CSOR W4231.002 – Spring, 2016

Homework 5 solutions

1. (a) We can prove this using the accounting method introduced in part 17.2 in textbook. Here we charge the operations as follows:

   - Setting a bit from 0 to 1: \$3, while the actual cost is \$1
   - Resetting a bit from 1 to 0: \$0, while the actual cost is \$1

   So, whenever a bit is set to 1, the bit 'carries' a credit of \$2 (i.e. 3 -1). Out of this a credit of \$1 can be used for resetting the bit to 0 (during the INCREMENT operation). Each bit that has been set or reset, still carries a credit of \$1, which can be used in the RESET operation. Thus the remaining credit after a sequence of n INCREMENT or RESET operations is always bigger or equal to 0. So we can conclude that starting from an initially zero counter, any sequence of n INCREMENT and RESET operations takes time $O(n)$, that is, the amortized time per operation in $O(1)$.

   (b) We can create a tree of height $\Omega(\log n)$ by $n-1$ UNIONs: first UNION($x1, x2$), UNION($x3, x4$), ..., UNION(x_{n-1}, x_n) of size 2; then create sets of size 4 by pairwise UNION of these, etc. This gives a binomial tree, which has $\binom{k}{i}$ of its $2k$ nodes at depth i, for $i = 0, \ldots, k$. Hence at least half of the n nodes are at depth greater than $(\log n)/2$, so each FIND on these nodes takes $\Omega(\log n)$ time. Letting $m \geq 3n$, we can have more than $m/3$ FINDs, so the total cost is $\Omega(m \log n)$.

2. This problems is similar to the minimum spanning tree. Instead of wanting to minimize the value of each edge, we want to maximize the value (bottleneck rate).

   We start with a root node s and try to greedily grow a tree from s outward. At each step, we simply add the node that can be attached with the highest bottleneck rate to the partial tree we already have. In other words, we maintain a set S (which is a subset) of V on which a spanning tree has been constructed so far. Initially, $S = s$. In each iteration, we grow S by one node, adding the node v that maximizes the achievable bottleneck rate, and including the edge $e = (u, v)$ that achieves this maximum bottleneck rate in the spanning tree.

   We want to show, that in each iteration, it only adds edges belonging to the tree with the best bottleneck rate. In each iteration of the algorithm, there is a set S (subset) of V on which a partial spanning tree has been constructed, and a node v and edge e are added that maximize the bottleneck rate. By definition, e is the edge that has the best maximum bottleneck rate with one end in S and the other end in $V - S$, and so by the Cut property (provided in the lecture notes) it is in every minimum spanning tree.

3. (a) Run the network flow algorithm to compute the max-flow. The critical edge has to be saturated; otherwise a small capacity reduction will not affect the flow. However, saturated edges may be not critical. Assuming e is saturated in the original graph, if there is a cycle that passes through the reverse e in the residual graph, we can push a flow through the cycle to reduce the flow in e. The flow conservation constraints will still be satisfied.

   The algorithm is: For each saturated edge (u, v) in the original network, run the depth first search algorithm to check whether there is a path from u to v in the residual network. If there is a path, (u, v) is not critical, and otherwise it is.

   (b) Algorithms:

   i. Let E_0 be the edges $e \in E$ for which $f(e) > 0$, and let $G_0 = (V, E_0)$. Find in G_0 a path P_1 from s to u and a path P_2 from v to t.

   ii. *Special case: If P_1 and P_2 have some edge e in common, then $P_1 \cup (u, v) \cup P_2$ has a directed cycle containing (u, v). In this case, the flow along this cycle can be reduced by one unit without changing the size of the overall flow. Return the resulting flow.*

   iii. Reduce the flow by one unit along $P_1 \cup (u, v) \cup P_2$.

   iv. Run Ford-Fulkerson with this starting flow.

Justification and running time:

Say the original flow has value F. Let's ignore the special case (2). After step (3) of the algorithm, we have a legal flow that satisfies the new capacity constraint and has value $F - 1$. Step (4), Ford-Fulkerson, then gives us the optimal flow under the new capacity constraint. However, we know the max flow has value at most F, and thus Ford-Fulkerson runs for just one iteration. Since each of the steps is linear, the total running time is linear, thus $O(m+n)$.

4. Flow Network With Supplies

i. Suppose there is a feasible circulation f. By definition

$$\begin{aligned}\sum_{v\in V} s_v &= \sum_{v\in V} f^{out}(v) - \sum_{v} f^{in}(v) \\ &= \sum_{v\in V}\sum_{(v,u)\in E} f(v,u) - \sum_{v\in V}\sum_{(u,v)\in E} f(u,v) \\ &= \sum_{v\in V}\sum_{(v,u)\in E} f(v,u) - \sum_{u\in V}\sum_{(v,u)\in E} f(v,u)\end{aligned}$$

In this sum, every edge appears twice: once as an outgoing edge of v and once as an incoming edge of u. Hence

$$\sum_{v\in V} s_v = 0.$$

It follows that

$$\sum_{v\in S} s_v = -\sum_{v\in T} s_v.$$

ii. Based on the original graph $G = (V, E)$ and its capacities, we

a) add a node s^* as the super source and a node t^* as the super sink.

b) and add an edge (s^*, v) with capacity s_v for every $v \in S$ and an edge (v, t^*) with capacity $-s_v$ for every $v \in T$.

Then there is a feasible circulation in G if and only if the max flow from s^* to t^* in G' has value $S = \sum_{v\in S} s_v$.

($\Rightarrow$) If there is a feasible circulation f in G, send flow s_v across (s^*, v) for every $v \in S$ and flow $-s_v$ across (v, t^*) for every $v \in T$, and combined with f they compose a max flow in G'. First, it's easy to show that this flow satisfies all capacity constraints and conservation constraints in G'. Second, this flow has value S and is maximum, since all outgoing edges of s are saturated.

($\Leftarrow$) Conversely, suppose there is a maximum flow of value S in G'. Delete edges out of s^* and edges into t^* and we can obtain a circulation f such that $f^{out}(v) - f^{in}(v) = s_v$ for every $v \in V$. Also, the capacity constraints are satisfied.

5. Conformal Decomposition

   (a) Find a single s-t path (by e.g. DFS).

   (b) Reduce the flow on every edge in the path by the minimal flow along the path, and remove the edges whose values are reduced to 0.

   (c) Add the path as a new element of the decomposition.

   (d) Repeat (a)-(c) until no edge is connected to s.

   (e) Start from a node still connected by some edges, find a cycle (by e.g. DFS).

   (f) Reduce the flow on the every edge in the cycle by the minimal flow along the cycle, and remove the edges whose values are reduced to 0.

   (g) Add the cycle as a new element of the decomposition.

   (h) Repeat (e)-(g) until no edge exists.

   The conservation constraints are always satisfied during our procedure so we can always find a s-t path in (a) until the condition in (c) becomes true or a cycle in (b) until the conditions in (h) becomes true. It's easy to show that the elements we get compose a valid conformal decomposition.

   We need $O(n)$ time to get an elements (a s-t path or a cycle), and we can find at most m elements (note we remove at least 1 edge each time we get an element). The total time is $O(mn)$.

6. Min-Cost Flow

(a) PASS

(b) Given a max flow problem, say a graph $G = (V, E)$ with capacities, a source $s \in V$, and a sink $t \in V$, we

i. assign 0 cost to all edges,

ii. add a new edge (t, s) with infinite capacity and -1 cost,

iii. and assign 0 supply to all nodes.

Then we get a min-cost flow problem. We can get a max flow in G by deleting (t, s) from a solution to this problem. The supply constraints degenerate to conservation constraints because all supplies are 0. The min cost problem minimizes $-f_{ts} = -f^{out}(s)$ (note (t, s) is the only edge with non-zero cost and the only incoming edge of s), and thus gives a max flow in G.

(c) Construct a graph G where we have

i. a node with supply 1 for every person and a node with supply -1 for every job,

ii. and an edge with capacity 1 and cost $-v_{ij}$ for each qualification pair (i, j).

Then we get a min-cost flow problem, which minimized $\sum_{(i,j)\in E} -v_{ij} f_{ij}$. We can always get an integral solution (if a feasible solution exists) to this problem because of the integral capacities. Only 0 or 1 flow values are permitted, since every capacity equals 1. For each person node, there is exactly one outgoing edge assigned flow 1 because its supply is 1. For each job node, there is exactly one incoming edge assigned flow 1 because its supply is -1. Hence the non-zero flows give a valid subset A for our original problem, and $\sum_{(i,j)\in V} v_{ij} f_{ij} = \sum_{(i,j)\in A} v_{ij} \cdot 1$ is maximized.