ECE608 CHAPTER 23 PROBLEMS

1) 23.1-1
2) 23.1-3
3) 23.1-6
4) 23.2-1
5) 23.2-7
6) 23-3

ECE608 - Chapter 23 answers

(1) CLR 23.1-1

Let A be the empty set and S be any set containing u but not v. Hence, the cut $(S, V - S)$ respects A and (u, v) is the light edge crossing that cut since it has the minimum weight of all edges in G. By Theorem 23.1 on p. 563 of CLR, the minimum weight edge in a graph G is safe for A, and hence is a member of a minimum spanning tree of G.

(2) CLR 23.1-3

Consider the minimum spanning tree T which contains (u, v) as an edge. Removing the edge (u, v) from the tree must break the tree into two disjoint trees. Say the sets of nodes in these two trees are X and Y respectively. Now imagine the cut of the graph (X, Y). Edge (u, v) crosses this cut and it must be a light edge for this cut, otherwise we could take the lighter weight edge crossing this cut and use it to replace (u, v) in T to get a lighter weight tree T' contradicting the fact that T is an MST. Hence, it must be the case that (u, v) is a light edge for the cut (X, Y).

(3) CLR 23.1-6

Consider a graph G and one of its possible minimum spanning trees T. Lets assume that for any possible cut $(C, V - C)$, there is only one unique light edge crossing that cut. An MST of G must contain this light edge and no other edge that crosses the cut. If it did not contain this edge the vertices $(C, V - C)$ would remain unconnected. If it used any other edge, then a lighter tree could be created by replacing that edge by the light edge, obtaining a lighter tree. Thus it must be the case that for every arbitrary cut, if there is one unique light edge, then each cut contributes one unique edge to the tree. Because there are $|V| - 1$ possible cuts and each contribues one unique edge, the MST is completely defined by the unique light edge from each cut. Thus, the MST must be unique.

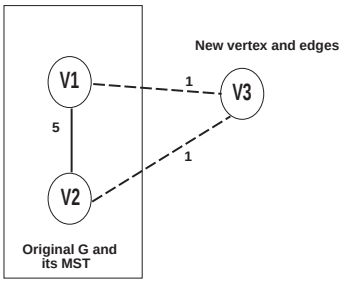
Now consider a graph G and assume that it has a unique MST T. In G a particular vertex w has only one edge that connects it to vertex v. Vertex v has a number of other edges that connect it to some other vertices of the graph. Out of all these other edges the minimum wieght is that of an edge to u. While retaining the uniqueness of T we can assume that $w[(v, u)] = w[(v, w)]$. Now for the cut $(\{w, v\}, V - \{w, v\})$, the edge (v, u) is the unique light edge and so it must be included in T. Also, for the cut $(\{w\}, V - \{w\})$ there is only one edge crossing the cut (w, v) so it must also be included in the MST. So both (w, v) and (u, v) are included in the unique MST. But for the cut $(\{v\}, V - \{v\})$ we have the case that both (w, v) and (u, v) are light edges and they are both included in T.

(4) CLR 23.2-1

Examine the desired minimum spanning tree T, and enlist all its edges. Now append all the remaining edges of the graph to this list. Apply a *stable* sort to this list and use this sorted list of edges to run KRUSKAL's algorithm on. If there are any non-unique light edges for any cut in the graph, they will be sorted such that the one contained in T will be ordered before the remaining ones. Therefore when running KRUSKAL, the edge in T will be selected instead of the rest of them when a light edge for the corresponding cut is required. Because this property is true for every cut, at the end of KRUSKAL's algorithm we will have obtained the desired MST T.

(5) CLR 23.2-7

The update of the MST when a new vertex and incident edges are added requires $\Omega(V)$ time. Consider $G = (V, E)$ with $V = \{v_1, v_2, \ldots, v_n\}$ and $E = \{(v_1, v_2), (v_2, v_3), (v_3, v_4), \ldots, (v_{n-1}, v_n), (v_n, v_1)\}$. To find the MST in this case (a simple cycle), we can simply omit the maximum weight edge. Now, consider adding a vertex v_{new} and incident edges to some arbitrary connected graph. Let e_{min} be the minimum weight edge incident to that new vertex v_{new}. Clearly e_{min} will be in the new MST; call it MST'. However, for each of the remaining edges incident to v_{new}, it is also necessary to determine whether adding that edge and removing an original edge will lead to a lower cost spanning tree. For example, consider the very simple example where G was originally comprised of two vertices ($V1$ and $V2$) and a single edge $(V1, V2)$ for which the MST is that edge, and we add a third vertex $V3$ and edges between $V3$ and each of the other vertices. In this case the MST should be $\{(V1, V3), (V2, V3)\}$.



To handle this, one must look at all the edges in the cycle that adding each new edge to MST' creates and remove the maximum weight edge. This process must examine all of the old MST edges in the general case, so $\Omega(V)$ time is the best we can do.

(6) CLR 23-3

(a) Suppose that an MST T of a graph G is not a bottle-neck spanning tree. Say that the maximum wieght edge in T is (u, v). Imagine removing (u, v), the resulting disconnected set of vertices can be viewed as a cut of the graph. It must be the case that (u, v) must be a light edge for this cut otherwise T would not be an MST. Because every possible MST must select one edge for this particular cut we guaranteed that every possible MST must contain (u, v) or another another edge for this cut with no

less weight then (u, v). Thus by contradiction, we cannot construct another spanning tree in which the heaviest edge is lighter than (u, v).

(b) Remove all edges from the graph G. Now for each edge, see if it's weight is less than or equal to b. If so, then add it back to the graph. If the resulting graph G' is connected then we know that it is possible to construct a spanning tree while using no edge with weight larger than b. This algorithm runs in linear time $O(E)$.

(c) Run the algorithm in part(b) repeatedly for a decreasing value of b each time. Start with the value of b set to the heaviest edge in the set of edges, and repeatedly run the algorithm for a smaller and smaller value of b until the algorithm returns a NO as the answer. We then know the bottleneck value b for this graph.

Obtain the set of all edges with weight less than or equal b. From this set randomly pick an edge and similar to KRUSKAL's algorithm, run two instances of FIND and one instance of UNION for that edge. If we use an implementation of FIND-UNION data structure that uses path-compression and union-by-rank heuristics we know that we can complete k such operations in $k\alpha(V)$ time. $\alpha(n)$ is a very slow growing function and can be approximated by $O(1)$ for values of n even as large as 10^{80}. Thus we perform $O(E)$ disjoint set operations in the worst case (each costing $O(1)$) and obtain a bottle-neck spanning tree.