HW 5 (due Wednesday, at noon, October 10, 2018)

CS 473: Algorithms, Fall 2018 Version: **1.0**

Submission guidelines and policies as in homework 1.

---

1 (100 PTS.) Storing fast.

Consider an array $A[1 \ldots n]$ which is initially empty. One can check in constant time whether an entry in array is empty, or store a value in an array. Think about this array as a cache. You are given a stream $x_1, \ldots, x_m$ of elements to store in the array. Let rand(n) be a library function that returns a random number chosen uniformly from $\{1, \ldots, n\}$.

In the ith iteration, the algorithm randomly select a location $r_i \leftarrow rand(n)$, and tries to store x_i in the location $A[r_i]$. If this location already contains a value, the algorithm retries by resampling $r_i \leftarrow rand(n)$, and repeat doing it till success.

1.A. (10 PTS.) For $m = n$ prove an exact bound on the expected number of calls to $rand(n)$.

1.B. (10 PTS.) For $m = n/2$ prove an exact bound on the expected number of calls to $rand(n)$.

1.C. (20 PTS.) Consider the variant that after trying 3 times for each element to store it, the algorithm simply reject it. For $m = n$, provide an upper bound (as tight as possible [constants matter here]) on the expected number of elements being rejected.

1.D. (20 PTS.) We take all the rejected elements from the previous part (for simplicity assume the number of these elements is exactly your upper bound from the previous part), and the algorithm tries to store them again, in a new array $B_2[1 \ldots, n]$, which is initially empty, again doing three tries before rejecting an element. Provide an upper bound (as tight as possible) on the expected number of elements being rejected.

1.E. (10 PTS.) Assume we do the above for i layers. Let R_{i-1} be the number of elements that got rejected in the first $i-1$ layers. Assume that $R_{i-1} \leq q_i n$, for some constant $q_i < 1/2$. Let Z_i be the number of elements that get rejected in the ith layer. Prove that $\mathbb{E}[Z_i] \leq q_i^3 n$.

1.F. (30 PTS.) Prove an upper bound (as small as possible), using the above, that holds with high probability, on the number of rounds one need to have till all elements are stored somewhere.

Note, that the assumption that the number elements getting rejected is exactly the expected bound, is of course false. The same analysis essentially holds, but the details becomes much hairier.

2 (100 PTS.) Cuts and stuff.

2.A. (10 PTS.) Consider a graph $G = (V, E)$ with n vertices, m edges and a min cut of size k. Let $\mathcal{F}$ be the collection of all min-cuts in G (i.e., all the cuts in $\mathcal{F}$ are of size k). What is the probability that MinCut (the simpler variant – see class notes) would output a specific min-cut $S \in \mathcal{F}$?

(Here, we are looking for a lower bound on the probability, since two minimum cuts of the same size might have different probabilities to be output.)

[And yes, this is easy.]

2.B. (10 PTS.) Let S be a set of numbers. Consider a randomized algorithm, that for any $x \in S$, outputs x with probability at least p. Prove that $|S| \leq 1/p$.

2.C. (10 PTS.) Bound the size of $\mathcal{F}$ using (B).

2.D. (10 PTS.) A ***good cut*** is a cut $(S, \overline{S})$ such that the induced subgraphs G_S and $G_{\overline{S}}$ are connected. Consider a specific good cut $C = (S, \overline{S})$ with kt edges in it, where $t \geq 1$. What is the probability that MinCut would output this cut? (Again, proof a lower bound on this probability.)

2.E. (40 PTS.) Consider the set $\mathcal{F}(t)$ of all good cuts in G of size at most kt. Bound the size of $\mathcal{F}(t)$.

2.F. (20 PTS.) Consider the set $\mathcal{F}'(t)$ of all cuts in G (not necessarily all good) of size at most kt. Bound the size of $\mathcal{F}'(t)$.

3 (100 PTS.) Cut, cut, cut.

3.A. (40 PTS.) Given a connected graph $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ with n vertices, and $m \geq n$ edges, consider the algorithm that assign edges random weights (say in $[0, 1]$), and computes the MST $\mathcal{T}$ of the resulting graph. Let e be the heaviest edge in T and consider the cut induced by the two connected components of $T - e$. Prove that with probability $\geq 2/n(n-1)$, this is a mincut of G.

3.B. (Not for submission - for fun.) Provide an algorithm that in (expected or deterministic) $O(m)$ time returns the heaviest edge in the MST of G. [Without computing the MST, of course.]

3.C. (Harder – not for submission.) Consider the task of computing any spanning tree $\mathcal{T}_1$ of G, then computing any spanning (forest) $\mathcal{T}_2$ of $\mathsf{G} - \mathsf{E}(\mathcal{T})$, and continuing in this fashion, where $\mathcal{T}_i$ is a spanning forest of the graph remaining from G after removing the edges of $\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}$. Prove, that one can compute a sequence of such spanning forests $\mathcal{T}_1, \mathcal{T}_2, \ldots$ in $O(m \operatorname{polylog} n)$ time.

3.D. (60 PTS.) Assume you are given that the mincut in G is of size k. Present an algorithm, with running time $O(m \operatorname{polylog} n)$, that computes a graph $\mathsf{H} \subseteq \mathsf{G}$, such that:

(i) The mincut of H is a mincut of G.

(ii) $|\mathsf{E}(\mathsf{H})| = O(nk)$.