

Böüncer

CS 543

- Final-deliverable



Background

Böuncer

An open source data streaming library for Python, built in C.

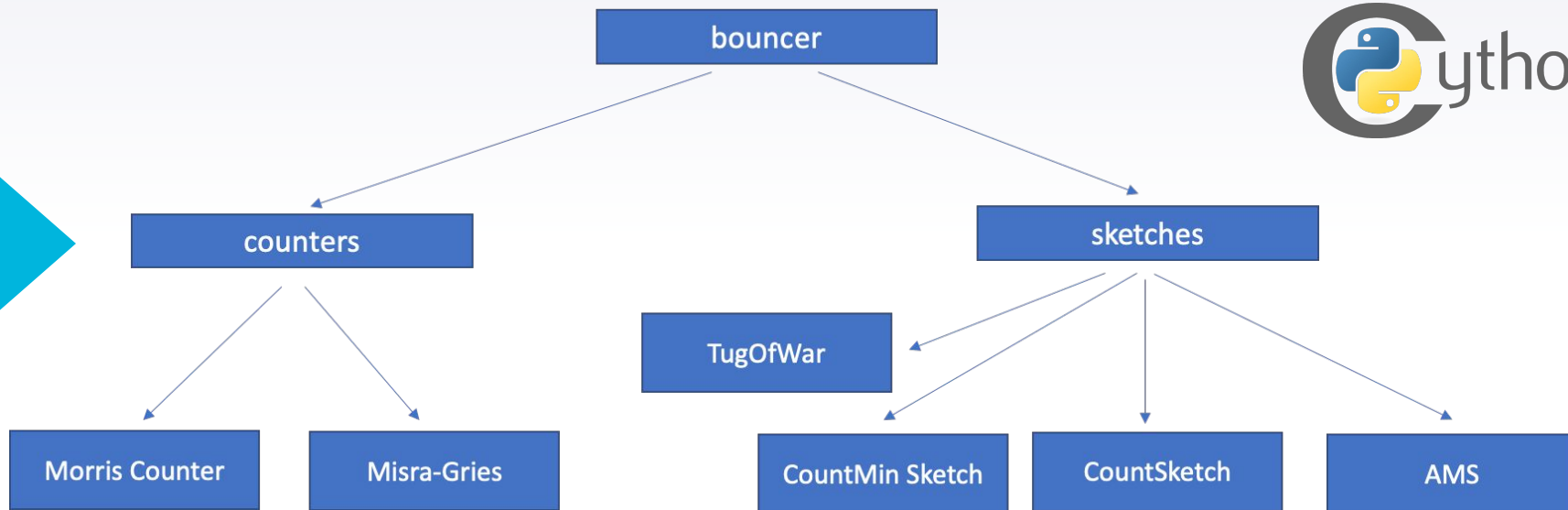
Just as a bouncer has to decide who is allowed in a venue and who is not, a data streaming algorithm has to 'decide' which data to 'let in' (use) and which to not in order to approximate the answer.



Class Architecture



1



Workflow

EDUCATION

- In depth analysis of all methods to be implemented: Jelani Nelson lecture notes
- Extensive c-python api documentation understanding: <https://docs.python.org/3/c-api/index.html>



Research

- Small space K-independent hashing algorithms.
- Translating between C code and python code.
- Engineering decisions about user friendliness vs efficiency.
- Class inheritance and meta-types via C python api.



WORK

- Hashing library to provide user options for which to use per linear sketch.
- Constructors to turn static c structs into python classes.
- Working implementations of Morris, CountMin Sketch, and CountMin

A word about python : Limitations

1

- Python is a beast!
Requires a lot of overhead in memory to get anything running.
- Python only has type long for integers, double for floats.
- Everything in python is a pyobject!

```
#pragma pack(1)
typedef struct {
    PyObject_HEAD
    byte* X; // counter
    byte version_flag : 2;
    byte epsilon : 7;
    byte delta : 7;
} Morris;
```

Size of any object instance will at least be the size of the members of its struct!

By default, to start a morris counter instance in python, would require, 16+ bytes!

A word about python : Limitations

1

- Usability of python library requires access to features inputted.
- An epsilon, delta value by default incur a cost of 8 bytes in C (but actually 16 in Python)!
- These are of course necessary prerequisites to instantiate the object instance.

```
static PyMethodDef Morris_methods[] = {
    {"update", (PyCFunction) Morris_update, METH_NOARGS,
     "Probabilistically update the morris counter"},
    {"output", (PyCFunction) Morris_output, METH_NOARGS,
     "Output the counter estimate"},
    },
    {NULL} /* Sentinel */
};
```

By storing the Python object version, a string object pointing to either of "Morris, Morris+, Morris++", we also incur at least another 10 bytes of memory by default.

One could find ways to elude this being a parameter, but at the cost of decreasing usability

```
PyGetSetDef get_Morris_sets[] = {
    {"version", /* name */
     (getter) get_version,
     NULL,
     NULL, /* doc */
     NULL /* closure */},

    {"epsilon", /* name */
     (getter) get_epsilon,
     NULL,
     NULL, /* doc */
     NULL /* closure */},

    {"delta", /* name */
     (getter) get_delta,
     NULL,
     NULL, /* doc */
     NULL /* closure */},

    {"shape", /* name */
     (getter) get_shape,
     NULL,
     NULL, /* doc */
     NULL /* closure */},

    {"raw_counts", /* name */
     (getter) get_raw_counts,
     NULL,
     NULL, /* doc */
     NULL /* closure */},
    {NULL}
};
```

Requirements and assumptions:

1

The logo for 'Böuncer' features the word in a bold, pink, sans-serif font. Above the 'o' in 'Böuncer', there are five small pink dots of varying sizes arranged in a slight arc, resembling a bounce or a trail of particles.

Böuncer

- Requires Python version $\geq 3.9.9$
- Class only works with integer tokens, cannot handle generic types (strings, vectors) as inputs.
- Assumes input domain does not exceed range of $2^{31} - 1$.

Counters module

```
# Non idealized with random.random() ?
class Morris():
    def __init__(self, epsilon, delta, version = 'morris'):
        assert(epsilon < 0 or epsilon > 1), "epsilon must be set between 0 and 1, non inclusive"
        assert(delta < 0 or delta > 1), "delta must be set between 0 and 1, non inclusive"
        assert(version == 'morris' or version == 'morris+' or version == 'morris++'), "Version must be morris, morris+, or morris++"

        self.delta = delta
        self.epsilon = epsilon
        self.update = lambda x: x + 1 if random.random() <= 1/(2**x) else 0
        self.output = lambda x: (2**x) - 1
        self.version = version

        if self.version == 'morris':
            self.X = 0

        elif self.version == 'morris+':
            self.s = math.ceil(1/(2 * (self.epsilon**2) * self.delta))
            self.X = np.zeros(shape= (self.s, ))

        else: # morris ++
            self.s = math.ceil(3/(2 * (self.epsilon**2)))
            self.t = math.ceil(18 * np.log(1/self.delta))
            self.X = np.zeros(shape= (self.t, self.s,))
```

Init function of Morris counter in python vs in C using the python api.

```
static int
Morris_init(Morris *self, PyObject *args, PyObject *kwargs)
{
    static char *kwlist[] = {"epsilon", "delta", "version", NULL};
    PyObject *version = NULL, *tmp;

    // passing in arguments
    if (!PyArg_ParseTupleAndKeywords(args, kwargs, "ff0", kwlist, &self->epsilon, &self->delta, &version))
        return -1;

    // error check for epsilon val
    if ((self->epsilon > 1) || (self->epsilon <= 0)) {
        PyErr_SetString(HyperParameterError, "epsilon must set between 0 and 1");
        return -1;
    }

    // error check for delta val
    if ((self->delta >= 1) || (self->delta <= 0)) {
        PyErr_SetString(HyperParameterError, "delta must be set between 0 and 1");
        return -1;
    }

    if (!PyUnicode_Check(version)) {
        PyErr_SetString(PyExc_TypeError,
                        "The version attribute must be a string");
        return -1;
    }

    const char *passed_in_version = PyUnicode_DATA(version);

    // if we only have to track one stream
    if (strcmp(passed_in_version, "Morris") == 0) {
        if ((self->X = (byte*) calloc(1, sizeof(byte))) == NULL) {
            PyErr_SetString(MemoryError, "Unable to allocate space, terminating");
            return -1;
        }
        self->version_flag = 0;
    }
    else if (strcmp(passed_in_version, "Morris+") == 0) {
        self->s = (uint32_t) ceil(1/((float) 2 * pow(self->epsilon, 2) * self->delta));
        if ((self->X = (byte*) calloc(self->s, sizeof(byte))) == NULL) {
            PyErr_SetString(MemoryError, "Unable to allocate space, terminating");
            return -1;
        }
        self->version_flag = 1;
    }
    else if (strcmp(passed_in_version, "Morris++") == 0) {
        self->s = (uint32_t) ceil(3/((float) 2 * pow(self->epsilon, 2)));
        self->t = (uint32_t) ceil(18 * log(1/ (float) self->delta));
        if ((self->X = (byte*) calloc(self->s * self->t, sizeof(byte))) == NULL) {
            PyErr_SetString(MemoryError, "Unable to allocate space, terminating");
            return -1;
        }
        self->version_flag = 2;
    }
    else // passed in incorrect version type
        PyErr_SetString(VersionError, "version must be set to: Morris, Morris+, Morris++");
        return -1;
}

tmp = self->version;
Py_INCREF(version);
self->version = version;
Py_DECREF(tmp);
return 0;
}
```


Compilation code snippets

```
counters > setup.py > ...
1 from distutils.core import setup, Extension
2
3 compile_args = ['-Os', '-Wno-nullability-completeness', '-Wno-expansion-to-defined', '-Waddress-of-packed-member']
4
5 def main():
6     setup(name="Ben Badnani",
7           version="1.0.0",
8           description="Python interface for the fpts C library function",
9           author="<your name>",
10          author_email="your_email@gmail.com",
11          ext_modules=[Extension("counters", sources=["counters_wrapper.c"], extra_compile_args=compile_args)]
12          # os flag is to optimize for space.
13 if __name__ == "__main__":
14     main()
15
```

```
counters git:(main) × python3 setup.py install
running install
running build
running build_ext
running install_lib
running install_egg_info
Writing /usr/local/lib/python3.9/site-packages/Ben_Badnani-1.0.0-py3.9.egg-info
```

```
counters > C counters_wrapper.c > PyInit_counters(void)
1 #include "headers/counters_wrapper.h"
2 #include "headers/morris_counter.h"
3 #include "morris_counter.c"
4
5 #pragma pack(1)
6
7
8 PyMODINIT_FUNC
9 PyInit_counters(void)
10 {
11     PyObject *m;
12     if (PyType_Ready(&Morris_type) < 0)
13         return NULL;
14
15     m = PyModule_Create(&counters_module);
16     if (m == NULL)
17         return NULL;
18
19     Py_INCREF(&Morris_type);
20     if (PyModule_AddObject(m, "Morris", (PyObject *) &Morris_type) < 0) {
21         Py_DECREF(&Morris_type);
22         Py_DECREF(m);
23         return NULL;
24     }
25
26     // Adding two custom errors to class
27     HyperParameterError = PyErr_NewException("HyperParameterError.error", NULL, NULL);
28     Py_XINCREF(HyperParameterError);
29     if (PyModule_AddObject(m, "error", HyperParameterError) < 0) {
30         Py_XDECREF(HyperParameterError);
31         Py_CLEAR(HyperParameterError);
32         Py_DECREF(&Morris_type);
33         Py_DECREF(m);
34         return NULL;
35     }
36
37     VersionError = PyErr_NewException("VersionError.error", NULL, NULL);
38     Py_XINCREF(VersionError);
39     if (PyModule_AddObject(m, "error", VersionError) < 0) {
40         Py_XDECREF(VersionError);
41         Py_CLEAR(VersionError);
42         Py_DECREF(&Morris_type);
43         Py_DECREF(m);
44         return NULL;
45     }
46
47     MemoryError = PyErr_NewException("MemoryError.error", NULL, NULL);
48     Py_XINCREF(MemoryError);
49     if (PyModule_AddObject(m, "error", MemoryError) < 0) {
50         Py_XDECREF(MemoryError);
51         Py_CLEAR(MemoryError);
52         Py_DECREF(&Morris_type);
53         Py_DECREF(m);
54         return NULL;
55     }
56
57     return m;
58 }
```

Counters module: results

```
>>> from counters import Morris
>>> m1 = Morris(epsilon = .2, delta = .5, version = "Morris+")
>>> for i in range(10):
...     m1.update()
...
>>> m1.output()
7.639999866485596
```

```
[>>> m1.epsilon
0.20000000298023224
```

```
[>>> m1.delta
0.5
```

```
[>>> m1.version
'Morris+'
```

```
>>> sys.getsizeof(m1)
26
```



CountMin module code snippets

```
import numpy as np
import math
from ..utils.utils import nextPrime, LinearSketch
from ..utils.utils import nextPrime

class CountMin(LinearSketch):
    # https://cs-web.bu.edu/faculty/homer/537/talks/SarahAdelBargal_UniversalHashingnotes.pdf , hashing algorithm
    def __init__(self, epsilon, delta, n):
        self.p = nextPrime(math.ceil(2/self.epsilon))
        self.t = math.ceil(math.log2(1/self.delta))

        self.random_bits = np.random.choice(range(self.p), size = (self.t, 2))
        self.hash = lambda token,row: ((self.random_bits[row][0]*token) + self.random_bits[row][1]) % self.p

        self.count_min_sketch = np.zeros(shape = (self.t, self.p))
```

Init function of CountMin sketch in python vs in C using the python api.

```
static inline
CountMin_Init(CountMin *self, PyObject *args, PyObject *kwargs) // add n as parameter
{
    PyObject *hashter, *tsp;
    hashter = PyObject_FromString("CP"); // setting hashter default value to "CP"

    float epsilon;
    float delta;

    static char *kwhitelist[] = {"epsilon", "delta", "hashter", NULL};

    // passing in arguments
    if (!PyArg_ParseTupleAndKeywords(args, kwhitelist, "ff|O", &epsilon, &delta, &hashter)) // optional argument for hashter, default is "tabulation"
        PyErr_SetString(MemoryError, "unable to allocate space, terminating");
    return -1;
}

// generating out arguments that super can handle
PyObject * super_args;
if ( (super_args = PyTuple_Pack(2, PyFloat_FromDouble((double) epsilon), PyFloat_FromDouble((double) delta))) == NULL)
    PyErr_NoMemory();
return -1;
}

// super().__init__(epsilon, delta)
if (LinearSketch_Init(self->super, super_args, NULL) != 0)
    PyErr_NoMemory();
return -1;
}

Py_XDECFREF(super_args);

if (!PyObject_Check(hashter)) {
    PyErr_SetString(PyExc_TypeError,
        "The version attribute must be a string");
    return -1;
}

self->os = (uint32_t) ceil(2/(float)self->super.epsilon);
self->ot = ceil(log1f((float) self->super.delta));

const char *passed_in_version = PyObject_DATA(hashter);
if (strcmp(passed_in_version, "tabulation") == 0)
    if ( (self->hash_table = (uint32_t *) calloc(self->ot, sizeof(uint32_t)) ) == NULL)
        PyErr_SetString(MemoryError, "Unable to allocate space, terminating");
    return -1;
}
for(uint32_t i = 0; i < self->ot; i++)
    self->hash_table[i] = rand(); // fill the table with random seed values
}
self->hash_retriever = Generate_random_bits_t;
self->hash_bit = 0;
}
else if (strcmp(passed_in_version, "CP") == 0) { // ON for Carter & Wegman
    // either if it was passed in or not, default value
    self->os = nextPrime(self->os); // n is set to the next prime
    if (self->hash_table = generate_random_bits_CW(self->os, self->ot, 2)) == NULL
        PyErr_SetString(MemoryError, "Unable to allocate space, terminating");
    return -1;
} // only requires 2 universal family
self->hash_retriever = Retrieve_hash_index_CW;
self->hash_bit = 1;
}
else if (strcmp(passed_in_version, "multishift") == 0) { // ON for Carter & Wegman
    if (self->hash_table = generate_random_bits_CW(self->os, self->ot, 2)) == NULL
        PyErr_SetString(MemoryError, "Unable to allocate space, terminating");
    return -1;
} // only requires 2 universal family
self->hash_retriever = Retrieve_hash_index_D;
self->hash_bit = 2;
}
}
else // passed in incorrect version type
    PyErr_SetString(VersionError, "version must be set to: tabulation, CW, multishift"); // need to fill in options later
return -1;
}

// generating sketch structure
if ( (self->super.X = (int *) calloc(self->os + self->ot, sizeof(int))) == NULL) // can use uint32_t pointer since it just allocates space > 0, prevents
    PyErr_SetString(MemoryError, "unable to allocate space, terminating");
return -1;
}

// set pystrring object in class to hashter class
tsp = self->hashter;
Py_INCREF(hashter);
self->hashter = hashter;
Py_XDECREF(hashter);

if ( (self->shape = PyTuple_Pack(2, PyLong_FromUnsignedLong((unsigned long) self->ot), PyLong_FromUnsignedLong((unsigned long) self->os))) == NULL)
    PyErr_NoMemory();
return -1;
}
return 0;
}
```

CountMin module: results

```
[>>> import sys  
[>>> sys.getsizeof(c1)  
48
```

```
[>>> from sketches import CountMin  
[>>> c1 = CountMin(epsilon = .2, delta = .5, hasher = "CW")  
[>>> sample_token_stream = [ (1, 1), (2,1), (1, -1), (2,-1), (1,2)]  
[>>> for token,count in sample_token_stream:  
...     c1.update(token, count)  
...  
[>>> c1.query(1)  
2  
[>>> c1.query(2)  
0
```

```
[>>> c1.delta  
0.5  
[>>> c1.epsilon  
0.20000000298023224  
[>>> c1.shape  
(1, 11)
```

```
[>>> c1.hasher  
'CW'
```



Hash library

```
// M. DIETZFELBINGER, Universal hashing and k-wise independent random variables via integer arithmetic without pri
/*****/
/*      2 independent hashing M. DIETZFELBINGER,      */
/*****/

uint32_t* generate_random_bits_D(uint32_t num_hash_buckets, uint32_t instantiations, uint8_t independence_degree)
{
    // assumes input can be represented with at most 32 bits.
    uint64_t i;
    uint32_t* hash_table;
    if ((hash_table = (uint32_t*) calloc(instantiations * 2, sizeof(uint32_t))) == NULL){
        return NULL;
    }
    for(i = 0; i < instantiations * independence_degree; i++){
        hash_table[i] = rand(); // pick a random number in 2^{input bit size}, 2^32
    }
    return hash_table;
}

uint32_t retrieve_hash_index_D(uint32_t num_hash_buckets, int token, int* rand_bits) // pointer into the table of
{
    uint32_t a, b; // to prevent overflow
    a = rand_bits[0];
    b = rand_bits[1];
    uint16_t hash_bits = log2(num_hash_buckets);
    return (a * token + b) >> (32 - hash_bits);
}
```

Hash library

```
// M. DIETZFELBINGER, Universal hashing and k-wise independent random variables via integer arithmetic without pri
/*****/
/*      2 independent hashing M. DIETZFELBINGER.      */
/*****/

uint32_t* generate_rand
{
    // assumes input ca
    uint64_t i;
    uint32_t* hash_tabl
    if ((hash_table = (
        | return NULL;
        | }
        | for(i = 0; i < inst
        | | hash_table[i] =
        | }
        | return hash_table;
    }

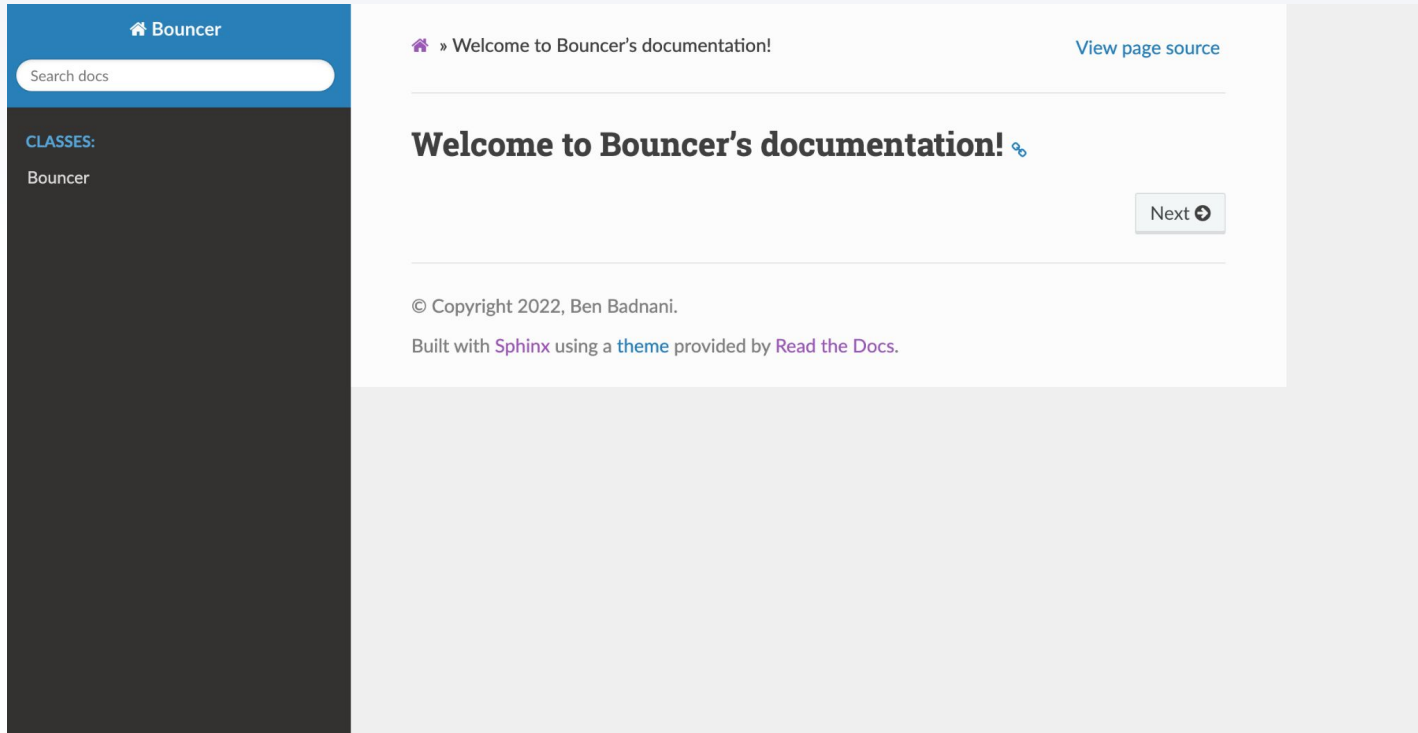
uint32_t retrieve_hash
{
    uint32_t a, b; //
    a = rand_bits[0];
    b = rand_bits[1];
    uint16_t hash_bits
    return (a * token +

uint32_t* generate_random_bits_CW(uint32_t num_hash_buckets, uint32_t instantiations, uint8_t independence_degree) // number of rand
{
    uint64_t i;

    uint32_t* hash_table;
    if ((hash_table = (uint32_t*) calloc(instantiations * independence_degree, sizeof(uint32_t))) == NULL){
        | return NULL;
        | }
        | for(i = 0; i < instantiations * independence_degree; i++){
        | | hash_table[i] = rand() / (RAND_MAX / num_hash_buckets + 1); // generate random number between 0 and prime for all indices
        | }
        | return hash_table;
    }

uint32_t retrieve_hash_index_CW(uint32_t num_hash_buckets, int token, int* rand_bits) // pointer into the table of random bits
{
    uint64_t hash_val = 0; // to prevent overflow
    for(uint32_t i = 0; i < num_hash_buckets; i++){
        | hash_val += pow(token, i) * rand_bits[i];
        | }
    hash_val = hash_val % num_hash_buckets; // modulus the number of buckets
    return (uint32_t) hash_val;
}
```

Sample documentation



The screenshot shows a web page for Bouncer documentation. On the left is a dark sidebar with a blue header containing the Bouncer logo and a search bar. Below the search bar, the text 'CLASSES:' is followed by 'Bouncer'. The main content area has a white background with a blue header containing a home icon, the text '» Welcome to Bouncer's documentation!', and a 'View page source' link. The main heading is 'Welcome to Bouncer's documentation!' with a small icon. Below it is a 'Next' button with a right arrow. At the bottom, there is a copyright notice '© Copyright 2022, Ben Badnani.' and a footer 'Built with Sphinx using a theme provided by Read the Docs.'

Bouncer


Search docs

CLASSES:

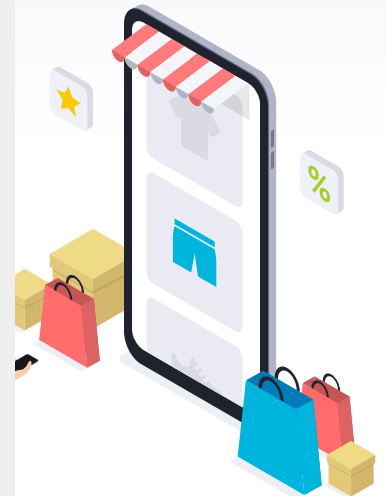
Bouncer

» Welcome to Bouncer's documentation! [View page source](#)

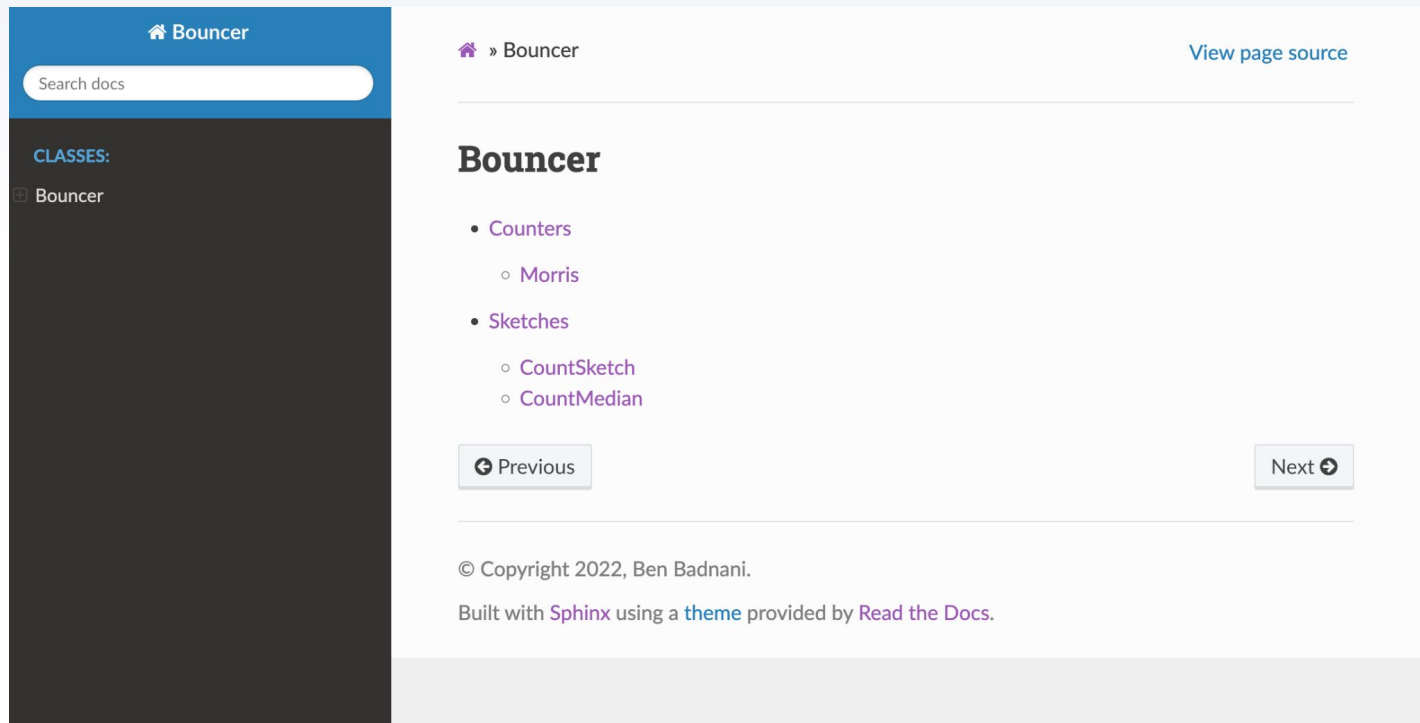
Welcome to Bouncer's documentation!

Next 

© Copyright 2022, Ben Badnani.
Built with [Sphinx](#) using a [theme](#) provided by [Read the Docs](#).



Sample documentation



The screenshot shows a web browser displaying a documentation page. The page has a dark blue header with the project name 'Bouncer' and a search bar. A left sidebar contains a 'CLASSES:' section with a sub-item 'Bouncer'. The main content area features a breadcrumb trail '» Bouncer', a 'View page source' link, and a large heading 'Bouncer'. Below the heading is a bulleted list of categories: 'Counters' (with a sub-item 'Morris') and 'Sketches' (with sub-items 'CountSketch' and 'CountMedian'). Navigation buttons for 'Previous' and 'Next' are present. The footer contains copyright information and mentions the use of Sphinx and Read the Docs.

🏠 Bouncer

Search docs

CLASSES:

- [-] Bouncer

» Bouncer [View page source](#)

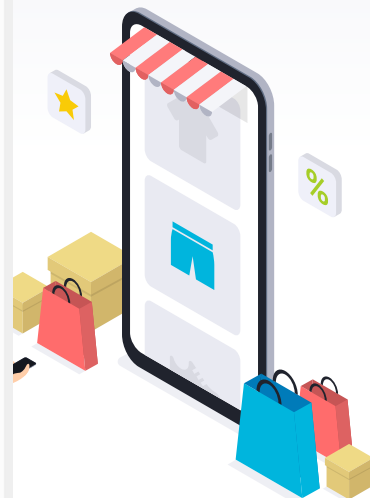
Bouncer

- Counters
 - Morris
- Sketches
 - CountSketch
 - CountMedian

⏪ Previous

Next ⏩

© Copyright 2022, Ben Badnani.
Built with Sphinx using a theme provided by Read the Docs.



Sample documentation

Bouncer

Search docs

CLASSES:

- Bouncer
 - Counters
 - Morris
 - Sketches

» Bouncer » Counters » Morris

[View page source](#)

Morris

The Morris counter is one of the first streaming algorithms ever discovered. For a stream of n tokens, the Morris counter seeks to approximate the number of items in the stream in only $O(\log n)$ bits of memory. Because the Morris counter acts as an unbiased estimator, we can approximate n up to value 2^{256} , using just one byte of memory for the counter. The algorithm works strictly on a vanilla model, meaning it can only record event insertions and does not allow for deletions.

```
class bouncer.counters.Morris(version, epsilon, delta)
```

Parameters

`version`: {"Morris", "Morris+", "Morris++", default="Morris"}

Specify the type of Morris counter.

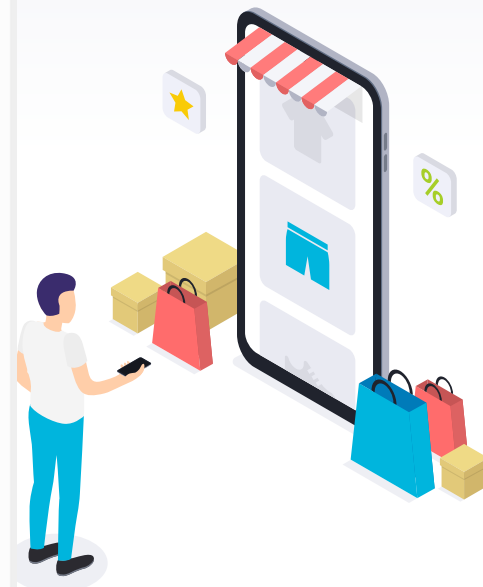
- `"Morris"`: One 8-bit Morris counter.
- `"Morris+"`: $s = \lceil \frac{1}{\epsilon^2 \delta} \rceil$ "Morris" counters.
- `"Morris++"`: $t = \lceil \frac{18}{\log \delta} \rceil$ "Morris+" counters, where $s = \lceil \frac{3}{2\epsilon^2} \rceil$.

`epsilon`: float, range=(0,1)

Accepted error range of true count. Only implemented for 'Morris+', 'Morris++'.

`delta`: float, range=(0,1)

Probability of exceeding epsilon error range. Only implemented for 'Morris+', 'Morris++'.



Examples

```
>> from bouncer.counters import Morris
>>
>> m = Morris(version="Morris")
>> for i in range(10):
...     m.update()
...
>>
>> m.output()
>> 15
>> m.shape
>> (1,)
>> m.raw_counts
>> [4]
>>
>> m1 = Morris(version="Morris+", epsilon = .2, delta = .5)
>> for i in range(10):
...     m1.update()
...
>>
>> m1.output()
>> 7.639999866485596
>> m1.shape
>> (1, 25)
>> m1.raw_counts
>> [2, 2, 3, 3, 3, 2, 3, 4, 3, 2, 3, 3, 2, 3, 3, 3, 3, 3, 2, 4, 3, 3, 4, 4, 4]
>>
>> m2 = Morris(version="Morris++", epsilon = .2, delta = .5)
>> for i in range(10):
...     m2.update()
...
>>
>> m2.output()
>> 10.894737243652344
>> m2.shape
>> (13, 38)
```



Examples

```
>> from bouncer.counters import Morris
>>
>> m = Morris(version="Morris")
>> for i in range(10):
...   m.update()
...
>>
>> m.output()
>> 15
>> m.shape
>> (1,)
>> m.raw_counts
>> [4]
>>
>> m1 = Morris(version="Morris+", epsilon = .2, delta = .5)
>> for i in range(10):
...   m1.update()
...
>>
>> m1.output()
>> 7.6399999866485596
>> m1.shape
>> (1, 25)
>> m1.raw_counts
>> [2, 2, 3, 3, 3, 2, 3, 4, 3, 2, 3, 3, 2, 3, 3, 3, 2, 3, 3, 2, 3, 3, 2, 3, 3]
>>
>> m2 = Morris(version="Morris++", epsilon = .2, delta = .5)
>> for i in range(10):
...   m2.update()
...
>>
>> m2.output()
>> 10.894737243652344
>> m2.shape
>> (13, 38)
```

Note

This implementation requires 26 bytes of overhead per instantiation. This is due to the overhead of PyObject_HEAD and other python fields required for all python object allocations.

Analysis and proof of correctness of the bounds can be found in Jelani Nelson's lecture notes under chapter 2.1.2: <https://www.sketchingbigdata.org/fall20/lec/notes.pdf>

← Previous

Next →

Credits and Acknowledgements

None of this would be possible without the following people:

- ▶ Krzysztof Onak
- ▶ Nadya Voronova
- ▶ Jelani Nelson
- ▶ Amit Chakrabarti
- ▶ My roommates Ben Blackman and Leor Lavi for the logo and name.



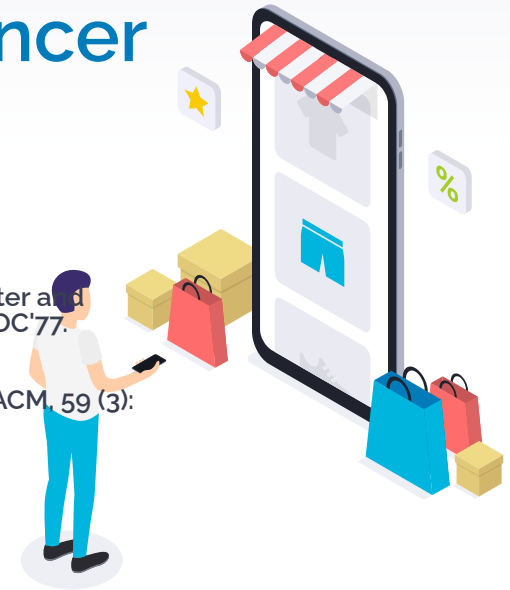
Library can be found at:

<https://github.com/KeyAleph/Bouncer>

Note: library is currently private

Sources:

1. Carter, Larry; Wegman, Mark N. (1979). "Universal Classes of Hash Functions". *Journal of Computer and System Sciences*. 18 (2): 143–154. doi:10.1016/0022-0000(79)90044-8. Conference version in STOC'77.
2. M. Dietzfelbinger. Universal hashing and k-wise independent random variables via integer arithmetic without primes. In *Proc. 13th STACS, LNCS 1046*, pages 569–580, 1996.
3. Pătrașcu, Mihai; Thorup, Mikkel (2012), "The power of simple tabulation hashing", *Journal of the ACM*, 59 (3): Art. 14, arXiv:1011.5200, doi:10.1145/2220357.2220361, MR 2946218.



What's Next?

What's next?

- ▶ Automatic Documentation with sphinx.
- ▶ Half-precision floating-point to decrease overhead!
- ▶ Implementation of Galois fields in C for universal hashing algorithms.
- ▶ Nisan's Pseudorandom Number Generator for derandomization and less random bit space usage!
- ▶ Add more algorithms!



THANKS!

Any questions?

