Bouncer

CS 543

- Final-deliverable

Background

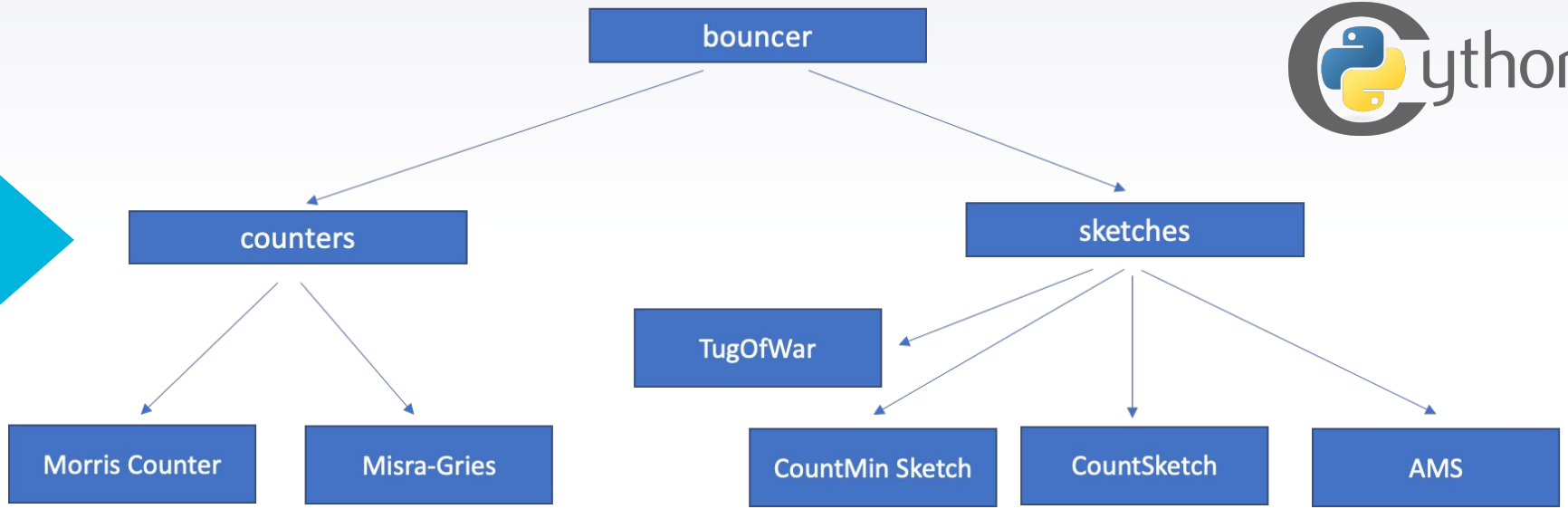
Bouncer

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 struts 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

- Requires Python version >= 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

```python
# 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,))
```

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

    // passing in arguments
    if (!PyArg_ParseTupleAndKeywords(args, kwds, "ffO", 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_vesion = PyUnicode_DATA(version);

    // if we only have to track one stream
    if(strcmp(passed_in_vesion, "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_vesion, "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_vesion, "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_XDECREF(tmp);
    return 0;
}
```

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

Compilation code snippets

counters > setup.py > ...

```python
from distutils.core import setup, Extension

compile_args = ['-Os','-Wno-nullability-completeness', '-Wno-expansion-to-defined', '-Waddress-of-packed-member']

def main():
    setup(name="Ben Badnani",
          version="1.0.0",
          description="Python interface for the fputs C library function",
          author="<your name>",
          author_email="your_email@gmail.com",
          ext_modules=[Extension("counters", sources=["counters_wrapper.c"], extra_compile_args=compile_args)])
          # os flag is to optimize for space.
if __name__ == "__main__":
    main()

```

```
→  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)

```c
#include "headers/counters_wrapper.h"
#include "headers/morris_counter.h"
#include "morris_counter.c"

#pragma pack(1)


PyMODINIT_FUNC
PyInit_counters(void)
{
    PyObject *m;
    if (PyType_Ready(&Morris_type) < 0)
        return NULL;

    m = PyModule_Create(&counters_module);
    if (m == NULL)
        return NULL;

    Py_INCREF(&Morris_type);
    if (PyModule_AddObject(m, "Morris", (PyObject *) &Morris_type) < 0) {
        Py_DECREF(&Morris_type);
        Py_DECREF(m);
        return NULL;
    }

    // Adding two custom errors to class
    HyperParameterError = PyErr_NewException("HyperParameterError.error", NULL, NULL);
    Py_XINCREF(HyperParameterError);
    if (PyModule_AddObject(m, "error", HyperParameterError) < 0) {
        Py_XDECREF(HyperParameterError);
        Py_CLEAR(HyperParameterError);
        Py_DECREF(&Morris_type);
        Py_DECREF(m);
        return NULL;
    }

    VersionError = PyErr_NewException("VersionError.error", NULL, NULL);
    Py_XINCREF(VersionError);
    if (PyModule_AddObject(m, "error", VersionError) < 0) {
        Py_XDECREF(VersionError);
        Py_CLEAR(VersionError);
        Py_DECREF(&Morris_type);
        Py_DECREF(m);
        return NULL;
    }

    MemoryError = PyErr_NewException("MemoryError.error", NULL, NULL);
    Py_XINCREF(MemoryError);
    if (PyModule_AddObject(m, "error", MemoryError) < 0) {
        Py_XDECREF(MemoryError);
        Py_CLEAR(MemoryError);
        Py_DECREF(&Morris_type);
        Py_DECREF(m);
        return NULL;
    }

    return m;
}
```

Counters module: results

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

```
>>> 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+'
```

CountMin module code snippets

```python
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.

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
0.200000002980232224
>>> 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 +
}
```

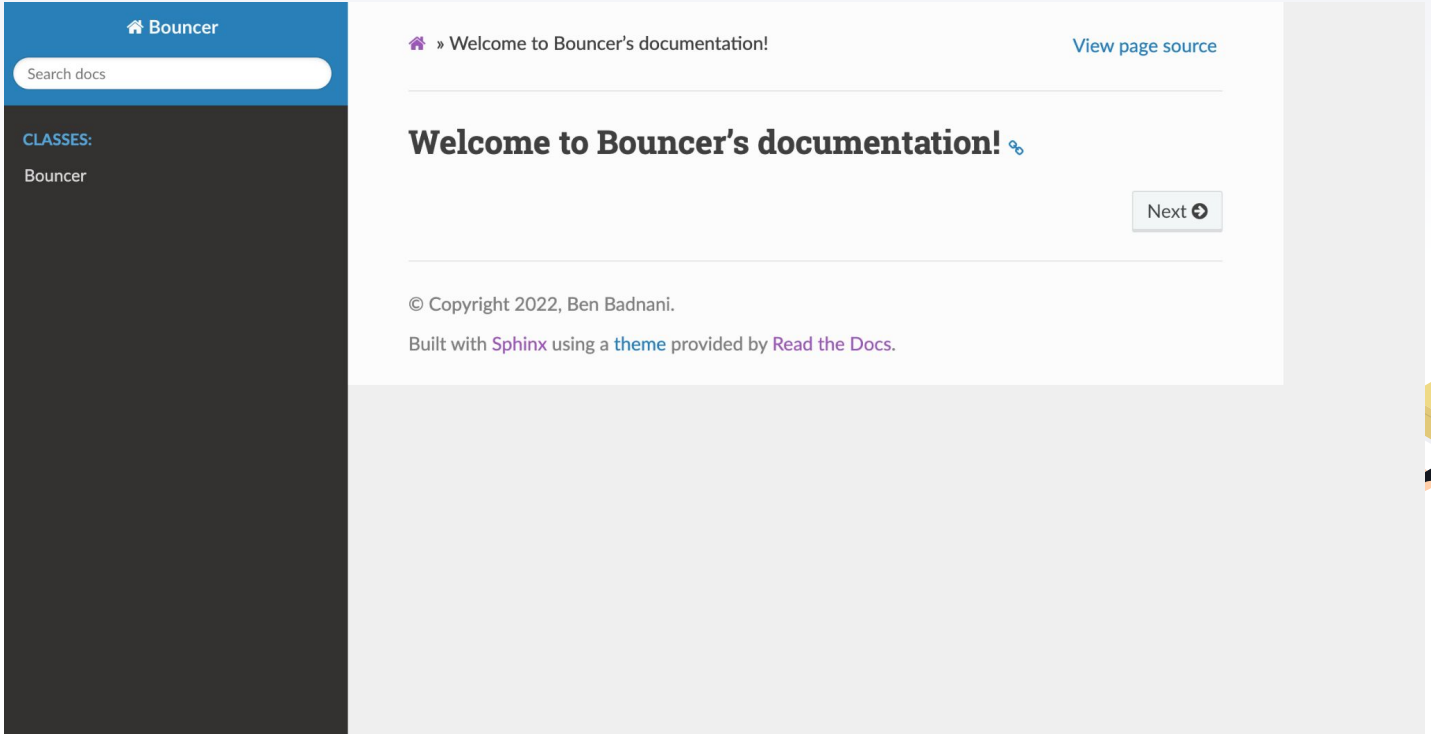
```
/*****************************************************/
/*  Carter & Wegman, arbitrary k independence          */
/*****************************************************/

uint32_t* generate_random_bits_CW(uint32_t num_hash_buckets, uint32_t instantiations, uint8_t independence_degree) // number of rando
{
    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



Sample documentation

Bouncer

Search docs

CLASSES:

Bouncer

» Bouncer View page source

Bouncer

- Counters
  - Morris
- Sketches
  - CountSketch
  - CountMedian

Previous Next

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)
```

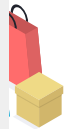
Examples

```
>> 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()
>> 7.639999866485596
>> m1.shape
>> (1, 25)
>> m1.raw_counts
>> [2, 2, 3, 3, 3, 2, 3, 4, 3, 2, 3, 3, 2, 3
>>
>> m2 = Morris(version="Morris++", epsilon =
>> 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 roomates 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
3. arithmetic without primes. In Proc. 13th STACS, LNCS 1046, pages 569–580, 1996.
4. Pătraşcu, 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?