CS535 Fall 2022 HW5 Sample Solutions

1. Delete the edge covering v from M, and delete all edges incident to v in the graph. Construct the alternating forest in $O(n^2)$. Notice that v is currently an unmatched even node in the forest. Now for each deleted edge (v, u), we check if u is an even vertex in the forest. If so, there exists an augmenting path with the addition of edge (v, u), after which we reach at a perfect matching where v is matched to u, so we can add u to the list.

2. Let L be the list of the n members sorted by the *starting* time of the departure interval in the *ascending* order. For each $1 \leq i \leq n$, let $[s_i, t_i]$ be the departure interval of the i-th member. The leximal matching (i.e., pairing) is repeatedly constructed as long as L is non-empty as follows: Remove the *last* member q from L. If q is matchable from member in L let p the *last* such member, pair up q with p, and then remove p from L. We first assert the optimality of the leximal pairing. It is sufficient to show that the first pair $\{p, q\}$ is contained in some maximum matching. We prove this via an exchange argument. Consider an optimal matching M^* which does not include the pair $\{p, q\}$. Then at least one of them is matched in M^*. If exactly one of them is matched in M^*, we replace the pair by $\{p, q\}$. Then M^* is still a maximum matching and it contains $\{p, q\}$. Henceforth, we assume that both are matched in M^*, say with $\{p, p'\}$ and $\{q, q'\}$. By the greedy choice of p, we have $q' < p$. Since $q' < p$ and $\{p, p'\}$ are matchable, we have $s_{q'} \leq s_p \leq t_{p'}$. Since $p' < q$ and $\{q, q'\}$ are matchable, we have $s_{p'} \leq s_q \leq t_{q'}$. Thus, $\{p', q'\}$ is matchable. Hence by replacing $\{p, p'\}$ and $\{q, q'\}$ with $\{p, q\}$ and $\{p', q'\}$, we obtain a maximum matching containing $\{p, q\}$.

   Now, we give a linearithmic-time implementation. Initially the matching M is empty. We maintain a balanced BST T initially consisting of the last member n only. Repeat the following iteration for $p = n - 1$ downto 1.

   - if $T = \emptyset$ then $T \leftarrow \{p\}$;
   - else if $\min_{j \in T} s_j > t_p$ then insert p to T;
   - else extract $q := \max\{j \in T : s_j \leq t_p\}$ from T and add $\{p, q\}$ to M.

   Each iteration can be performed in $O(\log n)$ time, hence the overall running time is $O(n \log n)$. It is straightforward to show that the output M is exactly the leximal matching.

3. Each car requires 2 technicians with 1 specific expertise, so we make 2 vertices v_{i1}, v_{i2} for every car i and connect them with an edge. Create vertices $U = \{u_1, \cdots, u_m\}$ for each technician. A technician may have multiple expertises. Create edges $(u_j, v_{i1}), (u_j, v_{i2})$ whenever technician j has the expertise required by car i. If k cars can be repaired, there will be $m - 2k$ technicians not assigned to any car. We create $m - 2k$ dummy vertices $D = \{d_1, \cdots, d_{m-2l}\}$. Connect between any vertex pair $d \in D$ and $u \in U$. There exists a perfect matching in this graph. If a car is repaired, its 2 vertices will be matched to 2 technicians assigned to 2. If a car is not repaired, its 2 vertices will be matched to each other. The $m - 2k$ unassigned technicians will each be matched to a separate dummy vertex. If k cars can be repaired, the max number of repairable cars is at least k, therefore we can binary search k_{max} from 0 to n by choosing k and try to compute a perfect matching in the current graph. The number of iterations is $O(\log(n))$, the entire process is polynomial time.

4. (a) Start from any unvisited vertex v then select its currently heaviest edge (v, u) to add to path p then delete all its remaining edges. Then move to the next vertex u and repeat. When u has no edges to be selected other than (v, u), we stop and add the current path p to the path set P. After all possible paths have been added to P, for each $p \in P$, we add p's odd edges to matching M_1 and even edges to M_2, and choose the heavier matching from these 2 to output.

Each edge will be inspected only once to find the current max-weighted edge at a vertex. Each edge then will be either added to the path or deleted. Therefore the time complexity is $O(n+m)$.

(b) Suppose M_1 is the matching chosen from P, we have $w(M_1) \geq \frac{1}{2}w(P)$. Let M_{max} be a max weighted matching. It suffices to show that $w(P) \geq w(M_{max})$. Each edge in M_{max} is either in P or not. If an edge e is not in P, it must have been deleted at a vertex v, where another incident edge e' of v is chosen to be added to P with $w(e') \geq w(e)$. All edges in M_{max} that are not in P can be mapped to a distinct edge e' and $w(M_{max})$ can be upper-bounded by $w(P)$.

5. A max weighted matching with exactly $\hat{k}$ matched edges with cover $2\hat{k}$ vertices and leave $n-2\hat{k}$ vertices. Create $n-2\hat{k}$ dummy vertices $D = \{d_1, \cdots, d_{n-2\hat{k}}\}$. For each vertex pair $v \in V$ and $d \in D$ create an edge (v, d) with 0 weight. If a max weighted perfect matching can be computed in the extended graph, it gives a max weighted matching with $\hat{k}$ matched edges. Now we iterate $\hat{k}$ from 1 to k, and compute the max weighted perfect matching. In the case where $k > m$ we stop at m. The number of iterations is $O(m)$ which is polynomial with respect to $|E|$, therefore the entire algorithm finishes in polynomial time.

6. For every edge, make its weight 2 if both endpoints are in X, 1 if only 1 endpoint is in X, 0 otherwise. Compute a max weight matching. By the construction, the weight of matching is exactly the number of vertices in X being matched. The max weighted matching can be computed in polynomial time.