HW3 - Data Science

S Deepak Narayanan
16110142

4)

a) No. This is because it could so happen that we have a sample like $\boxed{1}$ $\boxed{1}$ $\boxed{1}$ in the first case, whereas that won't occur in case two (second case).

In second case, we will not have reservoirs holding same values multiple times.

In case 1, it ~~always~~ could be the case that we have same items sampled multiple times.

We could have all possible samples that we have when we have a reservoir of size k in the multiple reservoir case but the vice versa is not true.

k size storage $\Rightarrow$ k 1 size storage

k 1 size storage $\Rightarrow$ k size storage

Eg: $\boxed{1}$ $\boxed{1}$ $\boxed{1}$ $\longleftrightarrow$ $\boxed{1 \mid 2 \mid 3}$, Never in case 2

b) By induction hypothesis, assume that we have i+1 elements in stream and we have reservoir of size k.

Suppose first i elements follow the fact that Pr[element in reservoir] = $\frac{k}{i}$

For the $(i+1)^{th}$ item,

Probability of it getting added is $\frac{k}{i+1}\left(\frac{1}{k} + \cdots \frac{1}{k}\right)$

$= \frac{k}{i+1}$ (Since we choose up $\frac{k}{i+1}$) $\boxed{\frac{k}{k} = 1}$

($\frac{1}{k} \rightarrow$ Pr of replacing an element) and dont choose up $1 - \frac{k}{i+1}$) $\rightarrow$ for n element

~~Probability of an existing element staying at time i a ~~new element getting chosen~~ and that it it staying in reservoir at time i is not replaced at time i+1.~~

$$\Rightarrow \cancel{\frac{k}{i}\left(1 - \left(\frac{k}{i+1}\right)\cdot\left(\frac{1}{k}\right)\right)} = \frac{k}{i+1}$$

(b) Probability an old element stayed in stream at time i, and the probability it wasn't kicked out at time i+1.

$$\Rightarrow \frac{k}{i}\left(1 - \left(\frac{k}{i+1}\right)\left(\frac{1}{k}\right)\right) = \frac{k}{i+1}$$

(c) No. This is not independent sampling. Our sample space reduces if we went to have the following. Suppose element x_1 is present in reservoir and we want to compute probability that x_2 is also added to reservoir.

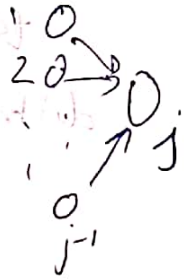
Scanned by CamScanner

without kicking out x_1. ~~This is~~ Assume at time $(t+1)$ x_2 comes. If we add x_2 to reservoir we need to ensure that x_1 is not kicked out, ie, $\Pr$ ~~of~~ $[x_2$ along with $x_1] = \left(\frac{k}{t+1}\right)\left(\frac{k-1}{k}\right)$

$\Rightarrow$ Not independent, since this varies ~~as~~ number of samples varies

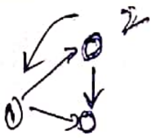
---

3) $p \rightarrow$ Probability Vector



$$T \quad \begin{array}{c|ccc} & & j & \\ \hline i & \cancel{p_2} & p_j & p_n \end{array} \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{bmatrix}$$

$p_{1j} \quad p_{2j} \cdot \quad p_{j-1j}$



$$\begin{bmatrix} p_1 & p_2 & p_3 \\ & p_3 & p_1 \\ & & p_2 \\ & & p_3 \end{bmatrix}$$

$p = Tp$ for stationarity

$$\delta_{ij} = \begin{cases} 1, & \text{edge from } i \to j \\ 0, & \text{else} \end{cases}$$

$$\begin{matrix} p_1 p_2 + p_3 p_3 \\ p_1 p_1 + p_3 p_3 \\ 0 \end{matrix} \begin{pmatrix} p_2^2 + p_3^2 \\ p_1^2 + p_3^2 \\ 0 \end{pmatrix} = \begin{bmatrix} \\ \\ \\ \end{bmatrix}$$

Added to make stationarity

$$\boxed{\delta_{ij} = \begin{cases} 1 & \text{edge from } i \to j \\ \cancel{0} & \text{else} \end{cases}}$$

$$p_i = \sum_{j=1}^{n} p_j^2 \delta_{ij} + x$$

~~xo~~ We can edit only the p_ith element if we have to add

Prc self loops. So, $x = p_i x'$

$$p_i = \sum_{j=1}^{n} p_j^2 \delta_{ij} + p_i x'$$

$$x' = \frac{1 - \sum_{j=1}^{n} p_j^2 \delta_{ij}}{p_i}$$

We need to add self loops summing upto the above value to make p, a stationary distribution.

Scanned by CamScanner

self loops. So, $x = p_i \cdot x'$

$$p_i = \sum_{j=1}^{n} p_j^2 \delta_{ij} + p_i x'$$

$$x' = \frac{1 - \sum_{j=1}^{n} p_j^2 \delta_{ij}}{p_i}$$

We need to add self loops summing upto the above value to make p, a stationary distribution.

In this sampling from a random surfer shouldnt affect the efficiency except for the simple fact that we can store the current node and only its neighbours in memory.

It is very efficient in this case because we already start off with a stationary distribution and dont need to run random walks a lot of times and because we have limited memory (Walk needs only neighbor data) its efficient.