CSCI 570 - Fall 2015 - HW 10

1. There is a precious diamond that is on display in a museum at m disjoint time intervals. There are n security guards who can be deployed to protect the precious diamond. Each guard has a list of intervals for which he/she is available to be deployed. Each guard can be deployed to at most A time slots and has to be deployed to at least B time slots. Design an algorithm that decides if there is a deployment of guards to intervals such that each interval has either exactly one or exactly two guards deployed.

We create a circulation network as follows. For the *i*th guard introduce a vertex g_i and for the *j*th time interval introduce a vertex t_j. If the *i*th guard is available for the *j*th interval, then introduce an edge from g_i to t_j of capacity 1. Add a source s and a sink t. To every guard vertex add an edge from s of capacity A and lower bound B. From every interval vertex add an edge to t of capacity 2 and lower bound 1. Add an edge from t to s of infinite capacity. We claim that there exists a valid deployment if and only if the above network has a valid circulation. The proof of the claim is virtually identical to the proof in section 7.8 of the text for the survey design problem. The algorithm proceeds by determining if the network has a circulation (by reducing it to a flow problem and then applying Ford-Fulkerson) and answers "yes" if and only if there is a circulation. The number of vertices and number of edges in the resulting flow problem are bounded by O(n) and $O(n^2)$ respectively. The running time of our algorithm is dominated by the flow computation which takes $O(n^3)$.

2. The computer science department course structure is represented as a directed acyclic graph G = (V, E) where the vertices correspond to courses and a directed edge (u, v) $\in$ E exists if and only if the course u is a prerequisite of the course v. By taking a course w $\in$ V, you gain a benefit of b_w which could be a positive or negative number. Design an algorithm that picks a subset A $\subseteq$ V of courses to take such that the total benefit $\sum_{\omega\in A} b_\omega$ is maximized. Remember that if v $\in$ A and (u, v) $\in$ E, then u has to be in A. That is, to take a course, you have to take all its prerequisites. The running time should be polynomial in |V |.

See solution to project selection problem from section 7.11.

3. Solve Kleinberg and Tardos, Chapter 7, Exercise 28.

We create a circulation network as follows. For the *i*th TA introduce a vertex p_i and for the *j*th time interval introduce a vertex t_j. If the *i*th TA is available for the *j*th interval, then introduce an edge from p_i to t_j of capacity 1. Add a source *s* and a sink *t*. To every TA vertex add an edge from s of capacity *b* and lower bound *a*. From every interval vertex add an edge to *t* of capacity 1. Add an edge from *t* to *s* of capacity *c* and lower bound c.

We claim that there exists a valid assignment if and only if the above network has a valid circulation.

Proof of Claim: We first show that if there is a valid assignment, then the above network has a valid circulation. Assume there is a valid assignment. If in the assignment, the *i*th TA is assigned to the *j*th interval, then assign a flow of 1 to the edge (p_i, t_j). Extend this flow to the rest of network using flow conservation laws to get a valid flow. That is, the flow on the edge (t_j, t) is 1 if a TA is assigned to t_j and 0 else, the flow on edge (t, s) is the number of intervals with TAs assigned and (s, p_i) is the number of intervals the *i*th TA is assigned to. The validity of the assignment implies that flow conservation holds at every vertex and capacity/lower bound constraints hold at every edge (check this !).

We next prove the converse, that is if there is a valid circulation, then there is a valid assignment. Assume that the network has a valid circulation. Since the capacities and lower bounds are integers, when we convert the circulation problem into a flow problem, we get a network with integer capacities. The correctness of Ford-Fulkerson algorithm implies that the resulting flow network has a maxflow where the flows are integers. Thus we can conclude that our original network has a valid circulation where the flow on each edge is an integer. Let f(e) denote the flow on edge *e* in this integer valued circulation. We next construct a TA assignment. Since the capacity of each edge from TA vertices to interval vertices is 1, the flow on these vertices is either 1 and 0. Assign the *i*th TA to the *j*th interval if and only if f((p_i , t_j)) = 1. The assignment is valid for the following reason. The *i*th TA is assigned to f(s, p_i) intervals (which is between *a* and *b*), every interval is assigned to at most one TA (since the flow out of an interval vertex is at most 1 due to the capacity constraint) and the number of intervals with TAs assigned is f(t, s) which is exactly *c* due to the lower bound/capacity constraint.

The algorithm proceeds by determining if the network has a circulation (by reducing it to a flow problem and then applying Ford-Fulkerson) and if so finding one with integer flows (denote by f). If there is a valid circulation, then Assign the *i*th TA to the *j*th interval if and only if f((p_i , t_j)) = 1.
Else, return “no assignment”.
As in problem 1 above, the running time is O(n^3).

4. A matching of an undirected graph is a subset of the edges of the graph such that no two edges in the subset share a vertex. Design an algorithm that given a tree T finds a matching of T with the maximum number of edges. The running time should be bonded by a polynomial in the number of vertices in the tree.

There are at least two ways of approaching this problem. One is through dynamic programming and the other through network flows.

Perhaps the easier solution is through network flows.

The first observation is that every tree is bipartite (There are no cycles in a tree which implies that there are no odd cycles in a tree which implies that a tree is bipartite). But from class (or Sec 7.5 in text) we know how to compute a maximum matching of bipartite graph! All that remains is to represent our tree as a bipartite graph (that is, find a partition of its vertices such that there are no edges contained within a single partition). This is accomplished by rooting the tree at an arbitrary vertex and considering all the odd level vertices on one side of the partition and the even level vertices on the other side. (This can be performed efficiently by BFS).

There is a very natural dynamic programming solution which we next briefly sketch. Root the tree T=(V,E) at an arbitrary vertex r. For a vertex x, let T_x denote the subtree rooted at x and let OPT(x) denote the size of the maximum matching of the subtree rooted at x.

Now consider a maximal matching M_v for the tree rooted at v. Either M_v does not contain an edge incident on v or it does. If it does, then M_v contains an edge (u,v) where u is a child of v. Further, M_v restricted to T_w (that is, the intersection of M_v with the edge set of T_w) where w is a child of v other than u is a maximum matching for T_w. Likewise, M_v restricted to T_z where z is a child of u is a maximum matching of T_z. Hence, we can deduce the following recurrence,

OPT(v) equals the minimum of the following two terms:

(i) Sum of OPT(u), where the summation is over children of v.

(ii) Min_{u is a child of v} {sum of OPT(w) where w ranges over the children of v other than u + sum of OPT(z) where z ranges over the children of u}.

The above recurrence can be solved bottom up (that is starting from the leaves). Keep track of the choice made at each iteration (that is, was an edge from v to a child selected in the matching ? and if so which one) and trace back the Maximum matching M_r from the computed OPT() values.

The initial condition is that OPT(y)=0 for all leaves y. When implemented naively the running time is O(n^2) since we have n subproblems and to compute OPT(v) given the OPT values of the descendants of v we at most need to spend O(n) time). Here n denotes the number of vertices in the tree.