Problem Set 5 Solutions

Problem 1. Given the graph G, we compute the max-flow F. Let G_F be the residual graph. Source s and sink t are disconnected in G_F.

(a) Increasing the capacity of arc $e = (v, w)$ increases the flow value iff there is an augmenting path in G_F after adding a small capacity in e. There exists in augmenting path iff v belongs to the component of s and w belongs to the component of t in G_F.

(b) Decreasing the capacity of arc $e = (v, w)$ by a small value decreases the flow value iff flow cannot be routed from v to w in $G_F - e$. There exists such a path iff v and w are disconnected in G_F.

Problem 2. Let G be the graph under consideration.

(a) We assume that the array creation takes constant time. There are n insert, m decrease-key and n delete-min operations. The insert and decrease-key operations take $O(1)$ time. Delete-min takes $O(1 + d)$ time, where d is the number of empty buckets skipped during the delete-min operation. The bucket number of the last element deleted is D. Thus the total cost of delete-min is $O(m + D)$.

(b) Given the shortest path P from s to v of length d_v, the path $P + vw$ is known to have length $d_v + l_{vw}$. Therefore

$$d_w \leq d_v + l_{vw} \tag{1}$$

and the reduced edge length l_{vw} is non-negative.

(c) For any path $P = (sv_2, v_2v_3, \ldots, v_{k-1}v_k)$, the total reduced edge length is

$$\begin{aligned} l^d_{sv_2} + \ldots + l^d_{v_{k-1}v_k} &= (l_{sv_2} + d_s - d_{v_2}) + \ldots + (l_{v_{k-1}v_k} + d_{v_{k-1}} - d_{v_k}) \\ &= d_s + (l_{sv_2} + \ldots + l_{v_{k-1}v_k}) - d_{v_k} \end{aligned}$$

Therefore all paths to a vertex v have reduced length as the length minus (constant) d_{v_k}. So the shortest path to v is the same and has length $d_v - d_v = 0$.

(d) The scaling algorithm works as follows. We initially start with edge lengths 0, and distance function $d^1(v) = 0$ for all v. In step k, we shift a bit of the length in each edge, and compute distances d_0^{k+1} with reduced edge lengths based on d^k. We use distance function $d^{k+1} = d^k + d_0^{k+1}$ for the reduced costs in the next step. After $\lceil \log C \rceil$ steps the exact distance will be computed. We will now prove the correctness of this algorithm and analyze its running time.

Consider graph $G' = (V, E)$ constructed from the original graph $G = (V, E)$ with edge length $l'_{vw} = \lfloor l_{vw}/2 \rfloor$. In a scaling step, the distances in G' are used to compute reduced edge length in G. Notice that part (b) and (c) of this problem work for any distance function satisfying (1). So if we define the distance function as distance in G', we still have the same shortest paths in G and G'. This proves the correctness of the algorithm. The length of shortest paths is 0 in the reduced graph G'. This means that in the original graph G the length of the shortest path is at most n. So Dial's algorithm takes $O(m+n) = O(m)$ time. The total time complexity for $\lceil \log C \rceil$ steps is therefore $O(m \log C)$.

(e) If a base b representation is used, there are $\lceil \log_b C \rceil$ scaling steps. The maximum distance D in each shortest path computation is bounded tightly by $n(b-1)$. Thus the time complexity of our scaling algorithm is $O((m + n(b-1)) \cdot \log_b C)$. If we set $b = 2 + m/n$ we achieve $O(m \log_{2+m/n} C)$ running time.

Problem 3. Let G be the graph under consideration.

(a) After $p + \sqrt{m} + 1$ blocking flows, the source and sink at least $p + \sqrt{m} + 1$ nodes apart. In other words, any path from the source to sink will have at least $p + \sqrt{m}$ other nodes. Even if all the p special nodes occur in the path, there will be at least $\sqrt{m}$ unit capacity edges. So the number of unit capacity edges saturated in the following $\sqrt{m}$ blocking flows will be at least $\sqrt{m} \cdot \sqrt{m} = m$. Thus we have bounded the number of blocking flows needed by $2\sqrt{m} + p = O(\sqrt{m} + p)$.

(b) Using dynamic trees, we can perform one blocking flow in $O(m \log n)$ time. The total time for solving max-flow is therefore $O(m^{3/2} \log n + mp \log n)$.

(c) There are only two unbounded capacity edge in any s-t path. After performing $\sqrt{m} + 2$ blocking flows, we have at least $\sqrt{m}$ unit capacity edges in each path. So we can bound the number of blocking flows by $2\sqrt{m} + 2 = O(\sqrt{m})$.

A blocking flow can be done using the advance-retreat method mentioned in class. Retreat is not done on any node incident on sink t however. Similarly the flow value of a path is set to the minimum of the capacities involved in the path. Both these contribute to $O(1)$ overhead in computing a path. Thus we have a $O(m)$-time blocking flow algorithm. The total time complexity is $O(m^{3/2})$.

Problem 4. During the push-relabel algorithm the distance labels increase monotonically. Moreover pushes cannot be done from node v to w if $d(v) > d(w) + 1$. So if there are no nodes with distance label l, no push or relabel done on nodes with greater distance label will cause flow to reach nodes with smaller distance label (including t).

Now consider applying discharge operations to the vertices with distance labels greater than l before applying discharge operations to any of the other vertices. No flow can reach the sink. So all excesses are sent back to the source. This suggests that we check for creation of such gaps in the distance labels, and set aside any nodes that end up on the source side of a gap. When all excess has either been processed or set aside, we can convert the preflow to flow efficiently as given in the solution to the next problem.

Problem 5. Consider a decomposition of the preflow. It consists of paths from s to excesses, paths from s to t, and cycles. Only the paths from s to t matter for the value, so we would like to remove the paths from s to excesses without removing the flow into the sink. To do this consider only the arcs already carrying flow. Start at some excess and search backwards on the flow carrying arcs until we reach the source or find a flow cycle. If we find a path to the source, push as much flow as we can on it (empty some arc or empty the node). If we find a flow cycle, push flow back around it to empty some arc. Repeat at until all of the excess is gone. There are several things to notice about this algorithm:

- We only remove flow from arcs that were carrying it, so there is no notion of creating a reverse arc, and once we empty an arc we never have to look at it again.
- Since there are no deficits except at the source, we can't ever "get stuck" searching back along flow paths, so we only have to pay for time spent advancing in our search and pushing flow.
- By construction we'll never remove flow entering the sink, so the value will remain unchanged.

(a) In the unit case, each arc only carries one unit of flow, so if we push on a path or cycle we remove all the flow from all of the arcs involved. Since we don't add flow at all, this eliminates them from all future consideration. So every time we look at an arc, we will soon eliminate it from all future consideration. This means that the total cost is only $O(m)$.

(b) In the capacitated case, there are two cases. Either we eliminate one arc when pushing flow on a path or cycle, or we remove the excess at the node we were working on. So we charge the path to the eliminated arc or the emptied node. The length of any path or cycle is only $O(n)$, so the total time is $O((m+n)n) = O(mn)$.

(c) We do better by using dynamic trees to push the flow. When we advance in our search we do a link operation. If we reach the source or notice that we are about to form a cycle, we use the find-min and add-path operations to find the bottleneck arc, push flow along the path, and identify the right place to restart our search. The same charging argument as (b) applies, but the cost per edge of the path is now only $O(\log n)$, so the total is $O((m+n)\log n) = O(m \log n)$.

Problem 6.

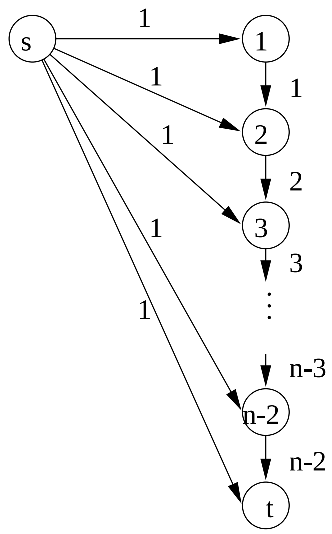
(a) There are clearly only $O(\sqrt{m})$ relabels per node, so the relabeling time is $O(m\sqrt{m})$ and the number of saturating pushes is $O(m\sqrt{m})$. There are no non-saturating pushes, so the total time is just $O(m\sqrt{m})$.

(b) If the capacities were bounded by 2, there could be only one non-saturating push on an edge before both endpoints increased label, because there could only be another non-saturating push if flow was pushed back. Thus there could only be $O(m\sqrt{m})$ non-saturating pushes, and the time bound would be unchanged.

(c) We know that the distance labels are a lower bound on actual distances to the sink, so if all excess is at labels greater than $\sqrt{m}$, then only $O(\sqrt{m})$ units of excess could possibly reach the sink. (Each unit would use its own path of $\sqrt{m}$ arcs, and there are only $O(m)$ arcs to be had.) So at this point we can use 4(a) to convert our preflow into a flow that is at most $O(\sqrt{m})$ units of flow shy of maximum. We can find the remaining flow with augmenting paths in $O(m\sqrt{m})$ time.

(d) Consider the graph from question 1 made unit capacity. Relabel $1, 2, 3, \ldots n-2$ (all to 1). Relabel $1, 2, 3 \ldots n-3$ (all to 2). Continue in this fashion, doing $O(n^2)$ relabels before doing any pushes. This graph has $m = 2n$, so n^2 is not $O(m\sqrt{m})$

Problem 7.



(a) The starting configuration has all nodes at label 0 and 1 unit of excess at each. So we could relabel node $n-2$ to 1, and push the unit of flow to the sink. Then we could relabel node $n-3$, push to $n-2$, and push to the sink. Then, node $n-4$, and so on. This will do only $O(n)$ relabels, but it will cause $O(n^2)$ pushes.

(b) If we initialize with distance from the sink, node i will get label $n-i-1$. Now we must push from 1 to 2, then 2 to 3, etc. So we initialize labels in linear time and then do $O(n)$ pushes.

(c) This suggests that it can be useful to have exact distance labels. So one obvious implementation heuristic is to start with exact distance labels. It also turns out to be useful to compute exact distances every now and then during the computation.