CSOR W4231 Analysis of Algorithms I - Spring 2023

Homework 4
Tong Wu, tw2906

Problem 1

The problem states that a graph has a starting node v_1, and at every time step t with $1 < t \leq n$, a single node v_t will be added into the graph. An outgoing edge is also added into the graph, the source is v_t and the destination is chosen uniformly at random.

Section a

For every possible time step t, there will be $t-1$ nodes in the graph, so there will be $t-1$ possible destination nodes for the outgoing edge. The probability that the destination of the edge is node v_j will be $\frac{1}{t-1}$ since it is chosen at uniformly random. Write X as the sum of the indicator random variable which state as the total number of ingoing edge for node v_j. Where X_t demonstrates the status of edge outgoing from node v_t is going to node v_j or not at specific time step t, such that:

$$X_t = \begin{cases} 1, & \text{if the outgoing edge from node } v_t \text{ has destination node } v_j \\ 0, & \text{otherwise} \end{cases}$$

Let G be the graph at the end of time step n, for all possible time step t, the total number of comparison be:

$$X = \sum_{t=j+1}^{n} (X_t)$$

Then the equation should be:

$$\begin{aligned} E[X] = E[\sum_{j+1}^{n} X_t] = \sum_{j+1}^{n} E[X_t] &= \sum_{j+1}^{n} Pr[X_t = 1] \\ &= \sum_{j+1}^{n} \frac{1}{t-1} \\ &= \frac{1}{j} + \frac{1}{j+1} + \ldots + \frac{1}{n-1} \\ &\approx \int_j^n \frac{1}{x}\, dx \\ &= \ln(n) - \ln(j) \\ &= \Theta(ln(\frac{n}{j})) \end{aligned}$$

Hence, the expected number of edges entering node v_j in G express using Θ notation is $\Theta(ln(\frac{n}{j}))$.

Section b

According to section a, the probability that the destination of the edge is node v_j will be $\frac{1}{t-1}$, hence the probability that the destination of edge is not v_j will be $1 - \frac{1}{t-1} = \frac{t-2}{t-1}$. For each time step, the probability should be all $\frac{t-2}{t-1}$, and each event are independent. Change value t to k, which represent the total number of nodes in the graph, the probability should now be: $\frac{k-1}{k}$.

In order to calculate the probability of a specific node has no incoming edges, it should first mention that at the last time step n, there are total $n-k$ times of random connection of the edge for node v_k. For each possible time step t with $k < t \leq n$, the probability that node v_k is not be connected is $\frac{t-2}{t-1}$ .The equation of the probability should be written as below using telescoping product:

$$\begin{aligned} Pr[\text{has no incoming edge at time step } n] &= \prod_{t=k+1}^{n} \frac{t-2}{t-1} \\ &= \frac{k-1}{\cancel{k}} \times \frac{\cancel{k}}{\cancel{k+1}} \times \frac{\cancel{k+1}}{\cancel{k+2}} \times \ldots \times \frac{\cancel{n-2}}{n-1} \\ &= \frac{k-1}{n-1} \end{aligned}$$

Write X as the sum of the indicator random variable which state as the total number of nodes in graph G where there are no incoming edges at time step n. Where X_k demonstrates the status of there is incoming edge for node v_k or not at time step n, such that:

$$X_k = \begin{cases} 1, & \text{if the incoming edge to node } v_k = 0 \\ 0, & \text{otherwise} \end{cases}$$

Let G be the graph at the end of time step n, for all possible time step t, the total number of comparison be:

$$X = \sum_{k=1}^{n} X_k$$

Then the equation should be:

$$\begin{aligned} E[X] = E[\sum_{k=2}^{n} X_k] = \sum_{k=2}^{n} E[X_k] &= \sum_{k=2}^{n} Pr[X_k = 1] \\ &= \sum_{k=2}^{n} \frac{k-1}{n-1} \\ &= \frac{1}{n-1} \sum_{k=2}^{n} (k-1) \\ &= \frac{1}{n-1} \sum_{k=1}^{n-1} (k) \\ &= \frac{1}{n-1} (1+2+3+\ldots+(n-1)) \\ &= \frac{1}{n-1} (\frac{n \times (n-1)}{2}) \\ &= \frac{n}{2} \end{aligned}$$

Hence, the expected number of nodes that with no incoming edge is $\frac{n}{2}$.

Problem 2

The problem state that a bidding system will update its maximum bidding price b_{max} to the highest price that bidding agent take at each time of bidding. There are n bidding agent and agent i has the bidding price $b_i > 0$. All agents appear in an order chosen uniformly at random.

- At the beginning, the $b_{max} = 0$ since no bidding. After the first bidding agent bids, b_{max} must update since $b_i \geq 0$.
- For the second bidding agent, since the bid price also follow the condition $b_i \geq 0$, so the price will have probability $\frac{1}{2}$ higher than the first agent to become the highest price, which makes b_{max} update.
- For the third agent, there are six possible order of bid price:
  1. b_3, b_2, b_1
  2. b_3, b_1, b_2
  3. b_2, b_1, b_3
  4. b_2, b_3, b_1
  5. b_1, b_2, b_3
  6. b_1, b_3, b_2

  There are two orders can let $b_3 > b_{max}$, which the probability should be $\frac{1}{3}$.
- In general, can summarize that the probability for the i-th agent update b_{max} should be $\frac{1}{i}$

Write X as the sum of the indicator random variable which state as the total number of times that b_{max} is updated. Where X_i demonstrates the status of there is an update of b_{max} or not, such that:

$$X_i = \begin{cases} 1, & \text{if the } i\text{-th bid causes } b_{max} \text{ update} \\ 0, & \text{otherwise} \end{cases}$$

For all possible bidding agent b_i, the total number of comparison be:

$$X = \sum_{i=1}^{n} X_i$$

Then the equation should be:

$$
\begin{aligned}
E[X] = E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[x_i] &= \sum_{i=1}^{n} Pr[X_i = 1] \\
&= \sum_{i=1}^{n} \frac{1}{i} = H_n \\
&= \Theta(ln(n))
\end{aligned}
$$

Hence, the maximum bidding price b_{max} will update $ln(n)$ times.

Problem 3

Section a

For this algorithm, the line 8 in the code shows that the algorithm will return error if the rank does not in the given bound. This statement indicates that the algorithm may return error or incorrect result with a probability. Hence, this algorithm is a Monte Carlo algorithm which has a fixed running time but cannot guarantee correct results.

Section b

In code of the algorithm:

1. Both line 1 and 2 costs $\mathrm{O}(1)$.
2. The for loop starts from line 3 to line 5, the loop takes $\mathrm{O}(n)$ to complete its iterations. Inside of the loop, the if condition and its statement both costs $\mathrm{O}(1)$.
3. Line 7 to line 9, each line costs $\mathrm{O}(1)$.

Hence, the overall running time of this algorithm should be $\mathrm{O}(n)$.

Section c

For the succession of the algorithm, it needs at least $\frac{n}{4}$ items are smaller than a_i and at least $\frac{n}{4}$ items greater than a_i. Let variable `rank` indicates the level of a_i. Denote the probability of the algorithm succession as:

$$\begin{aligned}
& Pr[\frac{n}{4} < \text{rank} \leq \frac{3n}{4}] \\
=& 1 - (Pr[\frac{n}{4} > \text{rank}] + Pr[\text{rank} \geq \frac{3n}{4}]) \\
=& 1 - (\frac{1}{4} + \frac{1}{4}) \\
=& 1 - \frac{1}{2} \\
=& \frac{1}{2}
\end{aligned}$$

Hence, the success probability of this algorithm is $\frac{1}{2}$, 50%.

Section d

In order to let the success probability of the algorithm over 99%, one way is to run the algorithm in multiple times. When run the algorithm in multiple times, the success probability of each execution will not influence other executions since its distinction.

Denote the algorithm needs to be multiplied run k times, the probability of success should be:

$$Pr = 1 - \frac{1}{2}^{k} > 0.99$$
$$\frac{1}{2}^{k} < 0.01$$
$$k \geq 7$$

Hence, if the algorithm runs 7 times, the success probability will be $1 - \frac{1}{2}^{7} \approx 99.2\%$. The running time of the resulting algorithm should be $\mathrm{O}(7n)$, and $\mathrm{O}(kn)$ in general. The value of k depends on how much probability would like to take. The running time now is still linear, so the overall running time is still $\mathrm{O}(n)$.