Solutions 5.

5-1: Suppose that \( e \) belongs to a minimal spanning tree, \( T \). Delete \( e \) from \( T \), and obtain a forest of two components, \( A \) and \( B \). Since \( G \) has a cycle containing \( e \), there is another edge \( f \) in the cycle in \( G \) that connects \( A \) to \( B \). By the choice of \( e \), \( f \) has larger cost. Replace \( e \) by \( f \) in \( T \), and obtain a smaller cost spanning tree, a contradiction.

5-2: Suppose there are two minimal weight spanning trees, \( T_1 \) and \( T_2 \). Consider all those edges of \( T_1 \) and \( T_2 \) that are only in \( T_1 \) or only in \( T_2 \). Among these edges, let \( e \) be the largest cost edge. We may assume that \( e \) is in \( T_1 \) and is not in \( T_2 \). Deleting \( e \) from \( T_1 \) disconnects \( T_1 \) into two components, \( A \) and \( B \). Since \( T_2 \) is a spanning tree, there is at least one edge in \( T_2 \) connecting \( A \) and \( B \). Let \( f \) be the smallest cost such edge. Now, \( f \) is in \( T_2 \) and not in \( T_1 \), so, by the choice of \( e \), \( f \) has smaller cost than \( e \). On the other hand, if in \( T_1 \), we replace \( e \) by \( f \), we obtain a smaller cost spanning tree, thus, \( T_1 \) is not of minimal cost, a contradiction.

5-3: Yes, the algorithm works. Add a large positive number to all edge costs. Then the minimal weight spanning tree produced by Kruskal’s algorithm will be the same.

5-4: The resulting tree:

5-5: The proof is similar to the proof the Kruskal’s algorithm produces a minimal weight spanning tree. We will show it in the case when the weights are all different. The case when some weights might be equal requires small modifications.

Suppose for a contradiction that there is a minimal weight spanning tree \( T \) whose weight is less than that of the tree found by Prim’s algorithm, which we will denote by \( P \).

Number the edges of \( P \) as \( e_1, e_2, \ldots \) by the order in which they are selected by the algorithm. Let \( i \) be the smallest index for which \( e_i \) is not in \( T \). Let \( A \) denote the tree whose edges are \( e_1, \ldots, e_{i-1} \). So, the edges in \( A \) are shared by \( P \) and \( T \). Clearly, one of the two endpoints of \( e_i \), denote it by \( u \) is in \( A \), and the other one, denote it by \( v \), is not.

Now, \( T \) is connected, so let \( p \) be a path in \( T \) from \( u \) to \( v \). Let \( f \) be the first edge of the path \( p \) that connects a vertex of \( A \) to a vertex outside of \( A \). By the algorithm, \( e \) has smaller cost than \( f \).

On the other hand, \( p \cup \{ e_i \} \) is a cycle, so, by replacing \( f \) by \( e_i \) in \( T \), we obtain a tree which is of lower cost than \( T \), a contradiction.

5-6: The answer is \( |V(T_1)| \ldots |V(T_k)| \cdot |V(T)|^{k-2} \).
We will use the following fact:
Let \( p(x_1, \ldots, x_n) \) be the following polynomial of \( n \) variables:
\[
p(x_1, \ldots, x_n) = \sum_T x_1^{d_T(1)-1} \cdots x_n^{d_T(n)-1},
\]
were the sum is taken over all labeled trees \( T \) on the set \([n]\), and \( d_T(i) \) denotes the degree of \( i \), that is the number of edges incident with \( i \) in \( T \).

We claim without proof here that
\[
p(x_1, \ldots, x_n) = (x_1 + \ldots + x_n)^{n-2}.
\] (1)

Now, consider the exercise. Contract \( T_1, \ldots, T_k \) to one point each. Then any labeled tree \( F \) on \( V \) becomes a labeled tree \( T \) on \( k \) vertices. We need to compute how many trees \( F \) are mapped to the same \( T \).

Let us try to reconstruct \( F \) from a fixed \( T \). Consider an edge of \( T \) between two vertices \( i \) and \( j \). There are \( |V(T_i)| \cdot |V(T_j)| \) ways of realizing this edge in \( F \). Thus, the number of trees \( F \) on \( V \) that are mapped to the same \( T \) is
\[
\prod_{e=(i,j): \text{an edge of } T} |V(T_i)| \cdot |V(T_j)|,
\]
which is
\[
\prod_{i=1}^{k} |V(T_i)|^{d_T(i)}.
\]

So, the total number of trees \( F \) on \( V \) is
\[
\sum_T \prod_{i=1}^{k} |V(T_i)|^{d_T(i)},
\]
which, by (1) is \( |V(T_1)| \ldots |V(T_k)| \cdot |V(T)|^{k-2} \).

5-7: Take an arbitrary tree \( T \) on the vertices \( v_1, \ldots, v_k \). Any forest of \( k \) components where \( v_1, \ldots, v_k \) are in distinct components, together with \( T \) is a tree on \( n \) vertices, and vica-versa. So, we need to compute how many labeled trees on \( n \) vertices contain a given labeled subtree on a given subset of \( k \) vertices. Using the previous exercise, we obtain \( k \cdot n^{n-k-1} \).