CS3230: Design and Analysis of Algorithms
Semester 2, 2022-23, School of Computing, NUS

Written Assignment 2

Deadline: 17th March, 2023 (Friday), 11:59 pm

Instructions

- Submit the soft copy at Canvas → Assignment → Written Assignment 2.
- Write legibly. If we cannot read what you write, we cannot give points. In case you CANNOT write legibly, please type out your answers.
- When you submit your answer, please try to make it concise. However, it should contain all the relevant details. **The page limit is 4**.
- All the proofs must be written formally; no marks will be given for hand-waving arguments.
- Before submitting your answers, please make sure you rename your submission file in the following way: <Student ID>.pdf. (Note, your student ID is that starts with "A".)
- If you need any clarification on any of the questions, we strongly encourage you to post in Canvas Forum (instead of emailing us) so that everybody can see our responses. You may also approach Steven, Diptarka or your tutor.
- All the logarithms are of base 2. We use $\mathbb{E}[\cdot]$ to denote the expectation.

Question 1 [20 marks]: Consider a standard (FIFO) queue that supports the following operations:

- PUSH(x): Add item x at the end of the queue.
- PULL(): Remove and return the first item present in the queue.
- SIZE(): Return the number of elements present in the queue.

We can easily implement such a queue using a doubly-linked list so that PUSH, PULL and SIZE operations take at most c_1, c_2, c_3 worst-case time, for some constants $c_1, c_2, c_3 > 0$. (No need to show this. You can assume such an implementation.) Now, suppose we are asked to consider the following new operation:

- DECIMATE(): Remove every tenth element from the queue starting from the beginning.

```
DECIMATE():
  n ← SIZE()
  for i ← 0 to n − 1
      if i mod 10 = 0
          PULL()    ⟨⟨result discarded⟩⟩
      else
          PUSH(PULL())
```

Figure 1: Pseudocode of the DECIMATE operation (assuming starting index to be 0)

The above DECIMATE operation takes at most $c_4\ell$ time (for some constant $c_4 > 0$) in the worst-case (where ℓ is the number of items present in the queue). Use **potential method** to show that in any intermixed sequence of PUSH, PULL, SIZE and DECIMATE operations, the amortized cost of each of them is $O(1)$. (Assume, you start with an empty queue.)

Question 2 [20 marks]: Let us now consider a variant of the above (FIFO) queue, where we are only interested in PUSH and SIZE operation (PULL operation is not allowed in the queue). Suppose there is a long sequence of PUSH operations performing in a queue Q, and we are interested in computing SIZE of the queue Q at any point of time. However, we do not have access to the main queue Q, rather we can use another small storage space. Now, our objective is to compute the SIZE of Q by using the data stored in that small storage.

You may compare the above with the following real life scenario. Suppose you are interested in counting how many tweets there are posted on a day. There could be enormous number of tweets in a day and they are all saved in some Twitter database, which of course you cannot access. Now, in your personal computer, you have a tiny amount (compare to Twitter database size) of storage space. Suppose m is an upper bound on the number of tweets in a day, whereas you only have $o(\log m)$ bits of storage space (so you cannot store the count of tweets because even to store the integer m, you need $\Theta(\log m)$ bits of space). However, you have learned a bit of randomized algorithms in your CS3230 module. So you use that knowledge and come up with the following simple randomized algorithm.

For simplicity, let us now talk in terms of the queue, instead of the above tweet example. You initialize a counter $z \leftarrow 0$. Then for each PUSH call, perform $z \leftarrow z + 1$ with probability 2^{-z}. Finally, in the end, output $2^z - 1$. So essentially throughout the whole algorithm, you only need to maintain z.

(a) **[2 marks]** For any integer $i \geq 1$, let Z_i be the random variable denoting the value of 2^z after i PUSH operations. Now, which of the followings options are correct:

(i) $Z_{i+1} = Z_i + 1$ with probability $\frac{1}{Z_i}$.

(ii) $Z_{i+1} = Z_i$ with probability $1 - \frac{1}{Z_i}$.

(iii) $Z_{i+1} = i + 1$ with probability $\frac{1}{Z_i}$.

(iv) $Z_{i+1} = 2Z_i$ with probability $\frac{1}{Z_i}$.

$Z_i = 2^z + 1 \cdot \frac{1}{2^z}$

There could be multiple correct answers, and you need to mention them all. (For this part (a), just writing the correct options suffices, no need to provide any explanation.)

(b) **[4 marks]** For any integer $i \geq 1$, let X_i be a random variable taking value 1 with probability $\frac{1}{Z_i}$; and 0 with probability $1 - \frac{1}{Z_i}$. Express Z_{i+1} as a function of X_i and Z_i.

(c) **[5 marks]** Show that $\mathbb{E}[Z_{i+1}] = 1 + \mathbb{E}[Z_i]$ for all integers $i \geq 1$.

(d) **[5 marks]** If there are total t PUSH operations, $Z_t - 1$ will denote the final output of your algorithm. Use the statement of part (c) to prove that $\mathbb{E}[Z_t - 1] = t$.

(e) **[4 marks]** Assuming m being an upper bound on the total number of PUSH operations (i.e., t can take any integer values between 1 to m), in expectation how much bits of space (as a function of m, using asymptotic notation that is as tight as possible) do you need to store z? (You may use the following inequality without proving it: For any positive-valued random variable Z, $\mathbb{E}[\log Z] \leq \log(\mathbb{E}[Z])$.)

Note that part (d) implies that your randomized algorithm indeed outputs the correct count in expectation. Is not it truly amazing what you can do using such a small amount of storage space just with the help of randomization!

Remarks: If you cannot answer any particular question, you can still assume that statement and answer later questions (without any further penalty).

(b) **[4 marks]** For any integer $i \geq 1$, let X_i be a random variable taking value 1 with probability $\frac{1}{Z_i}$; and 0 with probability $1 - \frac{1}{Z_i}$. Express Z_{i+1} as a function of X_i and Z_i.

$Z_{i+1} = Z_i + X_i$

(c) **[5 marks]** Show that $\mathbb{E}[Z_{i+1}] = 1 + \mathbb{E}[Z_i]$ for all integers $i \geq 1$.

$E[Z_i] = \lfloor \lg i \rfloor$

$E[Z_{i+1}] = \lfloor \lg(i+1) \rfloor$

$E[Z_{i+1}] = \lfloor \lg(i) \rfloor + \lg(1 + 1/i)$

$E[Z_{i+1}] = 1 + \lfloor \lg i \rfloor = 1 + E[Z_i]$

(d) **[5 marks]** If there are total t PUSH operations, $Z_t - 1$ will denote the final output of your algorithm. Use the statement of part (c) to prove that $\mathbb{E}[Z_t - 1] = t$.

$E[Z_{i+1}] = 1 + E[Z_i]$

$E[Z_t - 1] = \lfloor \lg(t-1) \rfloor = \lfloor \lg(t) \rfloor + \lfloor \lg(1 - 1/t) \rfloor$

$E[Z_t - 1] = E[Z_t] = t$

(e) **[4 marks]** Assuming m being an upper bound on the total number of PUSH operations (i.e., t can take any integer values between 1 to m), in expectation how much bits of space (as a function of m, using asymptotic notation that is as tight as possible) do you need to store z? (You may use the following inequality without proving it: For any positive-valued random variable Z, $\mathbb{E}[\log Z] \leq \log(\mathbb{E}[Z])$.)

$\lg(m) \Leftarrow$ bits to store m

$\lg\lg(m) \Leftarrow$ bits to store z

Question 1 [20 marks]: Consider a standard (FIFO) queue that supports the following operations:

- PUSH(x): Add item x at the end of the queue.
- PULL(): Remove and return the first item present in the queue.
- SIZE(): Return the number of elements present in the queue.

We can easily implement such a queue using a doubly-linked list so that PUSH, PULL and SIZE operations take at most c_1, c_2, c_3 worst-case time, for some constants $c_1, c_2, c_3 > 0$. (No need to show this. You can assume such an implementation.) Now, suppose we are asked to consider the following new operation:

- DECIMATE(): Remove every tenth element from the queue starting from the beginning.

```
DECIMATE():
  n ← SIZE()
  for i ← 0 to n − 1
      if i mod 10 = 0
          PULL()   ⟨⟨result discarded⟩⟩
      else
          PUSH(PULL())
```

Figure 1: Pseudocode of the DECIMATE operation (assuming starting index to be 0)

The above DECIMATE operation takes at most $c_4\ell$ time (for some constant $c_4 > 0$) in the worst-case (where ℓ is the number of items present in the queue). Use **potential method** to show that in any intermixed sequence of PUSH, PULL, SIZE and DECIMATE operations, the amortized cost of each of them is $O(1)$. (Assume, you start with an empty queue.)

$\phi(x) = 2\ell - 10k$ $\quad$ ℓ = size of elements
m = # of decimates

push():
$c_i = 1$
$\Delta\phi(x) = [2(\ell+1) - 10k] - [2\ell - 10k]$
$= 2$
$\hat{c_i} = c_i + 2 = 3 \Rightarrow O(1)$

pull():
$c_i = 1$
$\Delta\phi(x) = [2(\ell-1) - 10k] - (2\ell - 10k)$
$\Rightarrow -2$ it's okay for $\Delta\phi(x)$ to be negative
$\hat{c_i} = 1 + (-2) = -1 \Rightarrow O(1)$

size():
We need to assume the size is kept track of when pulling or pushing
$c_i = 1 \Rightarrow O(1)$
$\Delta\phi = 0$, doesn't change
$\hat{c_i} = c_i + \Delta\phi = c_i = 1 \Rightarrow O(1)$

decimate():
$c_i = 1$
$\Delta\phi = (2i + 10(k+1)) - (2i - 10k)$
$= -10k - 10 + 10k$
$= -10$
$\hat{c_{i+1}} = 1 - 10 \Rightarrow O(1)$

Overall, $\dfrac{O(1) + O(1) + O(1) + O(1)}{4} \Rightarrow \boxed{O(1)}$