dict-trie Documentation

Release latest

Apr 01, 2022

Contents:

This library provides a trie implementation using nested dictionaries. Apart from the basic operations, a number of functions for *approximate matching* are implemented.

Please see ReadTheDocs for the latest documentation.

CHAPTER 1

Installation

The software is distributed via PyPI, it can be installed with `pip`:

```
pip install dict-trie
```

1.1 From source

The source is hosted on GitHub, to install the latest development version, use the following commands.

```
git clone https://github.com/jfjlaros/dict-trie.git
cd dict-trie
pip install .
```

CHAPTER 2

Usage

The library provides the `Trie` class.

```
>>> from dict_trie import Trie
```

2.1 Basic operations

Initialisation of the trie is done via the constructor by providing a list of words.

```
>>> trie = Trie(['abc', 'te', 'test'])
```

Alternatively, an empty trie can be made to which words can be added with the `add` function.

```
>>> trie = Trie()
>>> trie.add('abc')
>>> trie.add('te')
>>> trie.add('test')
```

Membership can be tested with the `in` statement.

```
>>> 'abc' in trie
True
```

Test whether a prefix is present by using the `has_prefix` function.

```
>>> trie.has_prefix('ab')
True
```

Remove a word from the trie with the `remove` function. This function returns `False` if the word was not in the trie.

```
>>> trie.remove('abc')
True
```

(continues on next page)

(continued from previous page)

```
>>> 'abc' in trie
False
>>> trie.remove('abc')
False
```

Iterate over all words in a trie.

```
>>> list(trie)
['abc', 'te', 'test']
```

2.2 Approximate matching

A trie can be used to efficiently find a word that is similar to a query word. This is implemented via a number of functions that search for a word, allowing a given number of mismatches. These functions are divided in two families, one using the Hamming distance which only allows substitutions, the other using the Levenshtein distance which allows substitutions, insertions and deletions.

To find a word that has at most Hamming distance 2 to the word 'abe', the `hamming` function is used.

```
>>> trie = Trie(['abc', 'aaa', 'ccc'])
>>> trie.hamming('abe', 2)
'aaa'
```

To get all words that have at most Hamming distance 2 to the word 'abe', the `all_hamming` function is used. This function returns a generator.

```
>>> list(trie.all_hamming('abe', 2))
['aaa', 'abc']
```

In order to find a word that is closest to the query word, the `best_hamming` function is used. In this case a word with distance 1 is returned.

```
>>> trie.best_hamming('abe', 2)
'abc'
```

The functions `levenshtein`, `all_levenshtein` and `best_levenshtein` are used in a similar way.

2.3 Other functionalities

A trie can be populated with all words of a fixed length over an alphabet by using the `fill` function.

```
>>> trie = Trie()
>>> trie.fill(('a', 'b'), 2)
>>> list(trie)
['aa', 'ab', 'ba', 'bb']
```

The trie data structure can be accessed via the `root` member variable.

```
>>> trie.root
{'a': {'a': {'': 1}, 'b': {'': 1}}, 'b': {'a': {'': 1}, 'b': {'': 1}}}
>>> trie.root.keys()
['a', 'b']
```

The distance functions `all_hamming` and `all_levenshtein` also have counterparts that give the developer more information by returning a list of tuples containing not only the matched word, but also its distance to the query string and a CIGAR-like string.

The following encoding is used in the CIGAR-like string:

character	description
=	match
X	mismatch
I	insertion
D	deletion

In the following example, we search for all words with Hamming distance 1 to the word 'acc'. In the results we see a match with the word 'abc' having distance 1 and a mismatch at position 2.

```
>>> trie = Trie(['abc'])
>>> list(trie.all_hamming_('acc', 1))
[('abc', 1, '=X=')]
```

Similarly, we can search for all words having Levenshtein distance 2 to the word 'acb'. The word 'abc' matches three times, once by deleting the 'b' on position 2 and inserting a 'b' after position 3, once by inserting a 'c' after position 1 and deleting the last character and once by introducing two mismatches.

```
>>> list(trie.all_levenshtein_('acb', 2))
[('abc', 2, '=D=I'), ('abc', 2, '=XX'), ('abc', 2, '=I=D')]
```

CHAPTER 3

Contributors

- Jeroen F.J. Laros <J.F.J.Laros@lumc.nl> (Original author, maintainer)

Find out who contributed:

```
git shortlog -s -e
```