Problem Set 8 Solutions

This problem set is due **at 11:59pm** on **Thursday, November 17, 2016.**

EXERCISES (NOT TO BE TURNED IN)

- CLRS C.2-2, page 1195 [Probability Review]
- CLRS 5.4-6, page 142 [Probability Review]
- CLRS 9.2-4, page 220 [Randomized Algorithms & Worst-case Performance]

Problem 8-1. Unreliable Sorting [30 points]

Professor Jane, at the University of Monte Carlo, has a list of n numbers that she needs to sort using only her binary comparator, which takes as input two numbers and is supposed to output the larger of the two. Unfortunately, her comparator is unreliable, and sometimes gives an incorrect result. However, she knows that the result of each comparison she makes is independent of the results of all other comparisons, and that the probability of error is at most $\frac{1}{5}$.

Give an algorithm to sort this list of numbers that makes $O(n \log^2 n)$ comparisons and has a 99% chance of returning the correct sorted answer.

Solution: The idea to solve this problem is to amplify the accuracy by repeating the comparisons. Suppose Professor Jane is trying to compare two items a and b. Instead of comparing them once, Prof. Jane compares them k times (for an odd k) and takes the answer that appears in the majority of her attempts. (We will call this an amplified comparison.) In other words, if the comparison function indicates a is greater than b at least $\lceil k/2 \rceil$ times, she will assume that a is greater than b. Otherwise, she will assume b is greater than a. Therefore, the amplified comparison is incorrect if and only if the comparison function returns the wrong answer $X \geq k/2$ times.

Note that the expected number of wrong answers $\mathbb{E}[X]$ is $k/5$.

Let X_i be 1 if comparison i returns the wrong answer and 0 otherwise. Let $X = \sum_{i=1}^{k} X_i$, so that X is the total number of wrong answers made during a specific amplified comparison.

Because the X_is are independent (by assumption) Bernoulli random variables, we can apply the Chernoff bound:

$$P(\text{amplified comparison is wrong}) = P\left(X > \frac{k}{2}\right) = P\left(X > \left(1 + \frac{3}{2}\right) k/5\right) < e^{-(3/2)\frac{k}{5}/3} = e^{-k/10}$$

Let $k = 20 \log(n) \in O(\log(n))$. The probability of a specific amplified comparison failing is then $1/n^2$.

If we use Merge sort, we need to make $O(n \log(n))$ amplified comparisons. If no amplified comparison fails, then the entire sorting procedure will certainly succeed. Conversely, for the sorting procedure to fail, at least one amplified comparison must fail. Now by the union bound we have

$$P(\text{sorting fails}) \leq P(\bigcup_{\text{amplified comparisons}} \text{amplified comparison fails}) \leq O(n \log n)\frac{1}{n^2} < 0.01$$

for sufficiently large n.

Thus for large n there is at least a 99% chance of sorting correctly. The total number of comparisons made is $O(\log n)O(n \log n) = O(n \log^2 n)$.

(It is actually possible to solve this problem with $O(n \log n)$ comparisons!)

Problem 8-2. Matching [30 points]

In this problem we will develop a randomized algorithm for computing a perfect matching in some simple undirected bipartite graph $G = (V = L \cup R, E \subseteq L \times R)$ where $L = \{l_1, ..., l_n\}$ and $R = \{r_1, ..., r_n\}$ (so $|L| = |R| = n$). Let $m = |E|$.

Throughout this problem, it may be helpful to use the fact that a matching is perfect if and only if it consists of exactly $\frac{|V|}{2}$ edges.

(a) [10 points] Denote by $G_{\setminus ij}$ the subgraph of G induced on $V \setminus \{l_i, r_j\}$. That is, $G_{\setminus ij}$ is the graph obtained from G by removing vertices l_i and r_j as well as any edges connected to either l_i or r_j.

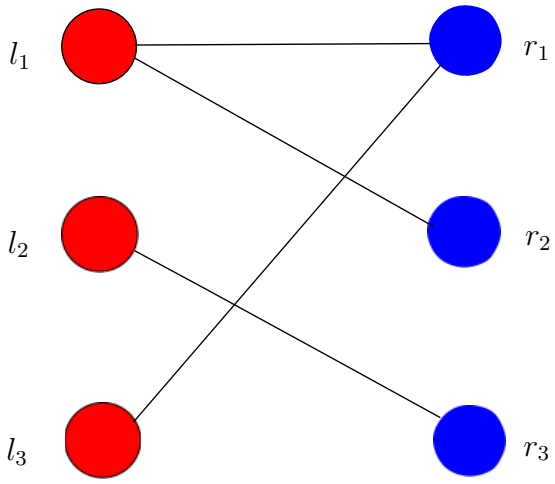
Prove that G contains a perfect matching if and only if for any $l_i \in L$, there exists some $r_j \in R$ such that the edge $\{l_i, r_j\}$ exists in G and $G_{\setminus ij}$ contains a perfect matching.

Solution:

- $\Rightarrow$: Given any perfect matching M, every $u \in L$ is matched to some $v \in R$. Removing that edge from M produces a set of edges M' of size $n - 1$. Every edge in M' is an edge of $G_{\setminus uv}$, because we cannot have more than one edge in M touching either u or v. Also, M' is a matching on $G_{\setminus uv}$, because if it were not, and two of its edges shared an endpoint, that sharing would mean that M itself was not a matching. Finally, $G_{\setminus uv}$ has $n-1$ vertices on each side, so we conclude that M' is perfect on $G_{\setminus uv}$.
- $\Leftarrow$: Pick any particular $u \in L$, and let $v \in R$ be the vertex guaranteed by the consequent. Construct a perfect matching by taking a perfect matching in $G_{\setminus uv}$ and adding to it the edge $\{u, v\}$. This is evidently a valid matching because u and v do not exist in $G_{\setminus uv}$, so the edge $\{u, v\}$ is compatible with any matching in $G_{\setminus uv}$. It is perfect because it has n edges.

(b) [15 points] Define M_G to be the symbolic matrix with entries $m_{ij} = x_{ij}$ if $\{l_i, r_j\} \in E$ and $m_{ij} = 0$ otherwise.

Example: Consider the graph G depicted below:



.

For this graph, we have

$$M_G = \begin{bmatrix} x_{11} & x_{12} & 0 \\ 0 & 0 & x_{23} \\ x_{31} & 0 & 0 \end{bmatrix}$$

.

Prove that the determinant of M_G is identically zero if and only if there does not exist a perfect matching in G. (An expression is identically zero if it equals 0 for all possible assignments of values to its variables. So for example $Det(M_G) = x_{12}x_{23}x_{31}$ is not identically zero, but $x_{11} - x_{11}$ is.)

Hint: Consider expanding the determinant by minors. Given an n-by-n matrix A with entries a_{ij}, denote by A_{ij} the submatrix obtained from A by removing the i^{th} column and j^{th} row. Then,

$$Det(A) = \sum_{j=1}^{n} (-1)^{i+j} a_{ij} Det(A_{ij})$$

for any $1 \leq i \leq n$.

Solution:

$\Rightarrow$: First we prove by induction the contrapositive of the statement, that if there exists a perfect matching in G, then $Det(M_G)$ is not identically zero.

For $n = 1$, the determinant is just a single variable if the edge exists or zero if it doesn't.

Now, suppose the claim has been proved for $1, ..., n-1$. Consider a graph with $|L| = |R| = n$. If the graph has a perfect matching, then apply part (a) with some

arbitrary vertex u—call its match v—and consider the expansion of the determinant by minors along the row corresponding to u. M_{uv} is $M_{G_{/uv}}$, so the inductive hypothesis tells us that $Det(M_{uv})$ is nonzero. Then all we need to show is that the term $(-1)^{u+v}x_{uv}Det(M_{uv})$ won't be canceled out, but this is clear because this is the only term that x_{uv} shows up in.

$\Leftarrow$: We now prove the inverse of the statement, that if $Det(M_G)$ is not identically zero, then there exists a perfect matching in G.

For $n = 1$, if $Det(M_G)$ is not identically zero, then it must be simply x_{11}, so the edge $\{l_1, r_1\}$ exists, so there is a perfect matching.

Now, suppose the claim has been proved for $1, ..., n-1$. Consider a graph with $|L| = |R| = n$. If the determinant is not identically zero, then some term corresponding to i, j in the sum is not identically zero. So the i, j entry in M_G is nonzero, and $Det((M_G)_{ij})$ is nonzero. By the inductive hypothesis, there exists a perfect matching in $G_{\backslash ij}$, and combining this perfect matching with the edge $\{l_i, r_j\}$ produces a perfect matching in G.

(c) [5 points] Suppose that you have a black box that can deterministically compute the determinant of a numerical n-by-n matrix in constant time.

Give a randomized algorithm for determining whether a bipartite graph with n vertices on each side contains a perfect matching. Your algorithm should run in $O(n^2 \log n)$ time and succeed with probability at least $\frac{3}{4}$.

You may use the following lemma without proof:

Lemma: Given a not-identically zero polynomial $P : \mathbb{R}^n \to \mathbb{R}$ of degree d, if $r_1, ..., r_n$ are selected independently and uniformly at random from a finite subset S of $\mathbb{R}$, then the probability that $P(r_1, ..., r_n) = 0$ is at most $\frac{d}{|S|}$.

Solution: The degree of the determinant in part b) is at most n (easily proved by induction). So it suffices to pick from $S = [1, 4n]$. We evaluate the polynomial by using the determinant black box. A random integer in S can be picked in $O(\log 4n) = O(\log n)$ time, and we need to do this $O(n^2)$ times. So the total runtime is $O(n^2 \log n)$.

Problem 8-3. Ice Cream Tournament [30 points]

Toscanini's (MIT's favorite ice cream shop) has developed a new ice cream flavor for MIT students. But before they start serving it at the store, they want to know whether or not MIT students like this new flavor. They have selected a group of n MIT students, where n is odd, and if a majority of the n students like the flavor, it passes the test. Otherwise, it fails the test.

Each student will either like the flavor or not. To determine whether a student likes the flavor, he/she must be given a sample of the ice cream (and this is also sufficient to determine the student's preference).

Toscanini's wants to conduct this test as efficiently as possible. That is, they would like to determine the outcome of the test while giving out as few samples as possible.

(a) [5 points] Prove that if Toscanini's wants to be guaranteed of a correct result, it cannot give out fewer than n samples.

Solution:

Roughly, if only k are tested, it is possible that $\lfloor \frac{k}{2} \rfloor$ like it and the rest don't. The remaining $n - k$ preferences could swing the overall result either way.

More precisely, suppose you have some deterministic algorithm A to test the students' preferences. We will adversarially construct a pair of inputs on which A cannot succeed.

Let a_1 be the index of the first student A picks to test. Our adversarial input I will set the preference of the a_i^{th} student to be Yes; i.e. $I[a_1] = Yes$. Given the outcome of this test, A now picks a second student a_2. We set $I[a_2] = No$. And so on, alternating back and forth, so that $I[a_{2k+1}] = Yes$ and $I[a_{2k}] = No$.

Now, for all students j that are not tested by A, set $I[j] = Yes$, so that the majority of students prefer the ice cream flavor. But now if we had instead set $I[j] = No$ for all students not tested by A, the majority of students would not prefer the ice cream flavor. A cannot output the correct answer in both cases.

Instead of testing each student's preference in a linear fashion to find the majority, Toscanini's decides to conduct a different test.

Each employee will ask three students if they liked the new flavor. A student will never be asked for their opinion more than once. The employee will report to its manager whether the majority of those three students liked the flavor. Each employee has exactly one manager.

The managers each have three employees. A manager will receive the judgments of its employees, take the majority among those, and report a judgment ("popular" or "unpopular") to *its* manager.

These judgments progress up the chain of management until reaching the CEO of Toscanini's, Inc. The CEO will hear the judgments of its three most senior advisers, take the majority of those, and

that will be the determination of the flavor's popularity – and whether to serve this ice cream flavor or not.

We model this procedure as a perfect ternary tree (each non-leaf node has 3 children, and all leaves are at the same depth) with $n = 3^h$ leaves for some positive integer h.

Each leaf node represents a student. For leaf nodes, we define the value of a node as 1 if the student likes the flavor and 0 otherwise. For internal nodes, we define the value of a node as the majority of the values of its three children.

If the root node's value is 1, the flavor passes the test. Otherwise, the flavor fails the test. (Note that this test does not necessarily find the majority in the original set.)

(b) [10 points] Using this tree construction for the taste test, prove that regardless of the values of the leaves, there exists a set of leaves of size $O(2^{\log_3 n}) = O(n^{\log_3 2})$ whose values determine the outcome of the test.

Solution: Let $v(x)$ denote the value of node x. For each node x, we know that there are at least two children x_1 and x_2 such that $v(x_1) = v(x_2)$. Thus, we only need to know the values of these two nodes by evaluating recursively, and we can ignore the value of third child since it will not affect the outcome. Let $T(n)$ be the minimum number of leaves needed to evaluate the outcome of a test where there are n leaves in the tree. Our recurrence is then $T(n) = 2T(n/3)$, where $T(1) = 1$. Finding $T(n)$ in closed form, we have $T(n) = 2^{\log_3 n} = n^{\log_3 2}$.

(c) [15 points] Based on your observation from part (b), give a randomized algorithm that correctly determines the outcome of the test and only gives ice cream samples to $O(n^{(\log_3 8)-1})$ students in expectation.

Solution: For a node x, randomly select two children x_1 and x_2 and evaluate them. If $v(x_1) = v(x_2)$, return the value of $v(x)$ as $v(x_1)$. Otherwise, evaluate $v(x_3)$, the third child, to break the tie.

If the three child values are equal, then we will always evaluate only two children. Otherwise, there is a $1/3$ probability that we will pick the two that are the same. The other $2/3$ of the time we will have to evaluate the third child as well. This gives us the worst-case recursion

$$T(n) = 2T(n/3) + 2/3T(n/3) = 8/3T(n/3) = n^{\log_3(8/3)}$$

.

Problem 8-4. Strange database [30 points]

You are working with a strange database containing n $(key, value)$ pairs. The keys are unique integers in the range $1, ..., n$. The only operations you have at your disposal are:

- *random*: Go to a random $(key, value)$ pair. key is picked independently and uniformly at random from $1, ..., n$.
- *successor*: Go to the pair with the key 1 greater than the key of the pair you are currently at (go from $key = i$ to $key = i + 1$). If you are currently at the pair with key n, *successor* has no effect.
- *predecessor*: Go to the pair with key 1 smaller than the key of the pair you are currently at (go from $key = i$ to $key = i - 1$). If you are currently at the pair with key 1, *predecessor* has no effect.

Your goal is to develop an efficient algorithm to get to the pair with key k.

(a) [3 points] Your first thought is to repeatedly call *random* until you find the pair with key k. What is the expected number of operations this algorithm will execute?

Solution: By symmetry, each call has a $\frac{1}{n}$ chance of taking you to k. So the expected number of calls until a success is n.

If you have not seen this sort of analysis before, here is a derivation:

Let p be the probability of success ($\frac{1}{n}$ in this case). Let X be the number of tries until success.

By definition of expectation, we can sum over all x where x is the number of failures before the first success:

$$\mathbb{E}[X] = \sum_{x=0}^{\infty}(x+1)(1-p)^x p$$

We can then manipulate this equation as follows:

$$\begin{aligned}\sum_{x=0}^{\infty}(x+1)(1-p)^x p &= \sum_{x=0}^{\infty} x(1-p)^x p + p\sum_{x=0}^{\infty}(1-p)^x \\ &= (1-p)\sum_{x=0}^{\infty} x(1-p)^{x-1} p + p\frac{1}{1-(1-p)} \\ &= (1-p)\sum_{x=0}^{\infty}(x+1)(1-p)^x p + 1 \\ &= (1-p)\mathbb{E}[X] + 1\end{aligned}$$

(Note that we could arrived at this directly by thinking recursively: on the first try, either we succeed, in which case we are done, or we fail and we're back to where we started.)

Solving this equation $\mathbb{E}[X] = \frac{1}{p} = n$.

(b) [7 points] Your next thought is to start by calling *random* to go to a random pair $(key, value)$. Then, if $key > k$, repeatedly call *predecessor* until you find the pair with key k. Otherwise, repeatedly call *successor* until you find the pair with key k.

What is the worst-case (with respect to possible inputs k) expected number of operations this algorithm will execute?

Solution: In the worst case, k is 1 or n, and your algorithm will execute $n/2$ operations in expectation.

(Exactly: if $k = 1$ and i is the index that random takes you to, the expected number of operations is $1 + \frac{1}{n}\sum_{i=1}^{n}(i-1) = 1 + \frac{1}{n}\frac{1}{2}(n-1)(n) = 1 + \frac{1}{2}(n-1) = \frac{1}{2}(n+1)$.

(c) [20 points] Design an algorithm that, for any k, makes $O(\sqrt{n})$ operations in expectation and always succeeds.

Hint: Use ideas from both of the previously analyzed algorithms.

Solution: Call *random* until you get something within $\sqrt{n}$ of k. Then call *predecessor* or *successor* to get to k. The chance of getting something within $\sqrt{n}$ of k is at least $\frac{\sqrt{n}}{n}$ (the worst case being when $k = 1$ or n), so the expected number of calls to *random* is $\frac{n}{\sqrt{n}} = \sqrt{n}$.

Problem 8-5. Friend Groups [40 points]

You are working on software for Socially Networked, Inc., who has acquired a database containing a friend graph of the entire world population. You are interested in finding out how many distinct, non-overlapping friend groups exist in this database.

Definition. A "friend graph" G is an undirected graph in which vertices represent people, and edges represent friendships: two vertices are connected by an edge if and only if the two people are friends.

Definition. A "friend group" in G is a subset of the people in G, where for any two people a and b in the friend group, there exists some list of friends which can be traversed from a to reach b, and vice versa. To use notation, for any pair of people a and b in the friend group, there exists a (possibly empty) list of friends $f_1, f_2, ..., f_k$ such that a and f_1 are friends, f_i and f_{i+1} are friends for all $1 \leq i \leq k-1$, and f_k and b are friends. Two people are in separate friend groups only if there is no way to reach from one person to the other through a chain of friendships.

Let c be the number of distinct, non-overlapping friend groups in G. In this problem, we will develop a sublinear algorithm which returns a value $\hat{c}$ which is an estimate for c.

More precisely, given G, and two parameters ϵ and δ, such that $0 < \epsilon, \delta < 1$, we want an algorithm with the following properties:

1. The algorithm outputs a value $\hat{c}$, which is a random variable such that,

$$|E[\hat{c}] - c| \leq \epsilon n$$

   with probability at least $\geq 1 - \delta$.

2. The algorithm makes $o(|V|)$ queries to the input graph.

(a) [2 points] Let n_v be the number of people in the friend group of person v (including v itself). Give a simple deterministic polynomial time algorithm to determine n_v, and determine its running time.

Solution: Use breadth first search starting from vertex v, counting how many unique vertices are visited. This runs in $O(|V| + |E|)$ time.

(b) [4 points] Using your algorithm from part (a), give a deterministic polynomial time algorithm that computes the exact value of c. Prove its correctness and analyze its running time.

Hint: If G has just one friend group, what would be a person's n_v value? How could your algorithm detect that there is only one friend group, that spans the entire graph, using just the n_v values it calculated? How would it know if there were two or more disjoint friend groups?

Solution: Using the algorithm from part (a), compute n_v for every vertex $v \in V$. Return $\sum_{v \in V} \frac{1}{n_v}$. This runs in $O(|V|(|V| + |E|))$ time.

To show correctness, we examine one friend group at a time. Suppose the vertices V are split into disjoint friend groups $C_1, C_2, ..., C_c$, so that $V = \cup_{k=1}^{c} C_k$.

Consider now a particular friend group, C_k. Note that all vertices in that friend group C_k have the same value of n_v: call this value x. If we restrict the summation above to just look at vertices in C_k, we have $\sum_{v \in C_k} \frac{1}{n_v} = |C_k| * \frac{1}{C_k} = 1$.

Then evaluating the entire sum, $\sum_{v \in V} \frac{1}{n_v} = \sum_{k=1}^{c} \sum_{v \in C_k} \frac{1}{n_v} = \sum_{k=1}^{c} 1 = c$. Thus this sum gives us the correct number of friend groups, c.

The following algorithm computes an approximation of c, with sublinear query complexity:

1. Select a multiset of k vertices, $v_1, ..., v_k$, uniformly and independently at random from V, where
$$k = \lceil \frac{2\ln(2/\delta)}{\epsilon^2} \rceil$$
.
2. For each i from 1 to k, compute $\tilde{n_v}$, a lower bound of how many vertices are in v_i's friend group (do this via a truncated BFS starting at v_i, visiting at most $\frac{2}{\epsilon}$ vertices, counting each unique vertex visited).
3. Return
$$\hat{c} = \frac{n}{k} \sum_{i=1}^{k} \frac{1}{\tilde{n_{v_i}}}$$
.

(c) [4 points] What is the running time of this algorithm?

Solution: The running time of this algorithm is the cost of doing k different truncated Breadth First Searches (BFS). BFS runs in $O(|V| + |E|)$ time. Since these BFSs are truncated after visiting $\frac{2}{\epsilon}$ vertices,

$$O(|V| + |E|) = O\left(\frac{1}{\epsilon^2}\right).$$

So the final running time is

$$O\left(k * \frac{1}{\epsilon^2}\right) = O\left(\frac{\ln(1/\delta)}{\epsilon^4}\right).$$

In the steps that follow, you will prove that this algorithm is correct. That is, if the algorithm outputs $\hat{c}$, you will show that

$$|\hat{c} - c| \leq \epsilon n$$

with high probability.

We will show this in two steps. Let $\mu = E[\hat{c}]$. Then notice that by the triangle inequality that:

$$|\hat{c} - c| \leq |\hat{c} - \mu| + |\mu - c|.$$

We will therefore show first that $|\mu - c| < \epsilon n/2$. And then we will show also that with high probability, $|\hat{c} - \mu| < \epsilon n/2$.

(d) [10 points] Show that $|\mu - c| \leq \epsilon n/2$

Hint: Give an expression for $\mu = E[\hat{c}]$ in terms of k and $\tilde{n_v}$, and compare with your expression for c from part (b).

Solution:

Let us first find an expression for μ.

$$\begin{aligned}
\mu &= E_i[\hat{c}] && (1)\\
&= E_i\left[\frac{n}{k}\sum_{i=1}^{k}\frac{1}{\tilde{n_{v_i}}}\right] && (2)\\
&= \frac{n}{k}\sum_{i=1}^{k}E_i\left[\frac{1}{\tilde{n_{v_i}}}\right] && (3)\\
&= \frac{n}{k} * kE_i\left[\frac{1}{\tilde{n_{v_1}}}\right] && (4)\\
&= n * \frac{1}{n}\sum_{v\in V}\frac{1}{\tilde{n_v}} && (5)\\
&= \sum_{v\in V}\frac{1}{\tilde{n_v}}. && (6)\\
& && (7)
\end{aligned}$$

To go from line (3) to line (4), notice that every vertex u has the same value for $E_i\left[\frac{1}{\tilde{n_u}}\right]$. This is by symmetry - all vertices have an equal chance of being chosen, and so this expectation is just going to be the average value

$$\sum_{v\in V}\frac{1}{\tilde{n_v}}P_i(v_i = v) = \frac{1}{n}\sum_{v\in V}\frac{1}{\tilde{n_v}}.$$

This means that we can replace the sum of k of these expectations, with just k times that expectation, since they are all the same.

We then use the expression above for this expectation to move from line (4) to line (5).

We can get some intuition for what μ represents: μ is the estimate we'd return if we did the truncated BFS from *every* node, rather than from k vertices, and then just added up the reciprocals of those values. This tells us that doing the truncated BFS estimate from a random k vertices, on average will give us the same answer as doing the truncated BFSs from all the vertices.

Claim 1 $|\mu - c| \leq \epsilon n$.

Proof. We can write the quantity of interest as follows:

$$\begin{aligned}|\mu - c| &= |\sum_{v \in V} \frac{1}{\tilde{n_v}} - \sum_{v \in V} \frac{1}{n_v}| \\ &= |\sum_{v \in V} \left(\frac{1}{\tilde{n_v}} - \frac{1}{n_v} \right)|\end{aligned}$$

For a given node, either $\tilde{n_v} = n_v$ or $\tilde{n_v} = \frac{2}{\epsilon} \leq n_v$ (because of the truncated BFS). So,

$$\tilde{n_v} \leq n_v \rightarrow \frac{1}{\tilde{n_v}} - \frac{1}{n_v} \geq 0.$$

We can use this fact in the above quantity, so that:

$$\begin{aligned}|\mu - c| &= |\sum_{v \in V} \left(\frac{1}{\tilde{n_v}} - \frac{1}{n_v} \right)| \\ &= \sum_{v \in V} |\frac{1}{\tilde{n_v}} - \frac{1}{n_v}|\end{aligned}$$

We can then bound the sum as follows:

$$\begin{aligned}|\mu - c| &\leq \sum_{v \in V} |\frac{1}{\tilde{n_v}}| \\ &\leq \sum_{v \in V} \frac{\epsilon}{2} \\ &= \frac{\epsilon n}{2}\end{aligned}$$

This is the desired bound.

(e) [20 points] Show that

$$|\hat{c} - \mu| \leq \frac{\epsilon n}{2}$$

with high probability [probability at least $1 - \delta$]).

Note: You may use the following fact without proof.

Fact: Given independent random variables $X_1, ..., X_k$, $Y = \frac{1}{k} \sum_{i=1}^{k} X_i$, and $0 \leq X_i \leq 1$ for all i, then:

$$P(|Y - E[Y]| \geq t) \leq 2 \exp\left(-2kt^2\right)$$

(This is a particular case of the Hoeffding bound.)

Solution:

Claim 2 $|\hat{c} - \mu| \leq \frac{\epsilon n}{2}$ *with high probability.*

Proof. We will examine the following probability:

$$P(|\hat{c} - \mu| \geq \frac{\epsilon n}{2}) = P(|\hat{c} - E[\hat{c}]| \geq \frac{\epsilon n}{2}) \tag{8}$$

$$= P(|\hat{c}/n - E[\hat{c}/n]| \geq \frac{\epsilon}{2}) \tag{9}$$

Let $X_i = 1/\tilde{n_{v_i}}$, so that

$$Y = \frac{1}{k}\sum_i X_i = \frac{1}{k}\sum_i \frac{1}{\tilde{n_{v_i}}} = \hat{c}/n.$$

We see now that the probability we are interested in, from equation 9, is in fact just

$$P(|Y - E[Y]| \geq \frac{\epsilon}{2}).$$

The Hoeffding bound states the following:
Given independent random variables $X_1, ..., X_k$, $Y = \frac{1}{k}\sum_{i=1}^{k} X_i$, and $a \leq X_i \leq b$ for all i, then:

$$P(|Y - E[Y]| \geq t) \leq 2\exp\left(\frac{-2kt^2}{(b-a)^2}\right)$$

The fact given in the problem statement exhibits one particular case, when $a = 0$ and $b = 1$:

$$P(|Y - E[Y]| \geq t) \leq 2\exp\left(-2kt^2\right)$$

Clearly $0 \leq X_i \leq 1$ for all i, because $\tilde{n_{v_i}}$ is a positive integer. So the Hoeffding bound applies, and we will use it below.

$$P\left(|Y - E[Y]| \geq \frac{\epsilon}{2}\right) \leq 2\exp\left(-2k\left(\frac{\epsilon}{2}\right)^2\right) = 2\exp\left(\frac{-k\epsilon^2}{2}\right)$$

If we want this probability to be less than or equal to δ, then we can solve for k:

$$k \geq \frac{2}{\epsilon^2}\ln(2/\delta)$$

Thus there is a valid choice of k for which $|\hat{c} - \mu| \leq \frac{\epsilon n}{2}$ with high probability.
The proof of claims 1 and 2 completes the proof that this algorithm is correct.