CPS 130 Homework 20 - Solutions

1. [CLRS 23.2-4] Suppose that all edge weights in a graph are integers in the range 1 to $|V|$. How fast can you make Kruskal's algorithm run? What if the edge weights are integers in the range from 1 to W for some constant W?

   Solution: Kruskal's algorithm sorts edges in nondecreasing order by weight. If the edge weights are integers in the range 1 to $|V|$, we can use Counting-Sort to sort the edges in $\Theta(V+E)$ time (recall Counting-Sort correctly sorts n integers in the range 0 to k in $\Theta(n+k)$ time). Then Kruskal's algorithm will run in $O(V+E+V\log V) = O(E+V\log V)$ time.

   If the edge weights are integers in the range from 1 to W for some constant W, we can use Counting-Sort to sort the edges in $\Theta(W+E)$ time and Kruskal's algorithm will run in $O(W+E+V\log V)$ time.

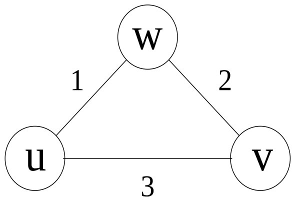
2. [CLRS 23-4]

   Solution:

   (a) Maybe-MST-A removes edges in non-increasing order as long as the graph remains connected, and the resulting T is a minimum spanning tree. To show correctness, let S be a MST of G. Each time we remove an edge e, either $e \in S$ or $e \notin S$. If $e \notin S$, then we can just remove it. If $e \in S$, removing e disconnects S (but not the graph) into two trees. There cannot be another edge connecting these trees with smaller weight than e, because by assumption S is a MST, and if a larger edge existed that connected the trees the algorithm would have removed it before removing e. We know a path exists since the graph is still connected, so it must be there is another edge with equal weight that hasn't been discovered yet, so we can remove e.

   Implementation: Use an adjacency list representation for T. We can sort the edges in $O(E\log E)$ using Merge-Sort. BFS or DFS can be used in $O(V+E)$ to check if $T-\{e\}$ is a connected graph. The edges are sorted once and BFS/DFS is performed E times, once for each edge. The total running time is $O(E\log E + E(V+E)) = O(E^2)$.

   (b) Maybe-MST-B will not give a minimum spanning tree. We prove this by counter-example. Consider the following graph G:

   The MST of G would have edges (w,u) and (v,w) with weight 3. Since Maybe-MST-B takes edges in arbitrary order, it could add edges (u,v) and (v,w) to T, then try to add (w,u) which forms a cycle, then return T (weight 5).

This implementation is similar to Kruskal's algorithm. Use a UNION-FIND data structure to maintain T. For each vertex v we need to MAKE-SET(v). For each edge $e = (u, v)$, if FIND-SET(u) $\neq$ FIND-SET(v) then there is no cycle in $T \cup \{e\}$, and we UNION-SET(u,v).

In total there are V MAKE-SET operations, $2E$ FIND-SET operations, and $V - 1$ UNION-SET operations. Using an improved UNION-FIND data structure we can perform FIND-SET in $O(1)$ and UNION-SET in $O(\log V)$ time amortized to give a running time of $O(V) + O(E) + O(V \log V) = O(V \log V + E)$.

(c) MAYBE-MST-C gives a minimum spanning tree. MAYBE-MST-C adds edges in arbitrary order to T and if a cycle c is detected removes the maximum-weight edge on c. Each time we add an edge e, either we form a cycle c or we do not. Suppose e is added to some tree T' in T and a cycle is c formed. Then we remove a maximum-weight edge e' from c and $T' - e' + e$ is of less or equal weight than T' (because e' is of greater or equal weight than e). If a cycle is not formed we either just add e to some tree in T or e connects two trees in T. In the second case if e is not the smallest weight edge that connects the trees then we haven't discovered it yet, and when we eventually form a cycle we have already shown MAYBE-MST-C performs correctly.

Implementation: Use an adjacency list representation for T. For each edge, we add it to T and then check $T \cup \{e\}$ for cycles. There can be at most one cycle, so we can use DFS to detect the cycle and output it. If there are no cycles, then we are done with this edge. Otherwise, we need to output the cycle, find the maximum-weight edge, and delete it from T.

Adding an edge takes $O(1)$. DFS takes $O(V + E) = O(V)$ in this case (as soon as T has a cycle we break it, so the number of edges in T at any point is no greater than V). Finding the maximum-weight edge on the cycle takes $O(V)$ and deleting an edge is $O(V)$. For each edge we add it to T, perform DFS, and possibly find a cycle, so the total running time is $O(EV)$.