marisa-trie Documentation

Release 0.7.5

Mikhail Korobov

Apr 10, 2018

Contents

1 Installation **3**

2 Usage **5**

3 Current limitations **7**

4 License **9**

4.1 Tutorial 9

4.2 Benchmarks 12

4.3 API reference 14

4.4 Contributing 14

4.5 CHANGES 14

Static memory-efficient Trie-like structures for Python (2.7 and 3.4+) based on marisa-trie C++ library.

String data in a MARISA-trie may take up to 50x-100x less memory than in a standard Python dict; the raw lookup speed is comparable; trie also provides fast advanced methods like prefix search.

Note: There are official SWIG-based Python bindings included in C++ library distribution; this package provides alternative Cython-based pip-installable Python bindings.

CHAPTER 1

Installation

```
pip install marisa-trie
```

CHAPTER 2

Usage

See *Tutorial* and *API* for details.

CHAPTER 3

Current limitations

- The library is not tested with mingw32 compiler;
- `.prefixes()` method of `BytesTrie` and `RecordTrie` is quite slow and doesn't have iterator counterpart;
- `read()` and `write()` methods don't work with file-like objects (they work only with real files; pickling works fine for file-like objects);
- there are `keys()` and `items()` methods but no `values()` method.

CHAPTER 4

License

Wrapper code is licensed under MIT License.

Bundled marisa-trie C++ library is dual-licensed under LGPL and BSD 2-clause license.

4.1 Tutorial

4.1.1 Tries

There are several trie classes in this package:

`marisa_trie.BinaryTrie`
`marisa_trie.Trie`
`marisa_trie.RecordTrie`
`marisa_trie.BytesTrie`

marisa_trie.Trie

Create a new trie from a list of keys:

```
>>> import marisa_trie
>>> trie = marisa_trie.Trie([u'key1', u'key2', u'key12'])
```

Check if a key is present:

```
>>> u'key1' in trie
True
>>> u'key20' in trie
False
```

Each key is assigned an unique ID from 0 to (n - 1), where n is the number of keys in a trie:

```
>>> trie[u'key2']
1
```

Note that you can't assign a value to a `marisa_trie.Trie` key, but can use the returned ID to store values in a separate data structure (e.g. in a Python list or NumPy array).

An ID can be mapped back to the corresponding key:

```
>>> trie.restore_key(1)
u'key2'
```

Query a trie

- Find all trie keys which are prefixes of a given key:

```
>>> trie.prefixes(u'key12')
[u'key1', u'key12']
```

- Find all trie keys which start with a given prefix:

```
>> trie.keys(u'key1')
[u'key1', u'key12']
```

- The latter is complemented by `items()` which returns all matching `(key, ID)` pairs.

All query methods have generator-based versions prefixed with `iter`.

Note: If you're looking for a trie with bytes keys, check out `BinaryTrie`.

marisa_trie.RecordTrie

Create a new trie from a list of `(key, data)` pairs:

```
>>> keys = [u'foo', u'bar', u'foobar', u'foo']
>>> values = [(1, 2), (2, 1), (3, 3), (2, 1)]
>>> fmt = "<HH"   # two short integers.
>>> trie = marisa_trie.RecordTrie(fmt, zip(keys, values))
```

Each data tuple would be converted to bytes using `struct.pack()`. Take a look at available format strings here.

Check if a key is present:

```
>>> u'foo' in trie
True
>>> u'spam' in trie
False
```

`marisa_trie.RecordTrie` allows duplicate keys. Therefore `__getitem__` and `get` return a list of values.

```
>>> trie[u'bar']
[(2, 1)]
>>> trie[u'foo']
[(1, 2), (2, 1)]
>>> trie.get(u'bar', 123)
[(2, 1)]
```

(continues on next page)

(continued from previous page)

```
>>> trie.get(u'BAAR', 123)  # default value.
123
```

Similarly, `keys()` and `items()` take into account key multiplicities:

```
>> trie.keys(u'fo')
[u'foo', u'foo', u'foobar']
>> trie.items(u'fo')
[(u'foo', (1, 2)), (u'foo', (2, 1), (u'foobar', (3, 3))]
```

marisa_trie.BytesTrie

`BytesTrie` is similar to `RecordTrie`, but the values are raw bytes, not tuples:

```
>>> keys = [u'foo', u'bar', u'foobar', u'foo']
>>> values = [b'foo-value', b'bar-value', b'foobar-value', b'foo-value2']
>>> trie = marisa_trie.BytesTrie(zip(keys, values))
>>> trie[u'bar']
[b'bar-value']
```

4.1.2 Persistence

Trie objects supports saving/loading, pickling/unpickling and memory mapped I/O.

Save trie to a file:

```
>>> trie.save('my_trie.marisa')
```

Load trie from a file:

```
>>> trie2 = marisa_trie.Trie()
>>> trie2.load('my_trie.marisa')
```

Note: You may also build a trie using `marisa-build` command-line utility (provided by underlying C++ library; it should be downloaded and compiled separately) and then load the trie from the resulting file using `load`.

Trie objects are picklable:

```
>>> import pickle
>>> data = pickle.dumps(trie)
>>> trie3 = pickle.loads(data)
```

4.1.3 Memory mapped I/O

It is possible to use memory mapped file as data source:

```
>>> trie = marisa_trie.RecordTrie(fmt).mmap('my_record_trie.marisa')
```

This way the whole dictionary won't be loaded fully to memory; memory mapped I/O is an easy way to share dictionary data among processes.

Warning: Memory mapped trie might cause lots of random disk accesses which considerably increases the search time.

4.1.4 Storage options

marisa-trie C++ library provides some configuration options for trie storage; See "Enumeration Constants" section in the library docs.

These options are exposed as `order`, `num_tries`, `cache_size` and `binary` keyword arguments for trie constructors.

For example, set `order` to `marisa_trie.LABEL_ORDER` in order to make trie functions return results in alphabetical oder:

```
>>> trie = marisa_trie.RecordTrie(fmt, data, order=marisa_trie.LABEL_ORDER)
```

Note that two tries constructed from identical data but with different `order` arguments will compare unequal:

```
>>> t1 = marisa_trie.Trie(order=marisa_trie.LABEL_ORDER)
>>> t2 = marisa_trie.Trie(order=marisa_trie.WEIGHT_ORDER)
>>> t1 == t2
False
```

4.2 Benchmarks

My quick tests show that memory usage is quite decent. For a list of 3000000 (3 million) Russian words memory consumption with different data structures (under Python 2.7):

- dict(unicode words -> word lenghts): about 600M
- list(unicode words) : about 300M
- BaseTrie from datrie library: about 70M
- `marisa_trie.RecordTrie` : 11M
- `marisa_trie.Trie`: 7M

Note: Lengths of words were stored as values in `datrie.BaseTrie` and `marisa_trie.RecordTrie`. `RecordTrie` compresses similar values and the key compression is better so it uses much less memory than `datrie.BaseTrie`.

`marisa_trie.Trie` provides auto-assigned IDs. It is not possible to store arbitrary values in `marisa_trie.Trie` so it uses less memory than `RecordTrie`.

Benchmark results (100k unicode words, integer values (lenghts of the words), Python 3.2, macbook air i5 1.8 Ghz):

```
dict building                    2.919M words/sec
Trie building                    0.394M words/sec
BytesTrie building               0.355M words/sec
RecordTrie building              0.354M words/sec

dict __getitem__ (hits)          8.239M ops/sec
```

(continues on next page)

(continued from previous page)

```
Trie __getitem__ (hits)           not supported
BytesTrie __getitem__ (hits)      0.498M ops/sec
RecordTrie __getitem__ (hits)     0.404M ops/sec

dict get() (hits)                 4.410M ops/sec
Trie get() (hits)                 not supported
BytesTrie get() (hits)            0.458M ops/sec
RecordTrie get() (hits)           0.364M ops/sec
dict get() (misses)               4.869M ops/sec
Trie get() (misses)               not supported
BytesTrie get() (misses)          0.849M ops/sec
RecordTrie get() (misses)         0.816M ops/sec

dict __contains__ (hits)          8.053M ops/sec
Trie __contains__ (hits)          1.018M ops/sec
BytesTrie __contains__ (hits)     0.605M ops/sec
RecordTrie __contains__ (hits)    0.618M ops/sec
dict __contains__ (misses)        6.489M ops/sec
Trie __contains__ (misses)        2.047M ops/sec
BytesTrie __contains__ (misses)   1.079M ops/sec
RecordTrie __contains__ (misses)  1.123M ops/sec

dict items()                      57.248 ops/sec
Trie items()                      not supported
BytesTrie items()                 11.691 ops/sec
RecordTrie items()                8.369 ops/sec

dict keys()                       217.920 ops/sec
Trie keys()                       19.589 ops/sec
BytesTrie keys()                  14.849 ops/sec
RecordTrie keys()                 15.369 ops/sec

Trie.prefixes (hits)              0.594M ops/sec
Trie.prefixes (mixed)             1.874M ops/sec
Trie.prefixes (misses)            1.447M ops/sec
RecordTrie.prefixes (hits)        0.103M ops/sec
RecordTrie.prefixes (mixed)       0.458M ops/sec
RecordTrie.prefixes (misses)      0.164M ops/sec
Trie.iter_prefixes (hits)         0.588M ops/sec
Trie.iter_prefixes (mixed)        1.470M ops/sec
Trie.iter_prefixes (misses)       1.170M ops/sec

Trie.keys(prefix="xxx"), avg_len(res)==415                  5.044K ops/sec
Trie.keys(prefix="xxxxx"), avg_len(res)==17                 89.363K ops/sec
Trie.keys(prefix="xxxxxxxx"), avg_len(res)==3               258.732K ops/sec
Trie.keys(prefix="xxxxx..xx"), avg_len(res)==1.4            293.199K ops/sec
Trie.keys(prefix="xxx"), NON_EXISTING                       1169.524K ops/sec

RecordTrie.keys(prefix="xxx"), avg_len(res)==415            3.836K ops/sec
RecordTrie.keys(prefix="xxxxx"), avg_len(res)==17           73.591K ops/sec
RecordTrie.keys(prefix="xxxxxxxx"), avg_len(res)==3         229.515K ops/sec
RecordTrie.keys(prefix="xxxxx..xx"), avg_len(res)==1.4      269.228K ops/sec
RecordTrie.keys(prefix="xxx"), NON_EXISTING                 1071.433K ops/sec
```

Tries from `marisa_trie` are static and uses less memory, tries from datrie are faster and can be updated.

You may also give DAWG a try - it is usually faster than `marisa-trie` and sometimes can use less memory (depending on data).

Please take this benchmark results with a grain of salt; this is a very simple benchmark on a single data set.

4.3 API reference

4.3.1 BinaryTrie

4.3.2 Trie

4.3.3 BytesTrie

4.3.4 RecordTrie

4.4 Contributing

Contributions are welcome! Development happens at GitHub. Feel free to submit ideas, bug reports and pull requests.

If you found a bug in a C++ part please report it to the original bug tracker.

4.4.1 Navigating the source code

There are 4 folders in repository:

- `bench` – benchmarks & benchmark data;
- `lib` – original unmodified marisa-trie C++ library which is a git submodule; if something is have to be fixed in this library consider fixing it in the original repo;
- `src` – wrapper code; `src/marisa_trie.pyx` is a wrapper implementation; `src/*.pxd` files are Cython headers for corresponding C++ headers; `src/*.cpp` files are the pre-built extension code and shouldn't be modified directly (they should be updated via `update_cpp.sh` script).
- `tests` – the test suite.

4.4.2 Running tests and benchmarks

Make sure tox is installed and run

```
$ tox
```

from the source checkout. Tests should pass under Python 2.7, 3.4 and 3.5.

In order to run benchmarks, type

```
$ tox -c bench.ini
```

4.5 CHANGES

4.5.1 0.7.5 (2018-04-10)

- Removed redundant `DeprecationWarning` messages in `Trie.save` and `Trie.load`.

- Dropped support for Python 2.6.
- Rebuild Cython wrapper with Cython 0.28.1.

4.5.2 0.7.4 (2017-03-27)

- Fixed packaging issue, `MANIFEST.in` was not updated after `libmarisa-trie` became a submodule.

4.5.3 0.7.3 (2017-02-14)

- Added `BinaryTrie` for storing arbitrary sequences of bytes, e.g. IP addresses (thanks Tomasz Melcer);
- Deprecated `Trie.has_keys_with_prefix` which can be trivially implemented in terms of `Trie.iterkeys`;
- Deprecated `Trie.read` and `Trie.write` which onlywork for "real" files and duplicate the functionality of `load` and `save`. See issue #31 on GitHub;
- Updated `libmarisa-trie` to the latest version. Yay, 64-bit Windows support.
- Rebuilt Cython wrapper with Cython 0.25.2.

4.5.4 0.7.2 (2015-04-21)

- packaging issue is fixed.

4.5.5 0.7.1 (2015-04-21)

- setup.py is switched to setuptools;
- a tiny speedup;
- wrapper is rebuilt with Cython 0.22.

4.5.6 0.7 (2014-12-15)

- `trie1 == trie2` and `trie1 != trie2` now work (thanks Sergei Lebedev);
- `for key in trie:` is fixed (thanks Sergei Lebedev);
- wrapper is rebuilt with Cython 0.21.1 (thanks Sergei Lebedev);
- https://bitbucket.org/kmike/marisa-trie repo is no longer supported.

4.5.7 0.6 (2014-02-22)

- New `Trie` methods: `__getitem__`, `get`, `items`, `iteritems`. `trie[u'key']` is now the same as `trie.key_id(u'key')`.
- small optimization for `BytesTrie.get`.
- wrapper is rebuilt with Cython 0.20.1.

4.5.8 0.5.3 (2014-02-08)

- small `Trie.restore_key` optimization (it should work 5-15% faster)

4.5.9 0.5.2 (2014-02-08)

- fix `Trie.restore_key` method - it was reading past declared string length;
- rebuild wrapper with Cython 0.20.

4.5.10 0.5.1 (2013-10-03)

- `has_keys_with_prefix(prefix)` method (thanks Matt Hickford)

4.5.11 0.5 (2013-05-07)

- `BytesTrie.iterkeys`, `BytesTrie.iteritems`, `RecordTrie.iterkeys` and `RecordTrie.iteritems` methods;
- wrapper is rebuilt with Cython 0.19;
- `value_separator` parameter for `BytesTrie` and `RecordTrie`.

4.5.12 0.4 (2013-02-28)

- improved trie building: `weights` optional parameter;
- improved trie building: unnecessary input sorting is removed;
- wrapper is rebuilt with Cython 0.18;
- bundled marisa-trie C++ library is updated to svn r133.

4.5.13 0.3.8 (2013-01-03)

- Rebuild wrapper with Cython pre-0.18;
- update benchmarks.

4.5.14 0.3.7 (2012-09-21)

- Update bundled marisa-trie C++ library (this may fix more mingw issues);
- Python 3.3 support is back.

4.5.15 0.3.6 (2012-09-05)

- much faster (3x-7x) `.items()` and `.keys()` methods for all tries; faster (up to 3x) `.prefixes()` method for `Trie`.

4.5.16 0.3.5 (2012-08-30)

- Pickling of RecordTrie is fixed (thanks lazarou for the report);
- error messages should become more useful.

4.5.17 0.3.4 (2012-08-29)

- Issues with mingw32 should be resolved (thanks Susumu Yata).

4.5.18 0.3.3 (2012-08-27)

- `.get(key, default=None)` method for `BytesTrie` and `RecordTrie`;
- small README improvements.

4.5.19 0.3.2 (2012-08-26)

- Small code cleanup;
- `load`, `read` and `mmap` methods returns 'self';
- I can't run tests (via tox) under Python 3.3 so it is removed from supported versions for now.

4.5.20 0.3.1 (2012-08-23)

- `.prefixes()` support for RecordTrie and BytesTrie.

4.5.21 0.3 (2012-08-23)

- RecordTrie and BytesTrie are introduced;
- IntTrie class is removed (probably temporary?);
- dumps/loads methods are renamed to tobytes/frombytes;
- benchmark & tests improvements;
- support for MARISA-trie config options is added.

4.5.22 0.2 (2012-08-19)

- Pickling/unpickling support;
- dumps/loads methods;
- python 3.3 workaround;
- improved tests;
- benchmarks.

4.5.23 0.1 (2012-08-17)

Initial release.