Massachusetts Institute of Technology
6.856J/18.416J: Randomized Algorithms
David Karger

Wednesday, February 13, 2019

Problem Set 1 Solutions

Problem 1. Homework policy: The guiding principle behind the homework policy is that every student must put in significant effort towards solving every problem, and understand the solution completely before writing up the solution on their own. Anything that hinders either of these violates the collaboration policy.

(a) *You gather for a pset party where ten of you work together to solve the problems before separating to write up your own solutions.*
Violates: You should work in groups of at most 3 or 4. Larger group doesn't allow everybody to think about the problem themselves.

(b) *You work in a pair to solve a problem, taking turns writing ideas and calculations jointly in the same notebook as you work out the solution, then photograph those notes so you can each have a copy of your sketch before you separate to write up your own solutions.*
Does not violate: Working out calculations together is acceptable, as long as each person writes up the solution independently.

(c) *You work together to write up a detailed solution; one of you then dictates it to the other so that each of you can turn in a solution you've written yourself.*
Violates: Writing your own solution means to think through all the steps of the solution on your own. Dictating to another doesn't serve this purpose.

(d) *After you've solved a problem, your friend who is still working on the problem asks you for feedback on a solution idea. Leveraging your knowledge of the right answer, you identify and point out a flaw in their proposed solution.*
Does not violate: This scenario is okay as long as the student with the solution does not give away the answer to the other student; instead helps him/her understand the flaw and get back on track.

(e) *After you've solved a problem, you try to help your friend by acting as a sounding board while they work towards the solution. With the deadline approaching, since they are still stuck, you allow them to skim your writeup to get a general idea of how to approach the problem, which allows them to go off and independently figure out their solution and write it up.*
Violates: Student skimming the solution might end up not fully understanding the solution.

(f) *After you've worked with your friend to solve a problem and each done your own writeup, you exchange writeups and examine them to identify and correct any flaws in one or the other.*

Violates: This can get problematic if there are indeed flaws in the write up. If you do want to cross check your solution, then you must verbally explain your solution.

(g) *You remember seeing the homework problem in a journal article you read a few years ago, so you go find the journal article and read it to remind yourself how it went.*
Violates: If you are unable to remember the solution, then you must try to think through it again and solve it. Looking it up again would mean that you are not thinking critically about the problem.

(h) *You know the answer to the problem because you read it in a journal article a few years ago, so you explain the solution to your pset collaborators.*
Violates: It's okay if you already knew the solution having read it earlier. But by conveying the solution to the collaborators, you are not allowing them to think of the solution on their own.

Problem 2. **(a)** Let $0.b_1b_2\ldots$ be the binary fraction representation of p_1. We can treat the random bit stream $r_1, r_2, \ldots$ also as a binary fraction and return the first item if $r = 0.r_1r_2\ldots < p_1$ and the second item otherwise. Since the random bit stream is unbiased, we will return the first item with probability p_1. We will make the comparison of the two binary fractions by comparing b_i and r_i starting from $i = 1$. If the two bits differ (with probability 0.5), we can immediately conclude if $r < p$ and return the appropriate item from S. If they are the same (with probability 0.5), we need to compare additional bits to determine the direction of the inequality. Thus, for any arbitrary p_1, the probability that it will take exactly n comparisons to choose one of the two items is $\left(\frac{1}{2}\right)^n$. Thus, the expected number of bits to examine is $B = \sum_{n=1}^{\infty}\left(\frac{1}{2}\right)^n n = \sum_{n=1}^{\infty}\left(\frac{1}{2}\right)^n + \sum_{n=2}^{\infty}\left(\frac{1}{2}\right)^n + \cdots + \sum_{n=i}^{\infty}\left(\frac{1}{2}\right)^n + \cdots = 1 + \frac{1}{2} + \cdots + \frac{1}{2^{i-1}} + \cdots = 2 \in O(1)$.

(b) When there are more than 2 items in S, we associate S with the root node of a binary tree that at each node divides the associated set into 2 roughly even partitions. For each node i, we can compute the probability p_i of drawing an element from the left subtree by summing up the probabilities associated with the elements associated with the subtrees and normalize such that probability of descending into the two subtrees sum up to 1. To draw a sample from the set, we recursively compare the current random bit stream against p_i at the node starting at the root, and descend down the appropriate subtree based on the result of the comparison. The recursion ends when we reach a subtree with only a single element and return that element. Since the height of the tree is $O(\log n)$ and each comparison uses $O(1)$ random bits in expectation, the overall algorithm uses $O(\log n)$ random bits in expectation per sample. [1]

[1] By computing cumulative probabilities $c_i = p_1 + \cdots + p_i$ and reusing the random bit stream when

(c) Assume that a finite number of bits n suffices for the uniform distribution on 1, 2, 3. This means that at the highest resolution, we are able to divide the [0,1] interval into 2^n equally sized subintervals. But we need to apportion intervals so that equal numbers of intervals are assigned to elements 1, 2, and 3. This is impossible since 3 does not divide 2^n. More generally, we have 2^n equi-probable states, which must be evenly divided among three elements. $\Rightarrow\Leftarrow$. Therefore, finite bits will not suffice in the worst case.

Problem 3. Let X_{ij} be the indicator random variable for whether the ith and jth items get compared or not. We use the law of total probability, breaking the analysis into three cases. First, the initial pivot is equal to one of the two items with probability $\frac{2}{n}$, in which case they are compared with probability 1. Next, the initial pivot separates the two items with probability $\frac{j-i-1}{n}$, in which case they are compared with probability 0. Finally, the initial pivot does not separate the two items with probability $\frac{n-j+i-1}{n} = 1 - \frac{j-i+1}{n}$. Our inductive hypothesis is that the probability that the ith and jth elements are compared (in a list containing both) is $\leq \frac{2}{j-i+1}$. Indeed, we can verify the base case of this induction in a list where the ith and jth elements are the smallest and largest, respectively: with probability $\frac{2}{j-i+1}$, one is chosen as a pivot and they are compared, and otherwise, they are not compared. Using our inductive hypothesis,

$$E[X_{ij}] \leq \frac{2}{n} \cdot 1 + \frac{j-i-1}{n} \cdot 0 + \left(1 - \frac{j-i+1}{n}\right) \frac{2}{j-i+1} = \frac{2}{j-i+1},$$

as desired.

Problem 4. **(a)** Since x is randomly selected, $s \in \{0, 1, \ldots, n-1\}$ with equal probability $\frac{1}{n}$. When $s \geq k$, the size of the recursive set is s. Otherwise, it is $n-s-1$ [2]. Thus, the expected size of the recursive set is

$$\begin{aligned}
\sum_{s=k}^{n-1} \frac{s}{n} + \sum_{s=0}^{k-2} \frac{n-s-1}{n} &= \frac{(n+k-1)(n-k)}{2n} + \frac{2(k-1)(n-1) - (k-2)(k-1)}{2n} \\
&= \frac{n^2 - 2k^2 + 2nk - 3n + 2k}{2n} = n\left(\frac{1}{2} - \left(\frac{k}{n}\right)^2 + \frac{k}{n}\right) - \frac{3}{2} + \frac{k}{n} \\
&< \left(-\left(\frac{k}{n}\right)^2 + \frac{k}{n} + \frac{1}{2}\right) n
\end{aligned}$$

Defining $b = -\left(\frac{k}{n}\right)^2 + \frac{k}{n} + \frac{1}{2}$, it obtains its maximum of $\frac{3}{4}$ at $\frac{k}{n} = \frac{1}{2}$. Thus, the expected size of the recursive set of is at most $\frac{3}{4}n$.

---

performing binary comparison over c_i, we can further reduce the constant factor of the $O(\log n)$ algorithm.

[2] We assume that if $s = k-1$, the algorithm terminates and returns x.

(b) Let $T(n)$ be the expected runtime of FIND on an n-element set. In the base case, $T(1) = 1$ since we can simply return the single element in the set. When $n > 1$, the algorithm picks a random pivot and divides the set into two in n steps. The recursion continues on a subset S' whose expected size is bounded by $\frac{3}{4}n$. Thus, assuming $T(\cdot)$ is linear in n, $T(n) = n + E[T(\text{sizeof}(S'))] = n + T(E[\text{sizeof}(S')]) \leq n + T(\frac{3}{4}n)$. This recurrence yields $T(n) \leq n\left(1 + \frac{3}{4} + \left(\frac{3}{4}\right)^2 + \cdots\right) = 4n$. Thus, the expected running time of FIND on an n-element set is at most $4n$.

(c) In the above analysis, we invoked the linearity of expectations $E[T(\cdot)] = T(E[\cdot])$ when computing the recursive runtime. If $T(\cdot)$ is not linear, we cannot move the expectation inside the function. Thus, the transformation and resulting time bound will be invalid.

Problem 5. **(a)** Since the edges are assigned a random score, the order of edge selection in Kruskal's minimum spanning tree algorithm is also random. In Kruskal's algorithm, an edge is only added if it connects two currently disjoint subsets in the graph. Thus, after adding an edge, we can remove all unselected edges across connected vertices in the graph without affecting the algorithm. This is equivalent to performing a contraction on the two disjoint subsets connected by the new edge. Therefore, we can interpret Kruskal's algorithm as a contraction algorithm that selects an edge to contract by their randomly assigned score. The contraction algorithm ends when we have two disjoint subsets. Kruskal's algorithm takes it one step further and connects the final two disjoint subsets to form a minimum spanning tree. Thus, without the final edge, Kruskal's algorithm is identical to the contraction algorithm for identifying the minimum cut, which finds the two sides of a minimum cut with probability $\Omega\left(\frac{1}{n^2}\right)$.

(b) It suffices to randomly assign distinct weights in $[m]$ to the edges, which takes $O(m)$ time. We note that Kruskal's algorithm runs in time $O(m \log n)$, and hence we have a combined $O(m \log n)$ implementation of the contraction algorithm.

Problem 6. **(a)** Consider the following graph $G = (V, E)$: there are vertices s and t, and there are k independent and similar components. The component i is as follows: there are nodes A_i and B_i, and edges $\{(s, A_i); (s, B_i); (A_i, B_i), (B_i, t)\}$. Clearly the size of the min cut is equal to k and the cut is: $T = \{t\}$ and $S = E \backslash T$ (i.e., all vertices except t). There is exactly one s-t cut. Therefore, let's call the edges $\{(B_i, t) \mid i = 1 \ldots k\}$ the cut edges, and the rest of the edges as non-cut edges.

Let's prove that the algorithm has an exponentially small probability of not contracting a cut edges. Suppose the algorithm performed t steps and did not contract a cut edge yet. Then there are k cut edges and at most $3k$ non-cut edges. Therefore, the probability that the algorithm chooses a non-cut edge is

$\leq \frac{3k}{4k} = 3/4$. Thus, there is at most $(3/4)^{n-2}$ probability that in each of the $n-2$ contractions the algorithm did not choose a cut edge.

Thus, the probability that the algorithm finds a min-cut is at most $(3/4)^{n-2} = e^{-\Omega(n)}$, which is exponentially small.

(b) The construction of the graph is similar: there are k independent similar components. The component i looks like: there is a vertex A_i and edges $\{(s, A_i); (A_i, t)\}$. Clearly any cut has value exactly k; and therefore every cut is a min-cut. There are in total $2^k = 2^{n-2}$ cuts in total (each A_i can be either in S or in T). Thus, there can be an exponential ($2^{\Omega(n)}$) number of min s-t cuts.

Problem 7. We use the same Contraction Algorithm but stop when there are 3 vertices remaining. We then select one of the three cuts in the graph, uniformly at random. Let c be the value of the second smallest cut in the original graph. Suppose that when the graph has r remaining vertices we have not yet contracted any edge of the second smallest cut. Each individual vertex defines a cut with that one vertex on one side. At most one of these vertices can correspond to (a side of) the minimum cut. Thus, every vertex but one has degree at least c. Since there are r vertices this yields a total of $(r-1)c$ edge "ends" which means there are at least $(r-1)c/2$ edges. Thus, a random edge is one of the c second-smallest cut edges with probability at most $2/(r-1)$. It follows, just as in the original analysis, that we do *not* contract any second-smallest cut edge with probability

$$\prod_{i=4}^{n}\left(1-\frac{2}{r-1}\right) = \prod_{i=3}^{n-1}\left(1-\frac{2}{r}\right) = \binom{n-1}{2}^{-1}$$

Thus, when we are down to three vertices, with probability $\Omega(1/n^2)$, we have not contracted any edge of the second smallest cut. Conditioned on this, choosing a cut uniformly at random chooses the second smallest cut with probability $1/3$, and we obtain the second smallest cut with probability $\Omega(1/n^2)$.