Assignment 3

by Diego Freire

Exercise 4.3.1: For the situation of our running example (8 billion bits, 1 billion members of the set S), calculate the false-positive rate if we use three hash functions? What if we use four hash functions?

Following what its explained on example 4.3:

S has m members, the array has n bits There are k hash functions

The number of targets is n, and the number of darts is y = km.

Thus, the probability that a bit remains 0 is e^(−km/n)

The Probability of a false positive is the probability of a 1 bit, which is the probability of a false positive False positive

(1 − e−km/n)^k

```
import math

m = 10**9
n = 8*10**9
k = 3

false_positives = (1 - math.e**(-k*m/n))**k
```

```
false_positives
```

```
0.030579354491777785
```

Exercise 4.4.1: Suppose our stream consists of the integers 3, 1, 4, 1, 5, 9, 2, 6, 5. Our hash functions will all be of the form h(x) = ax+b mod 32 for some a and b. You should treat the result as a 5-bit binary integer. Determine the tail length for each stream element and the resulting estimate of the number of distinct elements if the hash function is:

(a) h(x) = 2x+1 mod 32.

(b) h(x) = 3x+7 mod 32.

(c) h(x) = 4x mod 32.

```
l = [3, 1, 4, 1, 5, 9, 2, 6, 5]
lmin = 0
```

a) h(x) = 2x+1 mod 32.

```
for x in l:
    h1 = (2*x+1) % 32
    print(h1, '=', bin(h1))
```

```
7 = 0b111
3 = 0b11
9 = 0b1001
3 = 0b11
11 = 0b1011
19 = 0b10011
5 = 0b101
13 = 0b1101
11 = 0b1011
```

b) h(x) = 3x+7 mod 32.

```
for x in l:
    h2 = (3*x+7) % 32
    print(h2, '=',bin(h2))
```

```
16 = 0b10000
10 = 0b1010
19 = 0b10011
10 = 0b1010
22 = 0b10110
2 = 0b10
13 = 0b1101
25 = 0b11001
22 = 0b10110
```

c) h(x) = 4x mod 32.

```
for x in l:
    h3 = (4*x) % 32
    print(h3, '=',bin(h3))
```

```
12 = 0b1100
```

```
4 = 0b100
16 = 0b10000
4 = 0b100
20 = 0b10100
4 = 0b100
8 = 0b1000
24 = 0b11000
20 = 0b10100
```

Excercise 5.1.1: Compute the PageRank of each page inFig.5.7, assuming notaxation.

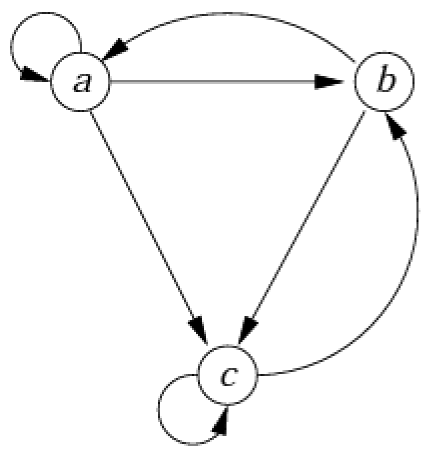

Figure 5.7: An example graph for exercises

```
import numpy as np

M = np.array([[1/3, 0, 0],
              [1/3,0,1/2],
              [1/3,1,1/2]])

r = np.array([[1/3],
             [1/3],
             [1/3]])

for i in range(100):
    r = M.dot(r)
    r = r
    if i % 10 == 0:
        print(r)
```

```
[[0.11111111]
 [0.27777778]
 [0.61111111]]
```

```
[[1.88167642e-06]
 [3.33300405e-01]
 [6.66697713e-01]]
[[3.18663555e-11]
 [3.33333302e-01]
 [6.66666698e-01]]
[[5.39659528e-16]
 [3.33333333e-01]
 [6.66666667e-01]]
[[9.13918149e-21]
 [3.33333333e-01]
 [6.66666667e-01]]
[[1.54772841e-25]
 [3.33333333e-01]
 [6.66666667e-01]]
[[2.62109165e-30]
 [3.33333333e-01]
 [6.66666667e-01]]
[[4.43884173e-35]
 [3.33333333e-01]
 [6.66666667e-01]]
[[7.51721745e-40]
 [3.33333333e-01]
 [6.66666667e-01]]
[[1.27304738e-44]
 [3.33333333e-01]
 [6.66666667e-01]]
```

Exercise 5.1.2: Compute the PageRank of each page in Fig. 5.7, assuming β =0.8.

```
beta = 0.8
N = len(M)

# Power iteration
for i in range(100):
    r = M.dot(r) * beta
    s = np.sum(r)
    r = r + np.ones([N, 1]) * (1 - s) / N
    if i % 10 == 0:
        print(r)

print(np.sum(r))
```

```
[[0.06666667]
 [0.33333333]
 [0.6       ]]
[[0.09090905]
 [0.32467573]
 [0.58441522]]
[[0.09090909]
```

```
 [0.32467532]
 [0.58441558]]
[[0.09090909]
 [0.32467532]
 [0.58441558]]
[[0.09090909]
 [0.32467532]
 [0.58441558]]
[[0.09090909]
 [0.32467532]
 [0.58441558]]
[[0.09090909]
 [0.32467532]
 [0.58441558]]
[[0.09090909]
 [0.32467532]
 [0.58441558]]
[[0.09090909]
 [0.32467532]
 [0.58441558]]
[[0.09090909]
 [0.32467532]
 [0.58441558]]
1.0
```