Randomized algorithms
HW2 – Bloom filters

Ildar Nurglaiev

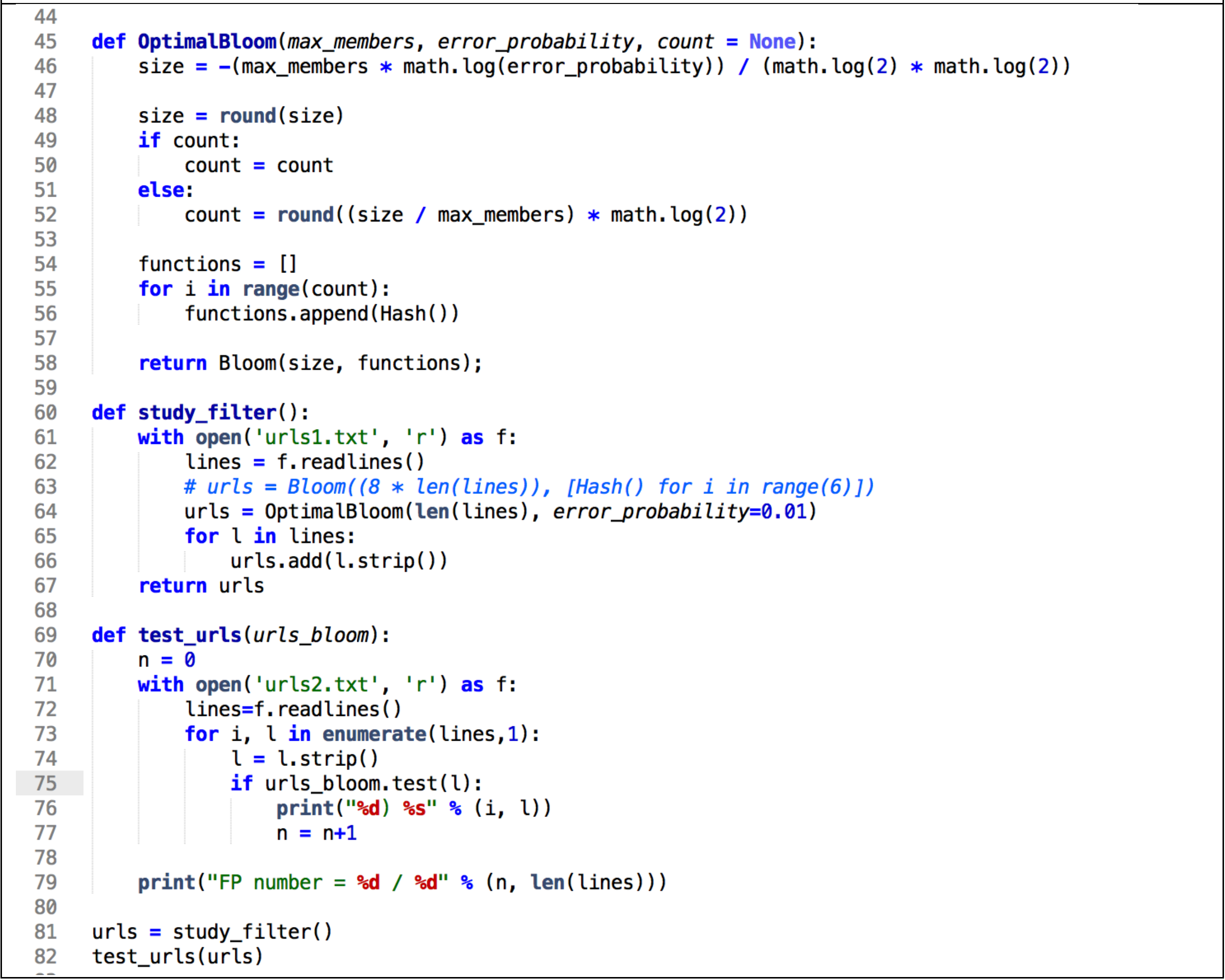
General Task:

1) Make a short program that takes 2 file names, one of which is bad sites. Add the bad sites to the filter.

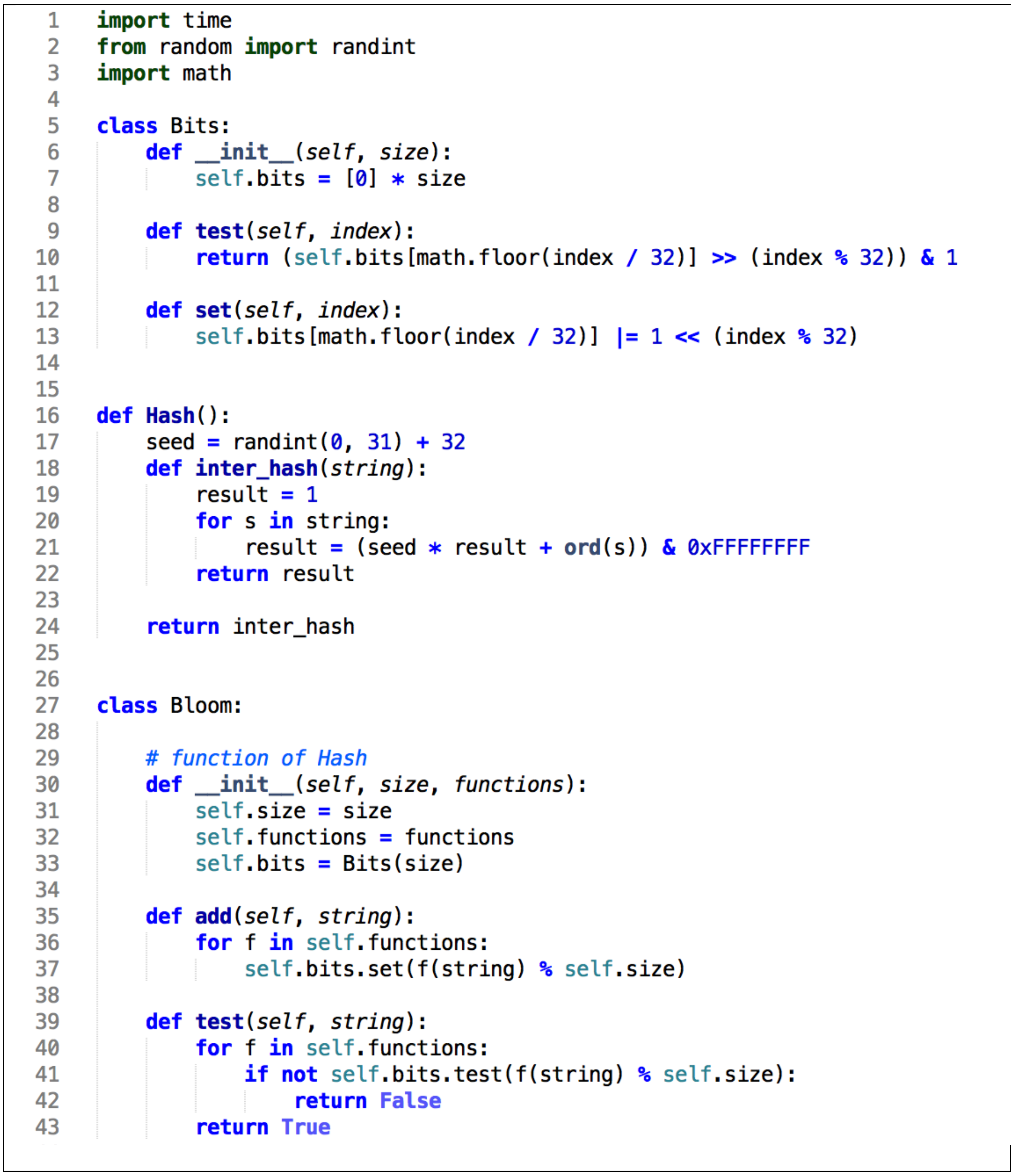
2) Check wherever URLs from other file is contained in the bloom filter. If the URL is in, print it.

3) Fill the report and answer to specified questions.

(10pts) Your implementation in Python



```python
def OptimalBloom(max_members, error_probability, count = None):
    size = -(max_members * math.log(error_probability)) / (math.log(2) * math.log(2))

    size = round(size)
    if count:
        count = count
    else:
        count = round((size / max_members) * math.log(2))

    functions = []
    for i in range(count):
        functions.append(Hash())

    return Bloom(size, functions);

def study_filter():
    with open('urls1.txt', 'r') as f:
        lines = f.readlines()
        # urls = Bloom((8 * len(lines)), [Hash() for i in range(6)])
        urls = OptimalBloom(len(lines), error_probability=0.01)
        for l in lines:
            urls.add(l.strip())
    return urls

def test_urls(urls_bloom):
    n = 0
    with open('urls2.txt', 'r') as f:
        lines=f.readlines()
        for i, l in enumerate(lines,1):
            l = l.strip()
            if urls_bloom.test(l):
                print("%d) %s" % (i, l))
                n = n+1

    print("FP number = %d / %d" % (n, len(lines)))

urls = study_filter()
test_urls(urls)
```



```python
import time
from random import randint
import math

class Bits:
    def __init__(self, size):
        self.bits = [0] * size

    def test(self, index):
        return (self.bits[math.floor(index / 32)] >> (index % 32)) & 1

    def set(self, index):
        self.bits[math.floor(index / 32)] |= 1 << (index % 32)


def Hash():
    seed = randint(0, 31) + 32
    def inter_hash(string):
        result = 1
        for s in string:
            result = (seed * result + ord(s)) & 0xFFFFFFFF
        return result

    return inter_hash


class Bloom:

    # function of Hash
    def __init__(self, size, functions):
        self.size = size
        self.functions = functions
        self.bits = Bits(size)

    def add(self, string):
        for f in self.functions:
            self.bits.set(f(string) % self.size)

    def test(self, string):
        for f in self.functions:
            if not self.bits.test(f(string) % self.size):
                return False
        return True
```

2) (20pts) Describe 1) how big is your bloom filter, 2) how many hash functions were used, and 3) what is false positive rate. 4) What is the relation among them?

We implemented parameterizable Bloom, so we use following formula to find size of filter, the size is determined from probability of error rate.

$$m = -\frac{n \ln p}{(\ln 2)^2} \quad (1)$$

To find number of hashes we will use following formula:

$$k = \frac{m}{n} \ln 2 \quad (2)$$

3) (20pts) How large was your bloom filter when it blocked around 1) 50, 2) 100, 3) 200 sites out of 1ml? (There are actually 13 sites known to be attack sites)

We have only 13 bad sites. We lookup 1ml. Assume that we detect 13 sites and also another not bad sites. Error rate for first case is (50 – 13)/1 ml. We will use formula (1), to find m:

a) Error rate = (50-13) / 1 ml = 0.000037

Bloom filter size will be approximately 21.2395 ml

b) Error rate = (100-13) / 1 ml = 0.000087

Bloom filter size will be approximately 19.4599 ml

c) Error rate = (200-13) / 1 ml = 0,000187

Bloom filter size will be approximately 17.8673 ml

(10pts) Explain hash functions you used in Bloom filter. (You can choose existing family of hash functions)

We used simple family of hash functions.

For every char of a string we retrieve its ASCII symbol number and add it to seed multiplied to previous resulting number afterwards we use mask in order to not go out of range of integer.

```
seed = randint(0, 31) + 32
def inter_hash(string):
    result = 1
    for s in string:
        result = (seed * result + ord(s)) & 0xFFFFFFFF
    return result
```

5. (10pts) Describe in 1) what ways Bloom filter better than other similar schemes. Also, 2) explain the case when k=1 (i.e., the number of hash functions used is 1) in Bloom filter.

a) Bloom filter better than other similar schemes (binary search trees, tries, hash tables, or simple arrays or linked lists of the entries) because of it's strong space. It's good for storing different data types, and requires only several bits per item, not depending on the type of the data. (We can construct a hash function to represent a string as index of array, instead of storing itself). Also it has probabilistic nature, and we can control the error rate of our filter. We can choose between accuracy and performance. All this makes Bloom filter good scheme.

b) Explain the situation with 1 hash functions:

Consider formula (2). From this formula we can see that size of filter m is: m= kn/ln2 = kn/0.69. So in case if k = 1, n/0.69 number of bits in Bloom filter we need where n is number of input strings, in case of 1 hash function.

6. (10pts) Can you 1) design a faster hash function? 2) Describe properties of good hash functions.

The main point is to have as few as possible hash collisions. Usually it doesn't make sense to use a cryptographically secure hash function, but something similar, for example xxhash or MurmurHash. If you want to use a bloom filter to avoid an in-memory lookup, then you probably want to use a faster hash function, for example just the XOR of the input, or use a Fletcher checksum. Also it is possible to use Murmur3 for the best trade-off between speed and uniformity as k hash functions (hash_i = hash1 + i x hash2).

Properties of good hash function:

1) How fast is the computation? the fastes is better.

2) How uniform is the output distribution? to disperse data points uniformly into n bits.

3) to securely identify the input data.

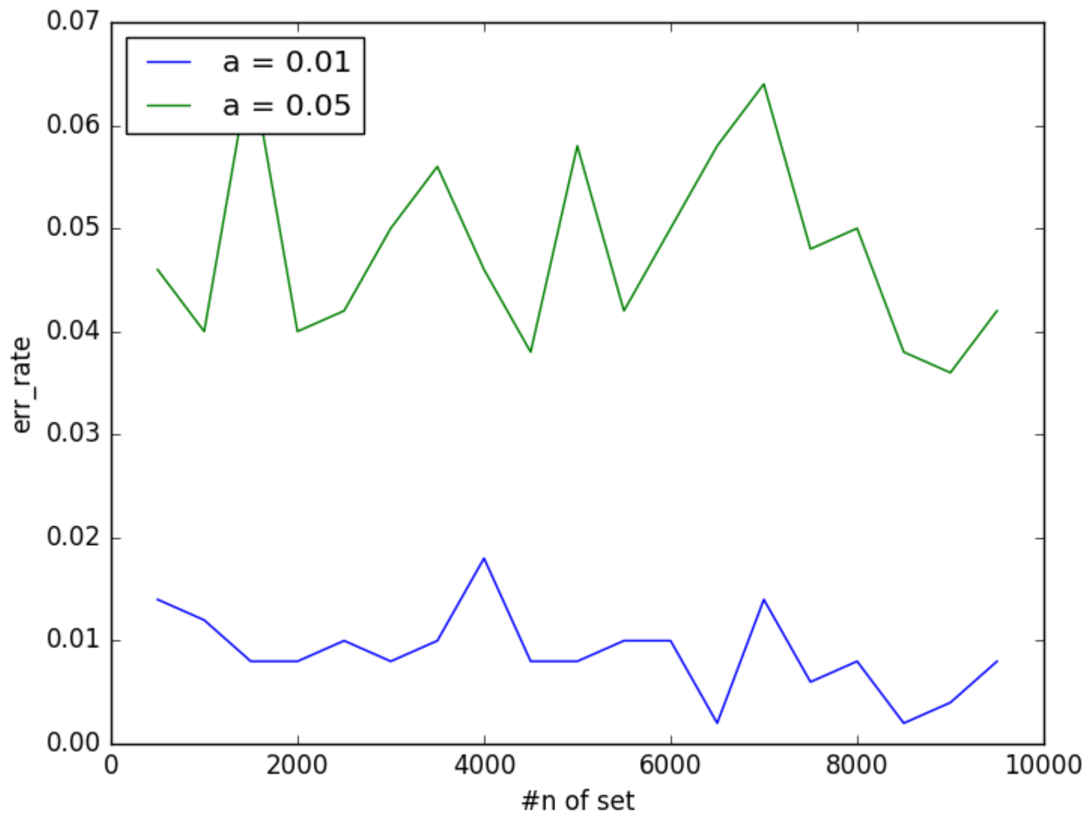
7. (10pts) Discuss where bloom filter is not suitable.

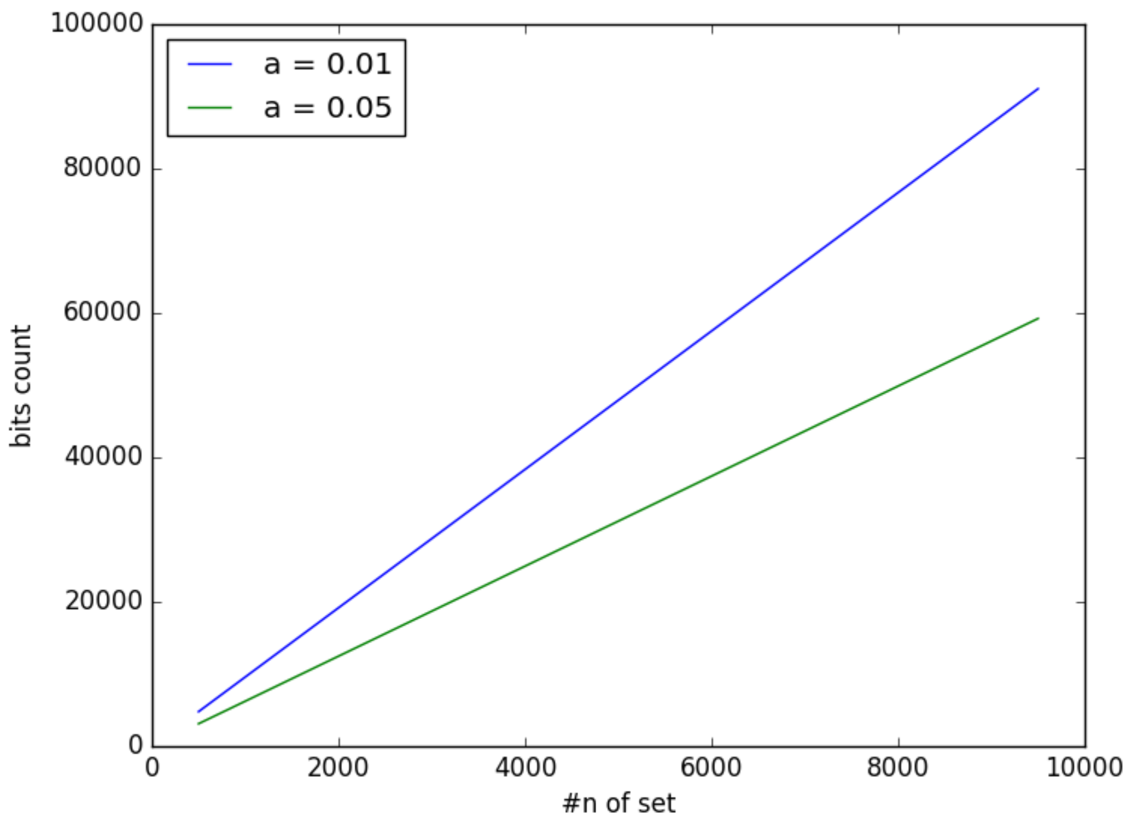
One of the limitations of filter is inability to delete element from filter. So, if your application data changes frequent, it would be problem to use bloom filter, however, we can use counting bloom filter.

Another limitation of Bloom Filter is inability to grow or shrink. In case of increasing data volume, we should rebuild and reconstruct bloom filter. So in application with fast growing data, we can't use Bloom Filter. However we can consider case of hierarchical Bloom Filter.

8. (10pts) Evaluate performance of your Bloom filter in terms of false positive rate and space efficiency. What if the expected false positive rate in terms of number of hash functions, size of Bloom filter and items?

For performance measurement we executed our implementation of parameter parameterizable Bloom Filter with error rate equal to 0.01 and 0.05.
The first graphic shows error_rate change with respect to input number of urls, the second graphic shows space change with respect to input number of urls.





expected false positive rate in terms of number of hash functions:

using formulas (1) and (2) we could reveal p – probability of FP

$\mathrm{E(FP)} = t * e^{-k*\ln(2)}$ where t – number of test strings