directories :

(all directories stored in **helper** file) **:**

i_index_dir -> base directory of inverted index

f_index_main_dir -> base directory of forward index

docs_subdir_log -> log of forward indexed docs

i_log -> log of inverted indexed docs

docs_path -> path where data_set is stored either in separate files or a single huge file.

stopwords_path -> path of stopwords file

Indexer :

```
while 1:
    index()
```

index() :

```
f_process = multiprocessing.Process( target = f_placer() )
i_process = multiprocessing.Process( target = i_placer() )
```

f_placer() :

```
# docs =  [ name of all documents from dos_path ]
# indexed_docs =  { documents that are already indexed, from docs_subdir_log }
# f_index_folders = [name of all folders in f_index directory]
# for doc in docs not in indexed_docs:
                    get_out_path_for_f_index ( doc, f_index_folder )
```

```
                path_of_nth_doc = docs_path + "\\" + str ( doc )
                forward_indexer ( stopwords_path, path_of_nth_doc,
                output_path )
                if forward_indexer return 0 update sub_dir_log
```

forward_indexer (stopwords_file, data_set, output_file)

```
        read stop words file
        open targeted file
            read it doc by doc
                lowercase all words
                substitute non alphanumeric with space
                tokenize
                remove stopwords
                stem each token
                enumerate through tokens and create desired dict pattern
                ( { ‘word’ : [ location_weight, x ], ‘word_n’ : [ location_weight, x ] } )
            out_batch_of_file = out_batch_of previous docs + ”,” + doc_no + ”,” + doc_size
            + ”,” forward_batch + ”\n”
        write forward_index onto file
```

i_placer ( ) :

```
    dir_dic_f  =  get_sub_dir_of_findex( docs_subdir_log )
    i_dict = read_ilog()
    for key in dir_dict_f :
```

```
if key not in i_dict :
    f_index = f_index_main_dir + "\\" + dir_dict_i[key] + "\\" + key
    inverted_batch = inverted_indexer(f_index)
    for word in inverted_batch :
        store_on_hashed_directory ( word, inverted_batch, i_index_dir, 1)
    update i_log
```

i_indexer (forwardindex_file) :

read forward index file and create your desired pattern for inverted index

I did

```
{ ‘word’ : { doc_id : [ total_words, x ] }, ‘word_n’ : { doc_id_n : [ total_words, x ] } }
```

return this inverted dictionary

helper:

Variables :

- directories (already mentioned above)
- dict_rest = dictionary of word restricted by Microsoft in filenaming.

Methods :

- get_size(path):
- query_parser(stopwords_path_, query):
- get_stopword_path():

get_qdict(path_list):

get_wposting_path(query_string):

unsorted_result(idict, query_list):

sort_result(r_doc):

get_hashed_directory(higher_directory, key_word, mask):

get_hashed_directory_tyag(higher_directory, key_word, mask):

check_for_path(hashed_path):

output_on_hashed_path(information, full_hashed_address, key_word, restricted, single_nested_dict_or_str):

get_sub_dir_of_findex(sub_dir_log_file_path):

read_doc_sub_directories():

read_ilog():

get_out_path_for_f_index(doc, f_index_folder):

store_on_hashed_directory(key_word, information, base_directory, single_nested_dict_or_str):

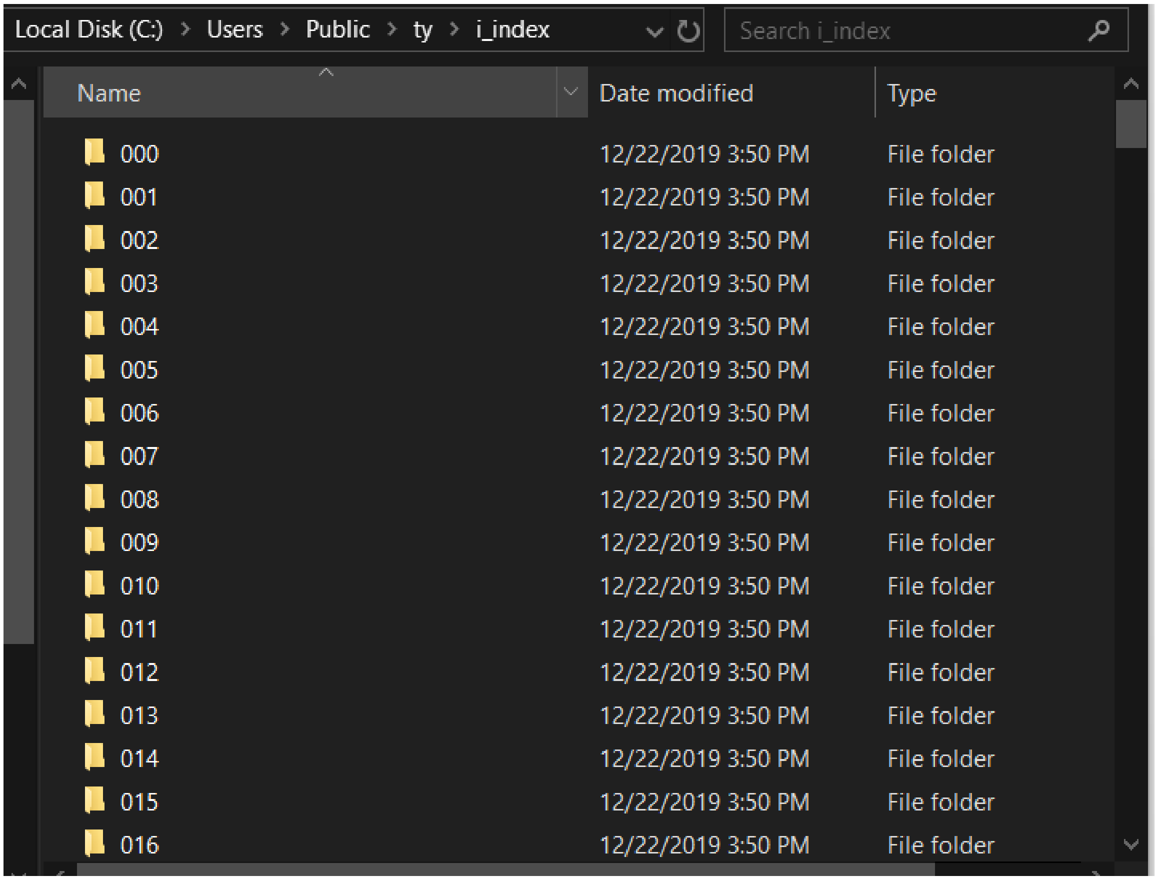
Hashing to generate directory :

One word index in one pickle file:

It is far better to get inverted index of word which we desire not of any other file, so inverted index is not stored in clustered form, each pickle file corresponds to inverted index of a single word. This saves file loading time, it takes 0.7 seconds to load a 3MB dictionary. If algorithm is to be scaled over a really huge dataset then storing indexes of words in clustered form could easily result in file size greater than 15 MB , this large pickle dictionary would take whopping 3.5 seconds to be loaded!

What about nesting of folders ?

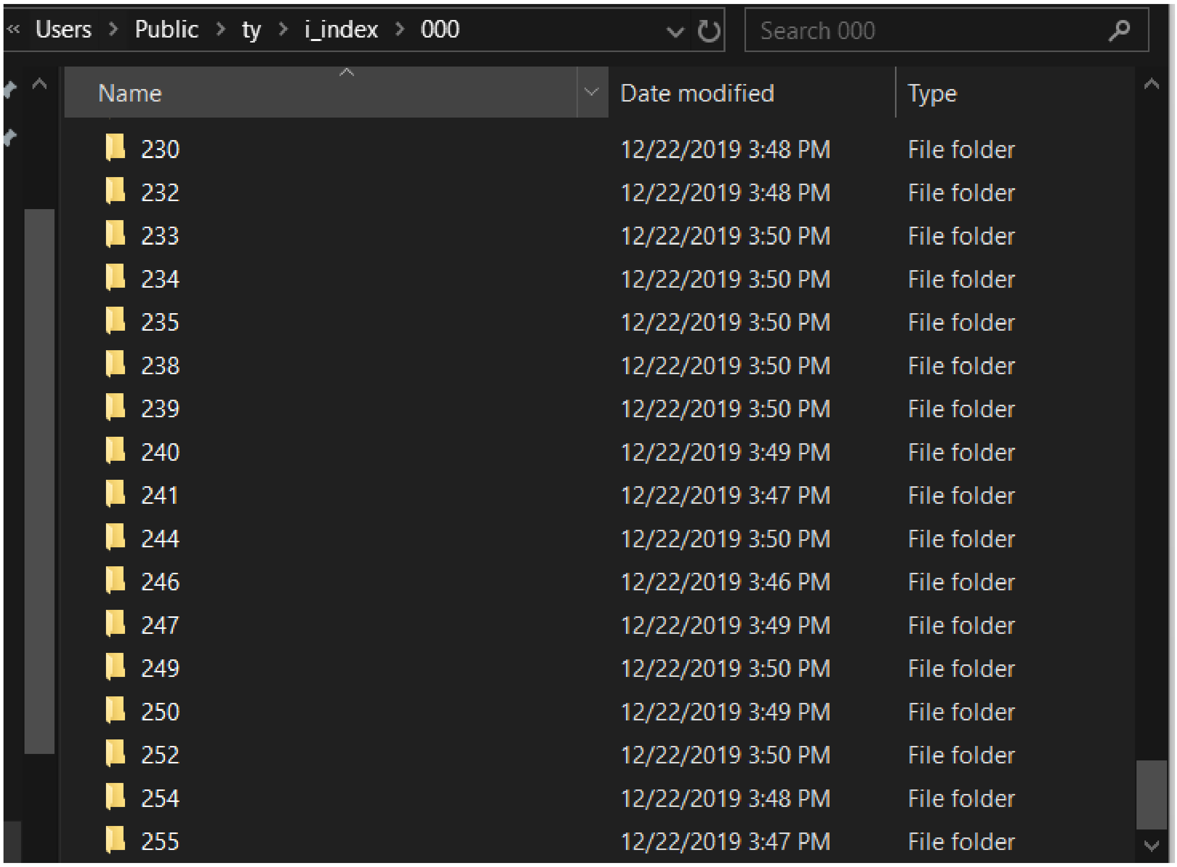
Outer chain of folders



Local Disk (C:) > Users > Public > ty > i_index

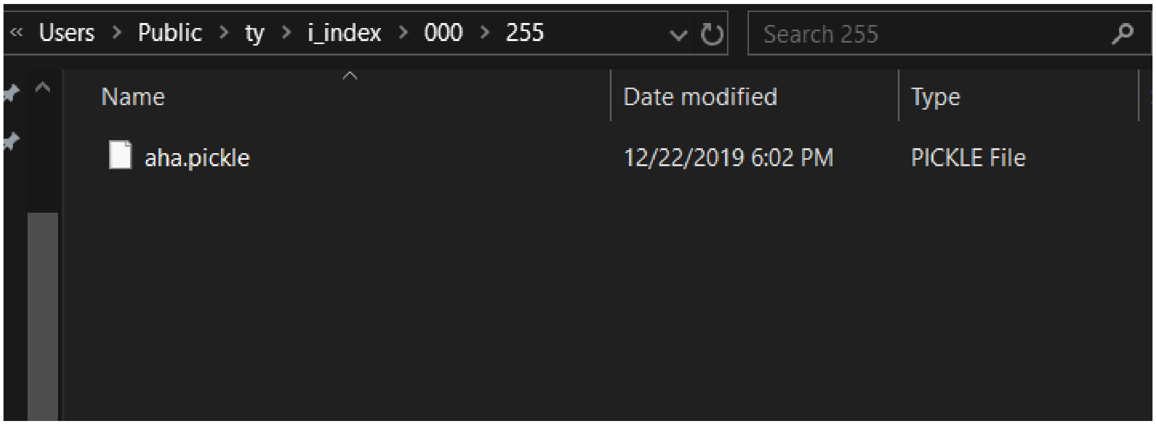
Name	Date modified	Type
000	12/22/2019 3:50 PM	File folder
001	12/22/2019 3:50 PM	File folder
002	12/22/2019 3:50 PM	File folder
003	12/22/2019 3:50 PM	File folder
004	12/22/2019 3:50 PM	File folder
005	12/22/2019 3:50 PM	File folder
006	12/22/2019 3:50 PM	File folder
007	12/22/2019 3:50 PM	File folder
008	12/22/2019 3:50 PM	File folder
009	12/22/2019 3:50 PM	File folder
010	12/22/2019 3:50 PM	File folder
011	12/22/2019 3:50 PM	File folder
012	12/22/2019 3:50 PM	File folder
013	12/22/2019 3:50 PM	File folder
014	12/22/2019 3:50 PM	File folder
015	12/22/2019 3:50 PM	File folder
016	12/22/2019 3:50 PM	File folder

Inner chain of folders



Name	Date modified	Type
230	12/22/2019 3:48 PM	File folder
232	12/22/2019 3:48 PM	File folder
233	12/22/2019 3:50 PM	File folder
234	12/22/2019 3:50 PM	File folder
235	12/22/2019 3:50 PM	File folder
238	12/22/2019 3:50 PM	File folder
239	12/22/2019 3:50 PM	File folder
240	12/22/2019 3:49 PM	File folder
241	12/22/2019 3:47 PM	File folder
244	12/22/2019 3:50 PM	File folder
246	12/22/2019 3:46 PM	File folder
247	12/22/2019 3:49 PM	File folder
249	12/22/2019 3:50 PM	File folder
250	12/22/2019 3:49 PM	File folder
252	12/22/2019 3:50 PM	File folder
254	12/22/2019 3:48 PM	File folder
255	12/22/2019 3:47 PM	File folder

Boundry :



Name	Date modified	Type
aha.pickle	12/22/2019 6:02 PM	PICKLE File

WHY ALL THIS NESTING ?

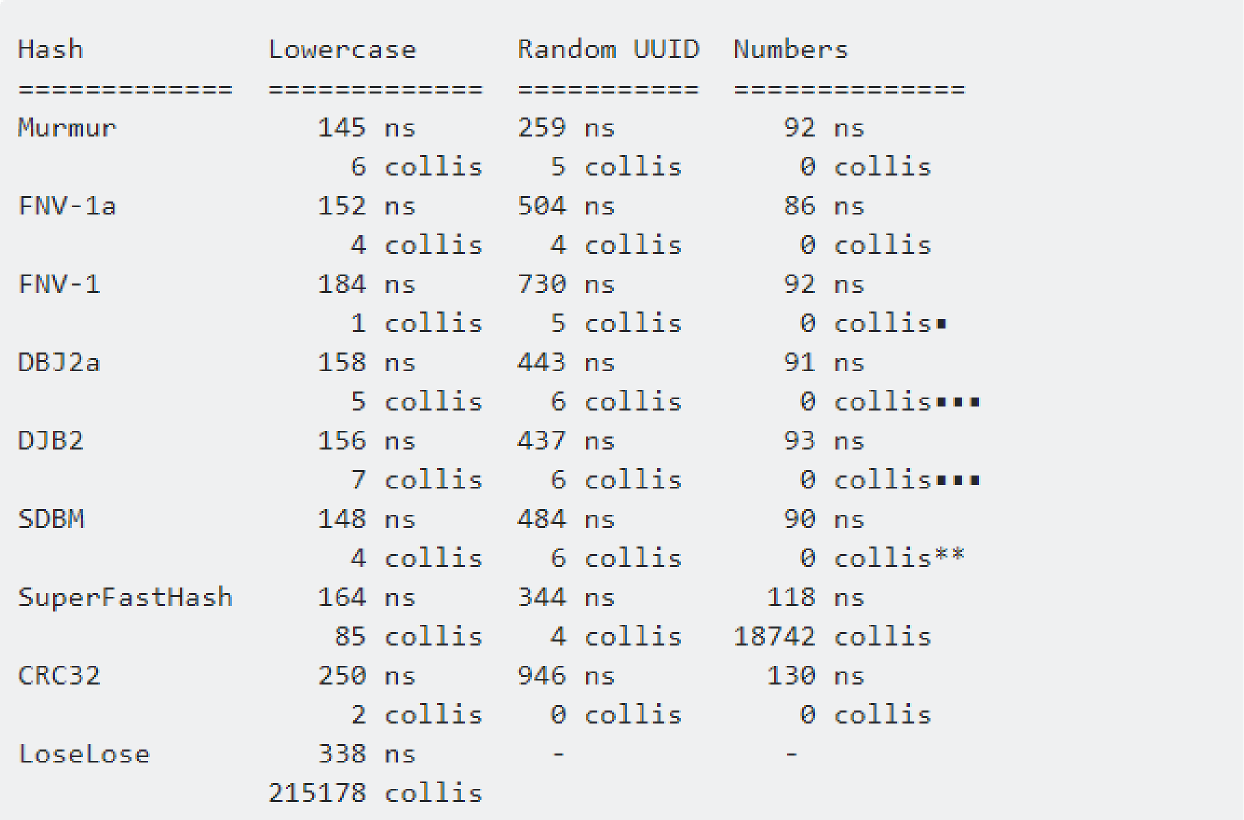
In window OS directory structure is a Tree. The time it took OS to search a query name handled to it by our Program (in this caase the name of our pickle file) is not actually O (1) even if wee know the complete path of file.

It depends on two factors the depth of node holding our required file and the number of childrens of it's parent node. The files in directory structure are actually leaf nodes, The time it takes OS to find a leaf node is O (n).

Since there could be tens of thousands of nodes in one directory then searching for node that is end of list could be expensive.

This nested folder structure reduces the number of file nodes in any folder node, hence minimizing load time.

Why chose Mur-Mur Hash :



```
Hash           Lowercase      Random UUID  Numbers
=============  =============  ===========  ==============
Murmur            145 ns      259 ns          92 ns
                    6 collis    5 collis       0 collis
FNV-1a            152 ns      504 ns          86 ns
                    4 collis    4 collis       0 collis
FNV-1             184 ns      730 ns          92 ns
                    1 collis    5 collis       0 collis▪
DBJ2a             158 ns      443 ns          91 ns
                    5 collis    6 collis       0 collis▪▪▪
DJB2              156 ns      437 ns          93 ns
                    7 collis    6 collis       0 collis▪▪▪
SDBM              148 ns      484 ns          90 ns
                    4 collis    6 collis       0 collis**
SuperFastHash     164 ns      344 ns         118 ns
                   85 collis    4 collis   18742 collis
CRC32             250 ns      946 ns         130 ns
                    2 collis    0 collis       0 collis
LoseLose          338 ns        -              -
               215178 collis
```

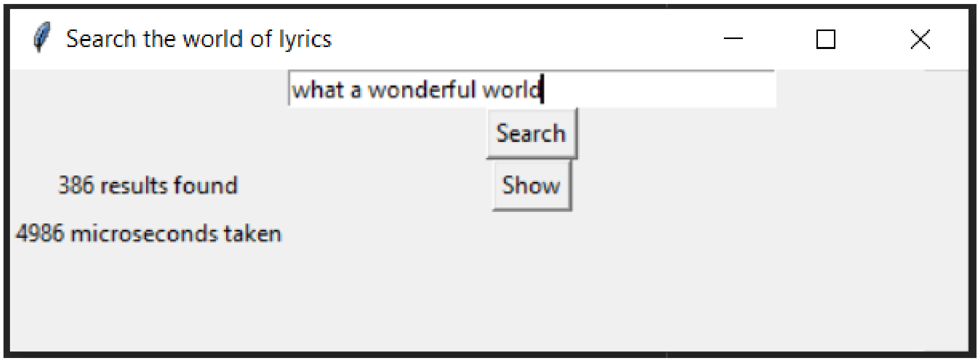
Why no just build a lexicon and use word ID as key word in hashing, it would be fast as seen above ?

Numbers are not hashable.

could use str (word_id) ?

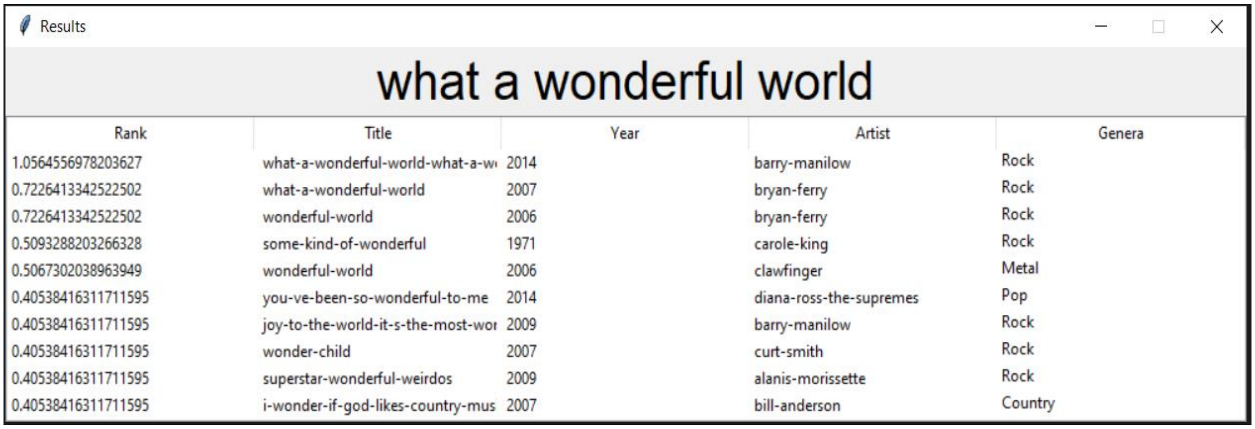
It would result in clustering of files within a folder, hence we won't be getting proper benefit from hashing. Greater the number of character in word we are hashing lesser the number of collisions, and wider the range of hashed positions.

Speed PERFORMANCE and Result relevance:



386 results found

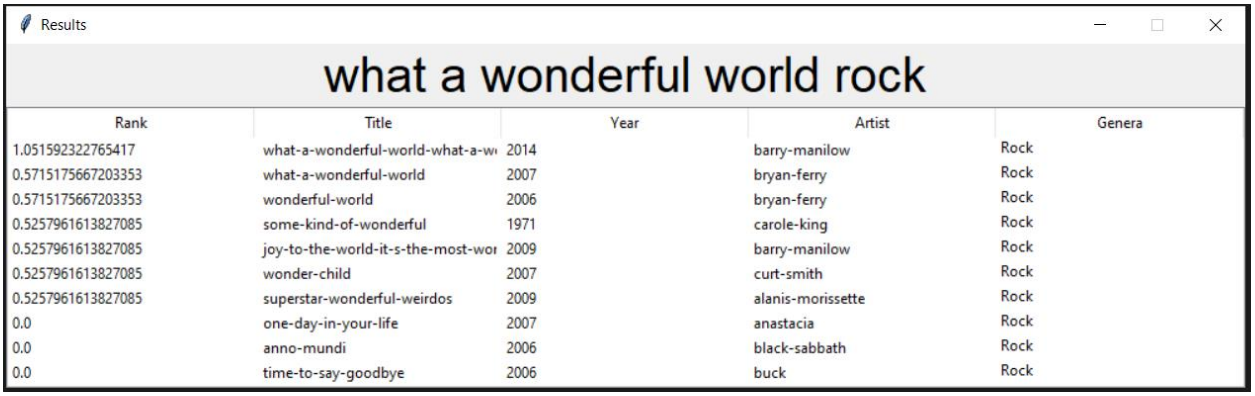
4986 microseconds taken


Results

what a wonderful world

Rank	Title	Year	Artist	Genera
1.0564556978203627	what-a-wonderful-world-what-a-w	2014	barry-manilow	Rock
0.7226413342522502	what-a-wonderful-world	2007	bryan-ferry	Rock
0.7226413342522502	wonderful-world	2006	bryan-ferry	Rock
0.5093288203266328	some-kind-of-wonderful	1971	carole-king	Rock
0.5067302038963949	wonderful-world	2006	clawfinger	Metal
0.40538416311711595	you-ve-been-so-wonderful-to-me	2014	diana-ross-the-supremes	Pop
0.40538416311711595	joy-to-the-world-it-s-the-most-wor	2009	barry-manilow	Rock
0.40538416311711595	wonder-child	2007	curt-smith	Rock
0.40538416311711595	superstar-wonderful-weirdos	2009	alanis-morissette	Rock
0.40538416311711595	i-wonder-if-god-likes-country-mus	2007	bill-anderson	Country

TO FILTER BY TITLE OR YEAR OR GENRE JUST TYPE it in QUERY



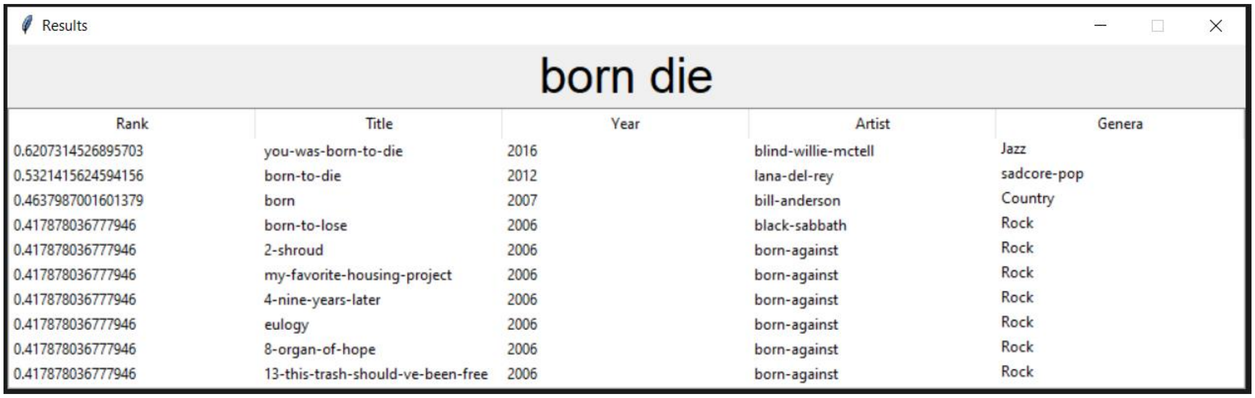
what a wonderful world rock

Rank	Title	Year	Artist	Genera
1.051592322765417	what-a-wonderful-world-what-a-w	2014	barry-manilow	Rock
0.5715175667203353	what-a-wonderful-world	2007	bryan-ferry	Rock
0.5715175667203353	wonderful-world	2006	bryan-ferry	Rock
0.5257961613827085	some-kind-of-wonderful	1971	carole-king	Rock
0.5257961613827085	joy-to-the-world-it-s-the-most-wor	2009	barry-manilow	Rock
0.5257961613827085	wonder-child	2007	curt-smith	Rock
0.5257961613827085	superstar-wonderful-weirdos	2009	alanis-morissette	Rock
0.0	one-day-in-your-life	2007	anastacia	Rock
0.0	anno-mundi	2006	black-sabbath	Rock
0.0	time-to-say-goodbye	2006	buck	Rock

WHAT ARE ZERO RANKED RESULTS in ABOVE SNIPPET ?

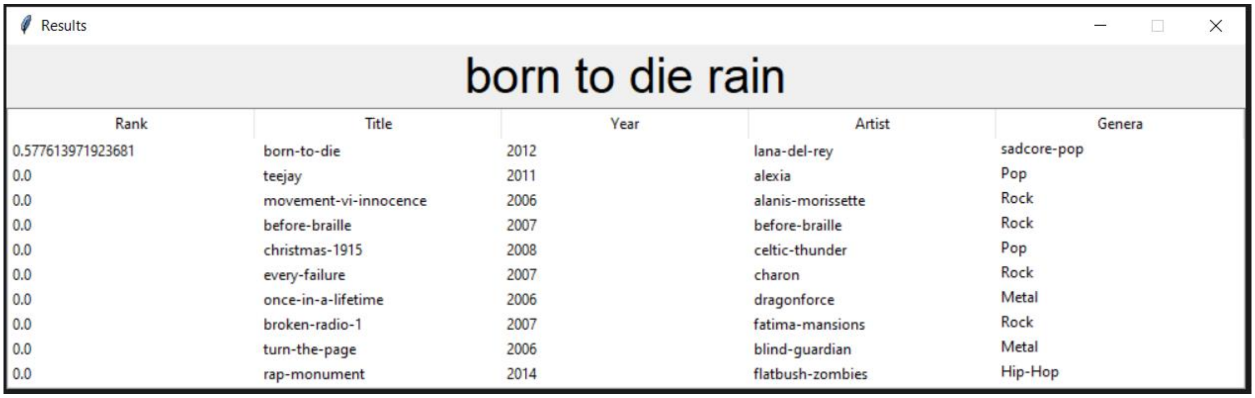
These are the documents do not contain entire phrase as a whole instead they have parts of it scattered in document

SEE THE CHANGED RANKING WITH CHANGED QUERY :



born die

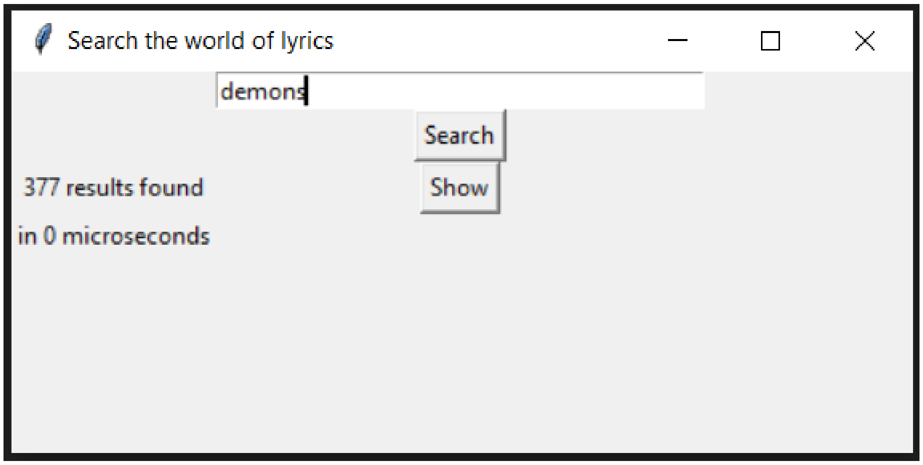
Rank	Title	Year	Artist	Genera
0.6207314526895703	you-was-born-to-die	2016	blind-willie-mctell	Jazz
0.5321415624594156	born-to-die	2012	lana-del-rey	sadcore-pop
0.4637987001601379	born	2007	bill-anderson	Country
0.417878036777946	born-to-lose	2006	black-sabbath	Rock
0.417878036777946	2-shroud	2006	born-against	Rock
0.417878036777946	my-favorite-housing-project	2006	born-against	Rock
0.417878036777946	4-nine-years-later	2006	born-against	Rock
0.417878036777946	eulogy	2006	born-against	Rock
0.417878036777946	8-organ-of-hope	2006	born-against	Rock
0.417878036777946	13-this-trash-should-ve-been-free	2006	born-against	Rock



born to die rain

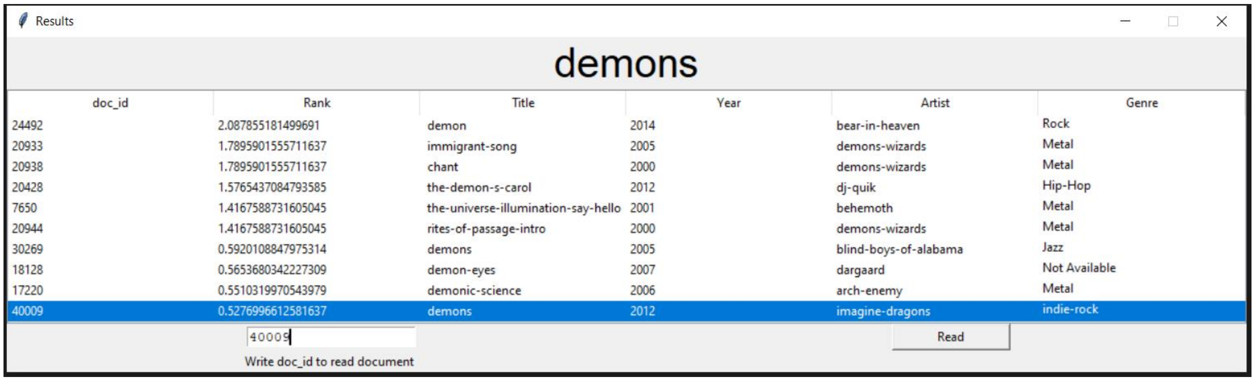
Rank	Title	Year	Artist	Genera
0.577613971923681	born-to-die	2012	lana-del-rey	sadcore-pop
0.0	teejay	2011	alexia	Pop
0.0	movement-vi-innocence	2006	alanis-morissette	Rock
0.0	before-braille	2007	before-braille	Rock
0.0	christmas-1915	2008	celtic-thunder	Pop
0.0	every-failure	2007	charon	Rock
0.0	once-in-a-lifetime	2006	dragonforce	Metal
0.0	broken-radio-1	2007	fatima-mansions	Rock
0.0	turn-the-page	2006	blind-guardian	Metal
0.0	rap-monument	2014	flatbush-zombies	Hip-Hop

SHOW DOC :



377 results found

in 0 microseconds

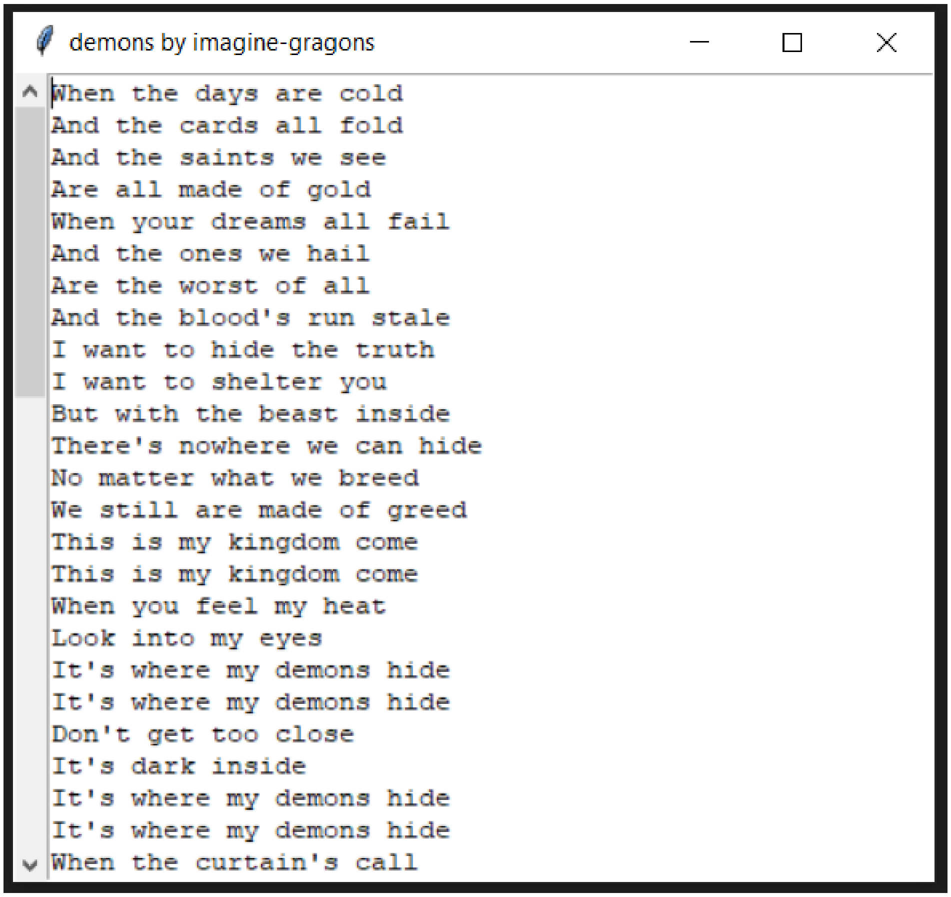


demons

doc_id	Rank	Title	Year	Artist	Genre
24492	2.087855181499691	demon	2014	bear-in-heaven	Rock
20933	1.7895901555711637	immigrant-song	2005	demons-wizards	Metal
20938	1.7895901555711637	chant	2000	demons-wizards	Metal
20428	1.5765437084793585	the-demon-s-carol	2012	dj-quik	Hip-Hop
7650	1.4167588731605045	the-universe-illumination-say-hello	2001	behemoth	Metal
20944	1.4167588731605045	rites-of-passage-intro	2000	demons-wizards	Metal
30269	0.5920108847975314	demons	2005	blind-boys-of-alabama	Jazz
18128	0.5653680342227309	demon-eyes	2007	dargaard	Not Available
17220	0.5510319970543979	demonic-science	2006	arch-enemy	Metal
40009	0.5276996612581637	demons	2012	imagine-dragons	indie-rock

40009

Write doc_id to read document



```
When the days are cold
And the cards all fold
And the saints we see
Are all made of gold
When your dreams all fail
And the ones we hail
Are the worst of all
And the blood's run stale
I want to hide the truth
I want to shelter you
But with the beast inside
There's nowhere we can hide
No matter what we breed
We still are made of greed
This is my kingdom come
This is my kingdom come
When you feel my heat
Look into my eyes
It's where my demons hide
It's where my demons hide
Don't get too close
It's dark inside
It's where my demons hide
It's where my demons hide
When the curtain's call
```