Search_engine_PII

Search engine using Positional Inverted Index & Free-text Queries with Proximity Operator
Matthieu Vilain

Requirements

- Python 3
- nltk (pip install nltk)
- BeautifulSoup 4 (apt install python3-bs4)

Project organisation

```
.
├── data
│   ├── documents.txt
│   └── wiki.txt
│
├── dictionary
│   ├── dictionary.pickle
│   └── dictionary.txt
│
├── doc
│   └── HW2.pdf
│   └── README.pdf
│
├── dictionary.py
├── query.py
├── search.py
└── README.md
```

How to use the PPI ?

Search

You can search very quickly with the commande :

```
python search.py "your 0(positional query)"
```

You can also add arguments for additional functionality :

- -b, --build : build the dictionary (otherwise the dictionary is load from a file)
- -s, --save : save the dictionary in a pickle file. Change the variable PATH_TO_DICTIONARY in search.py if you want to change the destination
- -t, --time : print the execution time for the different component

```
python search.py "your 0(positional query)" -b -s -t
```

Positional Inverted Index creation

1. Create a file with all your documents between `<DOC>...</DOC>`
2. Change the variable PATH_TO_DOCUMENTS in search.py
3. Run search.py with the argument --build (or -b) to build a new dictionary from your documents. You can save you dictionary using --save (or -s) argument

Some results

1. **"nexus like love happy"**

   ```
   No result
   ```

2. **"asus repair"**

   ```
   #1    docID: 7    tf-idf score: 1.5873253084721661
   #2    docID: 2    tf-idf score: 1.2218487496163566
   ```

3. **"0(touch screen) fix repair"**

   ```
   #1    docID: 2    tf-idf score: 2.4304034584882657
   ```

4. **"1(great tablet) 2(tablet fast)"**

   ```
   No result
   ```

5. **"tablet"**

   ```
   #1    docID: 3    tf-idf score: 0.2886318777514278
   #2    docID: 1    tf-idf score: 0.22184874961635637
   #3    docID: 2    tf-idf score: 0.22184874961635637
   #4    docID: 5    tf-idf score: 0.22184874961635637
   #5    docID: 8    tf-idf score: 0.22184874961635637
   #6    docID: 9    tf-idf score: 0.22184874961635637
   ```

Reflexion

For the query 1 an 4 we have no result, but for query 4 for exemple we have the following posting list :

```
1(great tablet)      Posting list: []
2(tablet fast)      Posting list: [1]
```

Because we take the intersection of the posting lists we have no result. But if we take the union we have 1 resulting document.
Because be have a scoring function it's more interesting to have to much result than too few.

Here are the result of the previous query using UNION of the posting lists.

1. **"nexus like love happy"**

```
#1    docID: 1    tf-idf score: 2.0782209403769034
#2    docID: 5    tf-idf score: 1.3792509360408844
#3    docID: 7    tf-idf score: 1.2218487496163566
#4    docID: 0    tf-idf score: 1.0457574905606752
#5    docID: 3    tf-idf score: 0.6989700043360189
```

2. **"asus repair"**

```
#1    docID: 7    tf-idf score: 1.5873253084721661
#2    docID: 2    tf-idf score: 1.2218487496163566
#3    docID: 0    tf-idf score: 0.6802809317048656
```

3. **"0(touch screen) fix repair"**

```
#1    docID: 2    tf-idf score: 2.4304034584882657
#2    docID: 7    tf-idf score: 2.3073209503825813
```

4. **"1(great tablet) 2(tablet fast)"**

```
#1    docID: 1    tf-idf score: 1.2218487496163566
```

5. **"tablet"**

```
#1    docID: 3    tf-idf score: 0.2886318777514278
#2    docID: 1    tf-idf score: 0.22184874961635637
#3    docID: 2    tf-idf score: 0.22184874961635637
#4    docID: 5    tf-idf score: 0.22184874961635637
#5    docID: 8    tf-idf score: 0.22184874961635637
#6    docID: 9    tf-idf score: 0.22184874961635637
```

We can see that it doesn't affect the order of the resulting docID. The documents that are in multiple posting lists are better ranked than thoses that appears in only one list