605.744: Information Retrieval
Programming Assignment #2: Inverted Files

Sabbir Ahmed

September 18, 2022

1 Introduction

This paper describes the enhancements and features added to the Information Retrieval program started in Assignment 1. Modifications include improvement in performance and efficiency in normalizing text and generating statistics from the pre-generated corpus and addition of binary inverted files.

2 Technical Background

All of the source code is in Python 3.10. The program is split into several modules and follows an object oriented structure. The following is the directory structure of the source code:

The source code for all of the files are attached in Appendix A.

The total number of non-empty lines of code for the program doubled to just under 400. However, the average execution time to process the sample files reduced to 37 seconds.

2.1 Existing Classes

2.1.1 Driver

The driver script for the program is split between `run.py` and `test.py`. The former script, while maintaining the responsibility for the same list of tasks as the previous iteration, also generates the binary inverted files and dictionaries for the sample corpus and saves it to disk. The latter script loads the inverted files and dictionaries and looks up the test terms as specified in the prompt.

2.1.2 `normalize.Normalizer`

The `Normalizer` class received the following modifications:

1. expansion of word contractions was replaced by removing all of the stopwords. The stopwords are a mixture of tokens from `nltk.corpus.stopwords.words("english")` and the contractions hash table keys. The larger list of "STOPWORDS" is represented as a Python set() object for efficient lookups.
2. all of the normalizing methods accept generators as input to improve performance on memory.

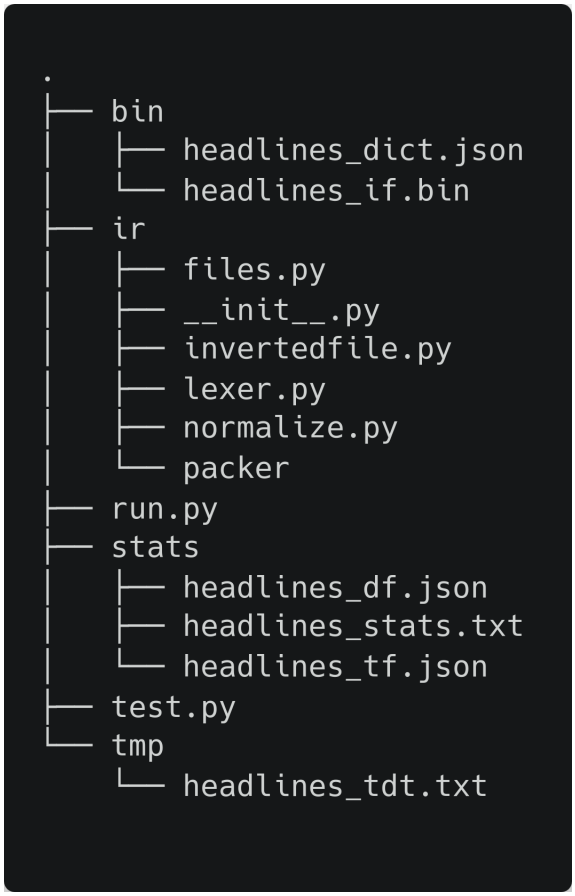

Figure 1: Directory Hierarchy of Assignment 2

2.1.3 `lexer.Lexer`

The `Lexer` class received the following modifications:

1. the Counter object `tf_in_doc` was added to maintain a term-frequency of the current document. This addition allows for the new method `term_doc_tf(doc_id: str)` that generates tuples of `(term-string, doc-ID, term-document-frequency)`. These tuples are saved on disk in files named "sample_tdt.txt" for further sorting and processing to generate the inverted files.

2.2 New Classes

2.2.1 `invertedfile.InvertedFile`

The `InvertedFile` class is responsible for building the binary inverted files and dictionaries. The class ingests the "sample_tdt.txt" files, sorts (in memory) the tuples as detailed in the prompt, and converts the values into 4-byte integer formats. The class also provides a method to look up tokens in the generated inverted files and dictionaries.

2.2.2 `Packer`

The `Packer` class is responsible for encoding and decoding fixed-format binary data.

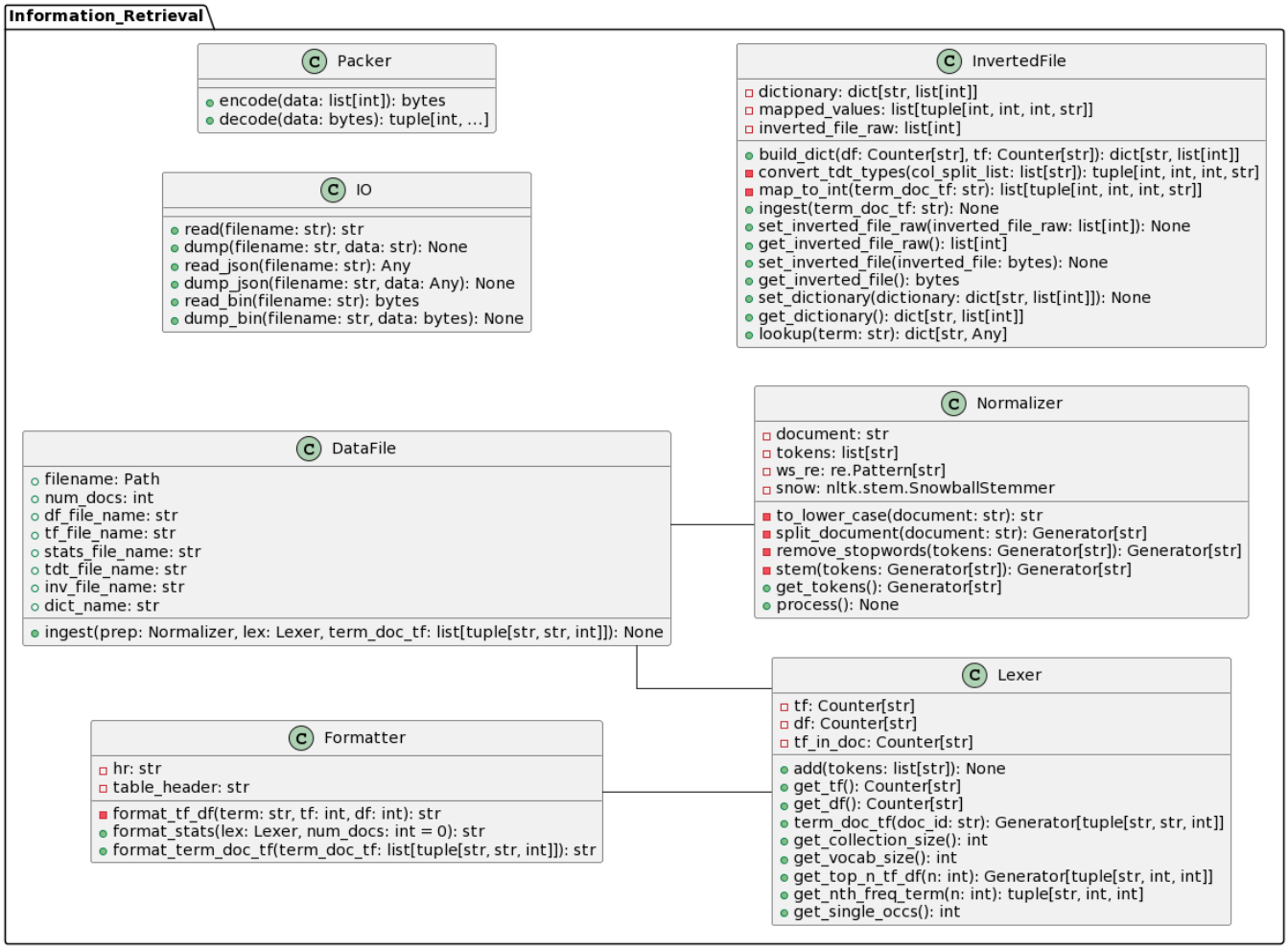

Figure 2: UML of Information Retrieval

2.2.3 `files` Classes

The `IO` class was split into 2 further classes, `Formatter` and `DataFile`. These classes provide support for file utility functions in the program, including reading and writing to plain files, JSON files, and binary files, formatting statistics outputs, generating file paths, etc.

3 Statistics and Observations

With the modifications made to the text normalization process, the statistics generated from the input file have changed. The new top 10 more frequent tokens now include “market”, “announc”, and “report”, which are terms that expected in headlines.

In terms of the inverted files and dictionaries, the space they occupy on disk combined are significantly lower than the original document.

File	Size (in bytes)	Description
headlines.txt	39381610	Input corpus file
headlines_dict.json	4275427	Generated dictionary JSON file
headlines_if.bin	27774312	Inverted binary file

Table 1: Sizes of Files Computed Through the `stat` Command on a Debian Based Linux

4 Testing

The tests described by the prompt were performed via the `test.py` driver script. The tokens to look up had to first be normalized through the `Normalizer` class.

Code Listing 1: Test 1: Document frequency and postings list for the terms: “Heidelberg” “cesium” “Trondheim” “crustacean”

```
# NOTE: the normalization stems "crustacean" to "crustacea"
# and "crustaceans" to "crustacean"
tokens1 = normalize_test_terms(
    prep, ("Heidelberg", "cesium", "Trondheim", "crustaceans")
)
results1 = read_inverted_file(
    invf,
    tokens1,
    ("term", "postings", "postings_len"),
)
```

```
[{'postings': (114330,
               135134,
               174781,
               221100,
               243838,
               285521,
               295687,
               452546,
               491140,
               491279),
  'postings_len': 10,
  'term': 'heidelberg'},
 {'postings': (50020, 280670, 348144,
391939, 394780),
  'postings_len': 5,
  'term': 'cesium'},
 {'postings': -1, 'postings_len': -1,
'term': 'trondheim'},
 {'postings': (230748, 234924, 426410),
  'postings_len': 3,
  'term': 'crustacean'}]
```

Figure 3: Output of Test 1

Code Listing 2: Test 2: Document frequency for the words: “Hopkins” “Stanford” “Brown” and “college”

```
tokens2 = normalize_test_terms(
    prep, ("Hopkins", "Stanford", "Brown", "college")
)
results2 = read_inverted_file(invf, tokens2, ("term", "postings_len"))
```

```
[{'postings_len': 73, 'term': 'hopkin'},
 {'postings_len': 154, 'term': 'stanford'},
 {'postings_len': 1083, 'term': 'brown'},
 {'postings_len': 2088, 'term': 'colleg'}]
```

Figure 4: Output of Test 2

Code Listing 3: Test 3: docids for documents that have both “Elon” and “Musk”

```
# NOTE: the normalization stems "Musks" to "Musk"
tokens3 = normalize_test_terms(prep, ("Elon", "Musks"))
results3 = read_inverted_file(invf, tokens3, ("postings",))
elon, musk = results3
elon_postings, musk_postings = set(elon["postings"]), set(musk["postings"])
elon_musk_postings = elon_postings & musk_postings
```

```
[3394, 16331, 19263, 21342, 29750, 44322,
45979, 52991, 57024, 57788, 84807, 98831,
115208, 122604, 127051, 128663, 131449, 131515,
146966, 159148, 186108, 194999, 197342, 239305,
240041, 245924, 249586, 274394, 283099, 297140,
303776, 305184, 306989, 341756, 342183, 354347,
369773, 383529, 399002, 399947, 420083, 431496,
431740, 449685, 456444, 482770]
```

Figure 5: Output of Test 3

A Source Code

Code Listing 4: ./ir/files.py

```
import json
from pathlib import Path
import re
from typing import Any

from .lexer import Lexer
from .normalize import Normalizer


class IO:
    @staticmethod
    def read(filename: str) -> str:
```

```
        with open(f"{filename}.txt") as fp:
            return fp.read()

    @staticmethod
    def dump(filename: str, data: str) -> None:

        with open(f"{filename}.txt", "w") as fp:
            fp.write(data)
        print(f"Dumped to '{filename}.txt'")

    @staticmethod
    def read_json(filename: str) -> Any:

        with open(f"{filename}.json") as fp:
            return json.loads(fp.read())

    @staticmethod
    def dump_json(filename: str, data: Any) -> None:

        with open(f"{filename}.json", "w") as fp:
            json.dump(data, fp)
        print(f"Dumped json to '{filename}.json'")

    @staticmethod
    def read_bin(filename: str) -> bytes:

        with open(f"{filename}.bin", "rb") as fp:
            return fp.read()

    @staticmethod
    def dump_bin(filename: str, data: bytes) -> None:

        with open(f"{filename}.bin", "wb") as fp:
            fp.write(data)
        print(f"Dumped binary to '{filename}.bin'")


class DataFile:
    def __init__(self, filename: Path) -> None:

        self.filename: Path = filename
        self.num_docs: int = 0

        self.df_file_name: str = f"stats/{filename.stem}_df"
        self.tf_file_name: str = f"stats/{filename.stem}_tf"
        self.stats_file_name: str = f"stats/{filename.stem}_stats"

        self.tdt_file_name: str = f"tmp/{filename.stem}_tdt"

        self.inv_file_name: str = f"bin/{filename.stem}_if"
        self.dict_name: str = f"bin/{filename.stem}_dict"

    def ingest(
        self,
        prep: Normalizer,
        lex: Lexer,
        term_doc_tf: list[tuple[str, str, int]],
    ) -> None:
```

```
        doc_id: str = ""
        doc_id_re = re.compile(r"\d+")
        line_num: int = 0

        with open(self.filename) as fp:
            for line in fp:
                match line_num % 4:

                    # line containing DocID
                    case 0:
                        doc_id = next(doc_id_re.finditer(line)).group()
                        self.num_docs += 1

                    # line containing document
                    case 1:

                        # normalize document through the preprocessing pipeline
                        prep.set_document(line)
                        prep.process()

                        # add processed tokens to the lexer
                        lex.add(prep.get_tokens())

                        # save records of term-DocID-tf
                        term_doc_tf.extend(lex.term_doc_tf(doc_id))

                    # empty lines
                    case _:
                        pass

                line_num += 1

        print("Processed", self.num_docs, "documents.")


class Formatter:

    hr: str = "------------------------------\n"
    table_header: str = f"{'Word':<12} | {'TF':<6} | {'DF':<6}\n{hr}"

    @staticmethod
    def __format_tf_df(term: str, tf: int, df: int) -> str:

        return f"{term:<12} | {tf:<6} | {df:<6}\n"

    @staticmethod
    def format_stats(lex: Lexer, num_docs: int = 0) -> str:

        contents: str = ""

        contents += f"{Formatter.hr}"
        contents += f"{num_docs} documents.\n"

        contents += f"{Formatter.hr}"
        contents += f"Collections size: {lex.get_collection_size()}\n"
        contents += f"Vocabulary size: {lex.get_vocab_size()}\n"
        contents += f"\n{Formatter.hr}"
```

```
        contents += "Top 100 most frequent words:\n"
        contents += Formatter.table_header
        for term in lex.get_top_n_tf_df(100):
            contents += Formatter.__format_tf_df(*term)

        contents += f"\n{Formatter.hr}"
        contents += "500th word:\n"
        contents += Formatter.table_header
        contents += Formatter.__format_tf_df(*lex.get_nth_freq_term(500))

        contents += f"\n{Formatter.hr}"
        contents += "1000th word:\n"
        contents += Formatter.table_header
        contents += Formatter.__format_tf_df(*lex.get_nth_freq_term(1000))

        contents += f"\n{Formatter.hr}"
        contents += "5000th word:\n"
        contents += Formatter.table_header
        contents += Formatter.__format_tf_df(*lex.get_nth_freq_term(5000))

        contents += f"\n{Formatter.hr}"
        single_occs: int = lex.get_single_occs()
        contents += "Number of words that occur in exactly one document:\n"
        contents += f"{single_occs} ({round(single_occs / lex.get_vocab_size() *
                                                100, 2)}%)\n"

        return contents

    @staticmethod
    def format_term_doc_tf(term_doc_tf: list[tuple[str, str, int]]) -> str:

        contents: str = ""
        for line in term_doc_tf:
            contents += " ".join(str(i) for i in line) + "\n"

        return contents
```

Code Listing 5: ./ir/invertedfile.py

```
from collections import Counter
from typing import Any

from .packer import Packer

# shared indices
TERM_ID_IDX = 0

# dictionary indices
OFFSET_IDX, LEN_IDX, DF_IDX, TF_IDX = (1, 2, 3, 4)

# inverted file indices
DOC_ID_IDX, TC_IDX, TERM_STR_IDX = (1, 2, 3)


class InvertedFile:
    def __init__(self) -> None:

        self.dictionary: dict[str, list[int]] = {}
        self.mapped_values: list[tuple[int, int, int, str]] = []
```

```
        self.inverted_file_raw: list[int] = []

    def build_dict(
        self, df: Counter[str], tf: Counter[str]
    ) -> dict[str, list[int]]:

        for idx, term in enumerate(sorted(tf.keys())):
            self.dictionary[term] = [idx, 0, 0, df[term], tf[term]]

        return self.dictionary

    def __convert_tdt_types(
        self, col_split_list: list[str]
    ) -> tuple[int, int, int, str]:
        return (
            self.dictionary[col_split_list[TERM_ID_IDX]][
                TERM_ID_IDX
            ],  # term ID
            int(col_split_list[DOC_ID_IDX]),  # doc ID
            int(col_split_list[TC_IDX]),  # term count,
            col_split_list[TERM_ID_IDX],  # term
        )

    def __map_to_int(self, term_doc_tf: str) -> list[tuple[int, int, int, str]]:

        return [
            self.__convert_tdt_types(i.split(" "))
            for i in term_doc_tf.split("\n")[:-1]
        ]

    def ingest(self, term_doc_tf: str) -> None:

        self.mapped_values = self.__map_to_int(term_doc_tf)

        # builtin timsort implicitly performs radix sort
        self.mapped_values.sort()

        cur: int = -1
        offset: int = 0
        width: int = 0

        for val in self.mapped_values:

            term_str: str = val[TERM_STR_IDX]

            if cur != val[TERM_ID_IDX]:
                cur = val[TERM_ID_IDX]
                self.dictionary[term_str][OFFSET_IDX] = offset  # update offset
                # width between the current term and the next
                width = self.dictionary[term_str][DF_IDX] * 2
                self.dictionary[term_str][LEN_IDX] = width  # update width
            offset += 2
            self.inverted_file_raw.extend(val[DOC_ID_IDX:TERM_STR_IDX])

    def set_inverted_file_raw(self, inverted_file_raw: list[int]) -> None:

        self.inverted_file_raw = inverted_file_raw

    def get_inverted_file_raw(self) -> list[int]:
```

```
        return self.inverted_file_raw

    def set_inverted_file(self, inverted_file: bytes) -> None:

        self.inverted_file = inverted_file

    def get_inverted_file(self) -> bytes:

        return self.inverted_file

    def set_dictionary(self, dictionary: dict[str, list[int]]) -> None:

        self.dictionary = dictionary

    def get_dictionary(self) -> dict[str, list[int]]:

        return self.dictionary

    def lookup(self, term: str) -> dict[str, Any]:

        metadata: dict[str, Any] = {"term": term}

        if term not in self.dictionary:
            return metadata

        of: int = self.dictionary[term][OFFSET_IDX]
        width: int = self.dictionary[term][LEN_IDX]
        postings: tuple[int, ...] = Packer.decode(self.inverted_file)[
            of : of + width : 2
        ]
        pos_len: int = len(postings)

        metadata["offset"] = of
        metadata["width"] = width
        metadata["postings"] = postings
        metadata["postings_len"] = pos_len

        return metadata
```

Code Listing 6: ./ir/normalize.py

```
import re
from typing import Generator

import nltk

# fmt: off
STOPWORDS: set[str] = {
    # contractions
    "aren't", "ain't", "can't", "could've", "couldn't", "didn't", "doesn't", "don'
                                                t",
    "hadn't", "hasn't", "haven't", "he'd", "he'll", "he's", "i'd", "i'll",
    "i'm", "i've", "isn't", "it'll", "it'd", "it's", "let's", "mightn't",
    "might've'", "mustn't", "must've'", "shan't", "she'd", "she'll", "she's", "
                                                should've",
    "shouldn't", "that'll", "that's", "there's", "they'd", "they'll", "they're", "
                                                they've",
    "wasn't", "we'd", "we'll", "we're", "we've", "weren't", "what'll", "what're",
```

```
    "what's", "what've", "where's", "who'd", "who'll", "who're", "who's", "who've"
                                                    ,
    "won't", "wouldn't", "would've", "y'all", "you'd", "you'll", "you're", "you've
                                                   ",
    # NLTK stopwords
    "a", "all", "am", "an", "and", "any", "are", "as",
    "at", "be", "because", "been", "being", "but", "by", "can",
    "cannot", "could", "did", "do", "does", "doing", "for", "from",
    "had", "has", "have", "having", "he", "her", "here", "hers",
    "herself", "him", "himself", "his", "how", "i", "if", "in",
    "is", "it", "its", "itself", "just", "let", "may", "me",
    "might", "must", "my", "myself", "need", "no", "nor", "not",
    "now", "o", "of", "off", "on", "once", "only", "or",
    "our", "ours", "ourselves", "shall", "she", "should", "so", "some",
    "such", "than", "that", "the", "their", "theirs", "them", "themselves",
    "then", "there", "these", "they", "this", "those", "to", "too",
    "very", "was", "we", "were", "what", "when", "where", "which",
    "who", "whom", "why", "will", "with", "would", "you", "your",
    "yours", "yourself", "yourselves",
}
# fmt: on


class Normalizer:
    def __init__(self) -> None:

        self.document: str = ""
        self.tokens: Generator[str, None, None]

        self.ws_re: re.Pattern[str] = re.compile(r"([A-Za-z]+'?[A-Za-z]+)")
        self.snow: nltk.stem.SnowballStemmer = nltk.stem.SnowballStemmer(
            "english"
        )

    def set_document(self, document: str) -> None:

        self.document = document[:-1]

    def __to_lower_case(self, document: str) -> str:

        return document.lower()

    def __split_document(self, document: str) -> Generator[str, None, None]:

        return (x.group(0) for x in self.ws_re.finditer(document))

    def __remove_stopwords(
        self, tokens: Generator[str, None, None]
    ) -> Generator[str, None, None]:
        return (word for word in tokens if word not in STOPWORDS)

    def __stem(
        self, tokens: Generator[str, None, None]
    ) -> Generator[str, None, None]:

        return (self.snow.stem(token) for token in tokens)

    def get_tokens(self) -> Generator[str, None, None]:
```

```
        return self.tokens

    def process(self) -> None:

        # convert the entire document to lower-case
        doc_lc: str = self.__to_lower_case(self.document)

        # split the document on its whitespace
        tokens: Generator[str, None, None] = self.__split_document(doc_lc)

        # remove contractions and stopwords
        no_sw: Generator[str, None, None] = self.__remove_stopwords(tokens)

        # stem tokens
        self.tokens = self.__stem(no_sw)
```

Code Listing 7: ./ir/lexer.py

```
from collections import Counter
from typing import Generator


class Lexer:
    def __init__(self) -> None:

        self.tf: Counter[str] = Counter()
        self.df: Counter[str] = Counter()
        self.tf_in_doc: Counter[str] = Counter()

    def add(self, tokens: Generator[str, None, None]) -> None:

        # create a Counter for the document
        self.tf_in_doc.clear()
        self.tf_in_doc.update(tokens)

        # update the total term-frequency values with the Counter
        self.tf.update(self.tf_in_doc)

        # increment the document-frequency values
        self.df.update(self.tf_in_doc.keys())

    def term_doc_tf(
        self, doc_id: str
    ) -> Generator[tuple[str, str, int], None, None]:

        for term in self.tf_in_doc:
            yield term, doc_id, self.tf_in_doc[term]

    def get_tf(self) -> Counter[str]:

        return self.tf

    def get_df(self) -> Counter[str]:

        return self.df

    def get_collection_size(self) -> int:

        return self.tf.total()
```

```
    def get_vocab_size(self) -> int:

        return len(self.tf)

    def get_top_n_tf_df(
        self, n: int
    ) -> Generator[tuple[str, int, int], None, None]:

        top_n_tf = self.tf.most_common(n)
        for tf in top_n_tf:
            term, freq = tf
            yield term, freq, self.df[term]

    def get_nth_freq_term(self, n: int) -> tuple[str, int, int]:

        term, freq = self.tf.most_common(n)[-1]
        return term, freq, self.df[term]

    def get_single_occs(self) -> int:

        single_occs: int = 0
        for df in self.df.values():
            if df == 1:
                single_occs += 1

        return single_occs
```

Code Listing 8: ./ir/packer.py

```
from struct import pack, unpack

BYTE_FMT_CHAR = "I"
BYTE_FMT_SIZE = 4


class Packer:
    @staticmethod
    def encode(data: list[int]) -> bytes:

        return pack(BYTE_FMT_CHAR * len(data), *data)

    @staticmethod
    def decode(data: bytes) -> tuple[int, ...]:

        return unpack(BYTE_FMT_CHAR * (len(data) // BYTE_FMT_SIZE), data)
```

Code Listing 9: ./run.py

```
from pathlib import Path

from ir.files import IO, Formatter, DataFile
from ir.invertedfile import InvertedFile
from ir.lexer import Lexer
from ir.normalize import Normalizer
from ir.packer import Packer


def process_document(filename: Path) -> None:
```

```
    prep = Normalizer()
    lex = Lexer()
    data = DataFile(filename)

    term_doc_tf: list[tuple[str, str, int]] = []
    data.ingest(prep, lex, term_doc_tf)
    term_doc_tf_str: str = Formatter.format_term_doc_tf(term_doc_tf)
    IO.dump(data.tdt_file_name, term_doc_tf_str)
    IO.dump_json(data.df_file_name, lex.get_df())
    IO.dump_json(data.tf_file_name, lex.get_tf())

    contents: str = Formatter.format_stats(lex, data.num_docs)
    IO.dump(data.stats_file_name, contents)

    invf = InvertedFile()
    invf.build_dict(lex.get_df(), lex.get_tf())
    invf.ingest(IO.read(data.tdt_file_name))
    inv_file = invf.get_inverted_file_raw()
    if_data = Packer.encode(inv_file)
    IO.dump_bin(data.inv_file_name, if_data)
    IO.dump_json(data.dict_name, invf.get_dictionary())


if __name__ == "__main__":

    process_document(Path(__file__).parent.parent / "data" / "headlines.txt")
```

Code Listing 10: ./test.py

```
from pathlib import Path
from pprint import pprint
from typing import Generator

from ir.files import IO, DataFile
from ir.invertedfile import InvertedFile
from ir.normalize import Normalizer


def read_inverted_file(
    invf: InvertedFile,
    tokens: Generator[str, None, None],
    keys: tuple[str, ...],
) -> Generator[dict[str, int | tuple[int, ...]], None, None]:

    for token in tokens:
        metadata = invf.lookup(token)
        yield {key: metadata.get(key, -1) for key in keys}


def normalize_test_terms(
    prep: Normalizer, terms: tuple[str, ...]
) -> Generator[str, None, None]:

    prep.set_document(" ".join(terms))
    prep.process()
    return prep.get_tokens()
```

```python
if __name__ == "__main__":

    data = DataFile(Path(__file__).parent.parent / "data" / "headlines.txt")

    invf = InvertedFile()
    inv_file = IO.read_bin(data.inv_file_name)
    dict_ = IO.read_json(data.dict_name)

    invf.set_inverted_file(inv_file)
    invf.set_dictionary(dict_)

    prep = Normalizer()

    # print out the document frequency and postings list for terms:
    # "Heidelberg", "cesium", "Trondheim", "crustacean".
    # NOTE: the normalization stems "crustacean" to "crustacea"
    # and "crustaceans" to "crustacean"
    tokens1 = normalize_test_terms(
        prep, ("Heidelberg", "cesium", "Trondheim", "crustaceans")
    )
    results1 = read_inverted_file(
        invf,
        tokens1,
        ("term", "postings", "postings_len"),
    )
    pprint(list(results1))

    # give document frequency, but do not print postings for the words:
    # "Hopkins", "Stanford", "Brown", and "college"
    tokens2 = normalize_test_terms(
        prep, ("Hopkins", "Stanford", "Brown", "college")
    )
    results2 = read_inverted_file(invf, tokens2, ("term", "postings_len"))
    pprint(list(results2))

    # Print out the docids for documents that have both "Elon" and "Musk" in the
                                                    text.
    tokens3 = normalize_test_terms(prep, ("Elon", "Musks"))
    results3 = read_inverted_file(invf, tokens3, ("postings",))
    elon, musk = results3
    elon_postings, musk_postings = set(elon["postings"]), set(musk["postings"])
    elon_musk_postings = elon_postings & musk_postings
    pprint(sorted(elon_musk_postings))
```

B Outputs

Code Listing 11: Statistics of 'headlines.txt'

```
500000 documents.
```

```
Collections size: 3518452
Vocabulary size: 111614
```

```
Top 100 most frequent words:
Word           | TF      | DF
```

new	27023	26505
market	14814	12930
after	13025	12977
announc	11090	11065
up	10814	10666
over	10007	9961
say	9875	9828
year	9442	9250
report	8920	8758
day	8578	8414
global	8272	8180
man	8211	8126
open	7930	7875
get	7838	7785
us	7754	7655
more	7682	7577
out	7638	7583
first	7591	7532
world	7268	7160
septemb	7266	7208
win	7226	7155
week	6879	6751
polic	6581	6520
launch	6426	6414
make	6398	6366
show	6138	6058
share	5886	5763
school	5817	5645
about	5741	5686
take	5725	5714
state	5712	5572
servic	5695	5566
back	5611	5538
home	5594	5480
video	5569	5499
top	5569	5505
plan	5515	5452
busi	5489	5329
one	5479	5289
time	5467	5379
industri	5405	5282
game	5248	5097
china	5216	5109
inc	5065	4496
help	4992	4950
call	4953	4925
set	4906	4890
th	4898	4749
citi	4891	4776
group	4883	4792
stock	4777	4708
into	4771	4760
live	4761	4677

review	4718	4691
updat	4714	4692
research	4667	4557
against	4662	4647
two	4647	4584
rate	4643	4551
fall	4635	4613
kill	4614	4597
compani	4448	4389
nation	4418	4361
award	4328	4221
best	4321	4250
high	4281	4225
million	4279	4220
releas	4262	4248
meet	4212	4190
star	4174	4114
car	4139	4070
price	4090	4052
offic	4067	4017
offer	4058	4036
watch	4035	3791
deal	4026	3981
refuge	3979	3936
chang	3906	3879
free	3903	3803
manag	3890	3780
down	3844	3822
big	3832	3749
name	3801	3777
season	3772	3749
use	3763	3741
hous	3751	3706
lead	3727	3720
news	3708	3641
look	3705	3677
hit	3705	3698
fire	3687	3591
sale	3685	3614
technolog	3683	3551
team	3677	3630
secur	3670	3541
charg	3663	3648
confer	3663	3637
footbal	3575	3527
start	3558	3541
work	3557	3502

500th word:

Word	TF	DF
agreement	1314	1310

1000th word:

Word	TF	DF
exhibit	736	724

5000th word:

Word	TF	DF
manipul	92	92

Number of words that occur in exactly one document:
52962 (47.45%)