Modern Algorithm Design (Monsoon 2024)
Midsem
Full Marks 60

1. (10+10 points) Consider an undirected graph $G = (V, E)$ with w_{uv} being the weight/cost of the edge uv (assume all non-negative weights). For each of the linear programs below, if you consider a $\{0, 1\}$ optimal solution, then they would correspond to a problem which you have seen in your lectures. State the correct problem and also justify your choice. In particular, you have to show why a $\{0, 1\}$ solution to the LP exactly corresponds to a solution to the particular problem.

   a) In the following $\delta(S, S)$ denotes the subset of edges in G that have both the endpoints inside S, where S is a subset of V.

$$\begin{aligned} \text{minimize} \sum_{(uv)\in E} w_{uv} \cdot x_{uv} \\ \text{subject to:} \sum_{(uv)\in E} x_{uv} = n - 1 \\ \sum_{(uv)\in\delta(S,S)} x_{uv} \leq |S| - 1, \forall S \subset V \\ x_{uv} \geq 0, \forall (uv) \in E \end{aligned}$$

   Solution. We claim that any $\{0, 1\}$ solution to the above corresponds to a *minimum spanning tree* of the given graph. (+4 points)

   (+6 points) Consider any such solution and let F denote the subset of edges uv for which $x_{uv} = 1$. Firstly, we observe that F can contain only one connected component. Suppose not. Then by the first constraint and pigeon hole principle, there exists one component, say F' which contains a cycle C. But then the set of vertices in C violates the second constraint. By a similar argument, it can be observed that this single connected component also cannot contain a cycle. Hence the proof. (+3 for each argument)

   b) Let s, t be two vertices in the graph. Let $\mathcal{S}_{s,t}$ be the family of all such cuts (S, S') such that $s \in S, t \in S'$ and $\delta(S)$ denotes the subset of edges crossing the cut S.

$$\begin{aligned} \text{minimize} \sum_{(uv)\in E} w_{uv} \cdot x_{uv} \\ \text{subject to:} \sum_{(uv)\in\delta(S)} x_{uv} \geq 1, \forall S \in \mathcal{S}_{s,t} \\ x_{uv} \geq 0, \forall (uv) \in E \end{aligned}$$

   Solution. We claim that any $\{0, 1\}$ solution to the above corresponds to a *shortest s-t path* of the given graph. (+4 points)

   Consider any such solution and let P denote the subset of edges uv for which $x_{uv} = 1$. Firstly, we claim that P will be a single connected component that contains both s and t. Suppose not and s and t lie in different components. Now consider all the vertices such that they belong to the the connected component containing s in P - call these vertices S. By our assumption, $t \in S$ implying that $V \setminus S$ is non-empty. There cannot be any edge $uv \in P$ such that $u \in S, v \in S'$ since that would violate the definition of S. But then the LP constraint is violated for S leading to a contradiction. Furthermore, suppose there is more than one connected component in P. But then we can remove all the edges from the component that does not contain s, t. Note that this cannot violate any constraint and gives us a strictly better/same cost.

Now we claim that the optimal solution corresponds to a path. Suppose not - then by an analogous argument as above, we can keep removing edges - thereby either improving the cost of the LP or keeping as it is - and end up at a path between s and t. (+3 for proving one component, +3 for proving path)

2. (20 points) Recall from the lectures a low diameter decomposition of graph $G = (V, E)$ with $d(uv)$ denoting the shortest path distance between vertices u, v (assume all edge lengths are non-negative and hence d defines a metric on the set of vertices V). Given a target diameter D, the goal is to define a random partition of the set of vertices V in to $V_1, V_2, \cdots V_k$, where $k \leq n$ such that

   (a) For any two vertices $u, v \in V_i$, $d(uv) \leq D$ for all $V_i, i = 1, 2, \cdots k$

   (b) For any two vertices u, v the probability that u and v belong to different partitions is at most $\beta \cdot \frac{d(uv)}{D}$, for some $\beta > 0$

   Now assume that the graph G is a tree rooted at a specific vertex r and all edges in the tree are of length 1. Design a LDD with $\beta = 2$. Clearly state the algorithm and prove correctness. (Hint: Hope you have understood the algorithm for path graphs !)

   Solution. We give a sketch of the algorithm here. Writing anything close to this will fetch full credit. First consider a deterministic algorithm analogous to the path case. Start at an arbitrary vertex v as a 'root'. First run a BFS to define levels of all other vertices. So $level(v) = 0$, $level(u) = 1, \forall u$ such that $d(uv) = 1$ etc.

   Now define $V_1 = \{u \mid d(uv) \leq \lfloor D/2 \rfloor\}$. Let $V' \subseteq V_1$ be the set of vertices such that $level(u), u \in V'$ are all equal and maximum in V_1. Now recurse with the children of each vertex in V' as new roots.

   Note that the above is deterministic. So the final ingredient is - instead of carving out the first set V_1 precisely at distance $D/2$, choose some distance D' uniformly between $[0, D/2]$. The rest of the algorithm remains exactly the same. Note carefully that the random choice happens only at the first step and not at the subsequent recursive calls.

   (+7 for the correct deterministic idea. +3 for the right randomization)

   Now we prove the two properties of the above decomposition. The first property follows immediately since for any two vertices u, v in the same partition, their distance from a common vertex (root of the recursion) is at most $D/2$.

   For the second property, consider two vertices u, v and let $e_1, e_2, \cdots e_k$ be the edges on the unique $u - v$ path in the given tree. Consider any edge $e_i = (u_i, v_i)$ in the above set. The probability of the two endpoints u_i, v_i being separated is precisely $\frac{1}{D/2}$. Further, the probability that any of the edges $e_1, e_2, \cdots e_k$ is separated by the algorithm is the union over the individual edges being cut. This is upper bounded by $\sum_{i=1}^{k} \frac{1}{D/2} = 2k/D = \frac{2d(u,v)}{D}$. (+10 for the proof. This is subjective)

3. (10+10 points) The following two problems are about *reductions*. That is, how to utilize algorithms for one problem to solve another one.

   (a) Recall that in the rooted min-cost arborescence problem, we are given a digraph $G = (V, A)$ with edge weights w_e and a root $r \in V$, and we want to find the min-cost r-arborescence (a directed subgraph such that every vertex other than r has only one outgoing edge and r has no outgoing edge). In the *unrooted* min-cost arborescence problem, we are given a digraph $G = (V, A)$ with edge weights w_e and want to find the min-cost arborescence in the graph, regardless of which vertex it is rooted at. Show that the rooted and unrooted problems are algorithmically equivalent. i.e., given an algorithm for one problem, show how to solve the other. Your reductions should take linear time in the size of the graph.

   Solution. One direction is almost trivial. Given an algorithm for the rooted version, we just run this algorithm for every choice of root - there can be $|V|$ many - and return the minimum among them.

   The other direction is slightly more subtle. The algorithm is simply the following - suppose r is the given root. Delete all outgoing edges from r and run the algorithm for unrooted version. We need two observations to justify that this gives us the right answer. Firstly, we claim that the algorithm will produce an r-arborescence if one exists. Suppose not and let r' be the root selected by the unrooted algorithm. But then there cannot be a path from r to r' in this arborescence since we have removed all outgoing edges from r. Conversely, every arborescence rooted at r is preserved in the modified graph and hence we get back the minimum cost one (if one exists). (+5 for each direction)

(b) Suppose you are given an algorithm $\mathcal{A}_{PM}$ which can check if a *perfect matching* exists in a graph. Show how to determine the size of the maximum matching in the graph with only $O(\log n)$ calls to $\mathcal{A}_{PM}$. You cannot use subroutine other than adding edges/vertices to your graph as and if required.

Solution The $O(\log n)$ tells us that there has to be some kind of binary search. Consider a number k such that $1 \leq k \leq n$. Consider a modified graph as follows - we introduce $n - 2k$ many extra vertices and each vertex in the given graph has an edge to each of the dummy vertices. It is not difficult to observe that there is a perfect matching in the modified graph if and only if the maximum matching in the given graph is of size at least $2k$. This can be checked by a single call to $\mathcal{A}_{PM}$. So the only remaining piece is to figure out the correct value of k. This can be done using a binary search for k for all values between 1 and n and hence we need only $O(\log n)$ many calls to $\mathcal{A}_{PM}$. As usual, start with $k = n/2$ (ignoring floor and ceiling). Now if $\mathcal{A}_{PM}$ returns a YES, double the value of k and continue. Otherwise, halve the value of k and continue.

(+5 for the correct construction. +5 for writing the correct binary search conditions.)