Delta Stepping Algorithm in Parallel : A beginner approach. Documentation

Release 0.0.1

Ivan Felipe Rodriguez Rodriguez

May 16, 2016

CONTENTS

1 Introduction **3**

1.1 Testing on the shell. 3
1.2 Search(): 4
1.3 The Attribute _pw_. 4
1.4 Searching by field: 5
1.5 add_wildcards and something 6
1.6 The output dictionary 7
1.7 use_dict: 7

2 Indices and tables **9**

Contents:

CHAPTER

ONE

INTRODUCTION

After you have registered the indexes where you want to store the searcheables. You can perform some testing by entering to the shell of your aplication:

```
$ python app.py shell
```

However if you'd rather prefer to "see" the results, you may access to the route '/ponywhoosh/' available when you run the server.

1.1 Testing on the shell.

.. code :: python

    PonyModel.search(query, **kwargs)

There are several options to perform a search with `flask_ponywhoosh`. For instance, to execute a simple search, choose the entity where you want to perform the search and then try something like the following code over a view function, or even from the shell,

```
>>> from app import *
>>> User.search("felipe")
{'cant_results': 2,
 'facet_names': [],
 'matched_terms': {'name': ['felipe']},
 'results': [{'docnum': 4L,
              'rank': 0,
              'pk': 5,
              'score': 2.540445040947149},
             {'docnum': 11L,
              'rank': 1,
              'pk': 12,
              'score': 2.540445040947149}],
 'runtime': 0.001981973648071289}
```

If you would prefer, you may use the function `search()`, which will run the same function but is quite more handy when writing

```
>>> from flask_ponywhoosh import search
>>> from app import *
>>> search(User,"felipe")
{'cant_results': 2,
 'facet_names': [],
 'matched_terms': {'name': ['felipe']},
 'results': [{'docnum': 4L,
```

```
              'rank': 0,
              'pk' : 5,
              'score': 2.540445040947149},
             {'docnum': 11L,
              'rank': 1,
              'pk' : 12,
              'score': 2.540445040947149}],
 'runtime': 0.001981973648071289}
```

1.2 Search():

The function `search()` takes up to three arguments. 1. A ponymodel, the databse entity where you want to perform the search. 2. The `search_string`, what you are looking for; and, 3. The arguments, some additional options for more refined searching.

```
search(PonyModel, "query", **kw)
```

For example, if you want the results to be sorted by some specific searcheable field, you have to indicate so, by adding the argument `sortedby="field"`.

In this case the search results object would show as a score the value of the item you choose for sorting. Please note that in order for one field to be sortable, you must indicate it when you are registering the model. (Refer to the *Usage* section above)

```
>>> from app import *
>>> from flask_ponywhoosh import search
>>> search(User,"harol", sortedby="age")
{'cant_results': 2,
 'facet_names': [],
 'matched_terms': {'name': ['felipe']},
 'results': [{'docnum': 4L,
              'rank': 0,
              'pk' : 5,,
              'score': '19'},
             {'docnum': 11L,
              'rank': 1,
              'pk' : 12,,
              'score': '19'}],
 'runtime': 0.0012810230255126953}
```

In synthesis, the options available are: `sortedby`, `scored`, `limit`, `optimize`, `reverse`. Which are widely described in the whoosh documentation.

1.3 The Attribute `_pw_`.

There are some special features avalaible for models from the database. You just have to call the model `PonyModel._pw_.`:

- `add_field`: This function is to add a desired field in the index.
- `charge_documents`: This function let you charge an index from an existing database.
- `delete_documents`: This function deletes all the documents stored in certain whoosh index.
- `delete_field`: This function works in case that you want to erase a determined field from a schema.

- `update_documents`: This function deletes all the documents and recharges them again.
- `counts`: This function counts all the documents existing in an index.

1.4 Searching by field:

.. code:: python

    search(PonyModel, query, field="field_name")

By default the function `search()` performs a multifield parser query, i.e. you will be searching in all the fields you have declared when you registered the model. However, sometimes you would like to perform searching in just one or some of all the fields. For these reasons we implemented the following extra options: The first one is refered as `field` all you have to do is indicate in which field you want to search. The output would be a results object containing only the information found in that field. And `fields` where you should write a list with all the fields you want to search.

```
>>> search(User,"harol",field="name")
     {'cant_results': 4,
     'facet_names': [],
     'matched_terms': {'name': ['harol']},
     'results': [{'docnum': 1L,
                  'pk': u'7',
                  'rank': 0,
                  'score': 2.0296194171811583},
                 {'docnum': 5L,
                  'pk': u'6',
                  'rank': 1,
                  'score': 2.0296194171811583},
                 {'docnum': 12L,
                  'pk': u'13',
                  'rank': 2,
                  'score': 2.0296194171811583},
                 {'docnum': 13L,
                  'pk': u'14',
                  'rank': 3,
                  'score': 2.0296194171811583}],
     'runtime': 0.005359172821044922}

>>> search(Attribute,"tejo", fields=["sport","name"])
    {'cant_results': 4,
     'facet_names': [],
     'matched_terms': {'name': ['tejo'], 'sport': ['tejo']},
     'results': [{'docnum': 1L,
                  'pk': u'7',
                  'rank': 0,
                  'score': 5.500610730717037},
                 {'docnum': 6L,
                  'pk': u'1',
                  'rank': 1,
                  'score': 5.500610730717037}],
     'runtime': 0.006212949752807617}
```

1.5 add_wildcards and something

```
search(PonyModel, query, add_wildcards=True)
```

Whoosh sets a wildcard *,‘‘?‘‘,‘‘!‘‘ by default to perform search for inexact terms, however sometimes is desirable to search by exact terms instead. For this reason we added two more options: `add_wildcards` and `something`.

The option *add_wildcards* (by default False) is a boolean argument that tells the searcher whether it should or not include wild cards. For example, if you want to search “harol” when `add_wildcards=False`, and you search by “har” the results would be 0. If `add_wildcards=True` , then “har” would be fair enough to get the result “harol” because searching was performed using wild cards.

```
>>> search(User, "har", add_wildcards=False)
  {'cant_results': 0,
   'facet_names': [],
   'matched_terms': {},
   'results': [],
   'runtime': 0.0003230571746826172
   }

>>> search(User, "har", add_wildcards=True)
  {'cant_results': 4,
   'facet_names': [],
   'matched_terms': {'name': ['harol']},
   'results': [{'docnum': 1L,
                'pk': u'7',
                'rank': 0,
                'score': 2.0296194171811583},
               {'docnum': 5L,
                'pk': u'6',
                'rank': 1,
                'score': 2.0296194171811583},
               {'docnum': 12L,
                'pk': u'13',
                'rank': 2,
                'score': 2.0296194171811583},
               {'docnum': 13L,
                'pk': u'14',
                'rank': 3,
                'score': 2.0296194171811583}],
   'runtime': 0.014926910400390625}
```

The `something=True` option, would run first a search with `add_wildcards=False` value, but in case results are empty it would automatically run a search adding wildcards to the result.

```
>>> search(Attribute, "tejo", something = True)
  {'cant_results': 4,
   'facet_names': [],
   'matched_terms': {'name': ['tejo'], 'sport': ['tejo']},
   'results': [{'docnum': 1L,
                'pk': u'7',
                'rank': 0,
                'score': 5.500610730717037},
               {'docnum': 6L,
                'pk': u'1',
                'rank': 1,
                'score': 5.500610730717037}],
   'runtime': 0.0036530494689941406}
```

1.6 The output dictionary

The `search()` function returns a dictionary with selected information.

- `cant_results`: is the total number of documents collected by the searcher.
- `facet_names`: is useful with the option `groupedby`, because it returns the item used to group the results.
- `matched_terms`: is a dictionary that saves the searcheable field and the match given by the query.
- `runtime`: how much time the searcher took to find it.
- `results`: is a dictionary's list for the individual results. i.e. a dictionary for every single result, containing:
  - 'rank': the position of the result,
  - 'result': indicating the primary key and the correspond value of the item,
  - 'score': the score for the item in the search, and
  - 'pk': the primary key Or the sets of primary keys.

1.7 use_dict:

If you want that the items look like a list rather than a dictionary. You can use the option use_dict: this option by default is set True. However if you choose false, results will look something like ('field', 'result')

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search