
Sthir

Release 0.0.2

Parth Parikh, Mrunank Mistry, Dhruvam Kothari

Jun 20, 2020

CONTENTS:

- 1 Sthir** **1**
- 1.1 CLI module 1
- 1.2 Test module 1
- 1.3 convert_2p15 module 2
- 1.4 convert_byte module 2
- 1.5 generate_search module 2
- 1.6 mmh3 module 3
- 1.7 parse module 3
- 1.8 scan module 3
- 1.9 spectral_bloom_filter module 4

- 2 Indices and tables** **7**

- Python Module Index** **9**

- Index** **11**

1.1 CLI module

- CLI.**chunk_size_arg** (*val*)
Validates the chunk_size for the arg parser
- CLI.**create_arg_parser** ()
Returns a well-setup argument-parser object
- CLI.**dir_path** (*path*)
Validates path to the source folder
- CLI.**error_rate_arg** (*val*)
Validates the error_rate for the arg parser

1.2 Test module

- class** Test.**Tester** (*doc_name: str, chunk_size: int = 4, fp_rate: int = 0.1*)
Bases: object
- Class for testing the spectral bloom filters
- generate_Filter** (*remove_stopwords, lemmetize*)
Returns the counter and params for the specified document in Testing Folder
- read_dict_words** ()
Reads and returns a list of words in the english_dict.txt file
- test_filter_for_FP** ()
Tests and logs the stats after testing the provided file
- Test.**create_logger** ()
Returns a logger object

1.3 convert_2p15 module

1.4 convert_byte module

1.5 generate_search module

@author: iotarepeat (<https://github.com/iotarepeat>)

This file contains methods for base2p15 encoding, a custom encoding that uses unicode. This encoding uses 2^{15} unicode characters. The goal is to make binary representation of Spectral Bloom Filter, to a JS string as small as possible.

`generate_search.base2p15_decode` (*base2p15: str*) → str
Encode/decode given base2p15 string as binary string

Parameters `base2p15` (*str*) – A base2p15 string, generated by `base2p15_encode`

Returns A binary string e.g. 00001001

Return type str

`generate_search.base2p15_encode` (*bit_string: str*) → str
Encode given bit_string to base2p15.

Parameters `bit_string` (*str*) – A binary string e.g. 00001001

Returns A base2p15 encoded string

Return type str

`generate_search.base2p15_get_range` (*base2p15: str, start: int, end: int*) → str
Get a range of bits from [start,end)

Parameters

- `base2p15` (*str*) – A base2p15 string generated by `base2p15_encode`
- `start` (*int*) – Starting number (inclusive)
- `end` (*int*) – End number (exclusive)

Returns A binary string e.g. 00001001

Return type str

`generate_search.gen_chunks` (*string: str, chunk_size: int, drop_remaining: bool = False*) → Iterable[str]
Yields an iterator of chunks of specified size

If `drop_remaining` is specified, the iterator is guaranteed to have all chunks of same size.

```
>>> list(gen_counter_chunks('123456789A', 4)) == ['1234', '5678', '9A']
>>> list(gen_counter_chunks('123456789A', 4, drop_remaining = True)) == ['1234',
↳ '5678']
```

1.6 mmh3 module

`mmh3.murmur3_x86_32` (*key*, *seed=0*)

1.7 parse module

`parse.extract_html_bs4` (*html_file_path*: *str*, *remove_stopwords*: *bool = True*, *enable_lemmetization*: *bool = False*)

Given a path to html file it will extract all text in it and return a list of words (using library: BeautifulSoup4)

Parameters

- **html_file_path** (*str*) – Path to html file, will be called with `open()`
- **remove_stopwords** (*bool*, *optional*) – Will remove stopwords like [“the”, “them”,etc], defaults to `False`
- **enable_lemmetization** (*bool*, *optional*) – Will lemmitize words if set to `True`.
Ex: `cats->cat`, defaults to `False`

Returns A list of words all in lowercase

Return type `List[str]`

`parse.extract_html_newspaper` (*html_file*: *str*, *remove_stopwords=True*, *enable_lemmetization=False*) → `List[str]`

Given a path to html file it will extract all text in it and return a list of words (using library: Newspaper3k)

Parameters

- **html_file_path** (*str*) – Path to html file, will be called with `open()`
- **remove_stopwords** (*bool*, *optional*) – Will remove stopwords like [“the”, “them”,etc], defaults to `False`

Returns A list of words all in lowercase

Return type `List[str]`

1.8 scan module

`scan.create_search_page` (*directory*, *output_file='search.html'*, *false_positive=0.1*, *chunk_size=4*, *remove_stopwords=True*)

Generates the search output file using the directory path.

Parameters

- **directory** – Directory path where HTML files are located
- **output_file** – name of the output file (Default - “search.html”)
- **false_positive** – Acceptable false positive rate during search (Default - 0.1) 0.01 is a better alternative, at the cost of increase in file size.
- **chunk_size** – Size of each counter in Spectral Bloom Filter (Default - 4) Default of 4 means that the maximum increment a counter can perform is 2^{*4} , which is 16.
- **remove_stopwords** – To remove stopwords (Default - `True`)

It saves the search file in the `output_file` path.

`scan.download_urls` (*json_file*, *output_file=""*)

Downloads and saves HTML files using a JSON file containing list of URLs. (For Debugging purposes)

`scan.generate_bloom_filter` (*file*, *false_positive=0.1*, *chunk_size=4*, *remove_stopwords=True*)

Generates a bloom filter and saves it in .bin file.

The saved .bin filename is same as that of the .html file name.

Returns a dictionary containing the -

length of the bitarray (m), no of hash functions used (k), chunk size (chunk_size), binary file name (bin_file), and HTML file's title (title).

This method is internally used in method - create_search_page

`scan.get_all_bin_files` (*directory*)

Returns list of bin files located in the directory

`scan.get_all_html_files` (*directory*)

Returns list of html files located in the directory

1.9 spectral_bloom_filter module

class `spectral_bloom_filter.Hash_Funcs` (*k: int*, *m: int*)

Bases: object

static `check_duplicates` (*indices_list: list*)

check_hashes (*word_list: list*)

Logs the duplicate hashed indices for words in words_list

Parameters `word_list` – List of words

get_hashes (*word: str*) → list

Returns a list of k hashed indices for the input word

Parameters `word` – Word to be hashed

Returns List of hashes of the word

class `spectral_bloom_filter.Spectral_Bloom_Filter` (*error_rate: float = 0.01*)

Bases: object

Creates a Spectral Bloom Filter using the words parsed from the documents

Paper: SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data, June 2003 Pages 241–252

DOI: <https://doi.org/10.1145/872757.872787>

create_filter (*tokens: list*, *m: int*, *chunk_size: int = 4*, *no_hashes: int = 5*, *method: str = 'minimum'*, *to_bitarray: bool = True*, *bitarray_path: str = 'document.bin'*) → `bitarray.bitarray`

Creates a spectral bloom filter.

Paper: SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data, June 2003 Pages 241–252

DOI: <https://doi.org/10.1145/872757.872787>

Parameters

- **tokens** – List of words to index in spectral bloom filter
- **m** – size of the bitarray
- **chunk_size** – Size of each counter in Spectral Bloom Filter (default: 4). Default of 4 means that the maximum increment a counter. Can perform is 2^{*4} , which is 16.
- **no_hashes** – No. of hashes to index word with, (default: 5)
- **method** – Currently only “minimum” is supported, (default: “minimum”). “minimum” stands for Minimum Increment
- **to_bitarray** – If True, will convert and save as bitarray in bitarray_path. If False, method will return list of lists containing the entire bitarray with chunks. (Default: True).
- **bitarray_path** – Path to store the bitarray, (default:”document.bin”).

create_hashes (*token: str, hashes: int, max_length: int*) → list

Get the hashed indices for the string

Parameters

- **token** – token to index
- **hashes** – no. of hashes (k)
- **max_length** – maximum length of the hash (m)

Returns list of hashes

gen_counter_chunks (*string: str, chunk_size: int, drop_remaining: bool = False*) → Iterable[str]

Yields an iterator of chunks of specified size

If drop_remaining is specified, the iterator is guaranteed to have all chunks of same size.

```
>>> list(gen_counter_chunks('123456789A', 4)) == ['1234', '5678', '9A']
>>> list(gen_counter_chunks('123456789A', 4, drop_remaining = True)) == ['1234', '5678']
```

Parameters

- **string** – bit string whose chunks are to be obtained
- **chunk_size** – size of each chunk (optimal: 4)
- **drop_remaining** – to drop the extra string, if left, (default: False)

Returns generator object containing the list of chunks

init_counter (*counter_length: int*) → dict

To initialize a binary counter for incrementing Spectral Bloom Filter’s counters.

Example: For counter_length = 2 Method returns - {'00': '01', '01': '10', '10': '11', '11': '11'}

Parameters **counter_length** – No. of bits in each counter

Returns Dictionary used for binary counter operation

initialize_string (*length: int*)

Returns string of zeros of width “length”.

Parameters **length** – size of the string

Returns string of 0s of the specified length

`optimal_m_k (n: int, p: int) → tuple`

From: <https://stackoverflow.com/questions/658439/how-many-hash-functions-does-my-bloom-filter-need>

Parameters

- **n** – items expected in filter
- **p** – false positive rate
- **chunk_size** – number of bits in each counter

Returns Tuple containing: m for number of bits needed in the bloom filter (index 0) and k for number of hash functions we should apply (index 1)

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

c

CLI, 1
convert_2p15, 2
convert_byte, 2

g

generate_search, 2

m

mmh3, 3

p

parse, 3

s

scan, 3
spectral_bloom_filter, 4

t

Test, 1

B

base2p15_decode() (in module generate_search), 2
 base2p15_encode() (in module generate_search), 2
 base2p15_get_range() (in module generate_search), 2

C

check_duplicates() (spectral_bloom_filter.Hash_Funcs static method), 4
 check_hashes() (spectral_bloom_filter.Hash_Funcs method), 4
 chunk_size_arg() (in module CLI), 1
 CLI
 module, 1
 convert_2p15
 module, 2
 convert_byte
 module, 2
 create_arg_parser() (in module CLI), 1
 create_filter() (spectral_bloom_filter.Spectral_Bloom_Filter method), 4
 create_hashes() (spectral_bloom_filter.Spectral_Bloom_Filter method), 5
 create_logger() (in module Test), 1
 create_search_page() (in module scan), 3

D

dir_path() (in module CLI), 1
 download_urls() (in module scan), 3

E

error_rate_arg() (in module CLI), 1
 extract_html_bs4() (in module parse), 3
 extract_html_newspaper() (in module parse), 3

G

gen_chunks() (in module generate_search), 2

gen_counter_chunks() (spectral_bloom_filter.Spectral_Bloom_Filter method), 5
 generate_bloom_filter() (in module scan), 4
 generate_Filter() (Test.Tester method), 1
 generate_search
 module, 2
 get_all_bin_files() (in module scan), 4
 get_all_html_files() (in module scan), 4
 get_hashes() (spectral_bloom_filter.Hash_Funcs method), 4

H

Hash_Funcs (class in spectral_bloom_filter), 4

I

init_counter() (spectral_bloom_filter.Spectral_Bloom_Filter method), 5
 initialize_string() (spectral_bloom_filter.Spectral_Bloom_Filter method), 5

M

mmh3
 module, 3
 module
 CLI, 1
 convert_2p15, 2
 convert_byte, 2
 generate_search, 2
 mmh3, 3
 parse, 3
 scan, 3
 spectral_bloom_filter, 4
 Test, 1
 murmur3_x86_32() (in module mmh3), 3

O

optimal_m_k() (spectral_bloom_filter.Spectral_Bloom_Filter method), 5

P

parse
 module, 3

R

read_dict_words() (*Test.Tester method*), 1

S

scan
 module, 3
spectral_bloom_filter
 module, 4
Spectral_Bloom_Filter (class in *spectral_bloom_filter*), 4

T

Test
 module, 1
test_filter_for_FP() (*Test.Tester method*), 1
Tester (class in *Test*), 1