8.1 Minimum Spanning Trees

Given an undirected and connected graph that has some numeric weight assigned to each edge, the minimum spanning tree problem seeks to create a spanning tree such that the sum of the weights selected for the edges of the spanning tree is minimized. We're assuming the weights here are non-negative. There are two classical algorithms for this problem. They include Kruskal's algorithm and Prim's algorithm.

---

procedure KRUSKAL(Graph $G = \{V(G), E(G), W(G)\}$)
▷ Note: $W(G)$ is a numeric *weight* for each edge in $E(G)$
  $V(T) \leftarrow V(G)$ ▷ Spanning tree T will have all vertices of G
  $E(T) \leftarrow \emptyset$ ▷ Edge set of T initially null
  sort $W(G)$ and correspondingly $E(G)$ by nondecreasing values in $W(G)$
  for all $w \in W(G), e \in E(G)$ **do**
    if numComponents$(T + e) <$ numComponents(T) **then**
      $E(T) \leftarrow E(T) + e$
    if numComponents$(T) = 1$ **then**
      break
  return T

Kruskal initializes a *spanning forest* T to include all vertices in the original graph G. A minimum weight edge is added as long as it reduces the number of components in T. The algorithm terminates once there is only a single component, i.e. T is now a spanning tree.

We can prove correctness of Kruskal's algorithm.

---

procedure PRIM(Graph $G = \{V(G), E(G), W(G)\}$)
▷ Note: $W(G)$ is a numeric *weight* for each edge in $E(G)$
  $V(T) \leftarrow \emptyset$ ▷ Spanning tree's vertices initially null
  $E(T) \leftarrow \emptyset$ ▷ Edge set of T initially null
  $V(T) \leftarrow$ randomSelect$(V(G))$ ▷ Randomly select one vertex from G
  while $V(T) \neq V(G)$ **do**
    $e \leftarrow \min(W(u, v)) \in E(G) : u \in V(T), v \notin V(T)$
    ▷ Minimum weight edge in G with only one vertex in T
    $E(T) \leftarrow E(T) + e$
    $V(T) \leftarrow V(T) + v$
  return T

Prim initializes by randomly selecting a vertex in G and adding it to T. Iteratively, the minimum-weight edge between a vertex in T and a vertex not yet in T is added to T along with the associated vertex. The algorithm terminates when T has all vertices in G. You can observe that both algorithms are rather similar.

8.2 Shortest Paths

Again, given an undirected and connected graph that has some numeric weight assigned to each edge, the shortest paths problem seeks to find the minimum distance between a vertex u and all other vertices $v \in V(G)$. Distance here is defined as the sum of the weights along a u, v-path. A classic algorithm for computing these distances is Dijkstra's algorithm.

---

procedure DIJKSTRA(Graph $G = \{V(G), E(G), W(G)\}$, vertex u)
▷ Finding all distances from u
  for all $v \in V(G)$ **do**
    $D(v) \leftarrow \infty$ ▷ Distances from u
  $D(u) \leftarrow 0$
  $S \leftarrow V(G)$ ▷ Unvisited set
  while $S \neq \emptyset$ **do**
    $w \leftarrow \min(D(v), v \in S)$
      ▷ Current vertex considered has minimum distance in unvisited set
    for all $x \in N(w)$, $x \in S$ **do**
      $t \leftarrow W(w, x)$ ▷ Weight of edge between w and x
      if $D(w) + t < D(x)$ **then**
        $D(x) \leftarrow D(w) + t$
    $S \leftarrow S - w$ ▷ Remove w from unvisited set
  return D

The algorithm initializes all distances $D(v \in V(G))$ to infinity except for the root vertex u, which has distance zero. The algorithm tracks an *unvisited* set of vertices, where on each iteration a vertex w with the minimum distance from u is selected. All edges adjacent to this vertex are examined, and if a distance from w to a neighbor x plus the distance from u to w is less than what is currently stored in the distance array for u to x, the distance array is updated for x. After examining all neighbors, w removed from the unvisited set. The algorithm terminates once the unvisited set is empty.

We can also prove correctness of Djikstra's algorithm.