input-directory beer_brand_search_terms \
     --output-file beer_brand_search_terms.log \
     --maximum-terms-in-memory 1000 \
  | grep 'processed in'
```

```
533332638 bytes (27858110 search terms) processed in 233.20 seconds (2.29 MB/s)
```

We can see one order of magnitude less throughput (~17 MB/s -> ~2 MB/s) by decreasing `maximum-terms-in-memory` three orders of magnitude (100000 -> 1000).

Let's see what happens when setting it to a much higher value, `10000000` (10 million). Based on the total number of search terms we've seen above (~27.8 million) the in-memory buffer will be flushed to disk at most thrice.

```
java -jar search-engine-indexer-0.2.0-SNAPSHOT-standalone.jar \
     process \
     --input-directory beer_brand_search_terms \
     --output-file beer_brand_search_terms.log \
     --maximum-terms-in-memory 10000000 \
  | grep 'processed in'
```

```
533332638 bytes (27858110 search terms) processed in 28.49 seconds (18.72 MB/s)
```

Only a marginal throughput improvement from `100000`. Likely means that at those parameter sizes the bottleneck isn't in intermediate state file disk writes.

Development

Work on search-engine-indexer is mostly done on Emacs. The workflow looks like:

1. A CIDER session is started with `M-x cider-jack-in`

2. Code is evaluated with with `cider-eval-sexp-at-point` or `cider-eval-buffer`
3. Tests are run with `cider-test-run-test` or `cider-test-run-ns-tests`

Dependencies

- Java
- Leiningen
- Docker

Check out repository

```
git clone git@github.com:mpereira/search-engine-indexer.git
```

`cd` into repository

```
cd search-engine-indexer
```

Running tests

```
lein test
```

Building uberjar

```
lein do clean, uberjar
```

Publishing uberjar

```
aws s3 cp --acl public-read \
  target/uberjar/search-engine-indexer-0.2.0-SNAPSHOT-standalone.jar \
  s3://search-engine-indexer/jars/search-engine-indexer-0.2.0-SNAPSHOT-standalone.jar
```

Building Docker image

```
docker build -t mpereira/search-engine-indexer:0.2.0 .
```

Publishing Docker image

```
docker login
```

```
docker push mpereira/search-engine-indexer:0.2.0
```

License