intbitset Documentation

Release 2.3.1.dev20160630

CERN

February 01, 2017

Contents

CHAPTER 1

About

Provides an `intbitset` data object holding unordered sets of unsigned integers with ultra fast set operations, implemented via bit vectors and *Python C extension* to optimize speed and memory usage.

Emulates the Python built-in set class interface with some additional specific methods such as its own fast dump and load marshalling functions.

`intbitset` additionally support the pickle protocol, the iterator protocol and can behave like a `sequence` type.

1.1 Usage

Example:

```
>>> from intbitset import intbitset
>>> x = intbitset([1,2,3])
>>> y = intbitset([3,4,5])
>>> x & y
intbitset([3])
>>> x | y
intbitset([1, 2, 3, 4, 5])
```

1.2 Notes

- Uses real bits to optimize memory usage, so may have issues with *endianness* if you transport serialized bitsets between various machine architectures.
- Please note that no bigger than __`maxelem`__ elements can be added to an `intbitset`.
- On modern CPUs, *vectorial instruction sets* (such as MMX/SSE) are exploited to further optimize speed.

CHAPTER 2

Performance

Here is an example of performance gain with respect to traditional `set` of positive integers (example of *ipython* session):

```
>>> ## preparation
>>> from intbitset import intbitset
>>> from random import sample
>>> sparse_population1 = sample(range(1000000), 10000)
>>> sparse_population2 = sample(range(1000000), 10000)
>>> dense_population1 = sample(range(1000000), 900000)
>>> dense_population2 = sample(range(1000000), 900000)
>>> sparse_set1 = set(sparse_population1)
>>> sparse_set2 = set(sparse_population2)
>>> sparse_intbitset1 = intbitset(sparse_population1)
>>> sparse_intbitset2 = intbitset(sparse_population2)
>>> dense_set1 = set(dense_population1)
>>> dense_set2 = set(dense_population2)
>>> dense_intbitset1 = intbitset(dense_population1)
>>> dense_intbitset2 = intbitset(dense_population2)
>>> sorted(sparse_population1)[5000:5002]
[500095, 500124]
>>> in_sparse = 500095
>>> not_in_sparse = 500096
>>> sorted(dense_population1)[500000:500002]
[555705, 555707]
>>> in_dense = 555705
>>> not_in_dense = 555706
```

For sparse sets, `intbitset` operations are typically **50 times faster** than set operations:

```
>>> ## Sparse sets operations
>>> %timeit sparse_set1 & sparse_set2
1000 loops, best of 3: 263 µs per loop
>>> %timeit sparse_intbitset1 & sparse_intbitset2 ## more than 20 times faster
100000 loops, best of 3: 11.6 µs per loop
>>> %timeit sparse_set1 | sparse_set2
1000 loops, best of 3: 891 µs per loop
>>> %timeit sparse_intbitset1 | sparse_intbitset2 ## almost 70 times faster
100000 loops, best of 3: 12.8 µs per loop
>>> %timeit sparse_set1 ^ sparse_set2
1000 loops, best of 3: 1.09 ms per loop
>>> %timeit sparse_intbitset1 ^ sparse_intbitset2 ## more than 80 times faster
100000 loops, best of 3: 12.9 µs per loop
>>> %timeit sparse_set1 - sparse_set2
```

```
1000 loops, best of 3: 739 µs per loop
>>> %timeit sparse_intbitset1 - sparse_intbitset2 ## almost 60 times faster
100000 loops, best of 3: 12.5 µs per loop
```

For dense sets, `intbitset` operations are typically **5000 times faster** than set operations:

```
>>> ## Dense sets operations
>>> %timeit dense_set1 & dense_set2
10 loops, best of 3: 62.1 ms per loop
>>> %timeit dense_intbitset1 & dense_intbitset2 ## more than 5000 times faster
100000 loops, best of 3: 12.3 µs per loop
>>> %timeit dense_set1 | dense_set2
10 loops, best of 3: 84.1 ms per loop
>>> %timeit dense_intbitset1 | dense_intbitset2 ## more than 6000 times faster
100000 loops, best of 3: 12.5 µs per loop
>>> %timeit dense_set1 ^ dense_set2
10 loops, best of 3: 64.2 ms per loop
>>> %timeit dense_intbitset1 ^ dense_intbitset2 ## more than 5000 times faster
100000 loops, best of 3: 12.6 µs per loop
>>> %timeit dense_set1 - dense_set2
10 loops, best of 3: 38.6 ms per loop
>>> timeit dense_intbitset1 - dense_intbitset2 ## more than 3000 times faster
100000 loops, best of 3: 12.8 µs per loop
```

Membership operations in `intbitset` behave in a comparable way than `set` objects, albeit with slightly better performance:

```
>>> ## Membership tests
>>> %timeit in_sparse in sparse_set1
10000000 loops, best of 3: 66.8 ns per loop
>>> %timeit in_sparse in sparse_intbitset1 ## 1.5 times faster
10000000 loops, best of 3: 42.8 ns per loop
>>> %timeit not_in_sparse in sparse_set1
10000000 loops, best of 3: 71.3 ns per loop
>>> %timeit not_in_sparse in sparse_intbitset1 ## 1.6 times faster
10000000 loops, best of 3: 44.7 ns per loop
>>> %timeit in_dense in dense_set1
10000000 loops, best of 3: 61.8 ns per loop
>>> %timeit in_dense in dense_intbitset1 ## 1.3 times faster
10000000 loops, best of 3: 45.3 ns per loop
>>> %timeit not_in_dense in dense_set1
10000000 loops, best of 3: 45.5 ns per loop
>>> %timeit not_in_dense in dense_intbitset1 ## similar speed
10000000 loops, best of 3: 41.4 ns per loop
```

Serialising can be up to **30 times faster**:

```
>>> ## serialization speed
>>> ## note: internally intbitset compress using zlib so we are
>>> ## going to also compress the equivalent set
>>> from zlib import compress, decompress
>>> from marshal import dumps, loads
>>> %timeit loads(decompress(compress(dumps(sparse_set1))))
100 loops, best of 3: 6.55 ms per loop
>>> %timeit intbitset(sparse_intbitset1.fastdump()) ## 15% faster
100 loops, best of 3: 5.63 ms per loop
>>> %timeit loads(decompress(compress(dumps(dense_set1))))
1 loops, best of 3: 565 ms per loop
>>> %timeit intbitset(dense_intbitset1.fastdump()) ## almost 30 times faster for dense sets
```

```
10 loops, best of 3: 20.9 ms per loop
```

Serialising can lead to **20 times smaller footprint**:

```
>>> len(compress(dumps(sparse_set1)))
29349
>>> len(sparse_intbitset1.fastdump()) ## almost half the space
16166
>>> len(compress(dumps(dense_set1)))
1363026
>>> len(dense_intbitset1.fastdump()) ## 5% of the space for dense set
70332
```

CHAPTER 3

Reference

class intbitset.**intbitset**

Defines an intbitset data object to hold unordered sets of unsigned integers with ultra fast set operations, implemented via bit vectors and Python C extension to optimize speed and memory usage.

Emulates the Python built-in set class interface with some additional specific methods such as its own fast dump and load marshalling functions. Uses real bits to optimize memory usage, so may have issues with endianness if you transport serialized bitsets between various machine architectures.

The constructor accept the following parameters: rhs=0, int preallocate=-1, int trailing_bits=0, bint sanity_checks=CFG_INTBITSET_ENABLE_SANITY_CHECKS, int no_allocate=0:

where:

- rhs can be:
  - `int/long` for creating allocating empty intbitset that will hold at least rhs elements, before being resized
  - `intbitset` for cloning
  - `str` (or `bytes` on Python 3) for retrieving an intbitset that was dumped into a string
  - `array` for retrieving an intbitset that was dumped into a string stored in an array
  - sequence made of integers for copying all the elements from the sequence. If minsize is specified than it is initially allocated enough space to hold up to minsize integers, otherwise the biggest element of the sequence will be used.
  - sequence made of tuples: then the first element of each tuple is considered as an integer (as in the sequence made of integers).
- `preallocate` is a suggested initial upper bound on the numbers that will be stored, by looking at rhs a sequence of number.
- `trailing_bits` is 1, then the set will contain "all" the positive integers
- `no_allocate` is used internally and should never be set.

__and__

Return the intersection of two intbitsets as a new set. (i.e. all elements that are in both intbitsets.)

__cmp__

__contains__

x.__contains__(y) <==> y in x

__deepcopy__()

__delitem__
x.__delitem__(y) <==> del x[y]

__eq__
x.__eq__(y) <==> x==y

__ge__
x.__ge__(y) <==> x>=y

__getitem__
x.__getitem__(y) <==> x[y]

__gt__
x.__gt__(y) <==> x>y

__hash__

__iadd__
x.__iadd__(y) <==> x+=y

__iand__
Update a intbitset with the intersection of itself and another.

__ior__
Update a intbitset with the union of itself and another.

__isub__
Remove all elements of another set from this set.

__iter__

__ixor__
Update an intbitset with the symmetric difference of itself and another.

__le__
x.__le__(y) <==> x<=y

__len__

__lt__
x.__lt__(y) <==> x<y

__ne__
x.__ne__(y) <==> x!=y

__new__ (*S*, ...) → a new object with type S, a subtype of T

__nonzero__
x.__nonzero__() <==> x != 0

__or__
Return the union of two intbitsets as a new set. (i.e. all elements that are in either intbitsets.)

__pyx_vtable__ = <capsule object NULL>

__rand__
x.__rand__(y) <==> y&x

__reduce__()

__repr__

__ror__
x.__ror__(y) <==> y|x

__rsub__
x.__rsub__(y) <==> y-x

__rxor__
x.__rxor__(y) <==> y^x

__safe_for_unpickling__ = True

__setitem__
x.__setitem__(i, y) <==> x[i]=y

__str__

__sub__
Return the difference of two intbitsets as a new set. (i.e. all elements that are in this intbitset but not the other.)

__xor__
Return the symmetric difference of two sets as a new set. (i.e. all elements that are in exactly one of the sets.)

add()
Add an element to a set. This has no effect if the element is already present.

clear()

copy()
Return a shallow copy of a set.

difference()
Return a new intbitset with elements from the intbitset that are not in the others.

difference_update()
Update the intbitset, removing elements found in others.

discard()
Remove an element from a intbitset if it is a member. If the element is not a member, do nothing.

extract_finite_list()
Return a finite list of elements sufficient to be passed to intbitset constructor toghether with the proper value of trailing_bits in order to reproduce this intbitset. At least up_to integer are looked for when they are inside the intbitset but not necessarily needed to build the intbitset

fastdump()
Return a compressed string representation suitable to be saved somewhere.

fastload()
Load a compressed string representation produced by a previous call to the fastdump method into the current intbitset. The previous content will be replaced.

get_allocated()

get_size()

get_wordbitsize()

get_wordbytsize()

intersection()
Return a new intbitset with elements common to the intbitset and all others.

intersection_update()
Update the intbitset, keeping only elements found in it and all others.

is_infinite()
Return True if the intbitset is infinite. (i.e. trailing_bits=True was used in the constructor.)

isdisjoint()
Return True if two intbitsets have a null intersection.

issubset()
Report whether another set contains this set.

issuperset()
Report whether this set contains another set.

pop()
Remove and return an arbitrary set element.

Note: intbitset implementation of .pop() differs from the native `set` implementation by guaranteeing returning always the largest element.

remove()
Remove an element from a set; it must be a member. If the element is not a member, raise a KeyError.

strbits()
Return a string of 0s and 1s representing the content in memory of the intbitset.

symmetric_difference
Return the symmetric difference of two sets as a new set. (i.e. all elements that are in exactly one of the sets.)

symmetric_difference_update
Update an intbitset with the symmetric difference of itself and another.

tolist()
Legacy method to retrieve a list of all the elements inside an intbitset.

union()
Return a new intbitset with elements from the intbitset and all others.

union_update()
Update the intbitset, adding elements from all others.

update()
Update the intbitset, adding elements from all others.

update_with_signs()
Given a dictionary rhs whose keys are integers, remove all the integers whose value are less than 0 and add every integer whose value is 0 or more

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

CHAPTER 5

Additional Notes

Notes on how to contribute, legal information and changelog are here for the interested.

5.1 Contributing

See <http://inveniosoftware.org/wiki/Development/Contributing> for now.

5.2 Changes

Here you can see the full list of changes between each intbitset release.

5.2.1 Version 2.3.0 (released 2016-06-21)

Bug fixes

- Fixes implementation of *del x[123]* operator which was wrongly defined as *__del__* rather than *__delitem__*. (#40)
- Amends license reST reference from gpl to lgpl to avoid detection as GPL when scanning the docs for licensing information.

5.2.2 Version 2.2.1 (released 2015-09-16)

Bug fixes

- Reorganizes MANIFEST.in and adds missing files. (#28) (#29)

5.2.3 Version 2.2.0

- Removes coverage because it is not really supported for Cython modules.
- Automatically generates intbitset documentation by using Sphinx automodule functionality.
- Overall, amends documentation to be compatible with reStructuredText.

- Amends .update() and corresponding methods to accept also non-intbitset objects, such as lists or sets of integers respecting the set interface.
- Raises TypeError rather than terminating current process with a segmentation fault when None is used on the left side of an operation with an intbitset.
- Initial release of Docker configuration suitable for local developments.
- No longer returns self in fastload().
- Stops using *-march=native* for compilation, because it makes the compiler to optimize the code for only the currently used processor.

5.2.4 Version 2.1.1

- PyBytes_FromStringAndSize() fix in Python 2

5.2.5 Version 2.1

- Adds type checking for &, |, etc. operators. The type of "self" was not checked.
- Adds support for new union() and isdisjoint() set methods.
- Updates intbitset interface to look like set built-in in Python 2.6.
- Supports initialization of an intbitset from a set.
- No crash when intbitset is on rhs.
- Complete Python 3.x support.

5.2.6 Version 2.0

- Packaged into a standalone git repository.

5.3 License

The full license text can be found below (*GNU Lesser General Public License*).

5.3.1 Authors

IntbitSet is developed for use in Invenio digital library software.

Contact us at info@inveniosoftware.org

Contributors

- Alessio Deiana <alessio.deiana@cern.ch>
- Jiri Kuncar <jiri.kuncar@cern.ch>
- Lars Holm Nielsen <lars.holm.nielsen@cern.ch>
- Marco Neumann <marco@crepererum.net>
- Nikola Yolov <nikola.yolov@cern.ch>
- Philippe Ombredanne <pombredanne@gmail.com>
- Samuele Kaplun <samuele.kaplun@cern.ch>
- Tibor Simko <tibor.simko@cern.ch>

5.3.2 GNU Lesser General Public License

GNU LESSER GENERAL PUBLIC LICENSE Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/> Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates

the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

0. Additional Definitions.

As used herein, “this License” refers to version 3 of the GNU Lesser

General Public License, and the “GNU GPL” refers to version 3 of the GNU General Public License.

“The Library” refers to a covered work governed by this License,

other than an Application or a Combined Work as defined below.

An “Application” is any work that makes use of an interface provided

by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A “Combined Work” is a work produced by combining or linking an

Application with the Library. The particular version of the Library with which the Combined Work was made is also called the “Linked Version”.

The “Minimal Corresponding Source” for a Combined Work means the

Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The "Corresponding Application Code" for a Combined Work means the

object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License

without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a

facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

> a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or
>
> b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from

a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

> a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.
>
> b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that,

taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

> a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.
>
> b) Accompany the Combined Work with a copy of the GNU GPL and this license document.
>
> c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.
>
> 4. Do one of the following:
>
> > 0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.
> >
> > 1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already

present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.

e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the

Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.

b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions

of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the

Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide

whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

i

Symbols

A

C

D

E

F

G

I

P

R

S

T

U