6.856 — Randomized Algorithms

Handout #6, March 6, 2011 — Homework 1 Solutions

1a.

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 $B = \sum_{n=1}^{\infty} \left(\frac{1}{2}\right)^n n = \sum_{n=1}^{\infty} \left(\frac{1}{2}\right)^n + \sum_{n=2}^{\infty} \left(\frac{1}{2}\right)^n + \cdots + \sum_{n=i}^{\infty} \left(\frac{1}{2}\right)^n + \cdots = 1 + \frac{1}{2} + \cdots + \frac{1}{2^{i-1}} + \cdots = 2 \in O(1)$.

1b.

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

1c.

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. ⇒⇐. Therefore, finite bits will not suffice in the worst case.

2a.

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

$$\sum_{s=k}^{n-1} \frac{s}{n} + \sum_{s=0}^{k-2} \frac{n-s-1}{n} = \frac{(n+k-1)(n-k)}{2n} + \frac{2(k-1)(n-1)-(k-2)(k-1)}{2n}$$

$$= \frac{n^2 - 2k^2 + 2nk - 3n + 2k}{2n} = n\left(\frac{1}{2} - \left(\frac{k}{n}\right)^2 + \frac{k}{n}\right) - \frac{3}{2} + \frac{k}{n}$$

$$< \left(-\left(\frac{k}{n}\right)^2 + \frac{k}{n} + \frac{1}{2}\right)n$$

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 $\frac{3}{4}n$.

2b.

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, $T(n) = n + E[T(sizeof(S'))] = n + T(E[sizeof(S')]) \leq n + T(\frac{3}{4}n)$. 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 $4n$.

---

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

2c.

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.

3.

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 $\Omega\left(\frac{1}{n^2}\right)$.

4.

We use the same Contraction Algorithm but stop when there are 3 vertices remaining and inspect all three of the possible cuts of that graph. Let c be the value of the second smallest cut. 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 the algorithm succeeds with probability $\Omega(1/n^2)$. Repeating the algorithm $n^2 \log n$ times succeeds with high probability.

Problem 5.

(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 $S = E \setminus T$ (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.