CS 577: Introduction to Algorithms — Homework 9 Solution

Out: 04/17/17 **Due: 04/24/17**

Problems

1. **(Page limit: 1 sheet; 2 sides)** (Taken from "Algorithm Design") A group of people are carpooling to work together, and want to come up with a fair schedule for who will drive on each day. This is complicated by the fact that the subset of people in the car varies from day to day due to different work schedules. We define fairness as follows. Let $S = \{1, \cdots, n\}$ denote set of people. Suppose on the i-th day a subset $S_i \subseteq S$ go to work. A schedule for d days is a sequence $i_1, i_2, \cdots, i_d$ specifying the driver for each day. A schedule is fair if for each driver j, the total number of times they drive is at most

$$\Delta_j = \left\lceil \sum_{i:j\in S_i} \frac{1}{|S_i|} \right\rceil .$$

   (a) Design and analyze an efficient algorithm for computing a (potentially partial) fair driving schedule that maximizes the total number of days to which a driver is assigned.

   (b) Prove that for any choice of sets $S_1, ..., S_d$, there exists a fair driving schedule that assigns a driver to *every* day.

   Hint: For the first part, reduce the problem to network flow. What can you say about the value of the maximum flow in your constructed network?

   SOLUTION:

   (a) We reduce the construction of an optimal fair schedule to an instance of MAX-FLOW. Add a source s, a sink t to the network. For each person j, we add a node P_j and an edge (s, P_j) of capacity Δ_j to enforce the constraint that person j may drive no more than Δ_j times. For each day i, we add a node S_i and an edge (S_i, t) to enforce the constraint that there is at most one driver on any given day. Finally, for each pair (P_j, S_i) such that person j goes to work on day i, we add an edge (P_j, S_i) of capacity 1 to indicate that person j can drive once on the i-th day.

   We now show that there is a one-to-one correspondence between integral flow in the network and fair schedules.

   Claim: There exists a fair schedule with a driver on k days if and only if there is an *integral* flow of value k.

   Proof. Suppose there is fair schedule that has a driver on k days. Then one can obtain an integral flow of value k as follows: If person j drives on day i, send 1 unit of flow along the path s, P_j, S_i, t. We do so for all days with a driver. Since we started with a fair schedule with a driver on k days, all capacity constraints in the network are satisfied (why?) and we send k units of flow from s to t. By the construction of the flow, it is integral.

   Now for the other direction, let us suppose that we have an integral flow of value k. We obtain a fair driving schedule with a driver on k days as follows: If an edge (P_j, S_i) carries a unit of flow, we have person j drive on day i. By the capacity constraints on edges leaving the source and edges entering the sink, each person j drives at most Δ_j times and each day has at most one driver. This tells us that the schedule we constructed is fair. Now, since the flow has value k, there are k days i such that the edge (S_i, t) carries a unit of flow. Since the flow into an vertex is equal to the flow out of a vertex, this means there are k days i for which there exists a person j such that (P_j, S_i) carries a unit of flow. This tells us that there are k days that have a driver in the schedule we constructed. □

By the claim, to find the optimal fair schedule, we only need to find an integral max-flow in the network. One can use the Ford-Fulkerson algorithm to do so. Given an integral max-flow, we construct a fair schedule as we did in the proof of the claim.

Running Time The running time of our algorithm is given by the time required to construct the network plus the time to find the max-flow in the network plus the time to retrieve a fair schedule from the max-flow. The size of the network we construct is $O(nd)$. As a result, the time required to construct the network is $O(nd)$, and one can retrieve a fair schedule given the max-flow in $O(nd)$ time. The running time of Ford-Fulkerson is $O(|E|F)$, where $|E|$ is the number of edges in the graph and F is the value of the maximum flow. Since there are d edges entering the sink each of capacity 1, the max-flow in the graph has value at most d. Therefore, finding the max-flow takes $O(nd^2)$ time (the number of edges in the graph is $O(nd)$) . It follows that our algorithm for finding the optimal fair schedule runs in $O(nd^2)$ time.

(b) By the claim, to show that there is a fair schedule that has a driver on each day, it is sufficient to exhibit an integral flow of value d in the network. To show that there is an integral flow of value d, it is sufficient to show that there is an s-t flow of value d (why?). We construct such a flow as follows: For each person j and for each day i on which j goes to work, send $\frac{1}{|S_i|}$ units of flow along the path s, P_j, S_i, t. The amount of flow through an edge (s, P_j) is exactly $\sum_{i:j\in S_i} \frac{1}{|S_i|} \leq \Delta_j$. The flow along an edge (P_j, S_i) is $\frac{1}{|S_i|}$. And the amount of flow along an edge (S_i, t) is $\sum_{j\in S_i} \frac{1}{|S_i|} = 1$. Therefore, each edge (S_i, t) carries one unit of flow and the total flow into the sink is d. This shows that there is a flow of value d. It follows that there is a fair schedule with a driver on all days.

2. **(Page limit: 1 sheet; 2 sides)** The manager at a local toy store, Algos-R-Us, is in need of some algorithmic expertise. A few days back he received a shipment of Russian nesting dolls that was damaged in transit. All of the sets were disassembled, that is, none of the dolls were nested inside another. There are n dolls in all and k boxes to pack them into. The manager needs to figure out how to assemble the n dolls into k or fewer nested sets, if at all possible. For any two dolls, it is possible to tell if one can be nested inside the other, but there is no other way of telling whether two dolls belong to the same set. For some pairs of dolls, neither can be nested inside the other (e.g. one may be taller than the other and the other wider than the first). Design and analyze an efficient algorithm to find an arrangement of the n dolls into k or fewer nested sets, if such a partition exists.

To be precise, the input to your algorithm consists of the numbers n and k, and for each pair $i, j \in [n]$, a bit that specifies whether or not doll i can be nested inside doll j. Your algorithm should return an arrangement of the n dolls into k or fewer nested sets, if such an arrangement exists, and return "Error" if not.

Hint: Think of the arrangement of dolls as a matching that matches each contained doll with the doll that directly contains it. So each doll is matched at most once to a doll containing it, and at most once to a doll contained in it.

SOLUTION:

We want to find a nesting of dolls that yields k or fewer nested sets. The first thing we observe is that this is equivalent to saying that we want to place at least $n-k$ dolls inside other dolls - every time we put a doll i inside a doll j, we reduce the total number of nested sets by 1, regardless of whether doll i already had other dolls in it. The constraints are that each doll only fits inside some other dolls, and that no two dolls can be placed inside the same third doll (unless they're nested one inside the other). These constraints are reminiscent of those we have in a maximum matching problem.

We model the given problem as a problem of maximum bipartite matching. Given n dolls and additional information specifying which dolls can be nested in which other dolls, we construct a bipartite graph G. G has $2n$ vertices, one on the left and one on the right for each doll. There is an edge from doll i on the left to doll j on the right if doll i fits inside doll j.

We claim that any matching M of G corresponds to a valid nesting of dolls that results in $n - |M|$ nested sets, and vice versa. To see this, note that a matching corresponds to an achievable nesting of dolls - no two edges share an endpoint, enforcing the conditions that no doll has more than one doll directly inside it and the condition that no doll is placed directly inside more than one doll. Furthermore, each edge of the matching represents a placement of a doll inside another doll, and so decreases the total number of nested sets by 1. This shows that a matching M corresponds to a nesting of dolls with $n - |M|$ nested sets. In the other direction, any valid placement of dolls inside other dolls corresponds to a matching in this graph: given a valid nesting A of dolls, we construct a matching of G by choosing the edge from i (on the left) to j (on the right) if doll i is placed *directly* inside doll j in A. These edges always exist in G since doll i fits inside doll j, and the set of such edges taken together is a matching, as no two of them can share an endpoint (since this would mean that either some doll i was directly inside more than one doll or more than one doll was placed directly inside some doll j). Furthermore, if there are m nested sets in A, $n - m$ dolls must have been placed inside other dolls. So, $|M| = n - m$.

This tells us that there is a nesting of dolls with at most k nested sets if and only if there is a matching of G with at least $n - k$ edges. Therefore, to determine whether a nesting with no more than k nested sets exists, one only needs to check whether the maximum matching in G has size at least $n - k$. To find the required nesting, we use the correspondence between matching and nesting of dolls described above.

Running Time The running time of the algorithm is equal to the time required to construct the graph G (denoted by $T_1(n)$) plus the time to find the maximum matching ($T_2(n)$) plus the time to retrieve a nesting of the dolls from the maximum matching ($T_3(n)$). Note that $T_1(n) = O(n^2)$ and $T_3(n) = O(n)$. The algorithm we saw in class for maximum matching runs in time $O(|V||E|)$, so $T_2(n) = O(n^3)$. Therefore, our algorithm for finding a nesting of dolls runs in $O(n^3)$ time.