CMPSCI 311: Introduction to Algorithms **Fall 2016**

Homework 5 Solution Sketch

Name: Solutions

Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

Instructions. You may work in groups, but you must individually write your solutions yourself. List your collaborators on your submission.

If you are asked to design an algorithm as part of a homework problem, please provide: (a) the pseudocode for the algorithm, (b) an explanation of the intuition for the algorithm, (c) a proof of correctness, (d) the running time of your algorithm, and (e) justification for your running time analysis.

Submission instructions. This assignment is due by 8:00pm on 11/11/2016 in Moodle. Please submit a pdf file. You may submit a scanned handwritten document, but a typed submission is preferred.

1. **Disaster Management, K&T Ch.7 Ex.9 (15 points).** The idea here is to reduce to network flow. We make a directed graph with $n+k+2$ vertices, one per person u_i, one per hospital v_j, and two extra nodes, one which is the source s and one which is the target t. We include edges (s, u_i) with capacity 1 for each person u_i. We also include edges (v_j, t) with capacity $\lceil n/k \rceil$ for each hospital v_j. Finally if $v_j \in S_i$, we include the edge (u_i, v_j) into the network with capacity 1.

   This graph has a flow of capacity n if and only if each person can be routed to a hospital, so that no hospital services more than $\lceil n/k \rceil$ patients.

   This construction will limit the amount of traffic to each hospital to $\lceil n/k \rceil$, as no more flow than this will be able to drain to t from each hospital node. Similarly, it prevents any patient from taking more than one slot at a hospital, as no person node can be routed to more than one due to the capacity of its edge from s. A person cannot give their flow to a hospital outside of their radius because we have only given the patient node an outgoing edge to the hospitals they can reach. We have encoded all of our constraints into this graph, so its solution from FF will properly map to the result that we seek.

   For the running time, constructing the graph will take $O(nk)$ time in the worst case, if we must add an edge from every patient to every hospital. Running FF takes $O((|E| + |V|)C)$, where C is any upper bound on the value of the maximum flow. We know from above that the number of edges is $O(nk)$, and since the sum of the edge weights leaving s provides an upper bound on the maximum flow, we have $C = n$. Thus the running time is $O(n^2k)$

2. **Network Flows, K&T Ch.7 Ex.12 (15 points).** The idea here is to find the minimum cut in the graph and delete k edges from it. There are two parts to the problem, first is how to find the minimum cut and second is proving that this is optimal.

   To find the minimum cut, we run Ford-Fulkerson until the residual network has s disconnected from t, which is when the algorithm terminates. At this point, the set of nodes reachable from s are on one side of the cut and the remaining nodes are on the other side. As we saw in class, this is a minimum cut since we cannot route any more flow.

   Now, if we delete k edges from this minimum cut, the minimum cut in this new graph is at least k smaller than the minimum cut in the old graph. To see why, suppose that (A, B) is the minimum cut in the old graph (which we found using FF), and suppose it has some capacity v. By deleting k edges, the same cut (A, B) in the new graph has value $v - k$. In the new graph, the minimum cut can only be smaller than this, and so by deleting k edges, we have decreased the value by at least k. By the Max-Flow Min-Cut theorem, we have also decreased the value of the maximum flow by at least k.

   To see why this is optimal, observe that since only a single unit of flow can traverse each edge (since all edges have unit cost), deleting an edge can reduce the maximum flow by at most 1 unit. Therefore, k is the largest reduction in flow we can hope for.

If the minimum cut has value $\leq k$, then we can disconnect s from t, so that both the minimum cut and maximum flow can be set to zero. Again this is the best we can do.

Running FF takes $\theta((|E|+|V|)C)$. The number of edges we remove cannot exceed $|E|$, so $\theta((|E|+|V|)C)$ is our final complexity. Since all edges have unit cost, a trivial upper bound on C is $|E|$, so the running time is $O(|E|^2 + |E||V|)$.

3. **Node Capacities, K&T Ch.7 Ex.13.** The idea here is that we can convert a graph with node capacities c_v into a standard edge-capacity graph in polynomial time. To do this we split each node v into two separate nodes v_{in} and v_{out}, where incoming edges to v connect to v_{in} and outgoing edges connect to v_{out}. We then connect v_{in} and v_{out} with an edge with capacity c_v. Since any flow going in to v_{in} must pass through the edge (v_{in}, v_{out}), the capacity c_v can never be exceeded. All the other edges have no capacity constraints.

   Then, solving for maximum flow on this new network is equivalent to solving for maximum flow on the node capacitated network.

   To define a cut in a node-capacitated network, let us just use the standard definition of a cut in this new graph. Hence we get the Max-Flow Min-Cut theorem for free. Note however that since in our reduction, the only edges with capacity constraints are of the form (v_{in}, v_{out}) for each v, the min cut will always only choose these edges. As such, we can equivalently think of cuts as a set S of vertices such that all $s-t$ paths must go through S. The capacity of a cut S is $\sum_{v \in S} c_v$. This is equivalent to the definition induced by the reduction.

4. **Bipartite Graphs, K&T Ch.7 Ex.15.**

   (a) Let $G = (V, E)$ be a bipartite graph with a node $p_i \in V$ representing each person and a node $d_j \in V$ representing each night. The edges consist of all pairs (p_i, d_j) for which $d_j \notin S_i$.

   Now, a perfect matching in G is a pairing of people and nights so that each person is paired with exactly one night, no two people are paired with the same night, and each person is available on the night they are paired with. Thus, if G has a perfect matching, then it has a feasible dinner schedule. Conversely, if G has a feasible dinner schedule, consisting of a set of pairs $S = \{(p_i, d_j)\}$, then each $(p_i, d_j) \in S$ is an edge of G, no two of these pairs share an endpoint in G, and hence these edges define a perfect matching in G.

   (b) First, construct the bipartite graph G from part (a): This takes at most $O(n^2)$ time. Now build the flow network that would arise in the reduction from bipartite matching to network flow (add a source and target node, etc.). Now use the partial solution provided to initialize the flow, in particular route flow along all edges provided, except one of the two with conflicts (say (p_j, d_k)). This is a matching and also a feasible flow in the flow network.

   We now try to find an augmenting flow in G with respect to the current flow solution, in time $O(|E|) = O(n^2)$. If we find such an augmenting flow P, then increasing Q' using P gives us a matching of size n, which is a perfect matching and hence by (a) corresponds to a feasible dinner schedule. If G has no augmenting path with respect to Q', then by a theorem from class, Q' must be a maximum matching in G. In particular, this means that G has no perfect matching, and hence by (a) there is no feasible dinner schedule.

   Finding an augmenting path here requires one iteration of FF, which in this graph just takes $O(n^2)$ time. We only require one iteration of FF, since we already have a flow of value $n-1$, the maximum value is n and either FF terminates, or we are guaranteed to increase the flow by 1. Note that if we ran FF from scratch, the algorithm would take $O(n^3)$ time!