Advanced Algorithms, Fall 2012

Prof. Bernard Moret

Homework Assignment #7: Solutions

Question 1. (Network Flow)
Let $G = (V, E)$ be a directed graph with source s and sink t. Let k be a positive integer.

1. *Propose a polynomial-time algorithm to decide whether there exist k edge-disjoint directed paths from s to t.*
2. *Propose a polynomial-time algorithm to decide whether there exist k node-disjoint directed paths from s to t.*
3. *If there exist k edge-disjoint directed paths from s to node u and there exist k edge-disjoint directed paths from u to t, do there exist k edge-disjoint directed paths from s to t? Give a proof or a counterexample.*
4. *If there exist k node-disjoint directed paths from s to node u and there exist k node-disjoint directed paths from u to t, do there exist k node-disjoint directed paths from s to t? Give a proof or a counterexample.*

Solution:

1. We set the capacity of each edge to 1 and then calculate the maximum s-t flow of the network. We can prove that there are k edge-disjoint directed paths from s to t if and only if the value of the maximum flow of the network is larger than or equal to k. If there are k such paths from s to t, then these paths form k independent augmenting paths, each of which can increase the value of flow by 1. Thus, the maximum flow of the network must be larger or equal to k. On the other hand, if the value of the maximum flow is v and $v \geq k$, since the capacity of each edge is integer, we have an integer maximum flow with value v. Thus, those edges with flow 1 yield v edge-disjoint paths from s to t.

2. We split each node v, except s and t, into 2 nodes, v_1 and v_2, and we add an directed edge from v_1 to v_2. For those edges that point to v, we move their heads to v_1, and for all edges that leave v, we move their tails to v_2. Clearly, there are k node-disjoint directed paths in the original graph if and only if there are k edge-disjoint directed paths in the new graph—and we just solved that problem.

3. As we have just seen, having k edge-disjoint paths from s to u means having a s-u flow with value k, or equivalently, an s-u cut containing k edges (or an an u-t cut with k edges). Now we prove that the minimum s-t cut contains at least k edges by contradiction. Suppose that the minimum s-t cut contains less than k edges. Consider the location of node u in this minimum cut. Without loss of generality, assume that u is on the side of s. Then this minimum cut is also a u-t cut, which contradicts the fact that the minimum u-t has value at least k. Thus, the value of the maximum s-t cut is also at least k, which means there exists k edge-disjoint edges from s-t.

4. This statement is false. Consider a directed graph with 3 nodes, which are s, u and t, and 4 edges, two of which point from s to u and the two two point from u to t. Clearly, there are 2 node-disjoint paths from s to u and there are 2 node-disjoint paths from u to t. But there is only one path from s to t, since every path must be through node u.

Question 2. (Network Flow)
A group of n people share a flat, in which there is only one kitchen. On any given day, not all of them will use the kitchen; of those who do use it, one (chosen uniformly at random) will clean it. Thus, during a period of m days, the expected number of days on which the i-th person should clean the kitchen is $D_i = \sum_{j=1}^{m} I_{\{i\in S_j\}}|S_j|^{-1}$*, where* S_j *is the set of people who use the kitchen on the j-th day, and* $I_{\{\cdot\}}$ *is the indicator function. Prove that we can always assign one person from* S_j *to clean the kitchen and make sure that the i-th person will clean the kitchen on at most* $\lceil D_i \rceil$ *days, and give a polynomial-time algorithm to compute such an assignment.*

Solution: We build an undirected network as follow. We create n nodes, $p_1, p_2, \cdots, p_n$, one for each person, and m nodes, $d_1, d_2 \cdots, d_m$, one for each day. We add an edge (p_i, d_j) if the i-th person uses the kitchen on the j-th day, i.e., $i \in S_j$, and set the capacity of each such edge to 1. We then add a source node s and, for each i, $1 \le i \le n$, we add edge (s, p_i) with capacity $\lceil D_i \rceil$. Finally we add a sink node t and add edge (d_j, t) with capacity 1 for all $1 \le j \le m$. Clearly, a feasible assignment (the i-th person cleans the kitchen on at most $\lceil D_i \rceil$ days, and on each day there must be one person to clean the kitchen) exists if and only if there exists an integer flow from s to t with value m in this network. We first show that there exists a fractional s-t flow with value m. This can be achieved by following the uniformly random strategy. We set the flow on edge (p_i, d_j) to $|S_j|^{-1}$, set that on edge (s, p_i) to D_i, and set that on edge (d_j, t) to 1. This assignment produces a legal flow with value m. Next, since all edge capacities are integers, we must have an integer s-t flow with value m, which means that we can find a feasible assignment.

Question 3. (Network Flow)
A group of n people share a washing machine. In each week, totally there are m disjoint available time-slots for washing. Each person choose a subset of them as his or her candidate washing time. A valid assignment is to choose at least l time-slots and at most h time-slots for each person from his or her candidate list, and make sure that these chosen time-slots are distinct. Propose a polynomial-time algorithm to decide whether there exist a valid assignment such that the total number of chosen time-slots is exactly a given integer k, $k \le m$*.*

Solution: We build an undirected network as follow. We create n nodes, $p_1, p_2, \cdots, p_n$, one for each person, and m nodes, $a_1, a_2 \cdots, a_m$, one for each time-slot. We add edge (p_i, a_j) if the j-th time-slot is in the candidate list of the i-th person, and set the capacity of each such edge to 1. We then add a source node s and, for each i, $1 \le i \le n$, add edge (s, p_i) with capacity h. Finally we add a sink node t and, for each j, $1 \le j \le m$, add edge (a_j, t) with capacity 1 Now a valid assignment with a total of k chosen time-slots exists if and only if in this network there exists an integer flow with value k such that the value of this flow on edges (s, p_i) is at least l.

Because of the presence of lower bounds, this problem cannot be solved directly by a maxflow solver. We need to build the lower bounds into the network. So we add a new source node s^* and link to each p_i with an edge of capacity l. We also link s^* to s with an edge of capacity $k - n \cdot l$. Finally, we change the capacity of edge (s, p_i) to $h - l$. Now, we claim that the

the maxflow in the new network is k if and only if the original network has an integer flow of value k such that the flow on each edge (s, p_i) is at least l. Note that, if the maxflow in the new network is k, then all edges adjacent to s^* must be saturated, which means we can build a flow with value k in the old network that satisfies the lower bound condition. Conversely, a flow with value k satisfying the lower bound condition in the old network can be trivially transformed to a flow of equal value in the new network.

Question 4. (Computational Geometry)
You are given n distinct points in the plane, no three of them collinear. Devise a $O(n \log n)$ algorithm to produce a simple polygon that has these n points as its n vertices.

Solution: The solution is based on sorting. We can sort along an axis or sort by angle. Sorting by angle is almost more intuitive here, but we shall see in class the algorithm known as the Graham scan (for building a convex hull), which is nearly identical in spirit, so we give a solution based on sorting along an axis.

We first identify in linear time the leftmost point p_l and the rightmost point p_r. (If there are ties, we take resolve them arbitrarily.) Then we split the remaining $n-2$ nodes into 2 sets: one set contains all points above (or on) the line determined by p_l and p_r, while the other set contains all points below the line. This takes linear time. For each set, we will create a simple path (that is, a path that visits a vertex at most once) from p_l to p_l that passes through all points in the set. Because of the way in which we partitioned the points, the simple path from one set intersects the simple path from the other set in exactly two points, p_l and p_r—hence the two simple paths form a simple polygon.

We build each simple path by sorting. We sort the points in a set along the axis determined by the line through p_l and p_r—-for instance, by considering their orthogonal projection onto that line. In case of ties, we break the ties by the distance from the points to the line through p_l and p_r. We then add each point to the simple path in sorted order, starting the path at p_l, adding the first sorted point, then the next, and so forth until the last point has been added, after which we add p_r. The running time is dominated by the sorting and so is $O(n \log n)$.