
PyTrie Documentation

Release 0.3

George Sakkis

December 06, 2016

1 Usage	3
2 Reference documentation	5
2.1 Classes	5
2.2 Trie methods	5
2.3 Extended mapping API methods	6
2.4 Original mapping API methods	7
2.5 Internals	7
Python Module Index	9

`pytrie` is a pure Python implementation of the `trie` (prefix tree) data structure.

A *trie* is a tree data structure that is used to store a mapping where the keys are sequences, usually strings over an alphabet. In addition to implementing the mapping interface, tries facilitate finding the items for a given prefix, and vice versa, finding the items whose keys are prefixes of a given key K . As a common special case, finding the longest-prefix item is also supported.

Algorithmically, tries are more efficient than binary search trees (BSTs) both in lookup time and memory when they contain many keys sharing relatively few prefixes. Unlike hash tables, trie keys don't need to be hashable. In the current implementation, a key can be any finite iterable with hashable elements.

Usage

```
>>> from pytrie import SortedStringTrie as Trie
>>> t = Trie(an=0, ant=1, all=2, allot=3, alloy=4, aloe=5, are=6, be=7)
>>> t
SortedStringTrie({'all': 2, 'allot': 3, 'alloy': 4, 'aloe': 5, 'an': 0, 'ant': 1, 'are': 6, 'be': 7})
>>> t.keys(prefix='al')
['all', 'allot', 'alloy', 'aloe']
>>> t.items(prefix='an')
[('an', 0), ('ant', 1)]
>>> t.longest_prefix('antonym')
'ant'
>>> t.longest_prefix_item('allstar')
('all', 2)
>>> t.longest_prefix_value('area', default='N/A')
6
>>> t.longest_prefix('alsa')
Traceback (most recent call last):
...
KeyError
>>> t.longest_prefix_value('alsa', default=-1)
-1
>>> list(t.iter_prefixes('allotment'))
['all', 'allot']
>>> list(t.iter_prefix_items('antonym'))
[('an', 0), ('ant', 1)]
```

Reference documentation

2.1 Classes

class `pytrie.Trie (*args, **kwargs)`
Bases: `_abcoll.MutableMapping`

Base trie class.

As with regular dicts, keys are not necessarily returned sorted. Use `SortedTrie` if sorting is required.

KeyFactory
alias of `tuple`

NodeFactory
alias of `Node`

__init__ (`*args, **kwargs`)
Create a new trie.

Parameters are the same with `dict()`.

classmethod fromkeys (`iterable, value=None`)
Create a new trie with keys from `iterable` and values set to `value`.

Parameters are the same with `dict.fromkeys()`.

class `pytrie.StringTrie (*args, **kwargs)`
Bases: `pytrie.Trie`

A more appropriate for string keys `Trie`.

class `pytrie.SortedTrie (*args, **kwargs)`
Bases: `pytrie.Trie`

A `Trie` that returns its keys (and associated values/items) sorted.

class `pytrie.SortedStringTrie (*args, **kwargs)`
Bases: `pytrie.SortedTrie, pytrie.StringTrie`

A `Trie` that is both a `StringTrie` and a `SortedTrie`

2.2 Trie methods

The following methods are specific to tries; they are not part of the mapping API.

`Trie.longest_prefix(key[, default])`

Return the longest key in this trie that is a prefix of `key`.

If the trie doesn't contain any prefix of key:

- if `default` is given, return it
- otherwise raise `KeyError`

`Trie.longest_prefix_value(key[, default])`

Return the value associated with the longest key in this trie that is a prefix of `key`.

If the trie doesn't contain any prefix of key:

- if `default` is given, return it
- otherwise raise `KeyError`

`Trie.longest_prefix_item(key[, default])`

Return the item ((`key`, `value`) tuple) associated with the longest key in this trie that is a prefix of `key`.

If the trie doesn't contain any prefix of key:

- if `default` is given, return it
- otherwise raise `KeyError`

`Trie.iter_prefixes(key)`

Return an iterator over the keys of this trie that are prefixes of `key`.

`Trie.iter_prefix_values(key)`

Return an iterator over the values of this trie that are associated with keys that are prefixes of `key`.

`Trie.iter_prefix_items(key)`

Return an iterator over the items ((`key`, `value`) tuples) of this trie that are associated with keys that are prefixes of `key`.

2.3 Extended mapping API methods

The following methods extend the respective mapping API methods with an optional `prefix` parameter. If not `None`, only keys (or associated values/items) that start with `prefix` are returned.

`Trie.keys(prefix=None)`

Return a list of this trie's keys.

Parameters `prefix` – If not `None`, return only the keys prefixed by `prefix`.

`Trie.values(prefix=None)`

Return a list of this trie's values.

Parameters `prefix` – If not `None`, return only the values associated with keys prefixed by `prefix`.

`Trie.items(prefix=None)`

Return a list of this trie's items ((`key`, `value`) tuples).

Parameters `prefix` – If not `None`, return only the items associated with keys prefixed by `prefix`.

`Trie.iterkeys(prefix=None)`

Return an iterator over this trie's keys.

Parameters `prefix` – If not `None`, yield only the keys prefixed by `prefix`.

`Trie.itervalues` (*prefix=None*)

Return an iterator over this trie's values.

Parameters `prefix` – If not None, yield only the values associated with keys prefixed by `prefix`.

`Trie.iteritems` (*prefix=None*)

Return an iterator over this trie's items ((`key`, `value`) tuples).

Parameters `prefix` – If not None, yield only the items associated with keys prefixed by `prefix`.

2.4 Original mapping API methods

The following methods have the standard mapping signature and semantics.

`Trie.__len__`()

`Trie.__nonzero__`()

`Trie.__iter__`()

`Trie.__contains__`(*key*)

`Trie.__getitem__`(*key*)

`Trie.__setitem__`(*key*, *value*)

`Trie.__delitem__`(*key*)

`Trie.__cmp__`(*other*)

`Trie.__repr__`()

`Trie.clear`()

`Trie.copy`()

`Trie.has_key`(*key*)

2.5 Internals

Tries are implemented as trees of *Node* instances. You don't need to worry about them unless you want to extend or replace *Node* with a new node factory and bind it to *Trie.NodeFactory*.

class `pytrie.Node`

Bases: `object`

Trie node class.

Subclasses may extend it to replace *ChildrenFactory* with a different mapping class (e.g. `sorteddict`)

Variables

- **value** – The value of the key corresponding to this node or NULL if there is no such key.
- **children** – A {key-part : child-node} mapping.

ChildrenFactory

alias of `dict`

numkeys()

Return the number of keys in the subtree rooted at this node.

p

pytrie, 3

Symbols

`__cmp__()` (pytrie.Trie method), 7
`__contains__()` (pytrie.Trie method), 7
`__delitem__()` (pytrie.Trie method), 7
`__getitem__()` (pytrie.Trie method), 7
`__init__()` (pytrie.Trie method), 5
`__iter__()` (pytrie.Trie method), 7
`__len__()` (pytrie.Trie method), 7
`__nonzero__()` (pytrie.Trie method), 7
`__repr__()` (pytrie.Trie method), 7
`__setitem__()` (pytrie.Trie method), 7

C

ChildrenFactory (pytrie.Node attribute), 7
`clear()` (pytrie.Trie method), 7
`copy()` (pytrie.Trie method), 7

F

`fromkeys()` (pytrie.Trie class method), 5

H

`has_key()` (pytrie.Trie method), 7

I

`items()` (pytrie.Trie method), 6
`iter_prefix_items()` (pytrie.Trie method), 6
`iter_prefix_values()` (pytrie.Trie method), 6
`iter_prefixes()` (pytrie.Trie method), 6
`iteritems()` (pytrie.Trie method), 7
`iterkeys()` (pytrie.Trie method), 6
`itervalues()` (pytrie.Trie method), 6

K

KeyFactory (pytrie.Trie attribute), 5
`keys()` (pytrie.Trie method), 6

L

`longest_prefix()` (pytrie.Trie method), 5
`longest_prefix_item()` (pytrie.Trie method), 6
`longest_prefix_value()` (pytrie.Trie method), 6

N

Node (class in pytrie), 7
NodeFactory (pytrie.Trie attribute), 5
`numkeys()` (pytrie.Node method), 7

P

pytrie (module), 1

S

SortedStringTrie (class in pytrie), 5
SortedTrie (class in pytrie), 5
StringTrie (class in pytrie), 5

T

Trie (class in pytrie), 5

V

`values()` (pytrie.Trie method), 6