Problem 1 [DPV] Problem 7.17 (e)

Give an efficient algorithm to identify all bottleneck edges in a network. **Do as fast as possible**. *(Hint: Start by running the usual network flow algorithm, and then examine the residual graph.)*

Answer: Recall that an edge in the graph G is call a *bottleneck edge* if increasing its capacity results in an increase in the maximum flow. We start by finding a maximum flow f on G. Then consider an egde $uv \in E(G)$. Increasing the capacity of uv results in an increase in maximum flow value if and only if there exist a path from s to u and a path from v to t in G^f. Intuitively, the condition implies that more flow can be sent from s to u and from v to t.

Therefore, our algorithm for finding bottleneck edges is as follows:

1. Find maximum flow f on G.
2. Do a DFS from s in G^f. Let S be the set of vertices reachable from s using a path in G^f.
3. Do a DFS from t in the reverse graph of G^f. Let T be the set of vertices reachable from t using a path in the reverse graph of G^f.
4. For each $(u, v) \in E(G)$, return (u, v) if $u \in S$ and $v \in T$.

Since steps 2 to 4 take $O(|E| + |V|)$ time, the running time is donimated by the running time of the maximum flow algorithm in step 1.

Problem 2 [DPV] Problem 7.18 (a) (b)

Please reducing (a) and (b) to the original max flow problem:

(a) There are many sources and many sinks, and we wish to maximize the total flow from all sources to all sinks.

(b) Each vertex also has a capacity on the maximum flow that can enter it.

Answer:

(a) Create a supersource and connect it to the sources using edges of infinite capacity. Similarly, create a supersink and connect all sinks to it using edges of infinite capacity. Run max flow on the new graph.

(b) For each vertex v, replace it with two new vertices v_1 and v_2 in the following way. Let all incoming edges of v come to v_1 and let all outgoing edges of v come from v_2. Finally, create a directed edge from v_1 to v_2 with capacity equal to the capacity of v. Run max flow on the new graph.

Problem 3 [DPV] Problem 5.22 (a)

Prove the following property carefully:
Pick any cycle in the graph, and let e be the heaviest edge in that cycle. Then there is a minimum spanning tree that does not contain e.

Answer: Consider a cycle C and let e be the heaviest edge in C. We will show that for any spanning tree T containing e, there exits a spanning tree T' such that T' does not contain e and the weight of T' is at most the weight of T. The property will immediately follow.

Since T is a spanning tree and $e \in T$, $T - e$ consists of two disjoint trees, namely T_1 and T_2. Note that $C - e$ is a path between a vertex in T_1 and a vertex T_2. Therefore, there must be an edge $e' \in C - e$ connecting T_1 and T_2. It follows that $T' = T_1 \cup T_2 \cup \{e'\}$ is a spanning tree that does not contain e. Moreover, since e is the heaviest edge in C, the weight of T' is at most the weight of T.

Problem 4 [DPV] Problem 5.9 (e)

Claim: The statement is true.

Proof:

Let $e = (u, v)$ be an edge in a given MST T of $G = (V, E)$. Removing e from T will break T into 2 connected components (trees) T_1 and T_2. Consider the cut $S, V - S$ where $S = \{v : v \in T_1\}$ and $V - S = \{v : v \in T_2\}$.

Assume there exists another edge e' across the cut and $w(e') < w(e)$.
Then we can construct another tree $T' = T - e + e'$. T' is a spanning tree and it has less cost than T, which contradicts to the fact that T is a MST. Thus such e' doesn't exist, which proves the statement.