Problem 2.9. *Suppose that $G = (V, E)$ is not connected. Show that in this case, when G is given to Kruskal's algorithm as input, the algorithm computs a spanning forest of G. First, precisely define a connected component and a spanning forest.*

Solution: Given an undirected graph $G = (V, E)$, a connected component $C = (V_c, E_c)$ of G is a nonempty subset V' of V (along with its included edges) such that for all pairs of vertices $u, v \in V'$, there is a path from u to v (which we'll state as "u and v are connected"), and moreover for all pairs of vertices x, y such that $x \in V'$ and $y \in V - V'$, x and y are *not* connected (i.e. there is no path from x to y). We can make a few quick obvservations about connected components:

(i) The connected components of any graph comprise a partition of its edge and vertex sets, as connectedness is an equivalence relation.

(ii) Given any edge, both of its endpoints are in the same component, as it defines a path connecting them.

(iii) Given any two vertices in a connected component, there is a path connecting them. Similarly, any two vertices in different components are necessarily not connected.

(iv) Given any path, every contained edge is in the same component.

A spanning forest is a collection of spanning trees - one for each connected component. That is, an edge set $F \subseteq E$ is a spanning forest of $G = (V, E)$ if and only if:

(i) F contains no cycles.

(ii) $(\forall u, v \in V)$, F connects u and v if and only if u and v are connected in G.

Let $G = (V, E)$ be a graph that is not connected. That is, G has > 1 components. Let T_i denote the state of T, in Kruskal's, after i iterations. Let $C = (V_c, E_c)$ be a component of G. We will use the following loop invariant as proof that Kruskal's results in a spanning forest for G:

$$\text{The edge set } T_i \cup \{e_{i+1}, \ldots, e_m\} \text{ connects all nodes in } V_c \tag{1}$$

The basis case clearly works; $T_0 \cup \{e_1, \ldots, e_m\} = E$. Every vertex in V_c is connected in G, and we have every edge in G at our disposal.

Assume that $T_{i-1} \cup \{e_i, \ldots, e_m\}$ connects all nodes in V_c.

Case 1: e_i is not in E_c. Clearly e_i has no effect on the connectedness of V_c, as any path in C must be a subset of E_c.

Case 2: $e_i \in E_c$ and $T_{i-1} \cup \{e_i\}$ contains a cycle. Let u, v be nodes adjacent to e_i. T_{i-1} does not contain a cycle by construction, so e_i completes a cycle in $T_{i-1} \cup \{e_i\}$. Thus, there is already a path (u, v) in T_{i-1}, which can be

used to replace e_i in any other path. Therefore, $T_{i-1} \cup \{e_{i+1}, \dots, e_m\}$ connects everything that was connected by $T_{i-1} \cup \{e_i, \dots, e_m\}$, so the assignment of $T_i = T_{i-1}$ works.
Case 3: $e_i \in E_c$ and $T_{i-1} \cup \{e_i\}$ does not contain a cycle. Then $T_i = T_{i-1} \cup \{e_i\}$, so $T_i \cup \{e_{i+1}, \dots, e_m\} = T_{i-1} \cup \{e_i, \dots, e_m\}$, so the loop invariant holds.

We have shown through induction that the loop invariant (1) holds. Note that C was an arbitrary connected component, so T_m for each component $C = (V_c, E_c)$ in G, T_m connects every node in V_c. Obviously, if any two nodes in V are not connected in G, T does not connect them; doing so would require edges not in E. Therefore, T_m meets both conditions imposed on a spanning forest above. $\square$