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

Handout 10
September 28, 1999

# Problem Set 4 Solutions

**Problem 1.** We augment the dynamic connectivity data structure proposed by Holm, Lichtenberg and Thorup to maintain key values on nodes. The dynamic connectivity data structure maintains a spanning forest $F_0$ at level 0 containing all vertices of the graph (say) $G$. We use the augmented ET-tree from problem 3 of problem set 3 to store $F_0$. Find, split and insert operations performed on $F_0$ can be done in the augmented ET-tree with $O(1)$ slowdown. We have not discussed the join operation on splay trees in the solution of the referenced problem. Join can be done by attaching the root of one splay tree to the other (preprocessed to contain no right subtree), and updating the min pointer of the new root. Therefore insert and delete take $O(\log n)$ time in $F_0$. The dynamic connectivity data structure achieves $O(\log^2 n)$-time insert-edge and delete-edge.

Find-min and decrease-key are supported on $F_0$ in $O(\log n)$ time. Since $F_0$ is a spanning forest, the components in $F_0$ are same as those of $G$. So we can perform find-min and decrease-key on $F_0$ achieving $O(\log n)$ time bound on these operations.

**Problem 2.** Again we use the Thorup *et al* data structure to maintain dynamic connectivity on the graph (say) $G$. Forest $F_0$ spans $G$ and therefore has the same components as $G$. We need to support output$(v, w)$ operation on the graph.

(a) Output$(v, w)$ performs a depth-first-search on $v$ to find a path to $w$. Since $F_0$ has at most $n - 1$ edges, output is an $O(n)$-time operation. This is an improvement over $O(m)$ which we get by a depth-first-search on $G$.

(b) Again we output the path from $v$ to $w$ in forest $F_0$. Let $l$ be the length of this path. We assume that $F_0$ is stored as an ET-sequence implemented by a splay tree. Each vertex stores the list of edges incident to it, called the adjacency list. Maintaining the adjacency lists on insert-edge and delete-edge operations takes constant time. So we do not have any asymptotic slowdown in the ET-tree operations.

We are interested in finding a path from $v$ to $w$. First we determine whether $v$ and $w$ are connected. If not, operation output returns that there is no existing path. We iterate on each edge $e$ incident on $v$ and remove it. Then, we check if $v$ and $w$ are connected.

**Lemma 1** *Edge $e$ separates $v$ and $w$ iff $e$ is on the path from $v$ to $w$.*

*Proof.* Since $F_0$ is a tree, every edge on the path separates $v$ and $w$ (proof for the "if" part). An edge that separates $v$ and $w$ exclusively connects components of $v$ and $w$ generated by its deletion. So can be no path that does not include $e$ (proof for the "only-if" part).

From Lemma 1, we can say that an edge that separates $v$ and $w$ belongs to the path from $v$ to $w$. The initial check made in operation output ensures that $v$ and $w$ are connected in the graph. So we will encounter an $e = (v, u)$ satisfying the above condition. We can then output $e +$ output$(u, w)$, recursively computing output$(u, w)$.

Finding an edge on the path involves $d$ delete-edge and find operations. So the cost of generating one edge on the path is $O(d \log n)$. Therefore output runs in $O(ld \log n)$ time where $l$ is the length of the path.

We can improve the runtime to $O(l \log d \log n)$ as follows. Adjacency lists of vertices are maintained in splay trees. Since $O(1)$ updates are made to adjacency lists during insert-edge and delete-edge, there is no asymptotic loss of performance in the data structure.

The adjecency list of a vertex $v$ starts with its active copy's outgoing edge in the ET-tree. The next edge corresponds to the subsequent copy of $v$ in the cyclic ET-sequence, and so on. Since insertion of edges is done next to the active copy, there is only an $O(\log d)$ overhead in maintaining the adjacency list during insertion. Deletion of an edge also takes $O(\log d)$ time. This overhead is asymptotically negligible when compared to the $O(\log n)$ work done by insert-edge and delete-edge operations.

On a path query from $v$ to $w$, we rotate the ET-sequence at the active copy of $v$. Now, the adjacency list of $v$ has the same order of edge appearence as in the ET-sequence. By splitting the ET-sequence at a copy of $v$, we can check whether the active copy of $w$ lies to the left or right of the copy of $v$. Therefore a binary search can be done on $v$'s active copies to compute the edge that encloses the active occurrence of $w$. This edge lies on the path from $v$ to $w$. Thus we reduce the number of splits and finds to $O(\log d)$ per vertex. The time complexity of output is now $O(l \log d \log n)$.

**Problem 3.** We consider the gross flow model.

(a) **False:** Consider vertices $v$ and $w$ having an edge from $v$ to $w$ and another from $w$ to $v$. Given a flow $f$ defined on these edges, we can increment both $f((v,w))$ and $f((w,v))$ by $\Delta$ to get another valid flow. (In the net flow model, note $f(v,w) = -f(w,v)$ so falseness is obvious.)

(b) **True:** Consider any pair $(v,w)$ with both $f(v,w)$ and $f(w,v)$ positive. Assume without loss of generality that $f(v,w) \leq f(w,v)$. Decrease both quantities by $f(v,w)$. One is now zero, but flow conservation and (since the flow values only decreased) capacity bounds have been maintained.

(c) **False:** Consider the graph in Figure 1. The max-flow value is 1. But this can be achieved by sending flow along $(s,v,t)$ or $(s,v,w,t)$.

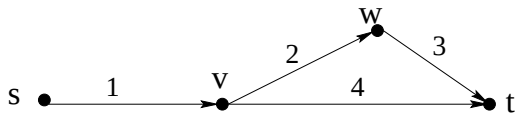


**Figure 1**: Counter-example for problem 3(c)

(d) **False:** A simple counter-example is given in Figure 2. If $(s,t)$ is directed the maximum flow value is 0. Otherwise, the max-flow value is 1.

**Problem 4.** Let $G$ be the directed graph of interest. A node $v$ with incoming edges $E_I(v)$, outgoing edges $E_O(v)$ and node capacity $w(i)$ can be converted to a regular edge-capacity graph as

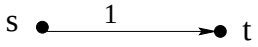


**Figure 2**: Counter-example for problem 3(d)

shown in Figure 3. Let $G'$ be the graph obtained by converting $G$.

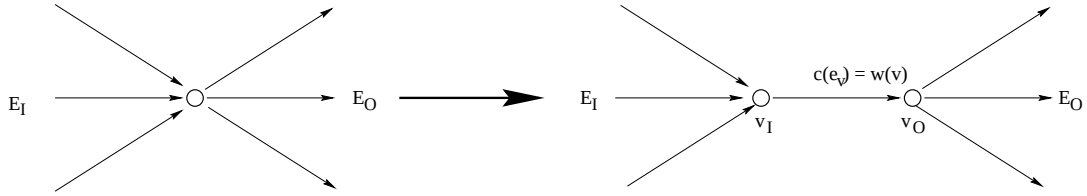


**Figure 3**: Converting node capacities to edge capacities

**Lemma 2** *A valid flow in $G$ corresponds to a valid flow in $G'$ with the same flow value and vice-versa.*

*Proof.* Let $f'(e)$ be the flow along edge $e$ in $G'$. Let $f(v)$ denote the flow into vertex $v$ in $G$. We can transform a valid flow on $G$ to one on $G'$ by satisfying $f(e) = f'(e)$ and $f'(e_v) = f(v)$ for all vertices and edges. It is evident that the flow values are the same.

Therefore a maximum flow on $G'$ yields a maximum flow on $G$. We can compute the flow in each edge using the equivalence used in the proof of Lemma 2.

The transformation takes time linear in the number of edges if we use an adjacency list representation of graph $G$. Graph $G'$ has $2n-2$ nodes and $m+n-2$ edges. We only consider the case when $m \geq n-1$. Otherwise, we can consider only the component containing $s$ and $t$ to perform the max-flow computation. An $O(m)$-time depth-first-search will give the component and we will be left with a graph having $m \geq n-1$. Therefore the number of vertices and edges in $G'$ are $O(n)$ and $O(m)$ respectively. There is no change in asymptotic performance of the max-flow algorithm.

**Problem 5.** Family $i$ can be represented as a source $s_i$ of members, with node capacity $a(i)$. Table $j$ can be represented as a sink $t_j$ of members, with node capacity $b(j)$. Since no two members of the same family can sit on the same table, we can ship at most one member from $s_i$ to $t_j$ for all $i, j$. Thus we have a fully bipartite directed graph with sources $s_i$ connected to sinks $t_j$ with unit capacity edges. We convert this to a standard max-flow problem by transforming node capacities to edge capacities (refer the solution to problem 4), transforming multiple sources and sinks to a single source and sink, and compressing paths to single edges. The graph (say) $G$ obtained after the conversion is shown in Figure 4.

**Lemma 3** *A valid flow with integral flow values on edges of graph $G$ corresponds to a valid seating arrangement and vice-versa. The flow value corresponds to the number of people successfully seated in tables.*

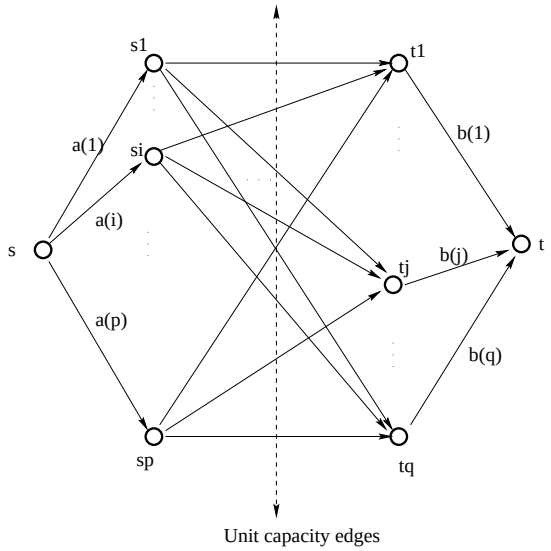


**Figure 4**: The max-flow equivalent of seating arrangement problem

*Proof.* We establish the following equivalence between flows and seating arrangements. Let $f(e)$ be the flow along edge $e$ in $G$. Flow value $f((s_i, t_j))$ is either 0 or 1. If it is 1, it corresponds to a member in family $i$ seated in table $j$. Otherwise, there is no member of family $i$ seated in table $j$. The lemma follows from the constraints we have established in the graph.

From Lemma 3, the seating arrangement problem is equivalent to the max-flow problem on $G$. We can use any max-flow algorithm that gives integral flows for problems with integral edge capacities. The objective of seating all family members is attained if the flow value is $\sum_{1 \leq i \leq p} a(i)$, i.e., all edges from $s$ are saturated. Lemma 3 guarantees that max-flow on $G$ gives the maximum number of people that can be seated successfully.

**Problem 6.** A student can attend lecture $i$ after attending lecture $j$ iff

$$b_i + r_{ij} \leq a_j \tag{1}$$

Thus we can generate a graph $G$ with edges from $i$ to $j$ if (1) is satisfied. A source of students can supply students to each of these lectures. A student can go from any lecture to a sink (i.e. a coffee shop). Now we impose the constraint that every lecture is attended, i.e., every lecture node has incoming flow of at least 1.

**Lemma 4** *A valid flow with integral flow values on edges of graph G is a valid covering of lectures and vice-versa. The flow value is equal to the number of students involved.*

*Proof.* Again we use present a one-to-one relationship from flows in the graph to lecture coverings. The flow can be decomposed to paths of unit flow. Each path is equivalent to a student covering

lectures involved in that path. The constraints imposed prove the lemma.

Now, we need to compute the *minimum flow* in $G$. We will do a series of simplifications to finally compute the flow. The lemmas involved in the process of simplification are presented later.

- Using the technique shown in problem 4, we can convert graph $G$ to an equivalent graph with lower bound edge capacities. Let $v_i$ and $w_i$ correspond to the "in" and "out" vertices for lecture $i$.
- Sending one student per lecture gives us a feasible lecture covering. We construct the equivalent flow $F$ in $G$, which sends one unit along the path $(s, v_i, w_i, t)$ for each $1 \leq i \leq p$. The residual graph $G_F$ is computed.
- Flow $F$ has flow 1 from each $v_i$ to $w_i$. So the residual graph has $w_i$ to $v_i$ edges with flow 1. We remove these backward flow edges, thereby ensuring that at least one unit is sent from each $v_i$ to $w_i$. Let $G'_F$ be the resulting graph.
- From Lemmas 5 and 6, the minimum flow value in $G$ can be found by computing the max-flow from sink $t$ to source $s$. We perform max-flow on $G'_F$ from $t$ to $s$. The resulting flow added with $F$ gives us the min-flow from $s$ to $t$ satisfying lower bounds.

Thus we can solve this problem by computing a max-flow on a graph with $O(n)$ and $O(m)$ nodes and upper bound capacities on edges, where $m$ is $n$ plus the number of $(i, j)$ pairs satisfying (1).

**Lemma 5** *Graph $G$ is a directed acyclic graph.*

*Proof.* A cycle in the graph can happen only if a lecture can be attended twice, which is absurd. Equation (1) shows that the starting time of lectures for nodes in any walk in $G$ are increasing provided $b_i > a_i$ for each $i$.

**Lemma 6** *For a directed acyclic graph $G = (V, E)$ with lower bound edge capacities $b(\cdot)$, the minimum flow from $s$ to $t$ is the negated maximum flow from $t$ to $s$.*

*Proof.* Since $G$ is a DAG, any cut isolating $s$ and $t$ will be such that all edges go from $s$'s cut to $t$'s cut. The max-cut of $G$ has flow in each edge as its lower bound capacity. The min-flow value is therefore

$$F_{min} = \max_{S} \sum_{e \in (S, V-S)} b(e)$$

While computing the maximum flow from $t$ to $s$, the minimum cut will be such that the flow in each edge $e$ is $-b(e)$.

$$\begin{aligned} F_{max} &= \min_{T} \sum_{e \in (V-T, T)} -b(e) \\ &= -\max_{T} \sum_{e \in (V-T, T)} b(e) \\ &= -F_{min} \end{aligned}$$

The result follows.

**Problem 7.** We can reduce the problem to one of computing max-flows as follows. Source $s_i$ denotes the row-sum of row $i$. Similarly sink $t_j$ denotes the column-sum of column $j$. Sources and sinks have node (upper bound) capacities as their value. Nodes $v_{ij}$ represent the matrix elements. We connect infinite capacity edges from $s_i$ to $v_{ij}$ and from $v_{ij}$ to $t_j$ for all $i, j$.

Matrix elements disclosed in $Y$ are processed as follows. The row and column sums are reduced by the disclosed value. Then the nodes corresponding to those entries are removed. This reduction is equivalent to forcing exactly $d_{ij}$ into $v_{ij}$ where $d_{ij}$ is a disclosed entry. We refer to the reduced graph as $G$.

A valid assignment of matrix elements corresponds to a flow in $G$ saturating the node capacities of sources and sinks. By performing max-flow on $G$, we can find such an assignment if it exists. Let $F$ be the max-flow computed.

**Lemma 7** *Element $d_{ij}$ is deducible iff there is no cycle from $v_{ij}$ with positive non-zero edge capacities in $G_F$.*

*Proof.* We refer to directed cycles with positive non-zero edge capacities as perturbations. It is evident that a perturbation incident on $v_{ij}$ can augment $G_F$ to get another solution to the system having a different value for $d_{ij}$. This proves the "only-if" part.

We will now prove the "if" part of the lemma. Assume that $d_{ij}$ is not deducible. Then there are two maximum flows $F$ and $F'$ satisfying $G$. Consider difference $d(e) = f(e) - f'(e)$ on each edge. The value $d(e)$ maintains the conservation and skew symmetric properties of flows. The flow value of $d(e)$ is 0. Now, we can decompose the flow defined by $d(e)$ to cycles. Since $d(e)$ is non-zero for edge $e$ incident on $v_{ij}$, there exists a cycle in the decomposition of the flow defined by $d(e)$. This cycle is a perturbation incident on $v_{ij}$, when flow $F$ is defined on the graph. Thus the "if" part is also proved.

Computing perturbations on $G_F$ takes $O(m) = O(pq)$ time. From Lemma 7, we can compute deducible elements in $O(p^2q^2) + M(O(pq), O(pq))$, where $M(n, m)$ is the time taken to compute max-flow on a graph with $n$ nodes and $m$ edges.