CS 270 Combinatorial Algorithms & Data Structures — Spring 2023

Problem Set 2

Due: 11:59pm, Wednesday, February 15th
Solution maximum page limit: 6 pages

See homework policy at `https://cs270.org/spring23/syllabus/#homework-policies`

Problem 1: In class it was shown that runtime plus potential difference per query to splay an item x is $O(\log(W/s(x))+1)$, where each item x has some positive weight $w(x)$ (which we can choose in our analysis), and $W := \sum_{x \in D} w(x)$. Here D is the database of all items. To prove static optimality, we showed that if some static tree T places item x on level ℓ_x (where $\ell = 0$ is the root), then we can set $w(x) := 3^{-\ell_x}$. Then $W = O(1)$ and $s(x) = O(3^{-\ell_x})$, so that $\log(W/s(x)) + 1 = O(\ell_x + 1)$. One then shows static optimality by letting T be the optimal static tree.

What though about the potentials? Recall that we used a potential function argument, so that our actual total runtime is of the form $t + \Phi_{\text{initial}} - \Phi_{\text{final}}$, where t is the actual total runtime of answering all queries, Φ_{final} is the final potential after answering all queries, and Φ_{initial} is the initial potential at the beginning. Thus, we showed in class that the total runtime to service a query sequence $\sigma = (x_1, x_2, \ldots, x_m)$ is bounded by $O(C_T(\sigma)) + \Phi_{\text{initial}} - \Phi_{\text{final}}$, where $C_T(\sigma)$ is the cost of servicing σ with T.

(a) (3 points) Show that for our weight function defined above, the potential Φ always lies in a range of size $O(n^2)$. That is, there exist some L, R, which are functions of n, such that (1) $\Phi \in [L, R]$ at all points in the operation sequence, (2) $R - L = O(n^2)$. Conclude that the total runtime for a splay tree to service σ starting from any initial tree shape is $O(C_T(\sigma) + n^2)$.

(b) (7 points) Show that in fact the total runtime is $O(C_T(\sigma) + n \log n)$. **Hint:** you may find it helpful to modify the weight function.

(c) (3 points) Deduce that if each $x \in \{1, \ldots, n\}$ appears in σ at least once (i.e., there is at least one i such that $x_i = x$ for each $x \in \{1, \ldots, n\}$), then the total runtime is $O(C_T(\sigma))$.

Problem 2: (15 points) In lecture we showed that the total runtime of m operations on a link-cut tree using splay trees to implement auxiliary trees is $O((\log n) \cdot (m + \text{PCC}))$, where PCC is the total number of preferred child changes over those m operations. Show that in fact the stronger bound $O(m \log n + \text{PCC})$ holds. Given that we showed $\text{PCC} = O(m \log n)$ in class, this implies an $O(\log n)$ amortized bound per operation. For this problem, we only require you to consider the case when all m operations are either link, cut, or access. **Hint:** Remember our main tool for analyzing splay trees: Claim 3.5 from Lecture 4. Choose the weight function $w(\cdot)$ appropriately.

Problem 3: (15 points) Give an $O(m(m + n \log n) \log U)$ time scaling algorithm for min cost max flow in graphs with capacities in $\{1, \ldots, U\}$. **Hint:** Initially all edges have capacity 0 and $p = 0$ is feasible. As capacity bits are shifted in, p must be fixed to stay feasible. Consider fully saturating all negative cost edges with flow. The result of course is not a feasible flow (it doesn't satisfy conservation of flow), but now some vertices have more flow coming in than out (excess) and some have the opposite (deficit). Fix this using a min-cost max-flow in the residual graph. What can you say about the edge costs in the residual graph?

Problem 4: (10 points) In Fibonacci heaps, when a node x loses 2 children, the subtree rooted at x is cut from x's parent and becomes a new tree in our top level forest. Suppose that instead we cut x's subtree away from its parent only after x loses k children.

(a) (5 points) Show that the amortized cost of decrease key is reduced as k increases. How does it decrease as a function of k? Note decrease key already has amortized cost $O(1)$ when $k = 2$, so the point here is just that the constant inside the big-Oh improves. **Hint:** modify the potential function from class.

(b) (5 points) Which operation(s) increase in amortized cost due to this change? Give a new bound as a function of k.

Problem 5: (1 point) How much time did you spend on this problem set? If you can remember the breakdown, please report this per problem. (sum of time spent solving problem and typing up your solution)