

9.2 Kruskal's Algorithm

In the previous section, we considered the greedy algorithm that “grows” a minimum spanning tree through a greedy inclusion of the nearest vertex to the vertices already in the tree.

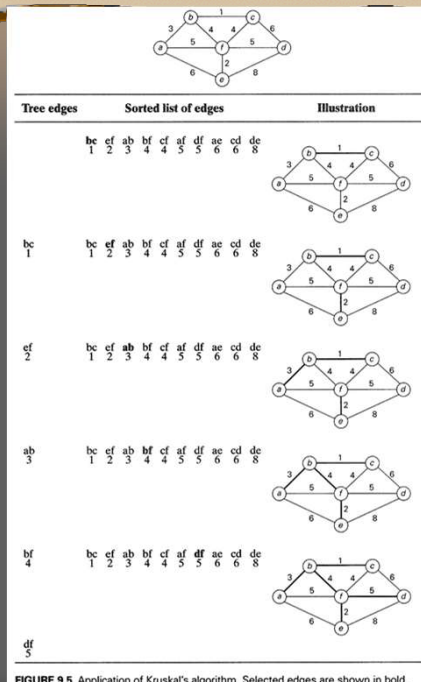
Remarkably, there is another greedy algorithm for the minimum spanning tree problem that also always yields an optimal solution. It is named *Kruskal's algorithm* after Joseph Kruskal, who discovered this algorithm when he was a second-year graduate student [Kru56]. Kruskal's algorithm looks at a minimum spanning tree of a weighted connected graph $G = (V, E)$ as an acyclic subgraph with $|V| - 1$ edges for which the sum of the edge weights is the smallest.

(It is not difficult to prove that such a subgraph must be a tree.)

Consequently, the algorithm constructs a minimum spanning tree as an expanding sequence of subgraphs that are always acyclic but are not necessarily connected on the intermediate stages of the algorithm.

ALGORITHM *Kruskal*(G)

//Kruskal's algorithm for constructing a minimum spanning tree

//Input: A weighted connected graph $G = (V, E)$ //Output: E_T , the set of edges composing a minimum spanning tree of G
sort E in nondecreasing order of the edge weights $w(e_{i_1}) \leq \dots \leq w(e_{i_{|E|}})$ $E_T \leftarrow \emptyset$; $ecounter \leftarrow 0$ //initialize the set of tree edges and its size $k \leftarrow 0$ //initialize the number of processed edgeswhile $ecounter < |V| - 1$ do $k \leftarrow k + 1$ if $E_T \cup \{e_{i_k}\}$ is acyclic $E_T \leftarrow E_T \cup \{e_{i_k}\}$; $ecounter \leftarrow ecounter + 1$ return E_T 

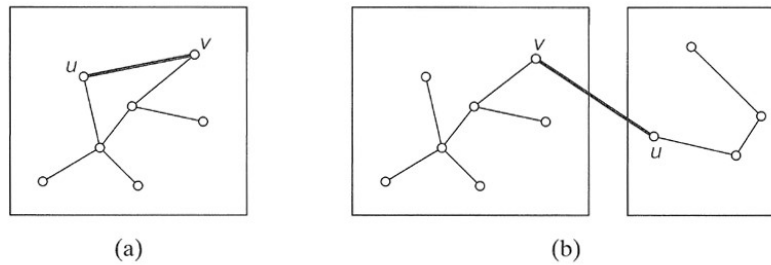


FIGURE 9.6 New edge connecting two vertices may (a) or may not (b) create a cycle.

Fortunately, there are efficient algorithms for doing so, including the crucial check for whether two vertices belong to the same tree. They are called *unionfind* algorithms. We discuss them in the following subsection. With an efficient union-find algorithm, the running time of Kruskal's algorithm will be dominated by the time needed for sorting the edge weights of a given graph. Hence, with an efficient sorting algorithm, the time efficiency of Kruskal's algorithm will be in $O(|E| \log |E|)$.