HW 9 (due Wednesday, at noon, November 7, 2018)

CS 473: Algorithms, Fall 2018 Version: **1.1**

Submission guidelines and policies as in homework 1.

1 (100 PTS.) Tikun Flow.

You are given an instance of network flow G, s, t with all capacities being integer numbers.

1.A. (30 PTS.) Given a flow f in G, and its residual network G_f, describe an algorithm for computing the augmenting path from s to t that has maximum residual capacity. Prove the correctness of your algorithm.

Solution:

For a given number K, we first show that an augmenting path of capacity at least K can be found in $O(|E|)$ time, if such a path exists.

We can obtain the residual graph in $O(|E|)$ time; here we assume that the graph is connected. Let $c_r(e)$ denote the capacity of the edge e in the residual graph. We create a graph G' with the nodes set V such that G' has an edge e if and only if $c_r(e) > K$. We run DFS starting with s on the graph G' and report a path from s to t (if such a path exists).

In order to find the highest capacity augmenting path in G_f, we let C_{max} to be the maximum capacity over the edges of G_f and we run the previous algorithm $\log C_{max}$ times performing binary search for the values of K in the range $[0, C_{max}]$. Since $C_{max} \leq C$, the total running time is $O(m \log C)$.

The faster algorithm is to do Dijkstra starting from s, with the following modification: The price of a path π starting at s and ending a vertex v is the smallest residual capacity of any edge of π. In particular, let $d[u]$ be the tentative cheapest path ending at u. The relax operation is now

```
relaxM($u$, $v$):+
    $x \leftarrow \min(d[u], c_f((u,v)))$
    if ( $x > d[v]$) then
        $d[v] := x$
```

Here $c_f((u,v))$ is the residual capacity of (u,v) under the given flow f. We initialize all the $d[u]$ to 0 in the beginning (and set, say, $d[s] = +\infty$). Now, we run Dijkstra with this modified relax operation, except that we always extract the vertex with the maximum value of $d[\cdot]$ that was not handled yet.

A simple induction shows that indeed $d[t]$ is the desired quantity. Indeed, consider the optimal solution $s = v_1 \to v_2 \to \cdots \to v_k = t$. Indeed, the algorithm first handles v_1, we are guaranteed after that iteration that $d[v_2] \geq c_f((v_1, v_2))$. As such, when the algorithm handles $d[v_2]$, we have that $d[v_3] \geq \min(c_f((v_1, v_2)), c_f((v_2, v_3)))$. Continuing by induction, we get that $d[t] \geq \min_{e \in \pi} c_f(e) = opt$. However, $d[t]$ is the capacity of a realizable path, we which implies that $d[t] \leq opt$. We conclude that $d[t] = opt$, as desired.

The running time of this faster variant is $O(n \log n + m)$.

1.B. (20 PTS.) Prove, that if the maximum flow in G_f has value τ, then the augmenting path you found in (A) has capacity at least τ/m.

Solution:

Let g be a maximum flow in G_f. We can assume that g is acyclic by removing cycles as seen in class. This implies that g uses at most m edges. Now, decompose g into paths, again, as seen in class. This decomposes g into m paths with flow on them.

Now, assume for the sake of contradiction, that there is no augmenting path of capacity T/m or more. Then, for at most $|E|$ paths, each of capacity $< T/m$, the max flow in $\mathsf{G}_f < m \cdot T/m = T$, a contradiction.

1.C. (10 PTS.) Consider the algorithm that starts with the empty flow f, and repeatedly use the above to find an augmenting path, until s and t are disconnected. Prove that this algorithm computes the maximum flow in G.

Solution:

The algorithm continues to add augmenting paths until there is no augmenting path of value larger than 0. Since G has integer capacities, any augmenting path has capacity at least 1. So, the algorithm is the familiar Ford-Fulkerson method, which returns a max-flow.

1.D. (20 PTS.) Consider the algorithm from the previous part, and the flow g it computes after m iterations. Prove that $|g| \geq C/\beta$, for some constant $\beta > 1$, where C is the value of the maximum flow. What is the value of β? (The smaller the better.)

Solution:

Let T_i be the capacity of the maximum augmenting path of G_f at iteration i of the algorithm. Let C_i be the maximum flow in the residual network used to compute T_i. We have that $T_i \geq C_i/m$. As such, if for any $i \leq m$, $T_i \leq C/2m$, then $C_i/m \leq C/2m$, which implies that $|g| \leq C_i \leq C/2$, as desired.

Otherwise, at each of the m iterations, the capacities of the augmenting paths $T_i > C/2m$, which implies that total capacities of the augmenting paths used in the m iterations is $\sum_i T_i \geq m(C/2m) \geq C/2$.

1.E. (20 PTS.) Give a bound, as tight as possible, on the running time of the above algorithm for computing maximum flow, as a function of n, m, and the maximum flow value in G.

Solution:

As argued at (A), finding the maximum capacity augmenting path at each iteration will take at most $O(n \log n + m)$ time. It remains to bound the number of iterations. From (D), after m iterations, the value maximum flow in the residual network shrinks by a factor of 0.9. As such, after $O(\log C)$ times of this happening, the value of the maximum flow in the residual network is < 1, which means that we are done. We conclude that the algorithm performs $O(m \log C)$ iterations, and every iteration takes $O(n \log n + m)$ time. As such, overall, the running time is $O(mn \log n \log C + m^2 \log C)$.

2 (100 PTS.) Matchless cut.

2.A. (10 PTS.) Let $\mathsf{G} = (V, E)$ be a directed graph, with source $s \in V$, sink $t \in V$, and nonnegative edge capacities $\{c_e\}$. Give a polynomial-time algorithm to decide whether G has a *unique* minimum s-t cut (i.e., an s-t of capacity strictly less than that of all other s-t cuts).

Solution:

Use (B). The cut is unique if and only if there is no central vertex in the network.

2.B. (30 pts.)

Suppose you're looking at a flow network G with source s and sink t, and you want to be able to express something like the following intuitive notion: Some nodes are clearly on the "source side" of the main bottlenecks; some nodes are clearly on the "sink side" of the main bottlenecks; and some nodes are in the middle. However, G can have many minimum cuts, so we have to be careful in how we try making this idea precise.

Here's one way to divide the nodes of G into three categories of this sort.

- We say a node v is ***upstream*** if, for all minimum s-t cuts (A, B), we have $v \in A$ – that is, v lies on the source side of every minimum cut.
- We say a node v is ***downstream*** if, for all minimum s-t cuts (A, B), we have $v \in B$ – that is, v lies on the sink side of every minimum cut.
- We say a node v is ***central*** if it is neither upstream nor downstream; there is at least one minimum s-t cut (A, B) for which $v \in A$, and at least one minimum s-t cut (A', B') for which $v \in B'$.

Give an algorithm that takes a flow network G and classifies each of its nodes as being upstream, downstream, or central. The running time of your algorithm should be within a constant factor of the time required to compute a *single* maximum flow.

Solution:

Run Edmonds-Karp algorithm on graph $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ to obtain maximum flow f and its associated residual network G_f. By $u \rightsquigarrow v$ we mean that there is a directed path starting from vertex u and ending in vertex v, in the graph G_f (all edges of that path must have non-zero capacity). Now consider the set $U = \left\{ u \,\middle|\, s \rightsquigarrow u \right\}$. Since G_f is disconnected under any min-cut, for any min-cut (A, B), $U \subseteq A$. This means that all elements of U are upstream vertices. Similarly we can see that all elements of $D = \left\{ u \,\middle|\, u \rightsquigarrow t \right\}$ are downstream vertices.

Now we show that all the vertices in $X = V \setminus (U \cup D)$ are central. Observe that $C_1 = (\mathsf{V} \setminus D, D)$ has no edge in the residual network G_f, and as such its capacity is $|f|$. Similarly, $C_2 = (U, \mathsf{V} \setminus U)$ has no edge in the residual network G_f, and as such its capacity is $|f|$. That is, C_1 and C_2 are both min-cuts. However, X is in one side of the cut in C_1 and on the other side in C_2, almost but not quite establishing the claim.

The final step is to observe that if a vertex u is central, then it is not possible for it to be in U. Indeed, if it is in U, then there is a min cut (A, B) that separates it from the source s (i.e., $u \in B$). But then, since u is reachable from s in the network G_f, it must be that this cut has an edge in G_f; that is, $|f| < c(A, B)$. That is, (A, B) is not a minimum cut. Similarly, one can argue that u can not be in D.

And now the claim indeed follows.

2.C. (30 pts.)

You are given three disjoint sets X, Y, Z of n vertices in a weighted graph G (G has N vertices and M edges) – the distance between any two vertices is the shortest path metric of the graph. Our purposes is to output a set $\mathcal{T}$ of n disjoint triplets of points $(x_i, y_i, z_i) \in X \times Y \times Z$, such that all the vertices in $X \cup Y \cup Z$ are included in exactly one such triplet, and furthermore, the price function

$$f(\mathcal{T}) = \max_{i=1}^{n} \Big(d(x_i, y_i) + d(y_i, z_i) + d(z_i, x_i) \Big)$$

is minimized, where $d(p, q)$ denotes the shortest path distance between p and q in G. Provide a polynomial time constant (the smaller the better) approximation algorithm for this problem. What is the approximation constant? (Hint: Network flow.)

Solution:

Assume x is the length of the longest used by any triangle of the optimal solution, and you know it. We build a network flow network as follows:

(A) A source s connected to all the vertices of R (with capacity 1 on each edge).
(B) For every $r \in R$ and $b \in B$, we put an edge (r, b) with capacity one, if and only if $d(r, b) \leq x$.
(C) For every $b \in B$ and $g \in G$, we put an edge (b, g) with capacity one, if and only if $d(b, g) \leq x$.
(D) For every $g \in G$, we put an edge with capacity 1 connecting it to the sink vertex t.

Let H be the resulting network, and compute the maximum flow in it between s and t. It can not be larger than n. If it is n, then there is a natural partition of $R \cup G \cup B$ into triples, such that the perimeter of each trip is at most $3x$. Furthermore, if the longest edge in the optimum is x, then this would succeed. Namely, given the longest edge in the optimum this yields a 3-approximation.

Now, compute all the pairwise distances of points in $R \cup B \cup G$, and let D be this set of distances (it has $O(n^2)$). Sort the set D, and let d_i be the ith distance in this set. Perform a binary search over d_i, finding the minimum i such that the maximum flow has value n. Clearly, this requires $O(n^2 \log n)$ time for computing distances and sorting them, and another $O(\log n)$ invocations of Edmonds-Karp max-flow algorithm, which takes $O(mn)$ time in this case. As such, the overall running time is $O\left(n^3 \log n\right)$.

2.D. (30 PTS.) You are given a set P of n vertices in a weighted graph G (as above, with N vertices and M edges), and a set Q of m vertices (i.e., base stations). The ith base station b_i, can serve at most α_i clients, and each client has to be in distance at most r_i from it, for $i = 1, \ldots, m$ (as before, we use the shortest path distances in the graph). Describe a polynomial time algorithm that computes for each client which base station it should use, and no base station is assigned more clients that it can use. The target is of course to serve as many clients as possible. What is the running time of your algorithm?

Solution:

This is a standard assignment problem. Create a graph H, having a $P \cup Q \cup \{s, t\}$ as its set of vertices, where s is the source, and t is the sink. Now, we have the following edges:

(A) For every $p \in P$ and $b \in Q$, if $d(p, b_i) \leq r_i$, add the edge (p, b_i) with capacity 1.
(B) For any $b_i \in Q$, we add the edge (b_i, t) with capacity α_i.
(C) For every $p \in P$, we add the edge (s, p) with capacity 1.

Now, compute a maximum flow in the graph G. The maximum flow is integral (assuming the α_i are integrals, of course, which we do). If the value of the maximum flow is $n = |P|$ then we can interpret the single active edge from a client $p \in P$, to a base station b_i as its assignment.

Of course, if the value of the flow is smaller than n, then there is no such possible assignment.

The running time of the algorithm is $O(mn)$, since the maximum value of the flow is $O(n)$, where m is the number of edges in the graph, which in the worst case can be $O(n^2)$.

3 (100 PTS.) Easy peasy.

3.A. (50 PTS.) Show that a maximum flow in a network $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ can always be found by a sequence of at most $|\mathsf{E}|$ augmenting paths. [Hint: Determine the paths after finding the maximum flow.]

Solution:

Let f be the given maximum flow. Next, consider the subgraph formed by edges that have non-zero flow on them, and let H be this graph. For the sake of simplicity assume that the source has only outgoing edges, and the sink t has only incoming edges.

As long as there is a directed cycle C in H, which can be detected in linear time using BFS (make sure you know how), we compute the edge on it with minimum flow c, and we push flow $-c$ on the cycle. This reduced the flow on the edges of C, but does not effect the value of the flow. In particular, one of the edges of the cycle has zero flow on it, and we can remove it. We repeat this process, till there are no cycle. What remains is a DAG J which is a subgraph of G, with a flow g on it, such that $|f| = |g|$.

Now, we just take any path in J from s to t, compute its residual flow, and subtract it from g. This removes one edge from J (since the flow on it become 0). We repeat this till there is no path remaining. By the conservation of flow, when the algorithm stops, all edges of J have zero flow on them. Thus implying the claim. As for the number of paths computed, observe that each path can be charged to the edge it removes. As such, the number of computed paths is $\leq m$.

The running time of this algorithm implemented naively is $O(m^2)$.

3.B. (50 PTS.) Suppose that a flow network $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ has symmetric edges, that is, $(u, v) \in \mathsf{E}$ if and only $(v, u) \in \mathsf{E}$. Show that the Edmonds-Karp algorithm terminates after at most $|\mathsf{V}||\mathsf{E}|/4$ iterations. [Hint: For any edge (u, v), consider how both $\delta(s, u)$ and $\delta(v, t)$ change between times at which (u, v) is critical.]

Solution:

This proof is nearly identical to the proof seen in class. There it is shown that $\delta(s, u)$ $(u \neq s)$ increases by at least 2 between each time $(u, v) \in E$ is a critical edge in the EK algorithm. A similar argument shows that $\delta(v, t)$ behaves in the same way. So, combining these results, we have that the length of the augmenting path $s \rightsquigarrow u \rightarrow v \rightsquigarrow t$ (if it again becomes augmenting) increases by at least 4 between each time that (u, v) is a critical edge. Since any augmenting path has length $\leq |V| - 1$, a given edge can be critical at most $|V|/4$ times. Here is where we stray from the proof of Theorem 26.9. Since G has symmetric edges, $|E_f| \leq |E|$. So, the residual graph has at most $|E|$ edges. From above, each of those augmenting edges can be critical at most $|V|/4$ times, for a total of at most $|E||V|/4$ iterations in the EK algorithm.