# Applied Algorithms Assignment 5

Fei Fan (Peter) Chen

February 6, 2021

## Exercise 1 (10 pts)

Initialize x to None. For each $a_i$, set $x = a_i$ with probability $\frac{1}{i}$.

**Proof**. Let $X_i$ be the random variable output of the above algorithm on a stream $a_0, a_1, ..., a_i$ of values $v_0, v_1, ..., v_j$. Then $P(X_i = v_j) = \frac{f_j^i}{i}$ where $f_j^i$ is number of occurrences of $v_j$ in the first $i$ elements.

We prove the above statement by induction. For $i = 0$, $X_0 = a_0$.

For $X_(i + 1)$, it remains $X_i$ with probability $\frac{i}{i+1}$ and changes to a new value $a_{i+1}$ with probability $\frac{1}{i}$. Now there are two cases for $X_{i+1} = v_j$, 1) $X_{i+1} = v_j$ and 2) $X_{i+1} = v_{j'} \neq v_j$.

**Case 1.** $X_{i+1} = v_j$ then $f_j^{i+1} = f_j^i + 1$ :

$$\begin{aligned} Pr\{X_{i+1} = v_j\} &= Pr\{X_i = v_j\}Pr\{X_{i+1}\ remains\ v_j\} + Pr\{X_{i+1}\ switches\ to\ v_j\} \\ &= \frac{f_j^i}{i}\frac{i}{i+1} + \frac{1}{i+1} \\ &= \frac{f_j^i + 1}{i+1} = \frac{f_j^{i+1}}{i+1} \end{aligned}$$

**Case 2.** $X_{i+1} = v_{j'}$ then $f_{j'}^{i+1} = f_{j'}^i$:

$$\begin{aligned} Pr\{X_{i+1} = v_{j'}\} &= Pr\{X_{i+1}\ switches\ to\ v_{j'}\} \\ &= \frac{f_{j'}^i}{i}\frac{i}{i+1} \\ &= \frac{f_{j'}^i}{i+1} = \frac{f_j^{i+1}}{i+1} \end{aligned}$$

## Exercise 2 (10 pts)

We can keep track of the last $ck$ values where $c$ is a constant. For simplicity, assume $c = 1$, we initialize by adding the first $k$ elements. Afterwards, we add the element with probability $\frac{k}{n}$ and evicting an existing element in $k$ with probability $\frac{1}{ck}$.

Over long run, an element that occurs more than $\frac{n}{k}$ times will be in the array and we output the unique elements.

This algorithm does result in double-sided errors though (e.g., we remove an element that did occur more than $\frac{n}{k}$ and we keep an element that did occur less than $\frac{n}{k}$ times). I think the degree of the error bias toward either side can be tuned with $c$.

# Exercise 3 (10 pts)

**a)**

$$Pr(|V_i| = 0) = (1-p)^{n-1}$$

**b)**

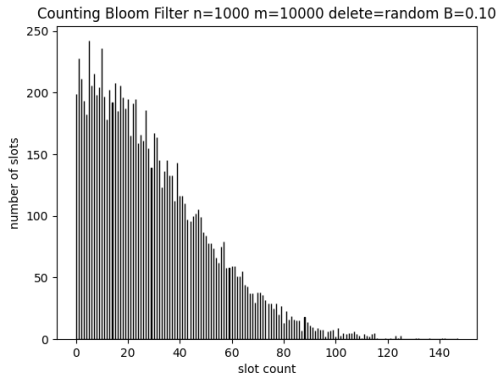
$$\begin{aligned} Pr(|V_i| = 0) &= (1 - \frac{2logn}{n})^{n-1} \\ &= \frac{(1 - \frac{2logn}{n})^n}{1 - \frac{2logn}{n}} \\ &\approx e^{-2logn} \; as \; n \longrightarrow \infty \\ &= \frac{1}{n^2} \end{aligned}$$

**c)** Let X be the random variable that counts the number of vertices that has degree 0. We give a vertices $i$ a value of 1 if it has degree 0 and 0 otherwise, then by Markov Inequality:

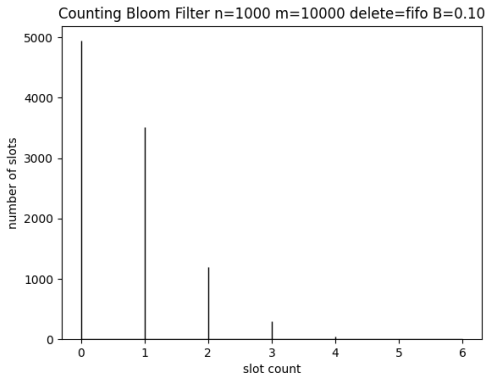
$$\begin{aligned} Pr(X \geq 1) &\leq E[X] \\ &= E[\sum_{i=1}^{n} X_i] \\ &= \sum_{i=1}^{n} E[X_i] \\ &= n \cdot \frac{1}{n^2} \\ &= \frac{1}{n} \end{aligned}$$

Alternatively we can actually calculate the exact probability of this happening:

$$\begin{aligned} Pr(no \; vertex \; has \; degree \; 0) &= 1 - Pr(every \; vertex \; has \; degree > 0) \\ &= 1 - (1 - Pr(a \; vertex \; has \; degree \; 0))^n \\ &= 1 - (1 - \frac{1}{n^2})^n \\ &= 1 - (1 - \frac{\frac{1}{n}}{n})^n \\ &= 1 - e^{-\frac{1}{n}} \end{aligned}$$

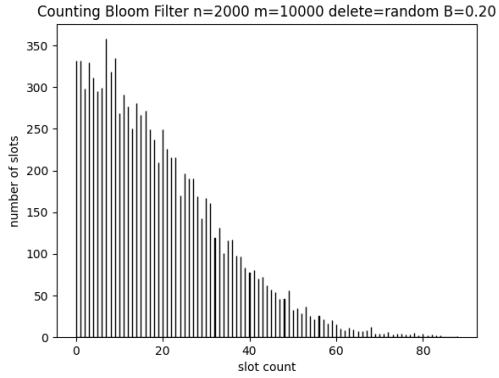



(a) Random

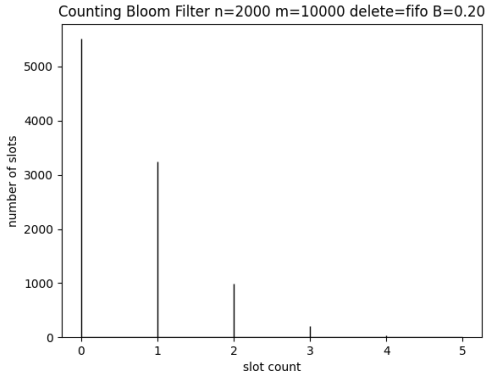



(b) FIFO

Figure 1: Slot Counter Size $\beta$=0.1 m=10000 n=1000




(a) Random



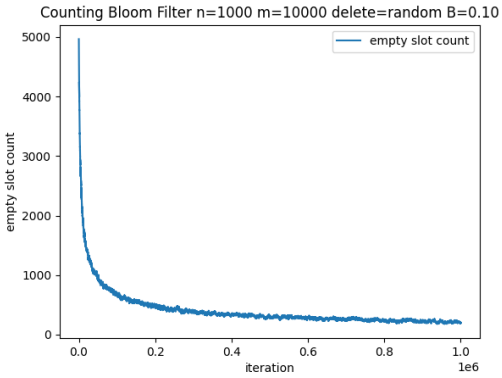

(b) FIFO

Figure 2: Slot Counter Size $\beta$=0.2 m=10000 n=1000

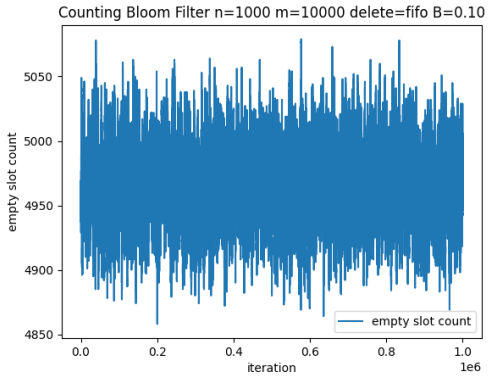
# Exercise 4 (10 pts)

We simulated a counting bloom filter. We incremented count when a slot is hashed for add and decremented count when a slot is hashed to for delete. We chose the optimum $k = \frac{m}{n} log2$ for a given $\beta = \frac{n}{m}$. We produce a histogram of slot counts and graph of available empty slots along several dimensions: load factor $\beta$, deletion method (random or FIFO). For each simulation, we ran for a million iterations after first pre-warming the bloom filter with given load factor $\beta$.

From the results it seems the data pattern affects the max counter size for each slot and the number of empty slots greatly. For "random" data pattern for deletions, the counter size must be larger (e.g., 10 bits) while empty slots falls progressively until it stabilizes. For "fifo" data pattern for deletions, the empty slot is roughly maintained at $\frac{1}{2}$ the total number of slots, while the counter size need to be no more than a few bits (e.g. 4 bits).

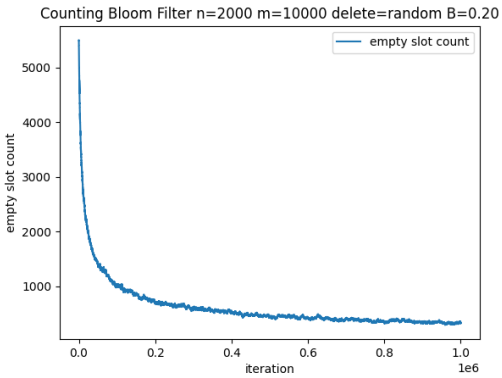



(a) Random

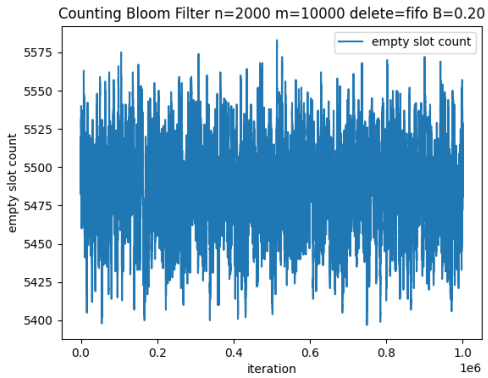



(b) FIFO

Figure 3: Empty Slot Count $\beta$=0.2 m=10000 n=1000




(a) Random




(b) FIFO

Figure 4: Empty Slot Count $\beta$=0.2 m=10000 n=1000

## Code

```
class BloomFilter():
  def __init__(self, m=10000, n=1000,
    iterations=1000000, method="random"):
    self._k = round( m/n * math.log(2) ) #optimal k given m and n
    self._B = n/m
    self._slots = [0] * m
    self._m = m
    self._iterations = iterations
    self._values = n*n # n^2 different values
    self._empty = m
    self._n = n
    self._deletion_method = method #random or fifo
    self._fifo = np.random.uniform(0, self._values, n)
    self._head = 0
    # simulate hashing the same item to the same location
    self._locations = {}

  def _add(self, i):
    if i not in self._locations:
      self._locations[i] = [ int(random.uniform(0, self._m))
        for _ in range(self._k) ]
    ilocs = self._locations[i]

    for iloc in ilocs:
      if self._slots[iloc] == 0:
        self._empty -= 1
      self._slots[iloc] += 1

  def _delete(self, i):
    if i not in self._locations:
      self._locations[i] = [ int(random.uniform(0, self._m))
        for _ in range(self._k) ]
    ilocs = self._locations[i]

    for iloc in ilocs:
      if self._slots[iloc] > 0:
        self._slots[iloc] -= 1
        if self._slots[iloc] == 0:
          self._empty += 1

  def run(self):
    # We fill the BloomFilter to half full, then
    # iterate self._n number of times with random
    # deletion or fifo deletion followed by another random
    # add.
```

```
# 1) pre-warm the bloom filter with self._fifo
for i in self._fifo:
  self._add(i)

empty_slots = [ self._empty ]

# 2) run random/fifo delete and add for iterations
for n in range(self._iterations):
  item_add = random.uniform(0, self._values)
  if self._deletion_method == "random":
    item_delete = random.uniform(0, self._values)
  elif self._deletion_method == "fifo":
    item_delete = self._fifo[self._head]
    self._fifo[self._head] = item_add
    self._head = (self._head + 1) % self._n

  self._delete(item_delete)
  self._add(item_add)
  empty_slots.append(self._empty)

return self._slots, empty_slots
```