Massachusetts Institute of Technology
6.854/18.415: Advanced Algorithms
David Karger

Handout 16
Nov 11, 1999

Problem Set 7 Solutions

Problem 1. Let n be the number of elements in the tree (say) T.

(a) Given a node x, we can perform an in-order traversal of its subtree T_x and store the (sorted) keys belonging to T_x in an auxiliary array. A 1/2-balanced tree can be constructed from the array by repeatedly computing the middle element. We refer to this operation as BALANCE.

(b) The search begins at the root node r which has $size[r] = n$. At each step, a traversal from (say) x to y is such that $size[y] \leq \alpha size[x]$. Therefore the size function drops from n to 1 in after k traversals where,

$$\alpha^k n \leq 1 < \alpha^{k-1} n$$

i.e.,

$$k = \left\lceil \frac{\log n}{\log(1/\alpha)} \right\rceil = O(\log n)$$

since α is a constant.

(c) For each node x, we define its potential $\varphi(x)$ as β times the difference of $size[left[x]]$ and $size[right[x]]$. The potential of the data structure

$$\varphi = \sum_{x \in T} \varphi(x).$$

Any α-balanced node x has $\varphi(x) \leq \beta(2\alpha - 1)|T_x|$. A BALANCE operation done on node x if $\varphi(x) > \beta(2\alpha - 1)|T_x|$. After the operation, the node x is 1/2-balanced and has its potential $\varphi(x) \leq \beta$. Therefore potential $\varphi(x)$ decreases by at least $\beta(2\alpha - 1)|T_x|$. The actual cost of BALANCE(x) is $O(|T_x|)$. By suitably adjusting β, we can make the amortized cost of BALANCE at most 0.

From (b), we know that the height of the tree is $O(\log n)$. So an INSERT operation has $O(\log n)$ actual cost. The potential increases by at most 1 in each traversed node. Since the tree has height $O(\log n)$, we can bound the potential increase by $O(\log n)$. Thus the amortized cost is also $O(\log n)$.

A DELETE can be performed by replacing the deleted node by its leftmost descendant in the right subtree. Again the actual cost is asymptotically bounded by the height which is $O(\log n)$. The potential increases in each traversed node by at most 1. By reasoning similar to the analysis of INSERT, we can bound the amortized cost of DELETE by $O(\log n)$.

Comments from graders: Parts (a) and (b) were trivial. Many alternative correct potential functions were used for this problem, such as sum of sizes of nodes in the tree. Some potential functions gave $\Omega(\log n)$ bounds for insert and delete.

Problem 2. Let G be the graph under consideration.

(a) Yes, every min-cost flow problem has an upward critical arc. Any edge belonging to a min-cut is an upward critical arc since it has to be saturated.

(b) We compute the min-cost max-flow using any efficient algorithm.We then compute the reduced costs for edges. The prices can be found using $O(mn)$-time Bellman-Ford shortest paths algorithm. Computing reduced costs from prices is an $O(m)$ time operation.

Given the reduced cost $c_p(e)$ for edge $e = (v, w)$ in residual graph G_F, we know the following:

$c_p(e) > 0$: There is no flow even after small changes in cost. So edge e is neither upward critical nor downward critical.

$c_p(e) < 0$: This case does not occur in G_F.

$c_p(e) = 0$: Let $f(e)$ be the flow on e. Using excesses, deficits and max-flows on 0 reduced cost edges, we can find out if there exists a min-cost max-flow with no flow on e, and one with saturated e. The former implies that e is not upward critical and the latter implies that e is not downward critical.

Lemma 1 *The above algorithm is correct.*

Proof. If an arc is computed to be "not upward critical" by the above algorithm, the algorithm also shows another flow with same value and cost. Therefore arcs that are not upward critical are correctly recognized by the algorithm. We will now prove that every arc that is reported by the algorithm as upward critical is in fact upward critical. The proof is by contradiction. If an arc is not upward critical, there is a flow F' with same cost. Flow F' has 0 flow through e. Flow $F - F'$ gives a circulation that has $f(e)$ through e. Moreover this circulation involves only 0 reduced cost edges. The max-flow with excess and deficit of $f(e)$ would have identified this circulation. So it is not possible that the algorithm reported e as upward critical. We can argue similarly for downward critical arcs. ∎

Comments from graders: There were many circuitous solutions to part (a). Some faulty solutions to part (b) computed single augmenting cycles containing e.

Problem 3. We use maximum augmenting paths to solve this problem.

(a) A maximum augmenting path algorithm achieves $O(m \log n)$ time bound (using a Dijkstra-like algorithm) and computes a flow with value at least $1/m$ times the maximum flow. The computed flow is feasible and is therefore smaller in value than the max-flow.

(b) Consider the edges incident on s. If edge capacities are rounded down to the nearest integer, the decrease in flow value is at most n, since edges incident on s cannot ship any more flow.

(c) We first get the maximum augmenting path value in G, say f. Initially we set parameter M' to mf. We know that M' is at least the max-flow of G and at most m times the max-flow of G. So the capacity of each edge e can be set to the value $\min\{c(e), M'\}$ without changing the flow value.

From (b), the flow value reduces by $n\delta$ if we set reduce the capacity of each edge to $\lceil c(e)/\delta \rceil \cdot \delta$. This reduction results in a $(1-\epsilon)$-approximation if $\delta \leq \epsilon M/n$. We do not have the value of M though. So we estimate M as M' which is in the range $[M, mM]$. So we have a lower bound $\epsilon M'/n$ for δ.

We now use the $O(mn \log U)$ algorithm to compute max-flow on the graph with costs $\lceil c(e)/\delta \rceil$. The algorithm is dependent on the value of U. We can upper bound the value of U however.

$$U \leq \frac{M'}{\delta} = \frac{M'}{\epsilon M'/n} \leq \frac{n}{\epsilon}$$

The resulting flow is multiplied by δ to get a $(1-\epsilon)$-approximation of the flow value. Thus we have a $O(mn \log n/\epsilon)$-time algorithm.

Comments from graders: Part (a) and (b) were easier than part (c). Solutions that did not use the $O(mn \log U)$-time scaling algorithm were usually faulty. Reducing the cost of edges to $\min\{c(e), M'\}$ is a crucial step that some solutions skipped.

Problem 4.

(a) All that changes between problems is that some capacities increase. Thus f_i can't possibly violate capacity constraints in p_{i+1}, and we do nothing that would disrupt conservation or antisymmetry constraints.

(b) Recall that for a distance labeling to be valid, we must have $d(v) \leq d(w) + 1$ for all residual arcs (v, w). When we switch to p_{i+1} we introduce new residual arcs from the source, and saturate the ones going to nodes with distance labels less than the source. Thus we get new residual arcs from nodes with distance labels less than the source to the source, and from the source to nodes with distance labels at least as big as the source. Thus all new arcs are uphill and don't violate the validity of the distance labeling.

(c) Distance labels are bounded by $2n$ in each problem, so they are bounded by $2n$ overall. Further, they never decrease, so there can be at most $O(n^2)$ relabelings overall.

It is still the case that for two saturating pushes to happen on the same arc both ends must be relabeled, so there can be at most $O(nm)$ saturating pushes. There are also the extra saturating pushes from the source, but only $O(np) = O(n^2)$ of them.

Recall that we bounded the number of non-saturating pushes for FIFO by looking at the potential function $\phi = \max_{\text{active}v} d(v)$. We call one pass through the queue a phase. A node is removed from the queue after a saturating push, so there can be at most $O(n)$ saturating pushes per phase. There are two cases for a phase. First, a relabeling occurs. Second, no relabeling occurs. In this case, all excess moves down one, so ϕ decreases by 1. The first case can be bounded by the number of relabelings, that is, $O(n^2)$. We bound the second by the total increase in the potential. Inside a problem, only a relabel can increase the maximum distance label, and we know that the total increase due to all relabelings is $O(n^2)$. The start of a problem can also increase the potential by $O(n)$. So the total number of phases is $O(n^2+np) = O(n^2)$, and the total number of non-saturating pushes is $O(n^3)$.

Comments from graders: Some solutions to part (b) did not use the correct definition of valid distance labelling. Many arguments to (c) had minor errors.

Problem 5. This problem is based on flows. We are given currencies $1,\ldots,n$, where currency 1 is dollar and currency n is yen. Let $c_1,\ldots c_m$ be the clients. We can set up a graph with each currency as a vertex and client i as an edge from vertex a_i to b_i with capacity u_i. Notice that u_i is the limit on amount of a_i converted. Rate r_i determines the multiplicative factor in the conversion. Because of the rates we cannot express this problem in terms of standard flow problems.

(a) Let X_i denote the amount of currency a_i traded by client i.

$$\mathrm{M}ax \sum_{i|b_i=k} r_i X_i$$

$$\begin{aligned} X_i &\le u_i \\ \sum_{i|a_i=j} X_i - \sum_{i|b_i=j} r_i X_i &\le 0 \\ \sum_{i|a_i=1} X_i &\le D \\ X_i &\ge 0 \end{aligned}$$

for all $1 \le i \le n$ and $1 \le j \le m$.

The constraint on trading D dollars is possible since recycling dollars will not result in more currency.

(b) We can consider the solution to the LP as a flow on a graph with currencies as nodes and clients as edges. The flow conservation rules are not the same due to the multiplicative factor.

We can decompose the flow into paths. Given any path in the graph starting from s, we compute the currency that is effectively traded by the path. This computation involves finding the minimum of $f(e_i)/\prod_{j<i} r(e_j)$. We will end up with a path and cycle decomposition. The paths can be executed one after another till all operations are performed. Cycles do not earn money due to the "no profit by arbitrage" condition. So we can ignore them. Thus no borrowing is necessary.

(c) Paths that do not end in yens can be ignored. Thus we can construct a flow based on the paths we extracted. This simplified flow will have the same value and will end the day with only dollars and yen.

Comments from graders: Some solutions applied constraints on the LP in part (a) that can be justified only in (b) and (c). Of these, some solutions skipped (b) and (c) by claiming that the formulation in part (a) avoided excesses and cycles. A few solutions had very large LP formulations.